МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ ІМЕНІ СЕМЕНА КУЗНЕЦЯ

Тарасов О. В.
Федько В. В.

КЛІЄНТ-СЕРВЕРНІ ТЕХНОЛОГІЇ СКБД ORACLE. МОВА SQL ORACLE

Навчально-практичний посібник
для самостійної підготовки студентів
з навчальної дисципліни
"Організація баз даних та знань"

Харків. ХНЕУ ім. С. Кузнеця, 2015

УДК 004.65(075)
ББК 32.973-018.2я7
Т 19

Рецензенти: докт. техн. наук, професор, завідувач кафедри інформаційних управляючих систем Харківського національного університету радіоелектроніки *Левикін В. М.*; канд. техн. наук, доцент кафедри геоінформаційних систем, оцінки землі та нерухомого майна Харківського національного університету міського господарства імені О. М. Бекетова *Поморцева О. Є.*

Рекомендовано до видання рішенням вченої ради Харківського національного економічного університету імені Семена Кузнеця.
Протокол № 6 від 19.12.2014 р.

Тарасов О. В.

Т 19 Клієнт-серверні технології СКБД Oracle. Мова SQL Oracle : навчально-практичний посібник для самостійної підготовки студентів з навчальної дисципліни "Організація баз даних та знань" для студентів напряму підготовки 6.050101 "Комп'ютерні науки" / О. В. Тарасов, В. В. Федько. – Х. : ХНЕУ ім. С. Кузнеця, 2015. – 384 с. (Укр. мов.)
ISBN 978-966-676-580-5

Розглянуто питання роботи з СКБД Oracle за допомогою утиліти SQL*PLUS та інтегрованого середовища розробки – Oracle SQL Developer.

Наведено велику кількість прикладів використання команд мови SQL Oracle для створення, модифікації та вибірки інформації з бази даних, а також розширення мовних конструкцій SQL для здійснення агрегацій у сховищах даних. Розглянуто питання забезпечення безпеки даних у СКБД Oracle та використання словника даних Oracle для отримання необхідної інформації про об'єкти бази даних.

Рекомендовано для студентів напряму підготовки 6.050101 "Комп'ютерні науки".

УДК 004.65(075)
ББК 32.973-018.2я7

ISBN 978-966-676-580-5

Вступ

У сучасному світі інформаційні системи є, безумовно, одними з найважливіших інструментів, що використовуються практично в усіх сферах людської діяльності. Головною складовою таких систем є база даних (БД), що є динамічною інформаційною моделлю конкретної предметної області, тобто певної частини реального світу. Використання баз даних є незамінним інструментом, який забезпечує значне підвищення якості та оперативності економічних розрахунків, підвищення ефективності процесу обґрунтування економічних рішень з метою стабільного функціонування та сталого розвитку організацій та підприємств.

До 80 % світового ринку комерційних систем керування базами даних (СКБД) охоплюють три найбільш поширені та потужні системи, а саме Oracle, DB2 та Microsoft SQL Server. З цих трьох систем слід звернути особливу увагу на СКБД Oracle, розробники якої постійно вдосконалюють та розширюють функціональні можливості системи. Тому логічно, що у навчальній дисципліні "Організація баз даних та знань" певна увага приділяється вивченню одного з світових флагманів систем керування базами даних Oracle.

Даний навчально-практичний посібник призначений, переважно для студентів напряму підготовки 6.050101 "Комп'ютерні науки" і орієнтований на вивчення та закріплення теоретичного матеріалу з навчальної дисципліни "Організація баз даних та знань".

Його розроблено відповідно до програми навчальної дисципліни "Організація баз даних та знань" [50] і безпосередньо охоплює такі теми навчальної дисципліни: "Середовище розробки і виконання в Oracle SQL*Plus. Інтегроване середовище розробки Oracle SQL Developer", "Особливості мови DDL та DML у СКБД Oracle. Використання стандартних функцій у СКБД Oracle, "Керування доступом у СКБД Oracle. Словник даних Oracle".

Крім того, наведений матеріал перегукується з такими темами, як: "Реляційна модель даних", "Мова SQL та огляд її можливостей", "Цілісність даних", "Захист даних у СКБД".

Освоєння цих важливих питань дозволяє студентам оволодіти такими вміннями в рамках компетентностей, визначених Галузевим

стандартом вищої освіти України з напряму підготовки 6.050101 "Комп'ютерні науки" [7].

Соціально-особистісні компетентності:

- уміння знаходити нові, нешаблонні рішення і засоби їх здійснення, діяти протягом тривалого часу, незважаючи на труднощі, проявляти гнучкість у подоланні перешкод;
- здатність до генерації нових ідей і варіантів розв'язання задач, до комбінування та експериментування, до оригінальності, конструктивності, економічності та простих рішень;
- уміння виявляти недоліки та помилки та виправляти їх, вирішувати суперечності.

Загальнонаукові компетентності:

- уміння застосовувати базові знання стандартів в області інформаційних технологій під час розроблення та впровадження інформаційних систем і технологій.

Інструментальні компетентності:

- уміння обробляти отримані результати, аналізувати, осмислювати та подавати їх, обґрунтовувати запропоновані рішення на сучасному науково-технічному рівні.

Загальнопрофесійні компетентності:

- підготовленість до розроблення нових математичних методів, ефективних алгоритмів та методів реалізації функцій інформаційних систем і технологій у прикладних галузях;
- здатність до програмної реалізації алгоритмів розв'язання задач, розроблення прикладного програмного забезпечення інформаційних систем.

Спеціалізовано-професійні компетентності:

- уміння використовувати основні поняття, ідеї та методи фундаментальної математики під час розв'язання конкретних задач в області комп'ютерних наук;
- уміння розробляти, аналізувати та застосовувати ефективні алгоритми для розв'язання професійних завдань в області комп'ютерних наук;
- уміння проектувати логічні та фізичні моделі баз даних, запити до них та використовувати різноманітні системи керування базами даних;

• уміння моделювати системи та процеси, стани та поведінку складних об'єктів інформатизації в процесі розроблення інформаційних систем і технологій.

Навчально-практичний посібник складається з трьох розділів, глосарію термінів, переліку використаної літератури та шести додатків.

Перший розділ присвячений ознайомленню з порядком інсталяції СКБД Oracle на комп'ютер користувача та порядком роботи з системою за допомогою програм SQL*Plus та інтегрованого середовища розробки Oracle SQL Developer. Головна увага приділяється знайомству з командами відповідних програм, а також використанню командних файлів для автоматизації процесу роботи з системою Oracle. Крім того розглядаються приклади використання команд SQL*Plus для форматування звітів, що формуються за даними у системі.

У другому розділі розглядаються питання використання команд мови SQL Oracle, для створення, модифікації та вибірки інформації з бази даних. Додатково наводяться мовні конструкції SQL для здійснення агрегації у сховищах даних, а також використання стандартних функцій СКБД Oracle для вирішення практичних задач. Матеріал розділу містить велику кількість практичних завдань та детальних пояснень щодо їхнього розв'язання.

Третій розділ присвячений вивченню засобів безпеки даних у СКБД Oracle та використання словника даних Oracle для отримання необхідної інформації про об'єкти бази даних. Роз'яснюється сенс системних та об'єктних привілеїв та їх відмінність у системі Oracle. На прикладах пояснюється використання мовних засобів SQL Oracle для створення ролей та користувачів у базі даних, а також для надання та скасування як системних, так і об'єктних привілеїв. Крім того наводиться опис системних подань Oracle та приклади отримання інформації зі словника за допомогою запитів на мові SQL.

Наприкінці кожного з розділів подано опис відповідної лабораторної роботи, самостійне виконання якої дозволить студенту закріпити теоретичний матеріал відповідного розділу та оволодіти практичними навичками роботи з СКБД Oracle.

У додатках наведена довідникова інформація щодо скриптів навчальної бази даних, додаткових параметрів команди Create Table, а також системних привілеїв СКБД Oracle та основних подань словника даних.

Вивчення матеріалу навчального посібника має відбуватися в такому порядку: спочатку прочитати матеріал, який подано у відповідному розділі; потім спробувати відповісти на запитання і виконати завдання, що наведені у якості прикладів. Після знайомства з теоретичним матеріалом розділу виконати лабораторну роботу – спочатку загальні завдання, а потім додаткові. Під час виконання лабораторної роботи слід звернути увагу на добір завдань. Він має бути посильним для розуміння студентом. З цією метою кожний студент самостійно обирає рівень складності з поміж таких:

1) фронтальний;

2) індивідуальний;

3) компетентнісний.

Якщо обрано фронтальний рівень, то студент виконує завдання базового рівня, детально описаного в інструкції. За його виконання студент отримує 60 балів за стобальною системою оцінювання.

Із метою випробування своїх сил і підвищення оцінки студент може самостійно розв'язати ще кілька задач, частина з яких репродуктивного, а інші – креативного типу. За правильне їх розв'язання додається ще до десяти балів. До отриманої суми балів студент може додати ще 10 балів, якщо самостійно запропонує і розв'яже оригінальну задачу за темою, що вивчається. Ця задача має бути з предметної області навчання чи майбутньої професії студента. Загальна оцінка за цим рівнем не перевищує 80 балів.

У разі вибору індивідуального рівня студент ознайомлюється з інструкцією щодо виконання завдання базового рівня і розв'язує аналогічну задачу з предметної області, визначеної індивідуальним завданням. За виконання такого індивідуального завдання студент отримує 70 балів. Ще 10 балів він може отримати, якщо адаптує до предметної області обраного варіанта задачі, які подані в інструкції, і розв'яже їх. Подібно до фронтального рівня, студент може додати ще 12 балів до отриманої суми балів, якщо сформулює та розв'яже оригінальну задачу за темою, що вивчається. Загальна оцінка за цим рівнем не перевищує 92 балів.

На компетентністному рівні студент демонструє можливість самостійно ставити та розв'язувати задачі за темою, що вивчається з предметної області навчання чи майбутньої професії. Спочатку він формулює і розв'язує задачу аналогічну базовій (74 бали), потім – аналогічну додатковим задачам (ще 11 балів) і насамкінець – оригінальну задачу (до 9 балів). Загальна оцінка за цим рівнем може досягати 100 балів.

Тема 1. Середовище розробки та виконання в Oracle SQL*Plus. Інтегроване середовище розробки Oracle SQL Developer

Мета розділу – ознайомитися з порядком інсталяції системи керування базами даних Oracle та програмними засобами адміністрування та роботи з СКБД Oracle, утилітою SQL*Plus та інтегрованим середовищем розробки Oracle SQL Developer.

Основні питання

Установка СКБД Oracle – роз'яснюється порядок інсталяції та налаштування СКБД Oracle 11g Express Edition на комп'ютер користувача.

Утиліта SQL*Plus Oracle – розглядаються основні функції та команди утиліти для роботи з СКБД Oracle.

Командні файли – пояснюється сенс командних файлів та наводяться приклади їх використання з метою автоматизації виконання задач.

Форматування звітів – розглядаються приклади використання команд SQL*Plus для форматування звітів, що формуються за даними у системі.

Інтегроване середовище розробки Oracle SQL Developer – описується порядок підключення та роботи з СКБД Oracle за допомогою інтегрованого середовища розробки Oracle SQL Developer.

1.1. Система керування базами даних Oracle на ринку корпоративних СКБД

На світовому ринку корпоративних систем керування базами даних (СКБД) домінує традиційна трійка продуктів: IBM DB2, Microsoft SQL Server і Oracle. Упродовж довгих років вони охоплюють понад 80% продажів на світовому ринку СКБД [40].

За статистичними даними, на ринках країн колишнього СНД провідне становище займає саме Oracle, бо охоплює понад 60% всього ринку серед інших СКБД і біля 30% світового ринку СКБД. Система випускається однойменною компанією та має багато різноманітних версій та типів.

Компанія Oracle була заснована нинішнім президентом компанії Леррі Елісоном разом з Робертом Майнором та Едом Уотсом у 1977 році у штаті Каліфорнія США під назвою Software Development Laboratories (SDL). У 1979 році компанія була перейменована у Relational Software, Inc (RSI), а згодом, у 1982 році, – у Oracle Systems Corporation. Остаточно у 1995 році назва компанії була змінена на Oracle.

Перша реляційна СКБД фірми базувалася на моделі IBM System/R і була першою системою, в якій використовувалась мова SQL, розроблена фірмою IBM [40; 76].

На сьогодні СКБД Oracle підтримує понад вісімдесят варіантів операційного середовища в широкому діапазоні, включаючи мейнфрейми IBM, міні-комп'ютери DEC VAX, UNIX, Windows та більшість інших платформ.

Остання версія СКБД Oracle Database 12c пропонує нову мультиарендну архітектуру, яка спрощує розгортання хмар баз даних та керування ними. Передові технології, що впроваджуються Oracle, у поєднанні з більшою доступністю, безпекою та підтримкою обробки великих обсягів даних роблять Oracle Database 12c ідеальною платформою для розгортання приватних і загальнодоступних хмарних середовищ [68].

Водночас СКБД є досить дорогою системою. Однак для навчальних цілей корпорацією Oracle була розроблена безкоштовна експрес-версія, яка є повнофункціональною системою, що дозволяє у повній мірі оволодіти роботою з СКБД Oracle як на рівні мови SQL, так і її процедурного розширення PL/SQL.

Перш ніж, як починати роботу з системою, її слід встановити на свій комп'ютер або мати вже встановлений серверний варіант системи. Порядок установки (інсталяції) системи Oracle Database 11g Express Edition, який можна безкоштовно завантажити з офіційного сайту корпорації Oracle за адресою [67], розглянуто у підрозділі 1.2.

Запитання і завдання

1. Назвіть системи керування базами даних, які домінують на світовому ринку корпоративних СКБД.
2. Опишіть основні можливості СКБД Oracle.
3. Яка остання версія СКБД Oracle вам відома? Що для неї характерне?

1.2. Інсталяція Oracle 11g Express Edition

Після завантаження інсталяційних файлів системи на комп'ютер, треба запустити на виконання файл setup.exe і активувавти процес установки системи. На першому кроці виконується підготовчий етап – витяг та декомпресія файлів (рис. 1.1, 1.2)

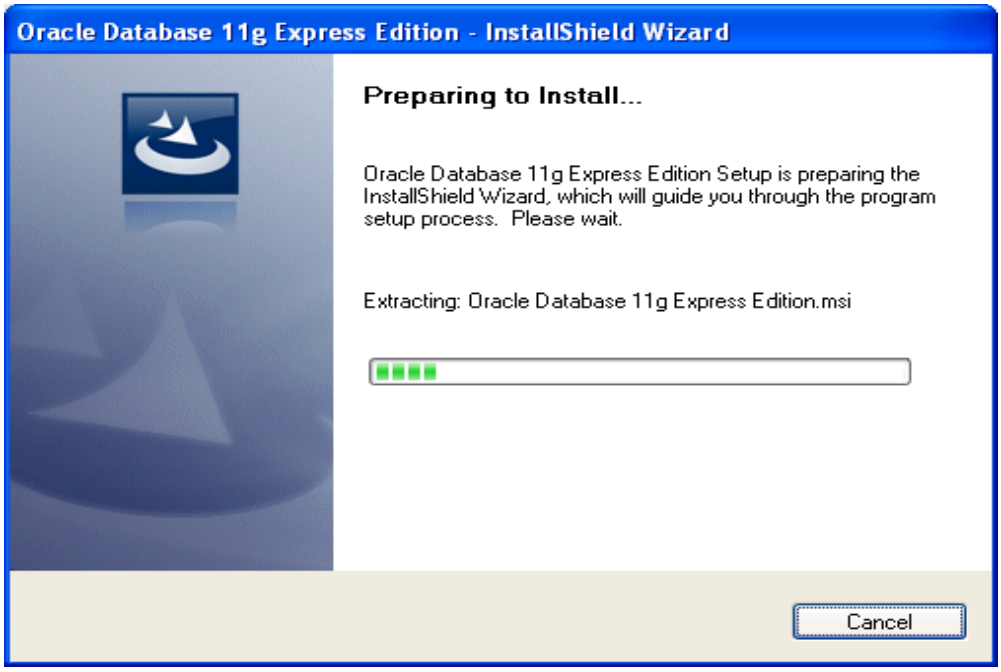


Рис. 1.1. **Установка СКБД Oracle** (витяг файлів)

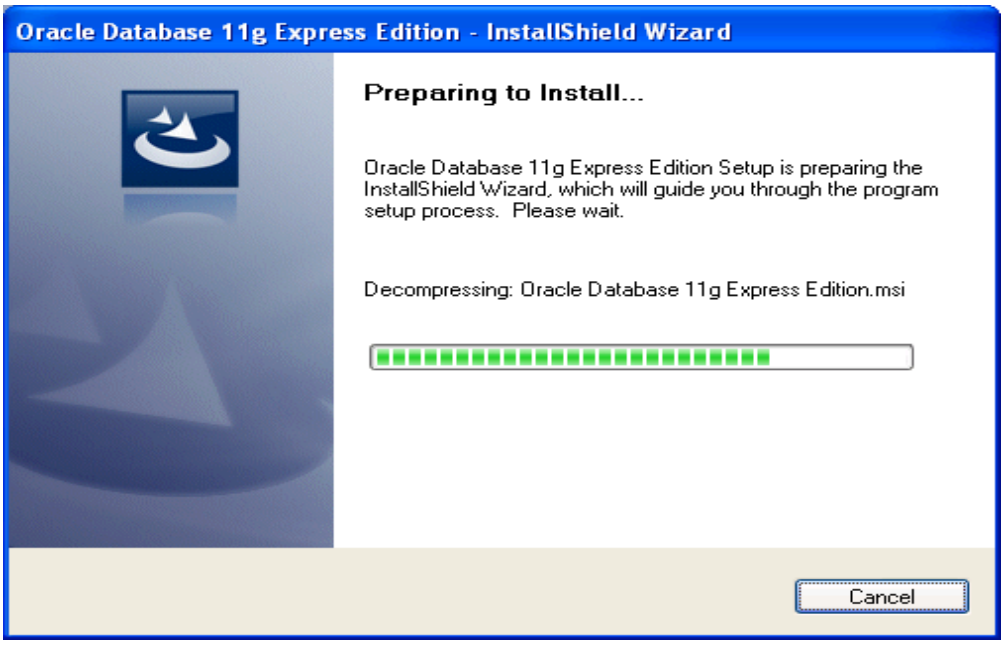


Рис. 1.2. **Установка СКБД Oracle** (декомпресія файлів)

Далі з'являється початкове вікно майстра установки системи (рис. 1.3), на якому слід натиснути кнопку Next.

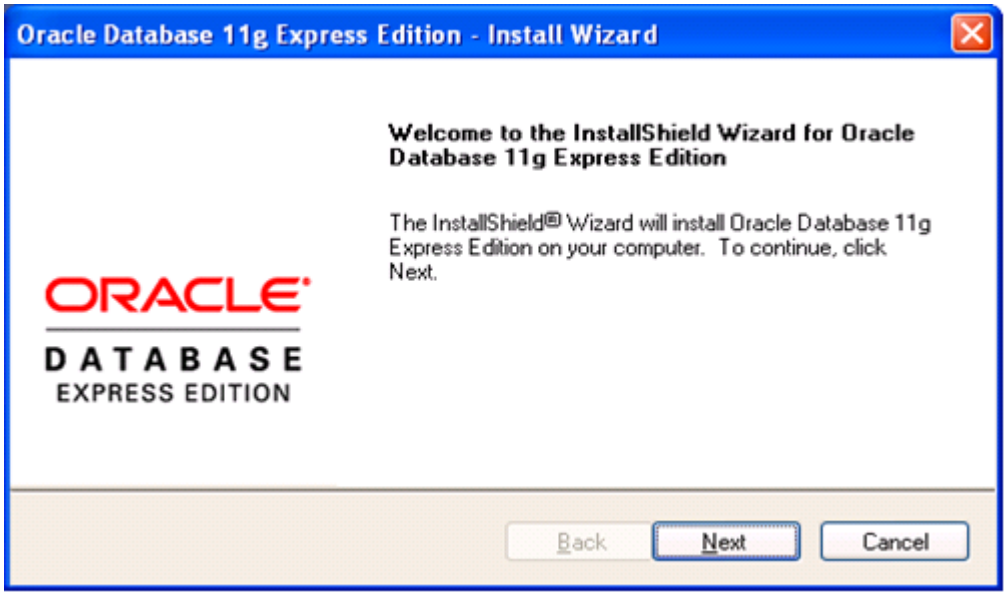


Рис. 1.3. **Вікно майстра установки СКБД Oracle**

У наступному вікні **License Agreement** слід погодитися з умовами ліцензійної угоди, вибравши відповідний перемикач, і натиснути кнопку Next (рис. 1.4).

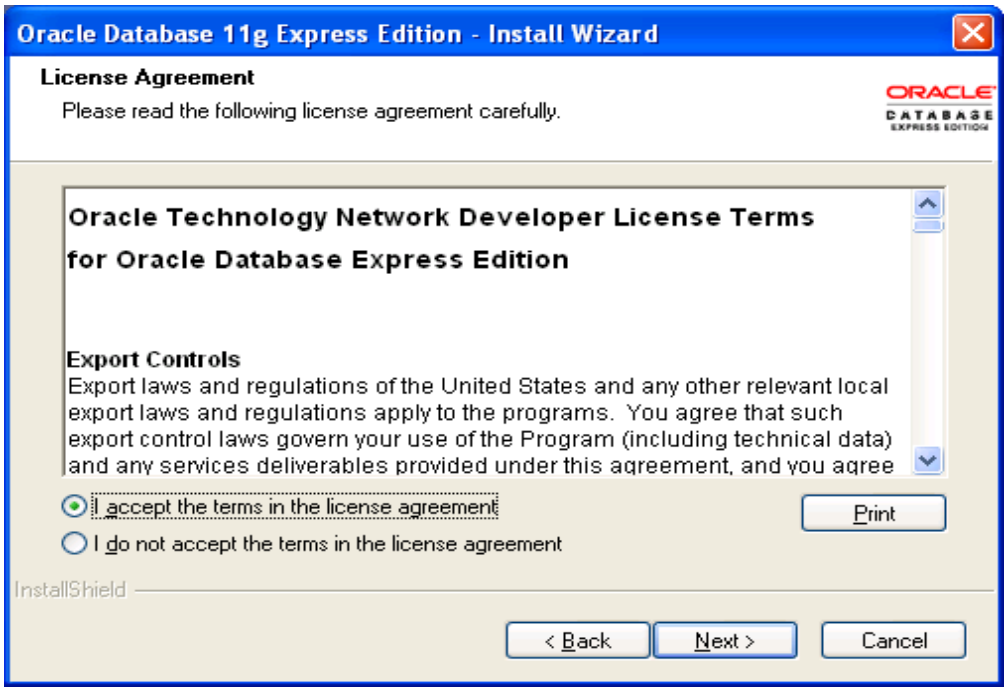


Рис. 1.4. **Вікно прийняття ліцензійної угоди**

У вікні **Choose Destination Location** слід вказати шлях, за яким буде встановлена система. Зазвичай таким шляхом є C:\ORACLEXE, але, за бажанням, можна вказати будь-який інший шлях, натиснувши кнопку Browse (рис. 1.5). Після вибору відповідного шляху слід натиснути кнопку Next.

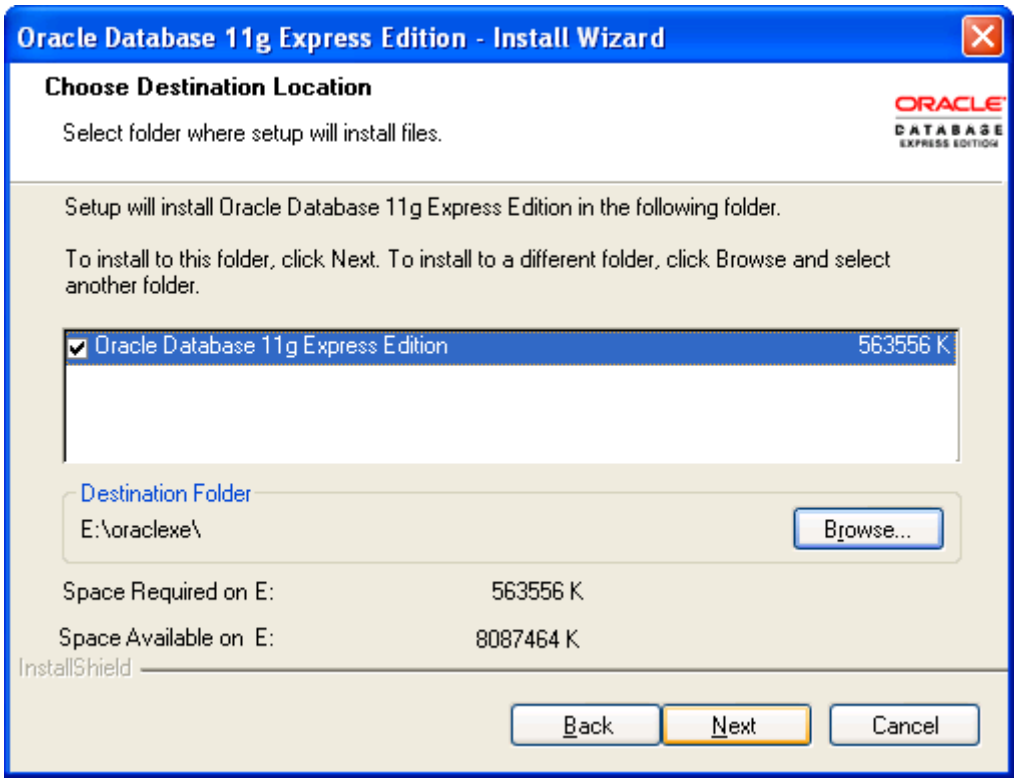


Рис. 1.5. **Вибір шляху для інсталяції СКБД Oracle**

Важливо

Наступний екран **Specify Database Passwords** є дуже важливим, оскільки запитується пароль адміністратора бази даних (рис. 1.6). Його треба вказати однаковим у двох рядках і запам'ятати, бо інакше буде неможливо розпочати роботу з системою. Після виконання відповідних дій активується кнопка Next, яку й слід натиснути для продовження інсталяції системи Oracle.

З'явиться вікно **Summary**, на якому будуть відображені поточні установки системи. Для продовження роботи слід натиснути кнопку Install (рис. 1.7).

Далі, після деякої паузи, з'являється вікно процесу інсталяції системи (рис. 1.8), після закінчення якого буде отримано повідомлення про успішне завершення установки Oracle Database Express Edition 11g на комп'ютер (рис. 1.9).

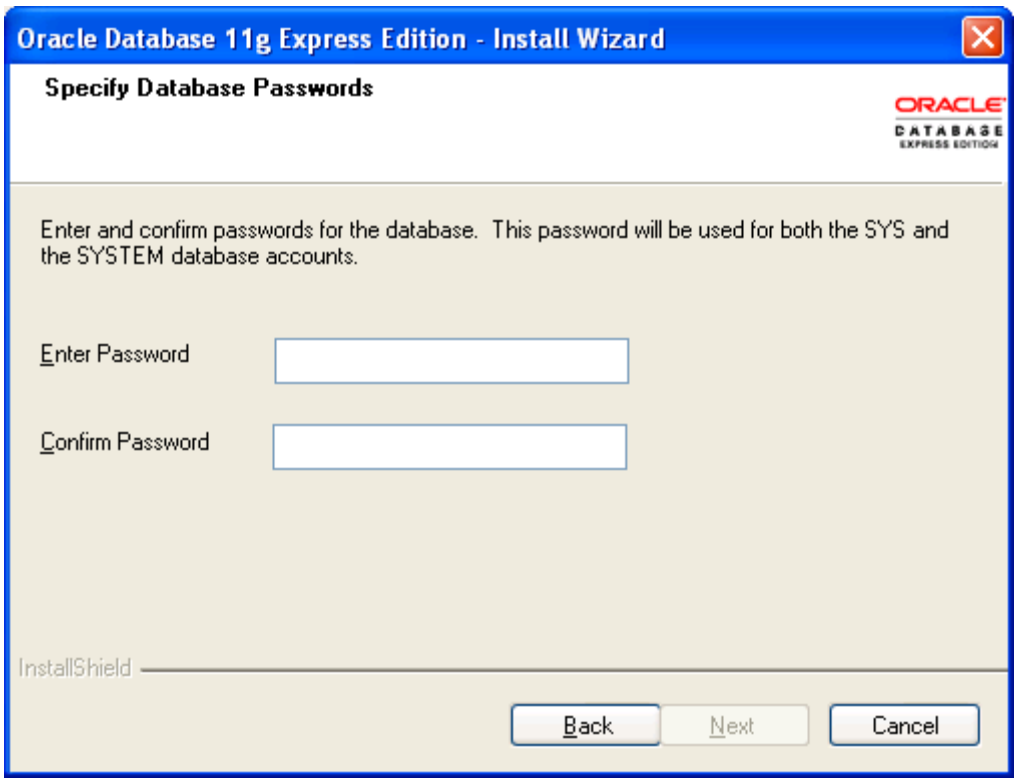


Рис. 1.6. **Вікно для створення пароля адміністратора БД**

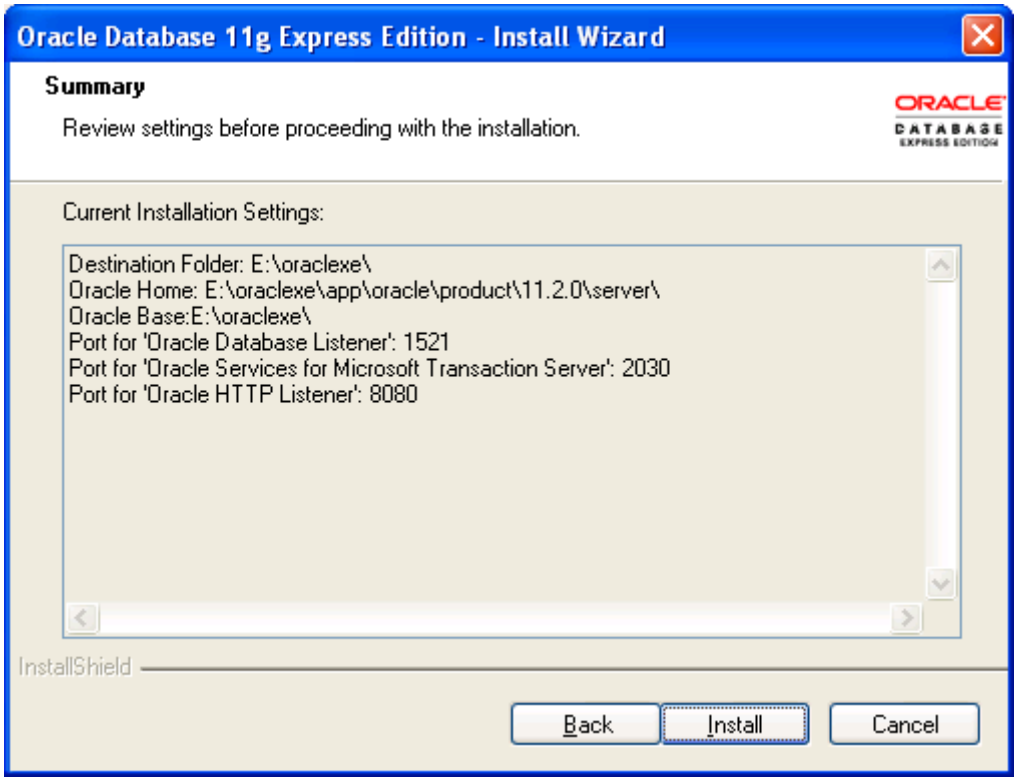


Рис. 1.7. **Вікно відображення поточних установок системи**

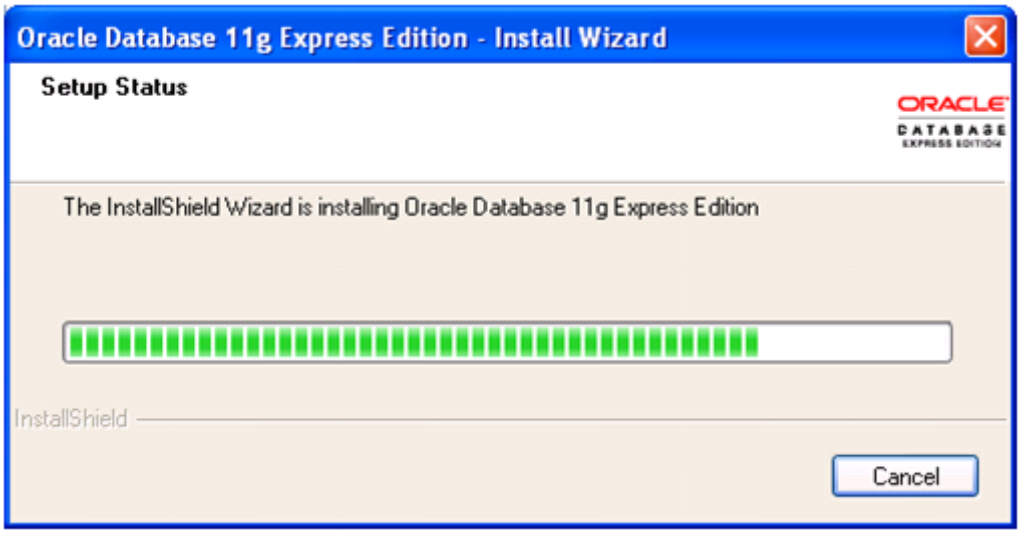


Рис. 1.8. **Вікно процесу інсталяції системи**

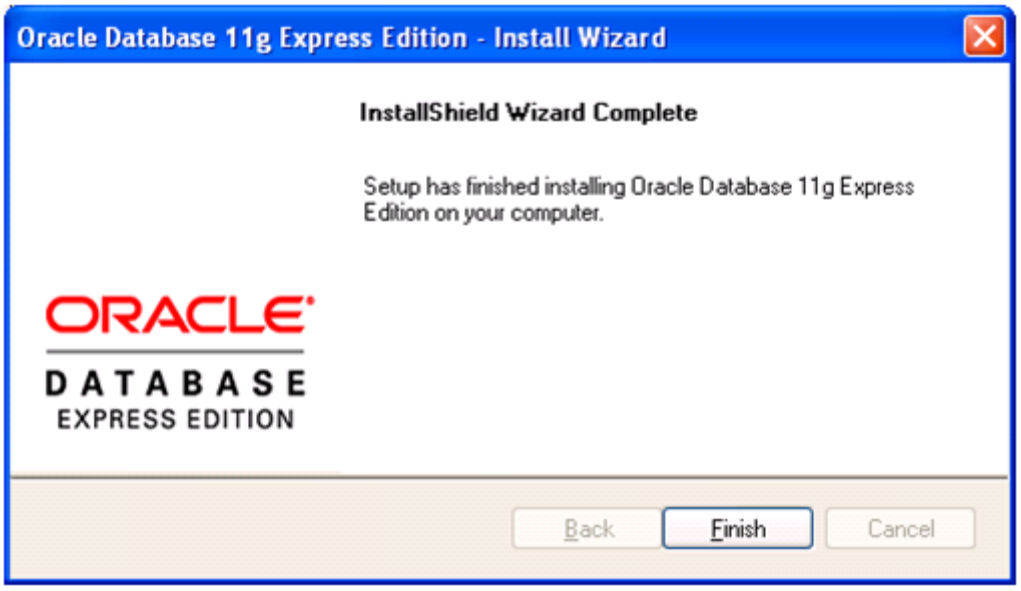


Рис. 1.9. **Повідомлення про успішну установку системи**

Після інсталяції системи Oracle в операційній системі (ОС) Windows з'являються відповідні посилання на встановлені програми (рис. 1.10). Як видно з рисунку, з'явилися програми, що дозволяють виконувати запуск та зупинку СКБД (Start Database, Stop Database), проводити резервне копіювання та відновлення бази даних (Backup Database, Restore Database), а також отримати доступ до бази даних за допомогою утиліти SQL*Plus (Run SQL Command Line) або Web-інтерфейсу (Get Started).

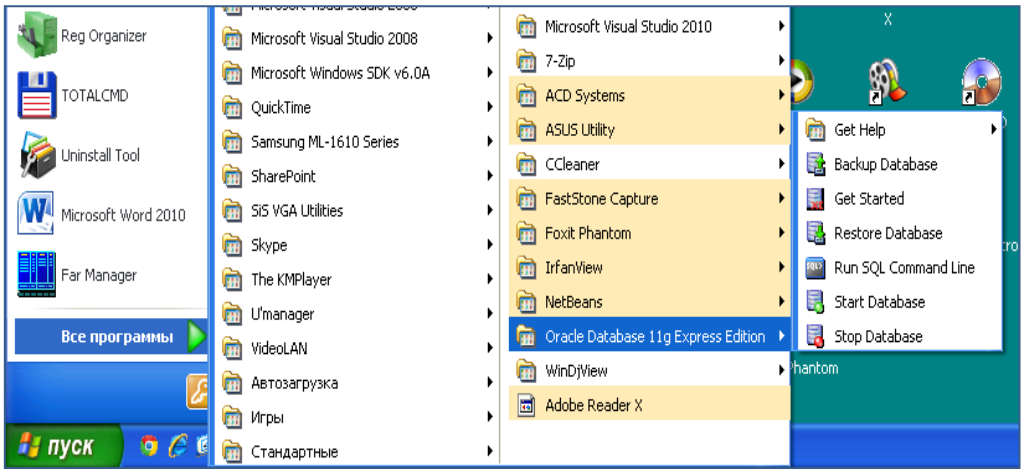


Рис. 1.10. **Доступ до СКБД Oracle у середовищі Windows**

У цьому посібнику розглядається саме утиліта SQL*Plus.

SQL*Plus в Oracle – це програмний інструмент з великою кількістю функцій, який забезпечує адміністратора бази даних (АБД) практично всіма засобами, необхідними для виконання його обов'язків: для створення; зберігання; модифікації; пошуку; виведення, друку та управління інформацією у БД Oracle. Він дозволяє маніпулювати командами SQL і блоками PL/SQL, а також виконувати інші додаткові завдання, що можуть знадобитися АБД [41].

Архітектура встановленої системи залежить від платформи, на яку вона встановлена та фізично розташована на декількох файлах ОС [15; 16]. Однак звичайний користувач може уявити БД Oracle як одну велику за розміром базу даних, логічно поділену на схеми користувачів. Кожний користувач у рамках своєї схеми має повний доступ до об'єктів бази які він створив, тобто він є власником цих об'єктів. До об'єктів, розташованих у інших схемах, користувач буде мати доступ тільки за умови надання йому відповідних привілеїв, причому імена цих об'єктів повинні уточнюватись через крапку іменем відповідної схеми.

Запитання і завдання

1. Який з етапів установки СКБД Oracle є найбільш відповідальним для подальшого використання СКБД?
2. Яку роль відіграє схема у роботі з базою даних Oracle?

1.3. Утиліта SQL*Plus Oracle, її основні функції, команди

Утиліта SQL*Plus дозволяє виконувати команди SQL і блоки PL/SQL, а також вирішувати ряд інших завдань. За допомогою SQL*Plus можна:

- вводити, редагувати, запам'ятовувати, завантажувати та виконувати команди SQL і блоки PL/SQL;
- виконувати налагодження SQL-операторів, блоків PL/SQL, збережених процедур і функцій перед їх використанням у розроблюваних додатках;
- форматувати, створювати, зберігати, друкувати та публікувати у Web результати виконання запитів (звіти);
- отримувати опис (імена та типи стовпців) будь-якої таблиці та подання;
- звертатися до віддалених баз даних і копіювати з них дані;
- посилати та приймати повідомлення від кінцевих користувачів;
- виконувати вбудовані команди;
- адмініструвати базу даних.

При роботі з SQL*Plus використовуються такі базові поняття [41].

- Команда – команда SQL*Plus, або оператор SQL Oracle.
- Блок PL/SQL – група взаємопов'язаних операторів PL/SQL, що оформлена у вигляді анонімного блоку.
- Таблиця – базова одиниця зберігання даних в Oracle.
- Запит SQL – оператор мови SQL, що створює нові об'єкти у базі даних або вибирає, модифікує чи видаляє інформацію з існуючих об'єктів БД.
- Результати запиту – дані, що повернуті запитом (для вибірки), чи зміна стану БД – для інших команд.
- Звіт – результати запиту на вибірку даних, що відформатовані за допомогою команд SQL*Plus.

За своєю сутнітю SQL*Plus – це програма-інтерпретатор командного рядка для роботи з системою керування базами даних Oracle Database, в якій можуть виконуватися команди безпосередньо самої утиліти SQL*Plus, команди мови SQL чи її процедурного розширення PL/SQL.

Виконання цих команд може здійснюватися як в інтерактивному режимі, так і за допомогою сценарію, який може зберігатися у текстовому файлі з розширенням SQL (за замовчуванням). Такі файли називають командними (більш детально вони будуть розглянуті в підрозділі 1.4.).

Програмісти й адміністратори СКБД зазвичай використовують SQL*Plus як інструмент за замовчуванням, бо його інтерфейс доступний у будь-якій версії програмного забезпечення Oracle.

Крім того, SQL*Plus надає редактор рядків для редагування команд SQL та дозволяє встановлювати параметри свого середовища, що впливають на роботу як самої утиліти, так і на вигляд отриманих результатів з БД.

У свою чергу, SQL є не процедурною мовою запитів [23], що використовується для комунікації з сервером Oracle з будь-якого інструменту або програми.

Коли вводиться SQL-оператор, він зберігається в частині пам'яті, що зветься ***буфером SQL,*** і залишається там, доки не буде введено новий SQL-оператор (рис. 1.11) [63].

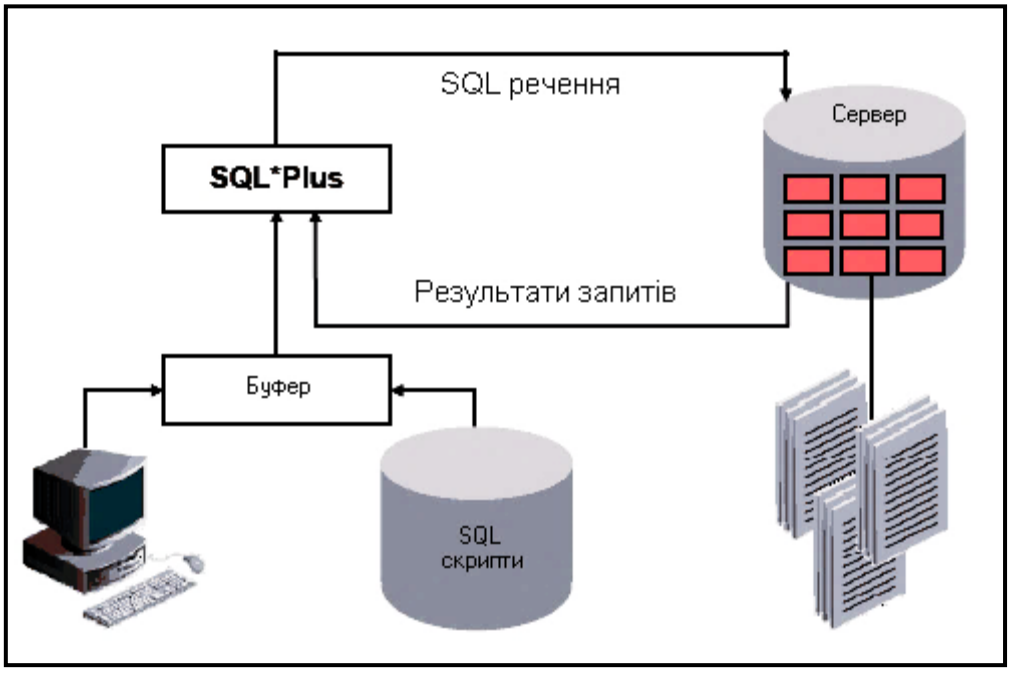


Рис. 1.11. **Взаємодія SQL та SQL*Plus**

Тобто буфер SQL – це спеціальна область, у якій SQL*Plus зберігає останню введену команду

У табл. 1.1 подано загальне порівняння команд мови SQL та команд SQL*Plus [62].

Таблиця 1.1

Порівняння команд SQL та SQL*Plus

SQL	SQL*Plus
Мова зв'язку з сервером Oracle для доступу до даних	Набір команд, що використовується для керування режимами роботи утиліти SQL*Plus, форматування результатів запитів та роботи з файлами. Розпізнає команди SQL та передає їх на сервер
Заснована на SQL ANSI (Американський інститут стандартів)	Інтерфейс для виконання команд SQL, розроблений корпорацією Oracle
Маніпулює даними та визначеннями таблиць у базі даних	Не дозволяє маніпулювати значеннями в базі даних
У буфер SQL вводиться одна команда мови SQL розташована на одному або декількох рядках	Вводяться по одному рядку, у буфері SQL не зберігаються
Не має символу продовження	Використовує символ продовження (-), якщо команда займає більше одного рядка
Не допускає скорочень у командах	Допускає скорочення
Використовує символ завершення для негайного виконання команди	Не вимагає символу завершення (;). Команди виконуються негайно

1.3.1. Запуск та закінчення роботи з SQL*Plus

Для запуску програми SQL*Plus слід засобами операційної системи Windows вибрати пункт меню "Run SQL Command Line" з переліку встановлених програм (див. рис. 1.10) чи двічі клацнути мишкою по відповідній іконці на робочому столі. Для підключення до вже встановленого сервера Oracle також можна скористатися каталогом XEClient з програмою SQL*Plus, що надається до лабораторної роботи "Використання утиліти SQL*Plus для роботи з базою даних Oracle" і архів якого можна завантажити з сайту дистанційного навчання Харківського національного економічного університету ім. С. Кузнеця за адресою [51].

В обох випадках на екрані з'явиться вікно утиліти SQL*Plus (рис. 1.12).

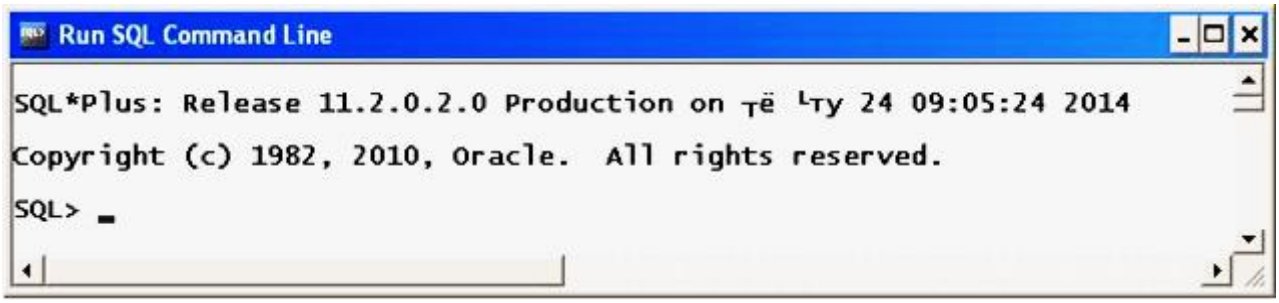


Рис. 1.12. **Вікно утиліти SQL*Plus**

Як видно з рисунку, на екрані з'являється інформація про версію встановленої утиліти SQL*Plus і потім – рядок запрошення з позначкою **SQL>**. Саме використання цього рядка дозволяє SQL*Plus вводити усі необхідні команди для роботи з самою утилітою або сервером бази даних.

Перше, що необхідно зробити, це підключитися до локальної або віддаленої бази даних. Після встановлення нової системи це можна зробити лише за допомогою пароля адміністратора, який вказувався під час інсталяції системи, оскільки інших користувачів бази даних ще не існує. Якщо ж підключення здійснюється до вже існуючої БД, яка містить певний перелік користувачів, необхідно отримати логін та пароль для доступу від адміністратора такої БД.

Приклад 1.1. Підключитися до створеної БД від імені адміністратора (рис. 1.13).

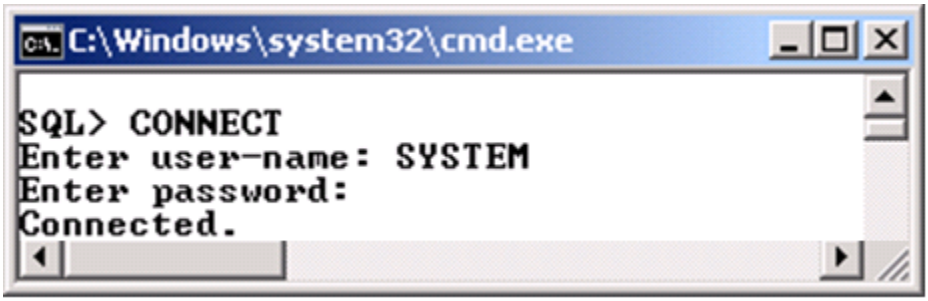


Рис. 1.13. **Підключення до бази даних**

Отже, для виконання підключення до БД необхідно ввести команду CONNECT (підключитися). Після введення команди та натиснення клавіші ENTER буде здійснено запит на введення логіну, а потім – пароля (див. рис. 1.13), і після успішного підключення з'явиться повідомлення **Connected**, тобто "з'єднання встановлено".

Слід зазначити, що підключитися можна і не входячи до діалогового режиму, а просто ввести команду

CONNECT SYSTEM/<SystemPassword>,

де SystemPassword – пароль адміністратора, який був вказаний при інсталяції системи Oracle.

Такий спосіб підключення здійснюється до бази даних за замовчуванням, тобто на тому комп'ютері, на якому була встановлена система. Якщо ж користувач підключається до віддаленої бази даних, то необхідно ще вказати так звану "специфікацію БД", яка додається до рядка підключення після паролю користувача з символом @.

Наприклад, підключення до сервера кафедри інформаційних систем ХНЕУ з мережі обчислюваних центрів здійснюється за допомогою команди

CONNECT studGGNN/stud@172.16.0.101/XE,

де GG – номер студентської групи;

NN – номер студента у групі (логін);

stud – пароль доступу до сервера;

@172.16.0.101/XE – специфікація віддаленої БД (у даному випадку сервера кафедри ІС).

Слід зазначити, що працювати з базою даних для користувача системи з правами адміністратора є не тільки недоречним, а й хибним шляхом, що може призвести до втрати працездатності системи. Тому для навчальних цілей і знайомства з можливостями системи слід створити користувачів з обмеженими правами. На щастя будь-яка інсталяція системи Oracle містить спеціальний командний файл (скрипт) для створення користувача системи і його тестової бази даних. Файл має назву **SCOTT.SQL** і він зазвичай зберігається у каталозі:

c:\Oraclexe\app\Oracle\product\11.2.0\server\rdbms\admin

або йому подібному – залежно від версії встановленої системи.

Аналізуючи текст скрипта (додаток А), можна побачити, що в ньому створюється користувач системи з логіном SCOTT та паролем TIGER, і для цього користувача створюються таблиці БД з іменами EMP, DEPT, SALGRADE та BONUS.

Таблиця EMP містить дані про співробітників певної фірми.

Таблиця DEPT – дані про відділи фірми.

Таблиця SALGRADE вказує на градацію (рівень) заробітної платні залежно від її кількості.

Таблиця BONUS порожня та не містить будь-яких даних.

Приклад 1.2. Створити схему користувача SCOTT з відповідними таблицями, використовуючи файл SCOTT.SQL.

Для виконання завдання потрібно, підключившись до бази даних як адміністратор, виконати команду:

START PATH\SCOTT.SQL,

де PATH – це шлях до скрипта SCOTT.SQL на комп'ютері. (рис. 1.14)

Рис. 1.14. **Запуск командного файла на створення БД**

Після виконання скрипта знову з'явиться рядок запрошення SQL>, після якого можна буде вводити нову команду.

Виконавши такі дії, можна підключатися вже до БД не тільки як адміністратор, але й як користувач SCOTT за допомогою команди CONNECT.

Приклад 1.3. Підключитися до БД як користувач SCOTT. Підключення здійснити командою CONNECT, вказавши при цьому відповідні логін та пароль (рис. 1.15).

CONNECT SCOTT / TIGER

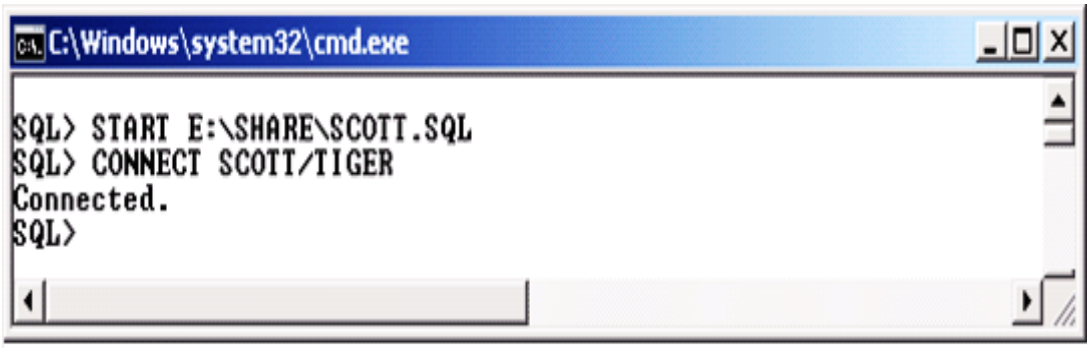


Рис. 1.15. **Підключення до БД від імені користувача SCOTT**

Якщо схема користувача створюється на віддаленому сервері, то слід вказати специфікацію віддаленої БД, наприклад:

CONNECT SCOTT / TIGER @172.16.0.101/XE

База даних, до якої здійснено підключення користувачем, стає базою за замовчуванням, поки не буде виконано підключення до іншої БД або не здійснено відключення від БД.

Приклад 1.4. Відключитися від поточної БД використовуючи, команду DISCONNECT (рис. 1.16).

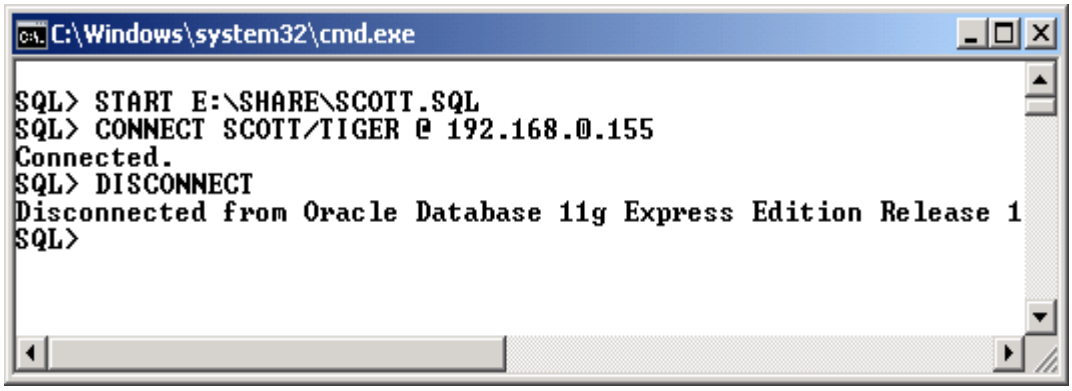


Рис. 1.16. **Відключення від БД**

Вихід з SQL*Plus здійснюється за допомогою команд QUIT або EXIT.

1.3.2. Основні команди SQL*Plus.

Як вже було зазначено, крім команд мови SQL та її процедурного розширення – мови PL/SQL, у рядку запрошення (командному рядку) SQL*Plus можна вводити спеціальні команди самої утиліти. Їх можна умовно поділити на декілька груп (табл. 1.2).

Таблиця 1.2

Класифікація команд SQL*Plus

Категорія	Призначення
Середовище	Впливають на загальну поведінку операторів SQL під час сеансу
Формат	Форматують результати запитів
Робота з файлами	Зберігають, завантажують та виконують скрипт-файли
Виконання	Передають команди SQL з буфера SQL на сервер Oracle

Закінчення табл. 1.2

Категорія	Призначення
Редагування	Змінюють команди SQL у буфері
Взаємодія	Дозволяють створювати змінні, передавати їх у команди SQL, друкувати значення змінних і посилати повідомлення на екран
Інше	Різноманітні команди для підключення до БД, маніпулювання середовищем SQL*Plus та виведення визначень стовпців

Загальний перелік команд SQL*Plus наведений у додатку Б.

Підключившись до системи, завжди можна отримати підказку, отримавши перелік команд SQL*Plus чи конкретно формат тієї чи іншої команди.

Приклад 1.5. Отримати підказку з переліку команд SQL*Plus. Для цього існує команда **HELP INDEX** (рис. 1.17).

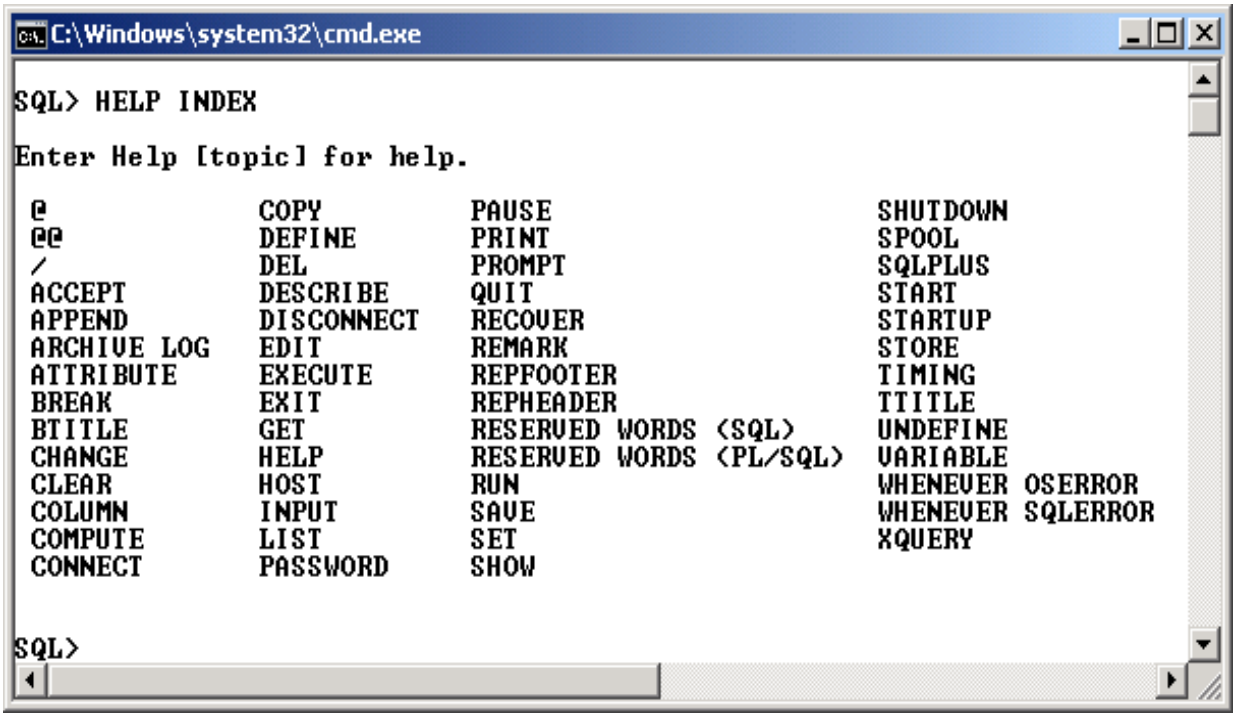


Рис. 1.17. **Отримання підказки з переліку команд SQL*Plus**

Щоб отримати детальну підказку за конкретною командою, необхідно виконати команду **HELP,** вказавши у якості параметра назву такої команди, тобто **HELP <Command>,**

Приклад 1.6. Отримати підказки для команд CONNECT та QUIT утиліти SQL*Plus (рис. 1.18).

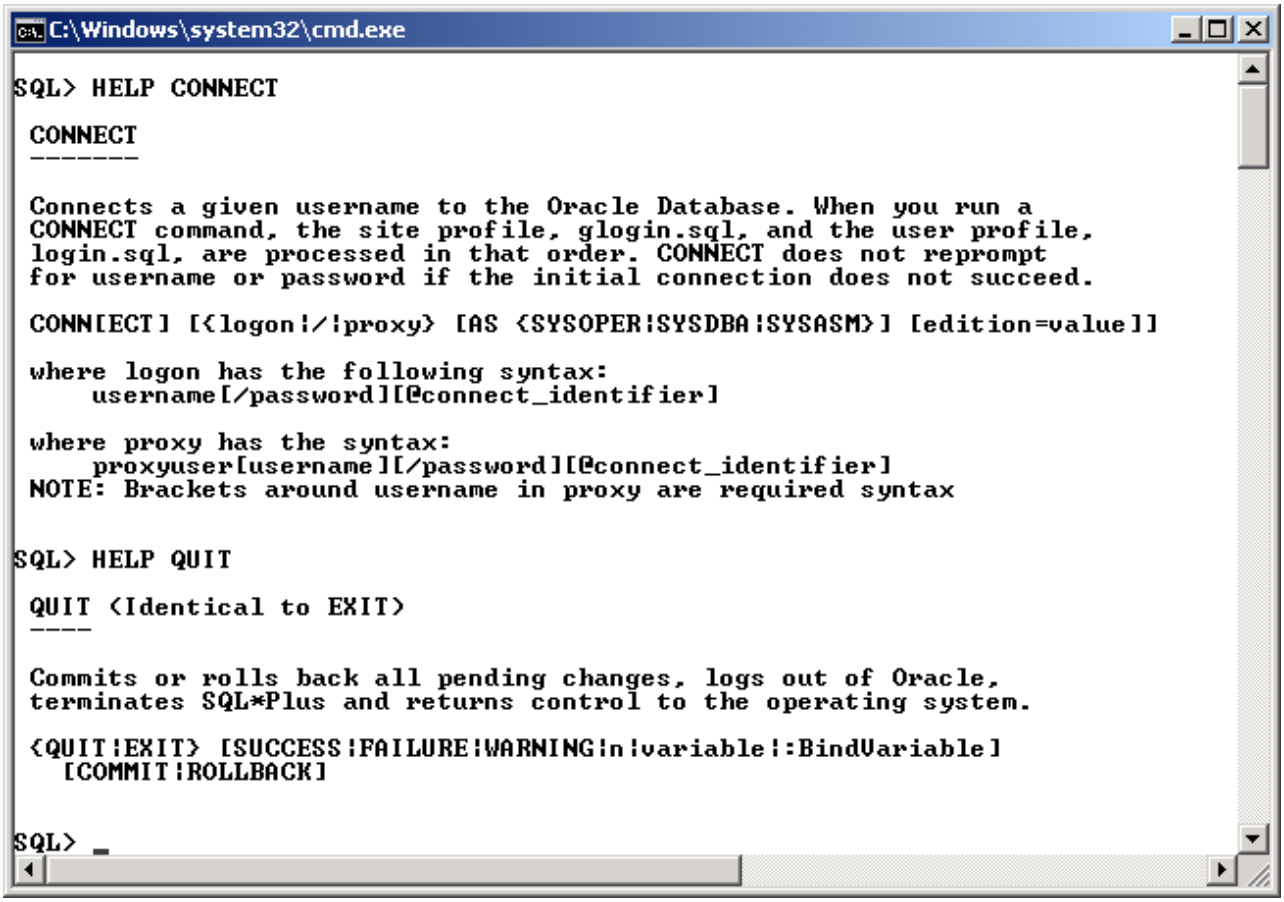


Рис. 1.18. **Підказка для команд CONNECT та QUIT**

Як видно з рисунка, команди SQL*Plus допускають скорочення, на відміну від команд мови SQL та PL/SQL. Наприклад, замість повного варіанту команди CONNECT можна було б ввести її скорочену форму CONN.

1.3.3 Використання команд мови SQL у середовищі SQL*Plus

Застосовуючи деякі прості правила, можна задавати правильні команди мови структурованих запитів SQL у середовищі SQL*Plus для обробки на сервері баз даних, які легко читати та редагувати [12].

- Вводити команди SQL можна в один або декілька рядків.
- Для зручності читання та редагування кожне речення зазвичай вводиться на окремому рядку.
- Для зручності читання команди можна використовувати табуляцію та відступи.

• Скорочення та перенесення слів в командах забороняються.

• Ключові слова та команди зазвичай вводяться символами верхнього регістру; решта слів (наприклад, імена таблиць і стовпців) – символами нижнього регістру, хоча це і не обов'язково.

• За відсутності особливих вказівок символи верхнього та нижнього регістрів в командах SQL не розрізняються.

• Команда SQL вводиться на тому ж рядку, що й запрошення SQL, а наступні рядки нумеруються. Ці рядки утворюють буфер SQL.

У кожен момент часу активною в буфері може бути тільки одна команда SQL, і виконати її можна декількома способами.

• Поставити крапку з комою (;) наприкінці останнього речення.

• Поставити крапку з комою або слеш (дробову риску) в останньому рядку буфера.

• Ввести слеш у відповідь на запрошення SQL.

• Задати команду RUN середовища SQL*Plus у відповідь на запрошення SQL.

Тобто, якщо є необхідність виконати команду знову, можна це зробити без повторного її введення.

SQL*Plus не запам'ятовує крапку з комою чи слеш у буфері, які були введені для виконання команди.

Різноманітні способи виконання команд ілюструються подальшими прикладами.

Приклад 1.7. Ввести команду SQL, що розташована на трьох рядках, та завершити її крапкою з комою (рис. 1.19). Команда вибирає співробітників з таблиці EMP, з заробітною платою більшою за 1 500.

Після натиснення [Enter] у останньому рядку команда SQL виконується миттєво.

1.3.4. Виконання блоків PL/SQL

Працюючи з SQL*Plus, можна використовувати програми мовою PL/SQL (часто називають блоками) для маніпулювання даними у базі даних. Особливості їх використання такі:

• Блоки PL/SQL починаються з ключових слів DECLARE, BEGIN або імені блоку SQL*Plus.

• Працювати з блоками PL/SQL можна так само, як і з командами SQL, враховуючи, що крапка з комою або порожній рядок не закінчують і не виконують блок.

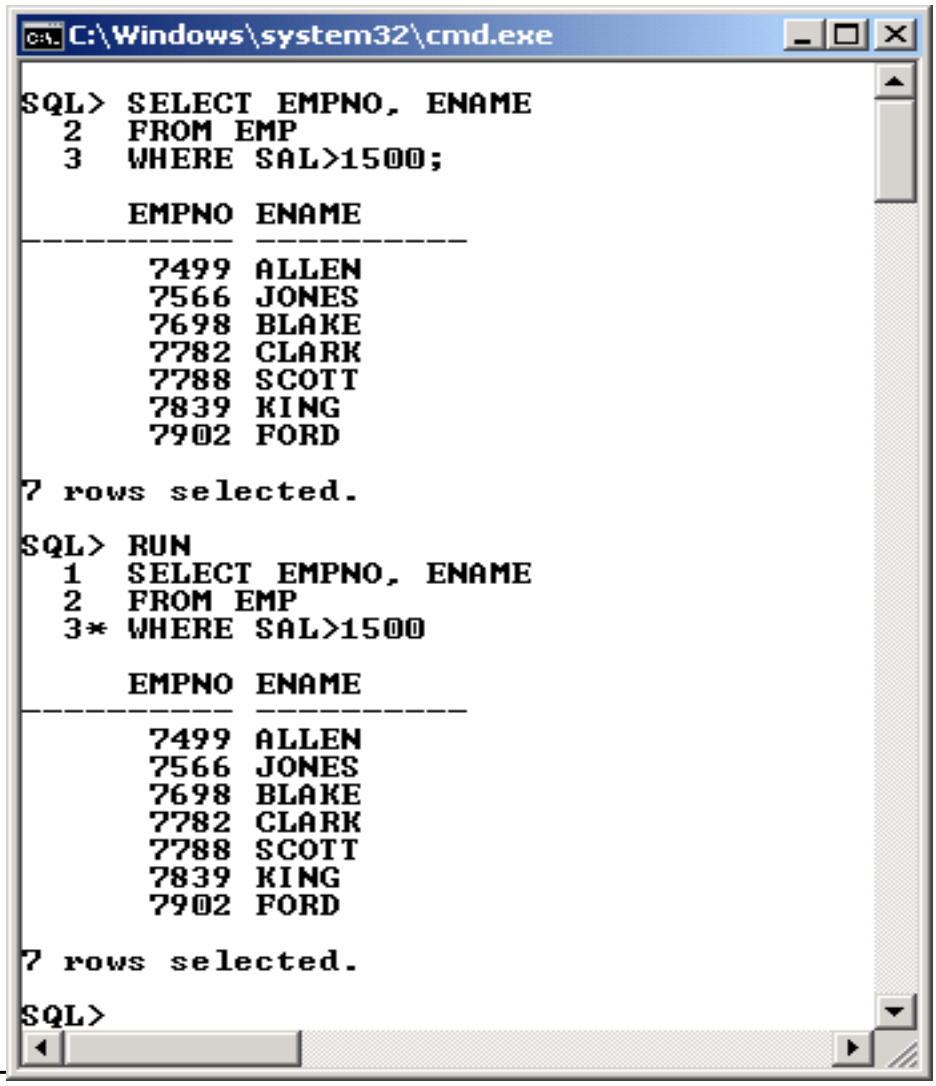


Рис. 1.19. **Введення команди SELECT на трьох рядках**

• Блок PL/SQL завершується крапкою (.) на новому рядку. SQL*Plus зберігає блоки, які були введені, у буфері SQL. Виконуючи поточний блок за допомогою команди RUN або "/", SQL*Plus посилає блок PL/SQL у СКБД Oracle для обробки.

Приклад 1.8. Ввести блок команд PL/SQL та запустити його на виконання (рис. 1.20).

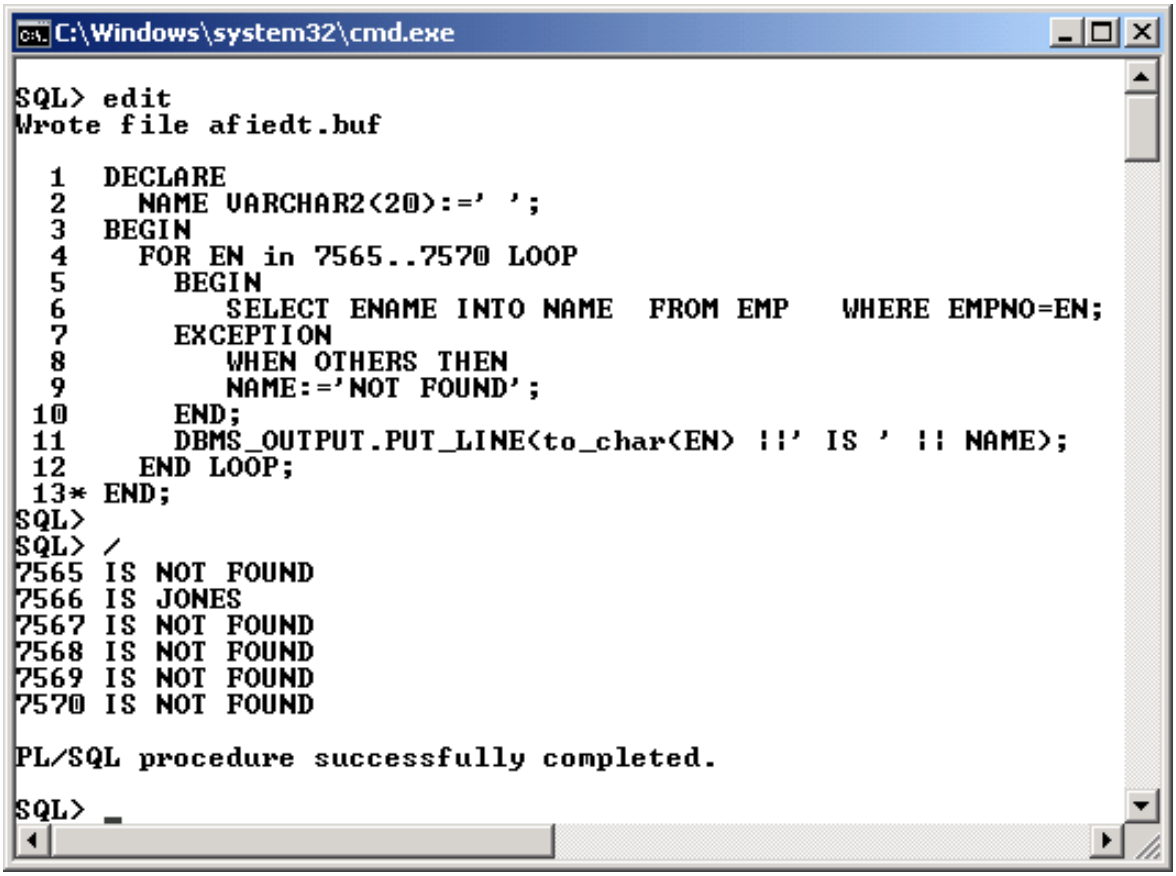


Рис. 1.20. **Виконання скрипту мовою PL/SQL**

Наведений блок PL/SQL у циклі перебирає номери від 7 565 до 7 570 та перевіряє, чи існують у таблиці EMP співробітники з такими номерами (поле EMPNO). Якщо "так", то виводиться прізвище співробітника, якщо "ні" – інформація, що такий співробітник не знайдений.

1.3.5. Узгодження синтаксису команд SQL*Plus

Команди SQL*Plus мають синтаксис, який є відмінним від синтаксису команд SQL та блоків PL/SQL. У зв'язку з цим потрібно дотримуватися певних правил, якщо одночасно треба вводити як команди SQL, так і команди SQL*Plus.

Продовження довгих команд SQL*Plus на додаткових рядках

Можна продовжити довгі команди SQL*Plus за допомогою введення дефіса в кінці рядка та натискання [Enter]. SQL*Plus виведе праву кутову дужку (>) як підказку для кожного додаткового рядка.

Закінчення команди SQL*Plus

Необов'язково закінчувати команду SQL*Plus за допомогою крапки з комою. Коли закінчено введення команди, можна одразу натиснути

[Enter]. Однак, за бажання, можна ввести крапку з комою в кінці команди SQL*Plus.

Приклад 1.9. Ввести команду SQL*Plus (COLUMN), розташовану на двох рядках, яка змінює формат відображення стовпця на ім'я SAL, та виконати команду SQL на вибірку даних з таблиці, що має стовпець з таким же ім'ям (рис. 1.21).

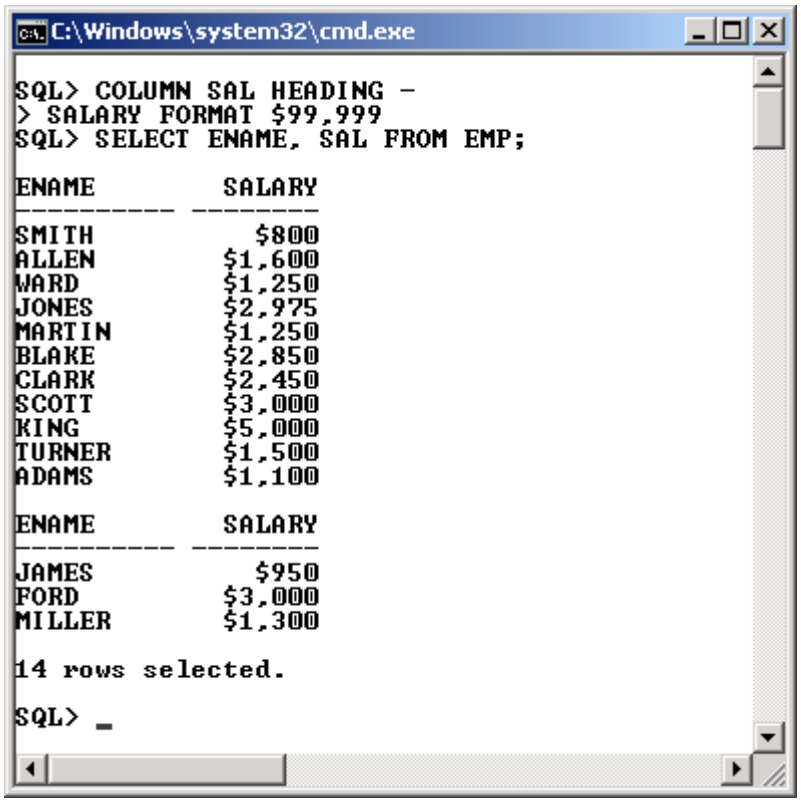


Рис. 1.21. **Команда SQL*Plus, що розташована на двох рядках**

1.3.6. Системні змінні, які впливають на виконання команд

Команда **SET** утиліти SQL*Plus керує більшістю змінних, які називаються системними змінними й установка яких впливає на спосіб виконання програмою SQL*Plus команд, що вводяться в інтерактивному режимі або під час виконання командних файлів.

Системні змінні керують великою кількістю параметрів всередині SQL*Plus, включаючи ширину стовпців за замовчуванням для виведення на екран, довжину та ширину сторінки для виведення результатів, автоматичним збереженням змін у базі даних тощо. Системні змінні називають також змінними команди SET.

Залежно від установки системних змінних результати, що виводяться на екран, можуть відрізнятися від наведених у прикладах. Щоб

роздрукувати поточні установки змінних команди SET, необхідно ввести після командної підказки **SHOW ALL** або **SHOW ім'я змінної**. Команда **SHOW ALL** – виводить поточні значення усіх змінних, а команда **SHOW ім'я змінної** – тільки значення конкретної змінної.

Приклад 1.10. Вивести на екран поточні значення усіх змінних (рис. 1.22) та значення змінних LINESIZE та USER(рис. 1.23).

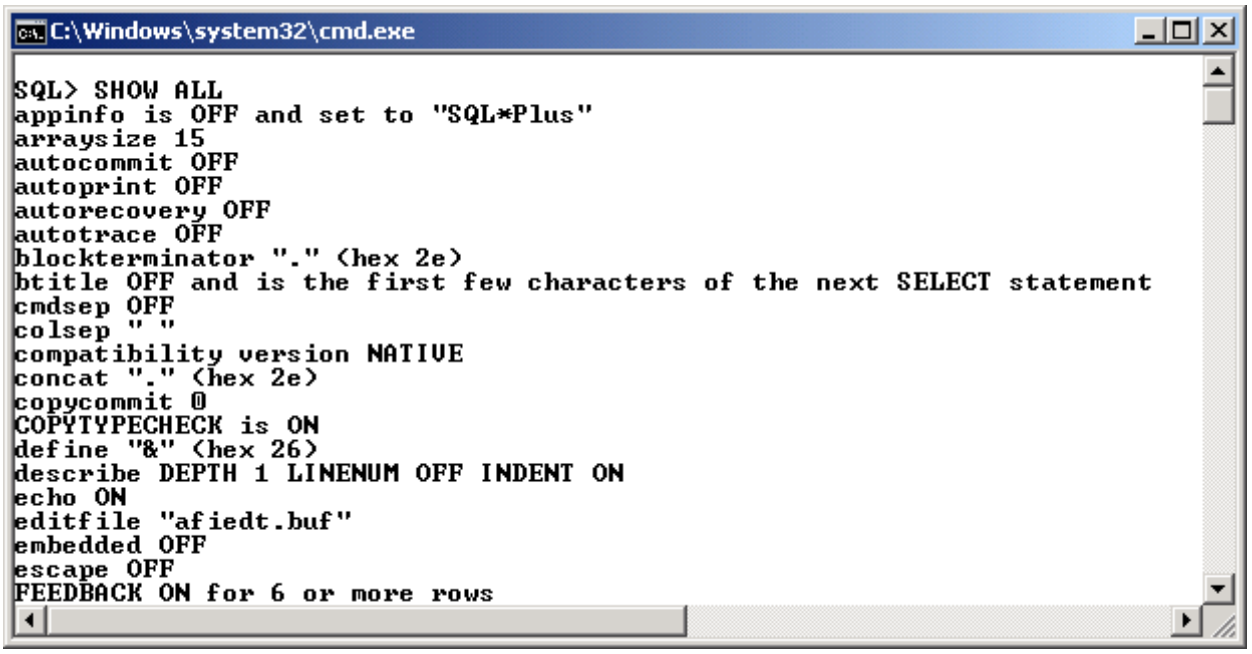


Рис. 1.22. **Відображення поточних значень системних змінних**

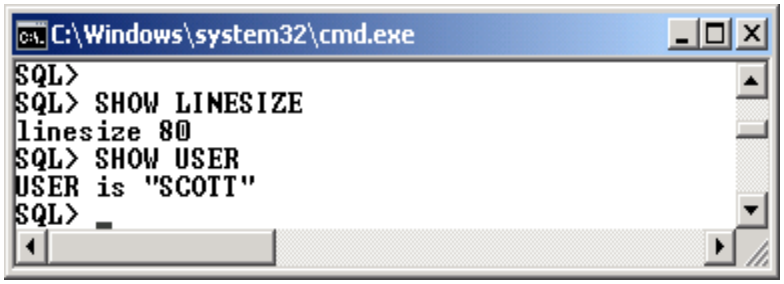


Рис. 1.23. **Відображення значень конкретних змінних**

Наприклад, команду SET можна використати для зміни установок за замовчуванням для кількості рядків на сторінці (чотирнадцять) або кількості символів в рядку (вісімдесят), що впливає на вигляд результату, отриманого на екрані.

Приклад 1.11. Отримати усі дані з таблиці EMP на основі значень параметрів LINESIZE PAGESIZE за замовчуванням (рис. 1.24).

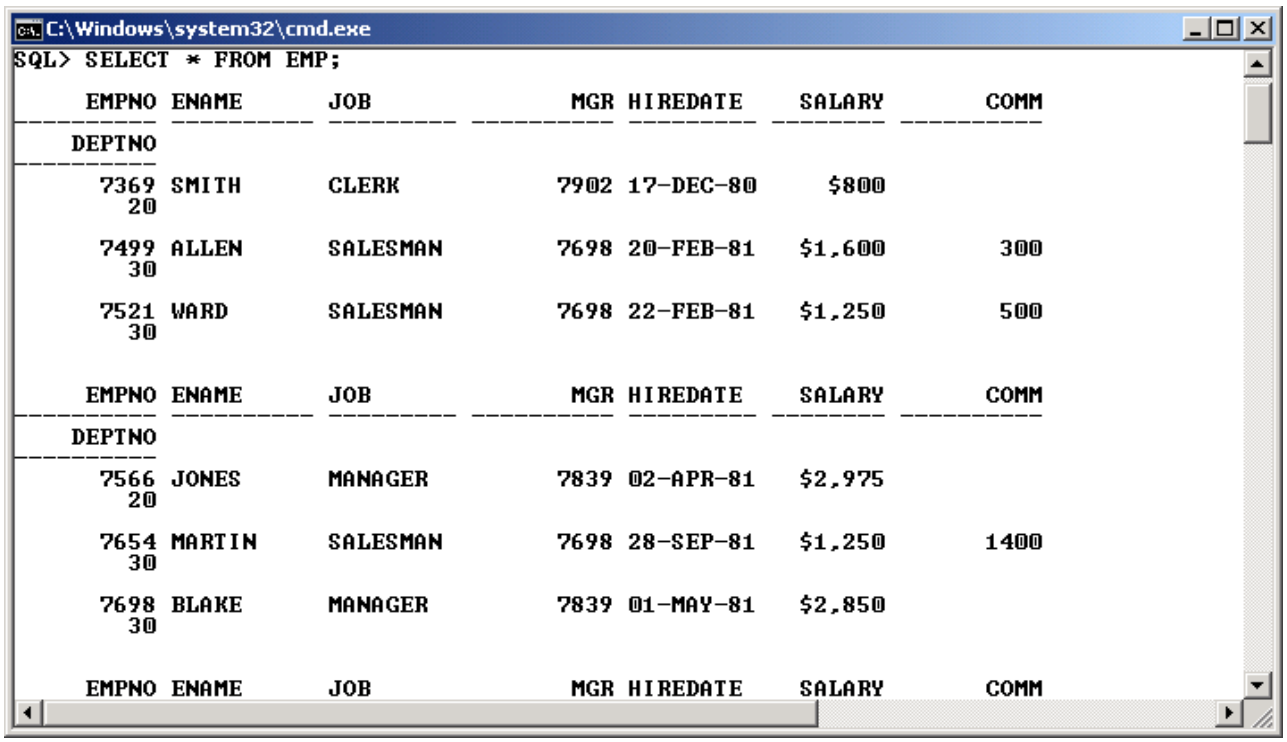


Рис. 1.24. **Стандартне відображення даних з таблиці EMP**

Як видно з рисунка, отриманий результат виглядає, з одного боку, некрасиво, а з іншого – не дуже зрозуміло. Це трапилось через те, що значення системної змінної LINESIZE за замовчуванням дорівнює 80-ти, а результат наведеного запиту потребує більше колонок.

Приклад 1.12. Збільшити до 120 значення параметра LINESIZE і повторно виконати запит до таблиці EMP (рис. 1.25).

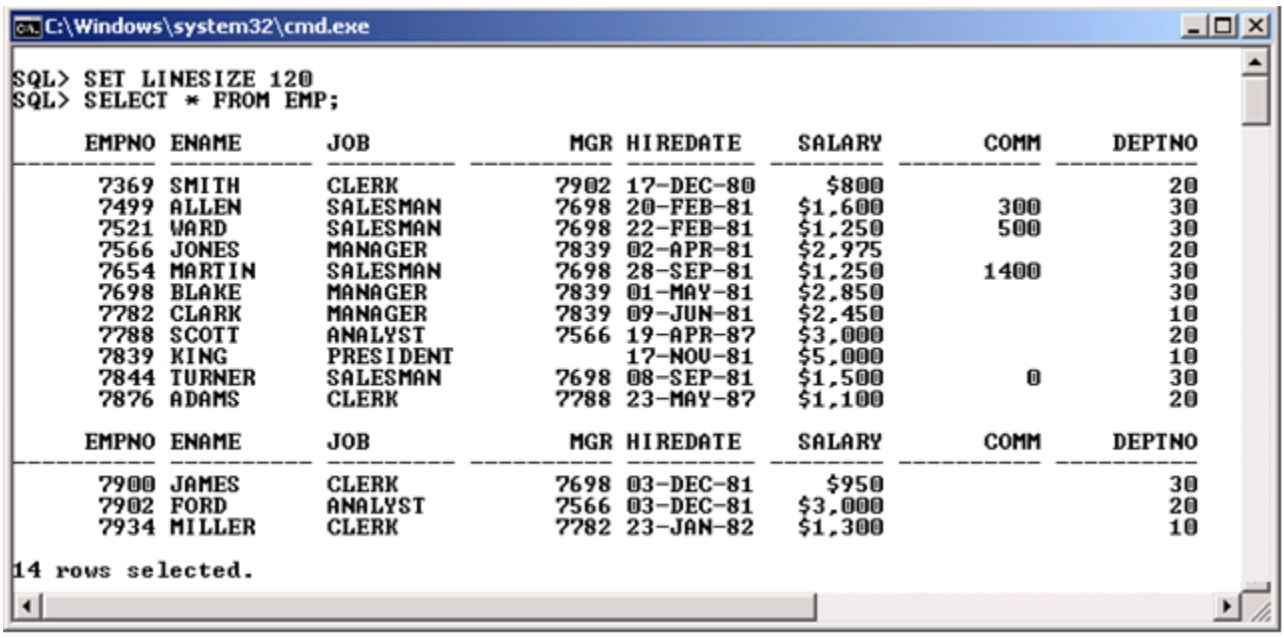


Рис. 1.25. **Відображення даних з таблиці EMP після зміни LINESIZE**

Як видно з рисунка, отриманий результат цілком зрозумілий. Певну незручність може викликати тільки повторення "шапки" таблиці через чотирнадцять рядків. Це викликано тим, що значення системної змінної PAGESIZE за замовчуванням дорівнює чотирнадцяти.

Приклад 1.13. Змінити значення параметра PAGESIZE на 60 та повторно виконати запит до таблиці EMP (рис. 1.26).

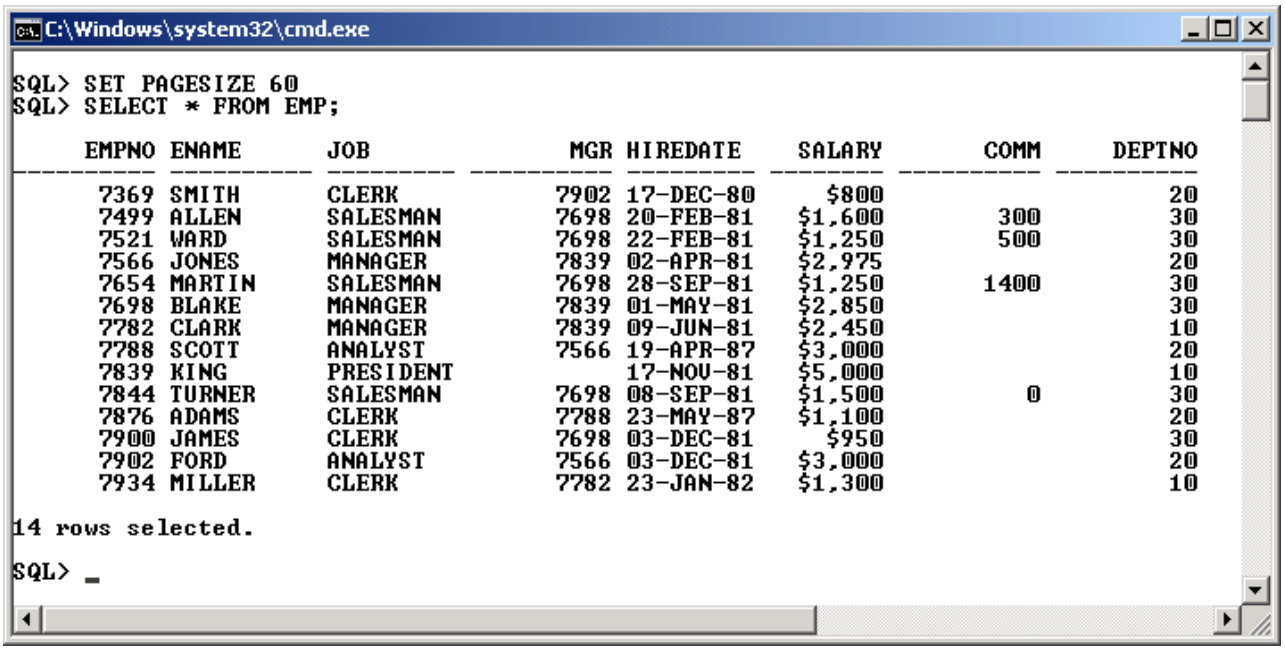


Рис. 1.26. **Відображення даних з таблиці EMP після зміни PAGESIZE**

Ще однією важливою змінною є SERVEROUTPUT, яка відповідає за виведення на консоль результатів роботи скриптів за допомогою команди **DBMS_OUTPUT.PUT_LINE** у мові PL/SQL**.** Ця команда є деяким аналогом команди **prinf** у мові C/C++ чи **writeln** у Паскалі.

Якщо повернутися до рис. 1.20, слід зауважити, що при значенні змінної SERVEROUTPUT рівному OFF, блок PL/SQL виконується, але результат на екрані не з'являється. Тому рекомендується завжди встановлювати значення цієї змінної у значення ON.

```
SQL> SET SERVEROUTPUT ON
```

Автоматичне збереження змін у БД

Завдяки командам SQL DML – UPDATE, INSERT, DELETE, які можуть використовуватися і в блоках PL/SQL, можна змінити дані у БД. Однак команда SQL фізично не змінює інформацію в БД, поки не буде

виконана команда SQL COMMIT або команди SQL DCL або DDL, наприклад, CREATE TABLE.

Тобто підключившись до БД, можна виконати декілька команд UPDATE, INSERT або DELETE, а потім завершити сеанс роботи з БД. Однак, згодом підключившись повторно до БД, на свій подив, виявити, що зміни у БД не збереглися і вона має стан на момент попереднього підключення. Це викликано тим, що не було введено команду COMMIT.

Однак можна встановити режим роботи SQL*Plus, за якого зміни в БД виконуються автоматично. Він задається за допомогою змінної AUTOCOMMIT команди SET і має такий формат:

SET AUTCOMMIT ON / OFF,

де **ON** – включає режим автоматичного збереження змін у БД, **OFF** – виключає (за замовчуванням) режим автоматичного збереження змін у БД.

1.3.7. Структура й опис таблиць

У SQL*Plus для виведення структури таблиці використовується команда DESCRIBE. Ця команда видає імена стовпців і типи даних таблиці, а також відомості про те, чи повинен стовпець містити дані.

Синтаксис: **DESC [RIBE] tablename,**

де **tablename** – ім'я будь-якої існуючої таблиці, подання або синоніма, які доступні користувачу.

Приклад 1.14. Отримати інформацію щодо структури таблиці EMP (рис.1.27).

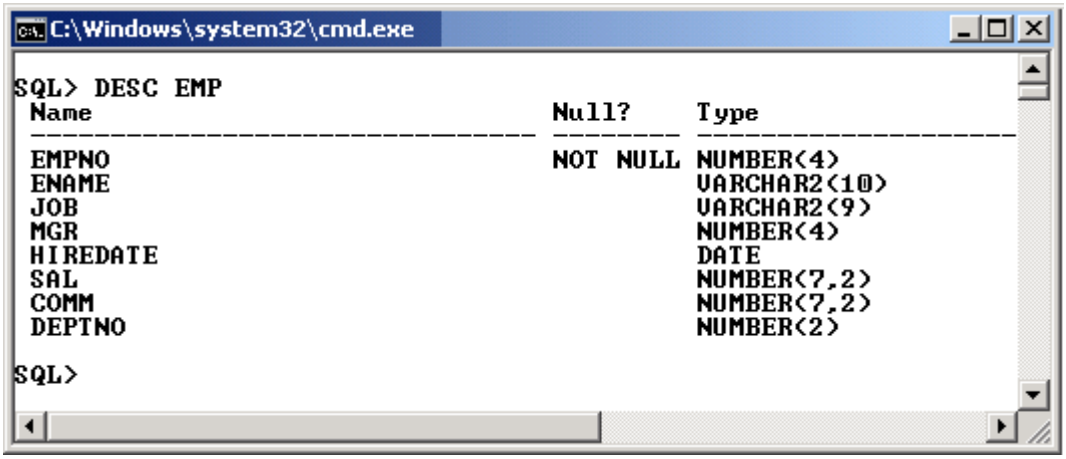


Рис. 1.27. **Отримання відомостей про структуру таблиці EMP**

У отриманому результаті перший стовпець (NAME) містить назви стовпців таблиці EMP. Другий стовпець (Null?) характеризує допустимість невизначених значень для конкретного стовпця таблиці, а третій (Type) – вказує на тип даних стовпця.

1.3.8. Повідомлення про помилки

Якщо SQL*Plus знайде помилку в команді, він вкаже на це за допомогою виведення повідомлення про помилку. Наприклад, якщо була допущена помилка в імені таблиці під час введення команди, повідомлення про помилку проінформує, що таблиці або подання (view) не існує:

```
SQL> DESCRIBE DPT
ERROR: ORA-0942: table or view does not exist
```

Якщо необхідно подальше пояснення, треба зробити один з таких кроків, щоб визначити причину помилки та виправити її:

- якщо помилка має номер, що починається з "ORA", потрібно знайти опис та роз'яснення у відповідному керівництві [69]:
- якщо помилка без номера, потрібно знайти правильний синтаксис команди, яка призвела до помилки, у довіднику команд SQL*Plus [73], "Довідкових керівництвах з мови SQL" [71; 72] або в "Керівництві користувача з PL/SQL" [70].

1.3.9. Додаткові можливості SQL*Plus

Збір часової статистики

Для збору та показу даних про часові ресурси, що витрачаються на виконання однієї або декількох команд або блоків, слід використовувати команду SQL*Plus TIMING. TIMING збирає дані за минулий період часу, зберігаючи інформацію про виконані команди в області хронометрування. Синтаксис та призначення параметрів команди TIMING такі:

<команда TIMING> :: = TIMI [NG] <команда таймера>
<команда таймера> :: = START [<ім'я таймера>] | SHOW | STOP.

де START – запускає таймер і дає йому вказане ім'я. Можна використовувати декілька активних таймерів, запускаючи додаткові за допомогою команди START. Останній запущений таймер стає поточним.

SHOW – видає ім'я та дані (час роботи) поточного таймера.

STOP – видає ім'я та дані (час роботи) поточного таймера, а потім зупиняє та видаляє таймер. Якщо активними є кілька таймерів, наступний, передостанній за часом запуску, таймер стає поточним.

Команда TIMING без параметрів видає кількість активних таймерів.

Приклад 1.15. Запустити таймер з ім'ям EXAMPLE1, після чого одразу виконати команди на отримання даних з таблиці DEPT і відображення часу роботи поточного таймера. Останнім кроком зупинити та видалити поточний таймер (рис. 1.28).

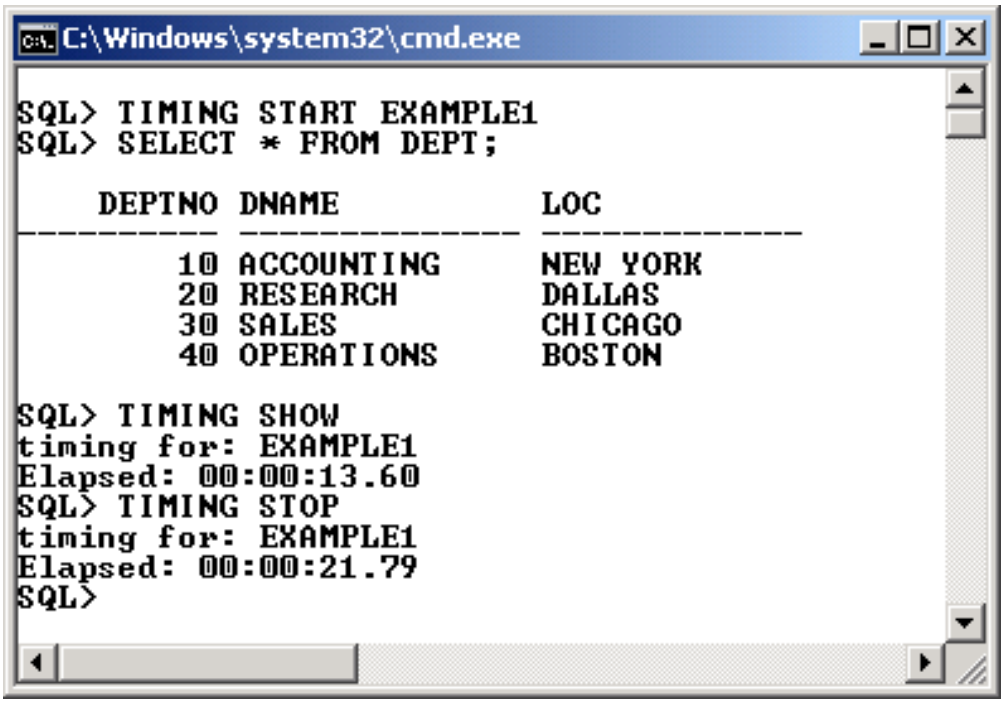


Рис. 1.28. **Приклад використання команди TIMING**

Звісно, що наведені результати пов'язані з тим, що команди виконувалися в інтерактивному режимі.

Можна створити командний файл (приклад 1.16), що містить аналогічні команди, і запустити його на виконання командою START, як це було зроблено у п. 1.3.1. під час створення тестової бази даних.

Приклад 1.16. Створити у будь-якому текстовому редакторі файл з такими рядками:

```
TIMING START EXAMPLE1
SELECT * FROM DEPT;
TIMING SHOW
TIMING STOP
```

та зберегти його під іменем TIMING.SQL. Запустити створений файл на виконання командою START. Результат виконання наведений на рис. 1.29.

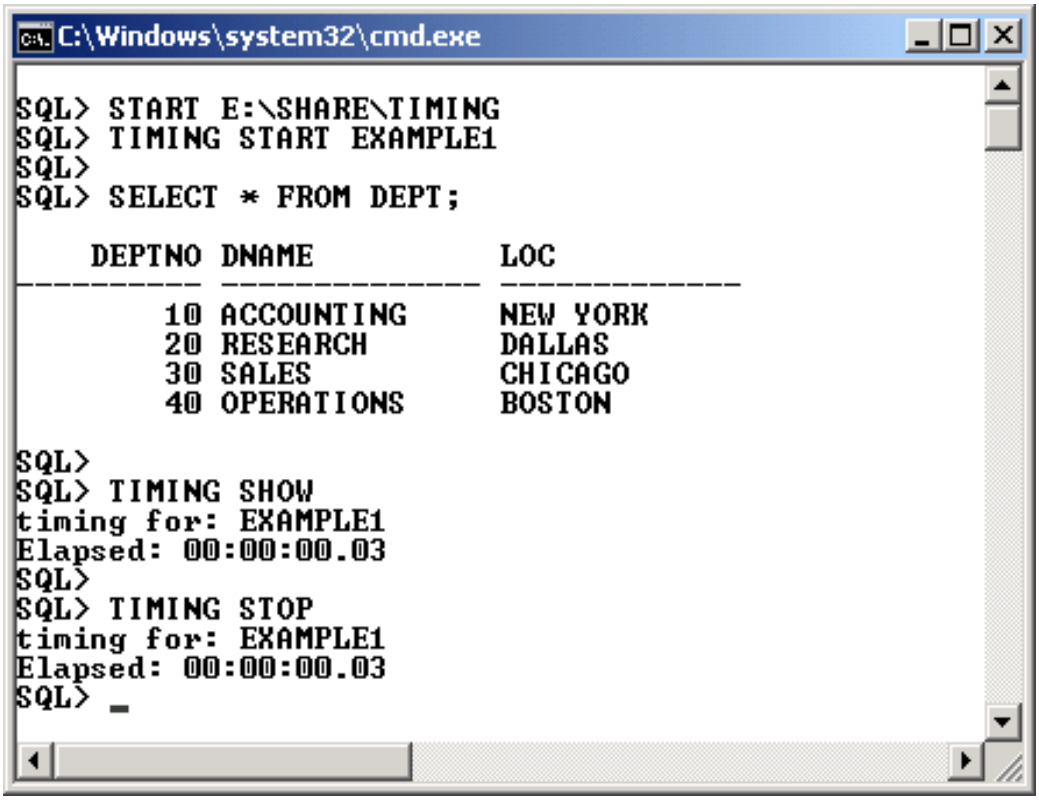


Рис. 1.29. **Хронометраж виконання запиту з командного файла**

Як видно з результатів тесту, час на виконання команд дорівнює не секундам, а сотим долям секунди, і більшу частину з них складає виведення на екран.

Виконання команд операційної системи

З командного рядка SQL*Plus можна виконувати команди ОС, не залишаючи SQL*Plus. Це корисно, коли необхідно виконати певне завдання (наприклад, друкування існуючого файла операційної системи).

Для виконання команди ОС необхідно ввести команду HOST з подальшою командою операційної системи.

Приклад 1.17. Не виходячи з програми SQL*Plus, отримати відомості щодо змісту диска C:\ (рис. 1.30).

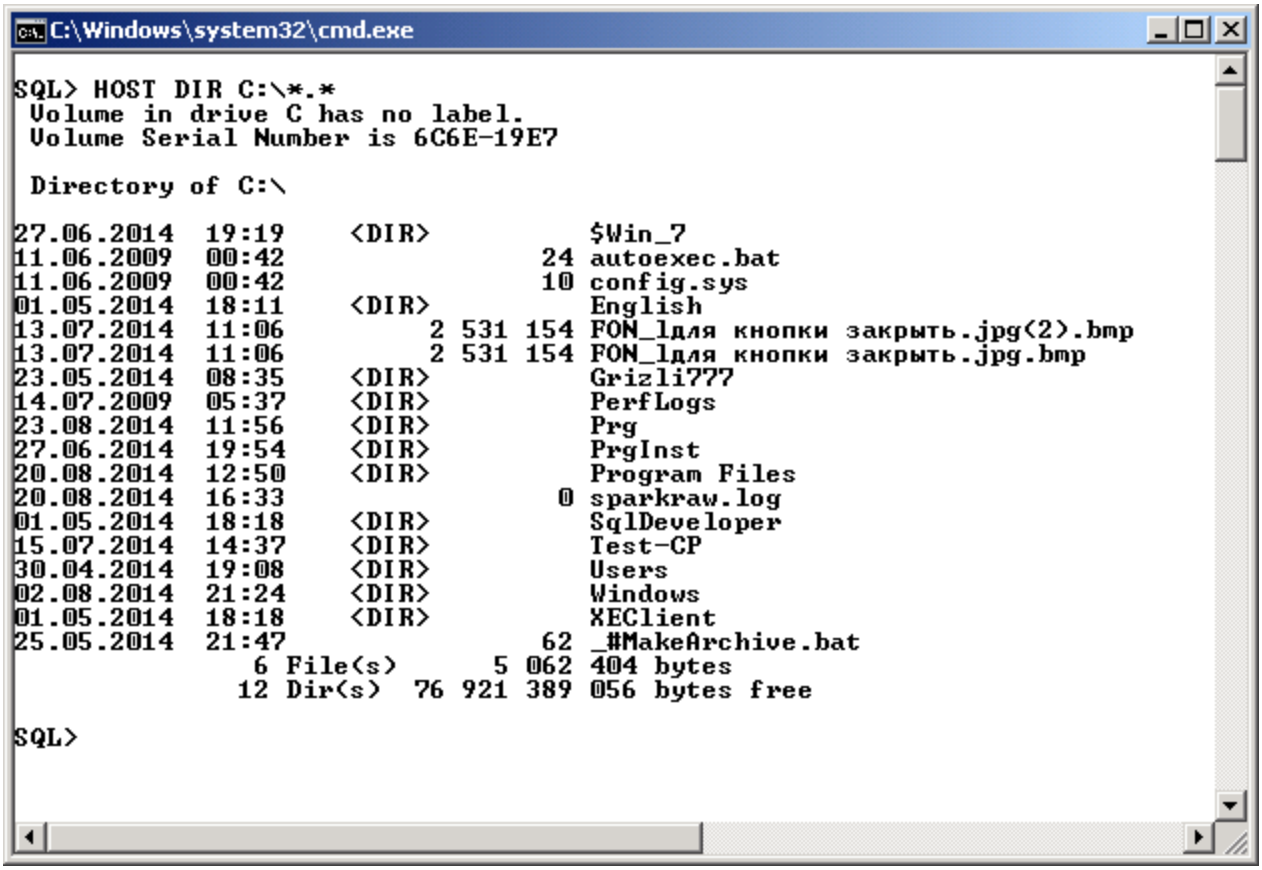


Рис. 1.30. **Виконання команд операційної системи**

Якщо просто ввести команду HOST, то у вікні з'явиться запрошення інтерпретатора команд операційної системи і можна буде вводити такі команди, доки не буде введено команду EXIT, яка поверне управління утиліті SQL*Plus (рис. 1.31.)

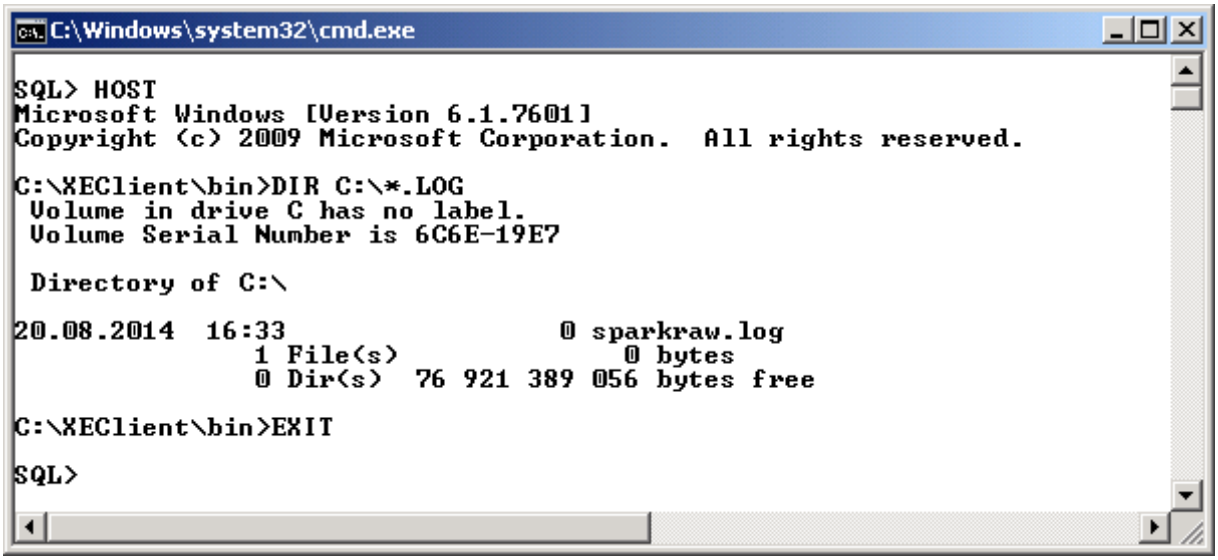


Рис. 1.31. **Перехід до інтерпретатора команд операційної системи**

Запис у файл і друкування результатів виконання запитів

Утиліта SQL*Plus дозволяє зберегти результати виконання команд у файлі ОС та роздрукувати їх на стандартному принтері. Такий процес називають ***спулінгом***. Для цього використовується команда SPOOL:

- **SPOOL <ім'я файла> –**- починає спулінг у файл;
- **SPOOL OFF** – припиняє спулінг і закриває файл;
- **SPOOL OUT** – припиняє спулінг і посилає файл на стандартний принтер базової операційної системи.

Спулінг дозволяє зберегти у файлі повний протокол роботи з системою, щоб у подальшому проаналізувати ті чи інші робочі моменти або виявити помилки.

Якщо процес спулінг був активізований, то перед завершенням роботи з SQL*Plus його бажано закрити командою **SPOOL OFF,** інакше файл може виявитись неповним, або навіть порожнім, оскільки певна частина інформації, що зберігалася у буферах операційної системи, не була збережена на зовнішні носії.

Якщо не вказано розширення файла, використовується стандартне розширення LST.

Очищення (скидання) значень параметрів

Команда CLEAR дозволяє скинути значення низки опцій утиліти SQL*Plus, зокрема, пов'язаних з форматуванням результатів. Наприклад, щоб очистити екран SQL*Plus слід ввести таку команду:

CLEAR SCREEN

Більш детально команда CLEAR розглянута в пункті 1.6.4.

1.3.10. Команди редагування буфера SQL

Команди редагування дозволяють виправити команду, що була введена, без повторного її набору на клавіатурі. Для цього можна використовувати номери рядків для редагування команд SQL або блоків PL/SQL, що зберігаються у даний момент у буфері. Також є можливість редагувати вміст буфера за допомогою редактора ОС.

У табл. 1.3 наводиться перелік команд SQL*Plus, що дозволяють переглядати та змінювати команду в буфері без її повторного введення.

Таблиця 1.3

Команди порядкового редагування тексту в буфері SQL

Команда	Скорочення	Призначення команди
APPEND text	A text	Додати текст у кінець рядка
CHANGE /old/new	C /old/new	Замінити old на new
CHANGE /text	C /text	Видалити text з рядка
CLEAR BUFFER	CL BUFF	Видалити усі рядки
DEL	(немає)	Видалити рядок
INPUT	I	Додати один або більше рядків
INPUT text	I text	Додати рядок, який містить text
LIST	L	Вивести на екран усі рядки буфера SQL
LIST n	L n або n	Виведення на екран n-го рядка
LIST *	L *	Виведення на екран поточного рядка
LIST LAST	L LAST	Виведення на екран останнього рядка
LIST m n	L m n	Виведення на екран діапазону рядків (m / n)

Виведення на екран (друкування) вмісту буфера

Будь-які команди редагування (крім LIST) впливають тільки на один рядок у буфері. Цей рядок називається поточним і позначається зірочкою, коли друкується поточна команда або блок.

Приклад 1.18. Використовуючи команду LIST, роздрукувати поточну команду, що зберігається у SQL буфері (рис. 1.32).

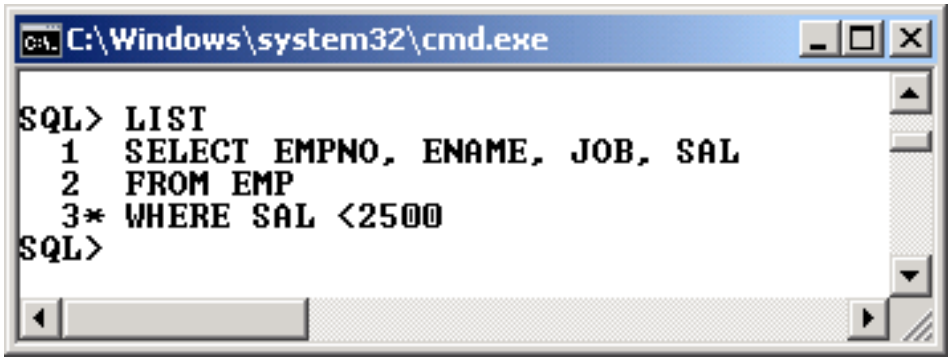


Рис. 1.32. Виведення на екран вмісту буфера SQL

Поточним, у даному випадку, є третій рядок.

Редагування поточного рядка

Команда SQL*Plus CHANGE дозволяє редагувати поточний рядок. Різноманітні дії визначають, який рядок є поточним:

• LIST даного рядка робить його поточним;

• коли друкується або виконується команда у буфері, останній рядок команди стає поточним (однак використання символу "/" для запуску команди не впливає на положення поточного рядка);

• якщо отримано повідомлення про помилку, рядок з помилкою автоматично стає поточним.

Замість повторного введення команди можна виправити помилку, редагуючи команду в буфері. Рядок, який містить помилку, стає поточним рядком. Використовуючи команду CHANGE, можна виправити помилку. Ця команда складається з трьох частин, розділених похилою рискою (слешем) або іншим не алфавітно-цифровим символом:

• слово CHANGE;

• послідовність символів, які потрібно змінити;

• послідовність символів, на які потрібно замінити.

Команда CHANGE знаходить перше входження послідовності символів в рядку, який необхідно змінити, і змінює її на нову послідовність символів.

За необхідності ввести рядок знову, не потрібно використовувати команду CHANGE. Для цього треба набрати рядок знову, вводячи номер рядка, через пропуск новий текст і натиснути [Enter].

Приклад 1.19. Замінити значення заробітної плати в умові пошуку з 2500 на 3000 за допомогою команди CHANGE:

CHANGE /2500/3000,

та змінити умови пошуку, щоб вибирати усіх менеджерів:

3 WHERE JOB='MANAGER'.

Результати виконання команд наведені на рис. 1.33.

Додавання нового рядка

Щоб вставити (додати) новий рядок після поточного, слід використати команду INPUT (її можна скорочувати: **I**).

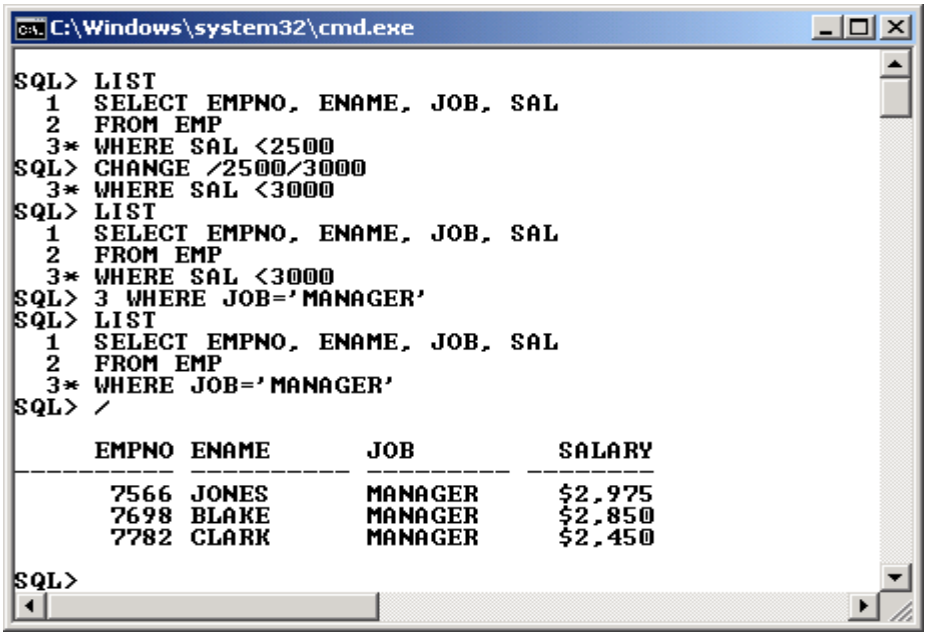


Рис. 1.33. **Приклади використання команди CHANGE**

Приклад 1.20. Додати четвертий рядок у команду SQL, яка змінювалася у попередньому прикладі (див. рис. 1.33), з додатковою умовою пошуку. Оскільки третій рядок вже є поточним, потрібно ввести INPUT і натиснути [Enter]. SQL*Plus видасть підказку для введення нового рядка:

```
SQL> INPUT
4
```

Ввести новий рядок і знов натиснути [Enter]. SQL*Plus знову видасть підказку для нового рядка:

```
4 AND SAL>2500
5
```

Натиснути [Enter], щоб повідомити SQL*Plus про припинення введення нових рядків, і потім використати команду RUN для перевірки та нового виконання запиту (рис. 1.34).

Додавання тексту у кінець рядка

Щоб додати текст в кінець рядка в буфері, треба використовувати команду APPEND. Для цього зазвичай спочатку використовують команду LIST (або безпосередньо номер рядка) для рядка, який необхідно змінити, щоб зробити його поточним. Потім вводять команду APPEND і текст, який необхідно додати. Якщо текст починається з пробілу, потрібно відокремити слово APPEND від першого символу тексту двома пробілами (перший пробіл – це роздільник, а другий – заноситься у буфер).

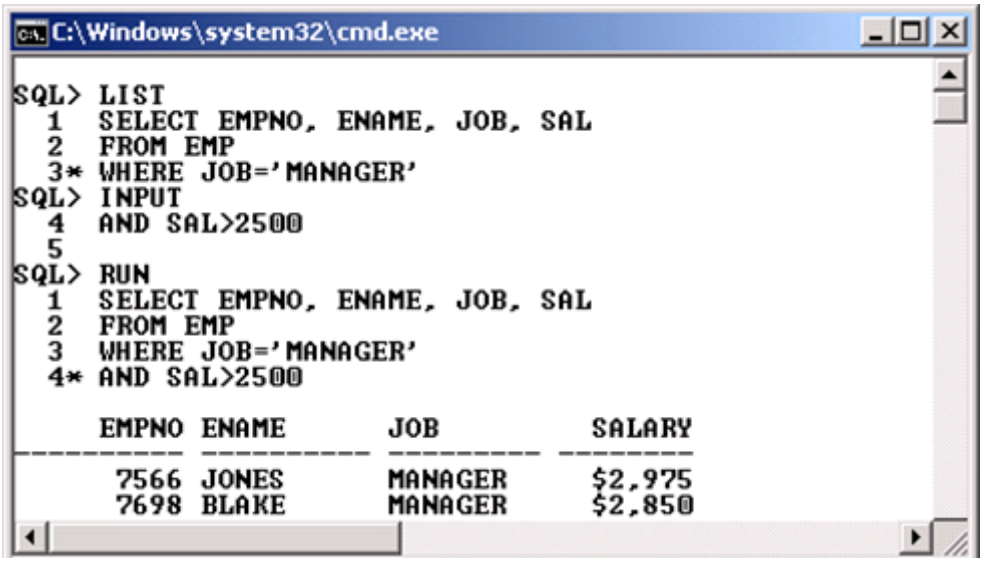


Рис. 1.34. **Додавання нового рядка у буфер SQL**

Приклад 1.21. Додати у результат з останнього прикладу (див. рис. 1.34) ще один стовпець – DEPTNO.

Для вирішення поставленої задачі потрібно змінити перший рядок запиту. Виконати команду **LIST 1**, щоб зробити рядок поточним, потім команду **APPEND.DEPTNO,** щоб додати у кінець першого рядка додатковий стовпець для виведення. Результати виконання наведені на рис. 1.35.

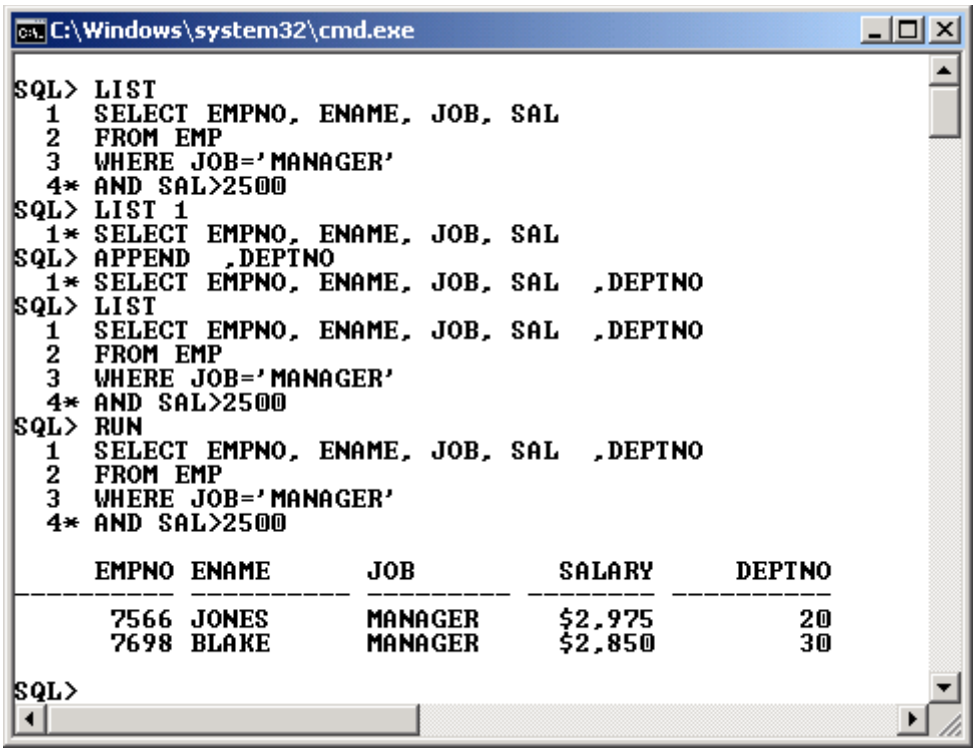


Рис. 1.35. **Приклад додавання тексту в кінець рядка**

Видалення рядків

Для видалення поточного рядка з буфера використовується команда DEL. Тобто спочатку необхідно командою **LIST <номер рядка>** роздрукувати рядок, який необхідно видалити, потім ввести команду DEL.

Команда DEL робить наступний рядок у буфері поточним. Таким чином, послідовно можна видаляти кілька рядків, спочатку роблячи рядок поточним, а потім вводячи команду DEL.

Нові версії SQL*Plus мають розширені можливості для команди DEL. Наприклад, вказавши DEL 3 * можна видалити рядки з третього по поточний, а команда DEL 5 LAST видаляє усі рядки з п'ятого до останнього.

Приклад 1.22. За допомогою команди DEL вилучити третій та четвертий рядки з команди SQL, що знаходиться у буфері. Результат операції наведено на рис. 1.36.

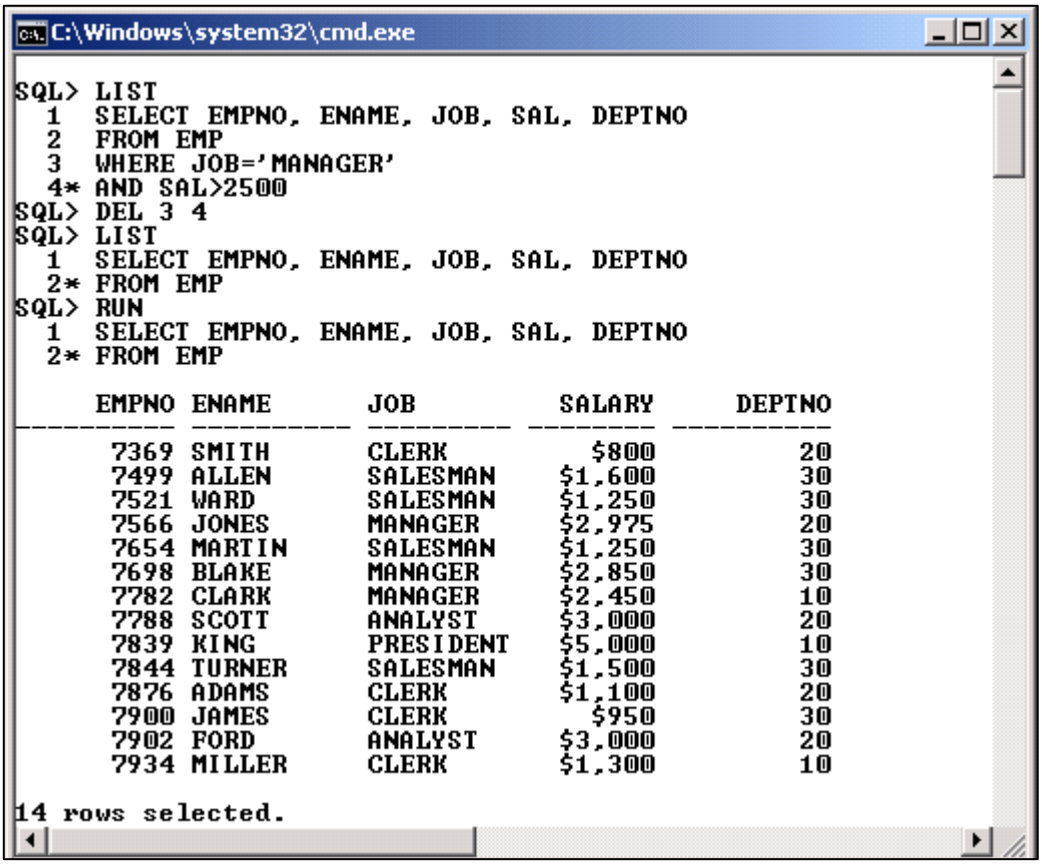


Рис. 1.36. **Приклад видалення рядків з буферу SQL**

Редагування команди системним текстовим редактором

Зазвичай, операційна система має один або декілька текстових редакторів, які можна використовувати для створення та редагування

файлів. Текстовий редактор виконує ті самі функції, що і команди редагування SQL*Plus, але він значно зручніший.

Можна запускати системний текстовий редактор без виходу із SQL*Plus за допомогою введення команди EDIT. При цьому за замовчуванням викликається стандартний системний редактор, наприклад, **Notepad** операційної системи Windows.

Закінчивши редагування редактором, слід його закрити, при цьому команда SQL, або блок PL/SQL, що редагувався, з'являється у вікні SQL*Plus і може бути запущений на виконання за допомогою команди RUN або слешу "/" (рис. 1.37, 1.38).

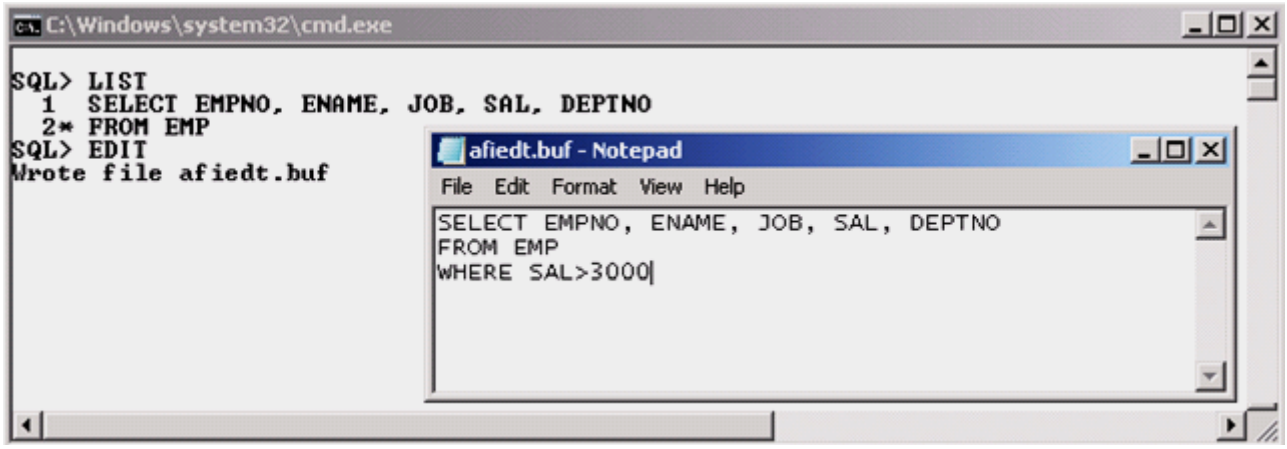


Рис. 1.37. **Виклик та редагування буфера SQL текстовим редактором**

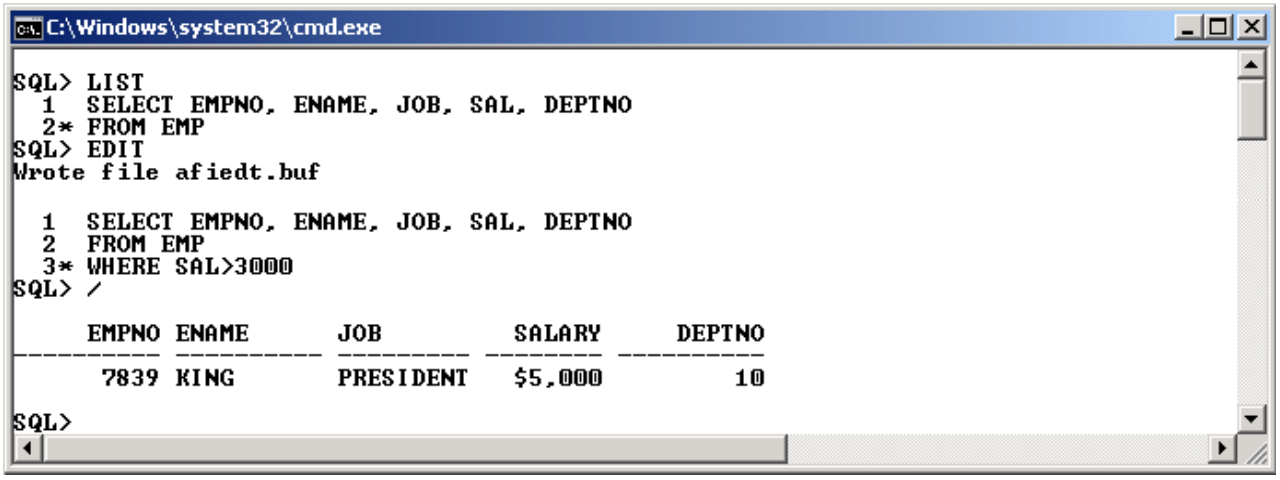


Рис. 1.38. **Виконання команди після редагування редактором**

1.3.11. Змінні користувача та змінні підстановки

Працюючи із СКБД Oracle за допомогою SQL*Plus, користувач має можливість вживати змінні, які певною мірою нагадують змінні в мовах

програмування. По-перше, користувач може визначати безпосередньо "свої" змінні, додавши їх до списку системних, і використовувати їх у подальшій роботі; по-друге, використовувати так звані змінні підстановки, значення яких зазвичай визначаються в інтерактивному режимі при виконанні запитів.

Визначення змінних користувача

Можна визначати змінні, які називаються змінними користувача, і повторно їх використовувати під час виконання тих чи інших команд. Створити (визначити) змінну користувача можна за допомогою команди SQL*Plus **DEFINE**. Змінну користувача можна використовувати і в заголовках звітів, що розглянуті у підрозділі 1.6.

Приклад 1.23. Визначити змінну користувача EMPLOYEE і присвоїти їй значення "SMITH".

Для визначення змінної користувача EMPLOYEE і надання їй значення "SMITH", потрібно ввести команду:

DEFINE EMPLOYEE = SMITH

Для перевірки опису існуючих у системі змінних використовується команда DEFINE без параметрів (рис. 1.39).

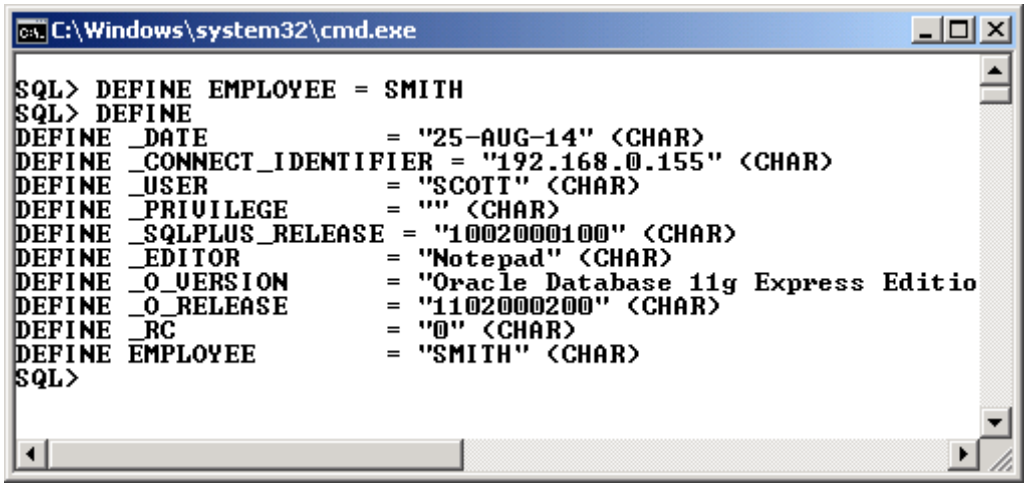


Рис. 1.39. **Перегляд значень змінних**

Пояснення. Імена системних змінних починаються з підкреслення, і вони у прикладі роздруковані спочатку, потім виводиться інформація про змінну користувача EMPLOYEE.

Для перевірки опису конкретної змінної вводиться команда DEFINE з іменем змінної. Наприклад на команду DEFINE EMPLOYEE – SQL*Plus роздрукує опис: DEFINE EMPLOYEE = "SMITH" (CHAR)

Усі змінні, які явно визначені командою DEFINE, отримують тип даних CHAR.

Використання змінних підстановки

Змінна підстановки – це ім'я змінної користувача з одним або двома попередніми амперсандами (&). Коли SQL*Plus зустрічає змінну підстановки у команді, він виконує команду так, ніби вона містить значення цієї змінної, а не саму змінну.

Приклад 1.24. Визначити змінну SORTCOL та MYTABLE і надати їм значення JOB та EMP, відповідно. З використанням створених змінних створити та виконати запит

```
SELECT ENAME, & SORTCOL, SAL
       FROM & MYTABLE
      ORDER BY & SORTCOL;
```

до бази даних (рис. 1.40).

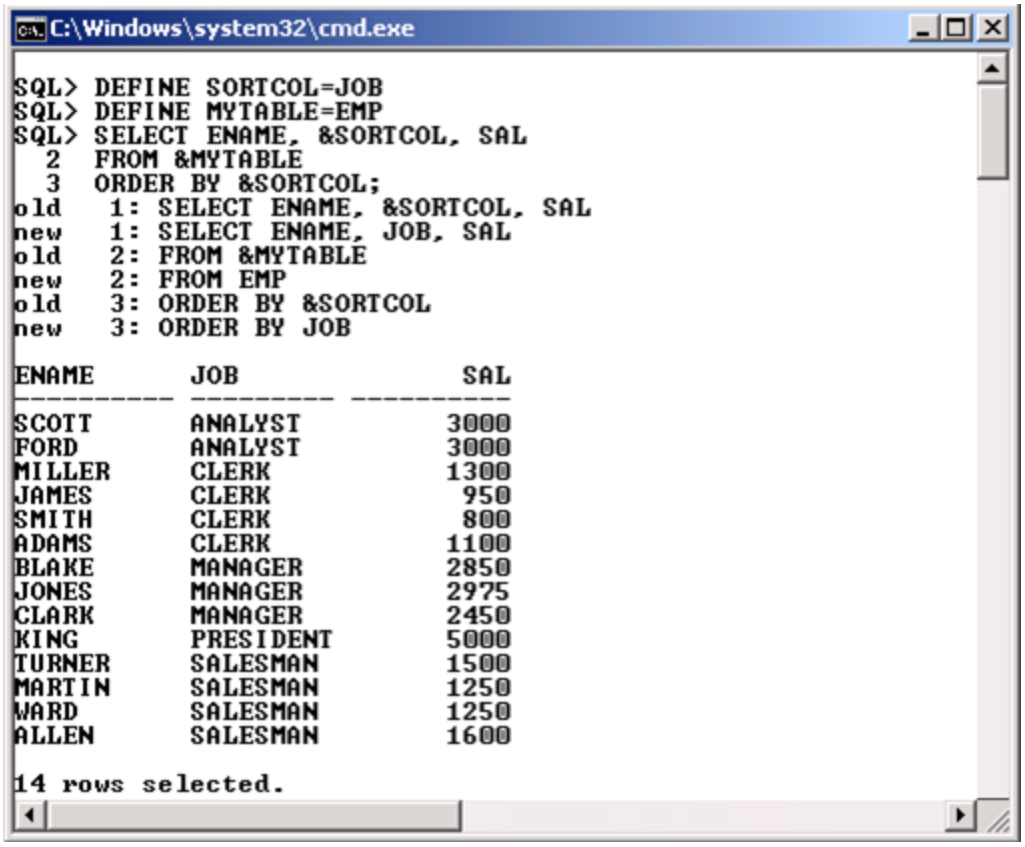


Рис. 1.40. **Приклад виконання команди зі змінними користувача**

Пояснення. При виконання запиту рядки, що містили змінні, були змінені, і замість виразів &SORTCOL та &MYTABLE були підставлені їх значення, а саме – JOB та EMP. Тобто фактично був виконаний запит

```
SELECT ENAME, JOB, SAL
       FROM EMP
       ORDER BY JOB;
```

Приклад 1.25. Змінити рядок SELECT ENAME, & SORTCOL, SAL попередньої команди, додавши у його кінець ще одну змінну (&NUM), і знов виконати запит (рис.1.41).

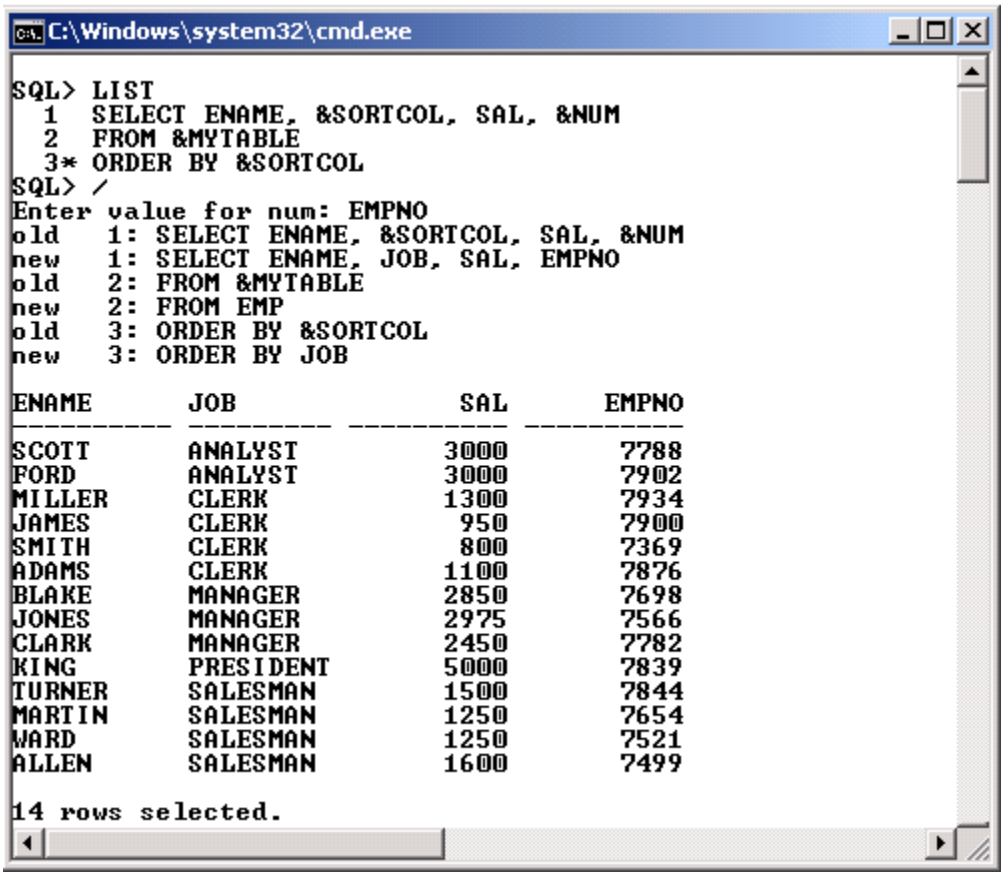


Рис. 1.41. **Приклад виконання команди зі змінними підстановки**

Пояснення. Що трапилося під час виконання запиту? Оскільки змінні &SORTCOL та &MYTABLE вже визначені, замість них підставилися відповідні значення. Що ж стосується змінної &NUM, то вона не була до цього часу визначена, тому SQL*Plus у діалозі з користувачем просить ввести значення для цієї змінної. Цей момент відображено рядком

Enter value for num:

У відповідь було введено слово EMPNO (це назва стовпця з таблиці EMP), і у результаті був отриманий новий стовпець.

Якщо знову запустити на виконання останній запит, то SQL*Plus знову запропонує ввести значення змінної NUM.

Усунення необов'язкової підказки для введення значення

Можна уникнути повторного введення значень змінних за допомогою вказівки другого амперсанда перед іменем змінної. SQL*Plus автоматично визначить змінну підстановки з двома амперсандами, але не зробить цього для змінної з одним амперсандом.

Таким чином, якщо змінна підстановки зустрічається протягом сеансу більше одного разу, SQL*Plus використає певне значення для змінних з двома амперсандами і попросить введення значення для змінних з одним амперсандом (рис. 1.42).

Приклад 1.26. Змінити попередній запит (приклад 1.25), додавши до змінної NUM ще один амперсанд. Виконати запит два рази поспіль (рис. 1.42).

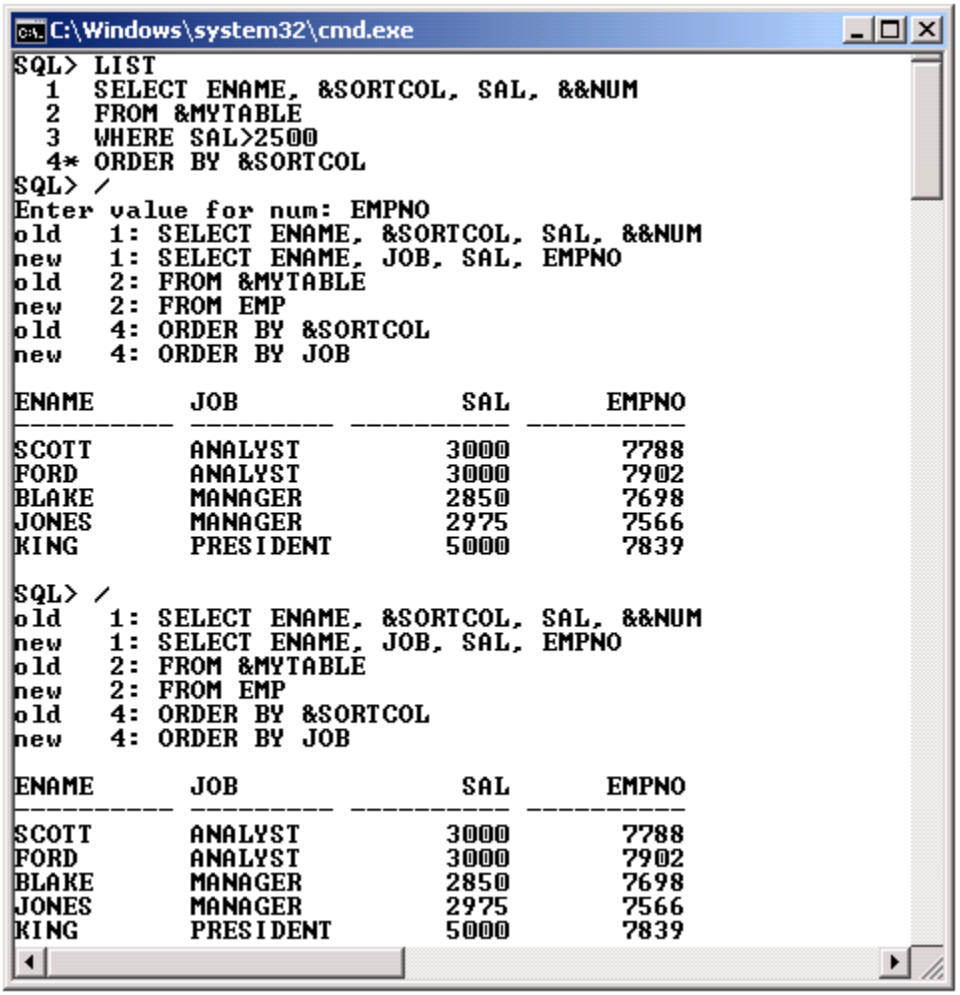


Рис. 1.42. **Використання змінної підстановки з двома амперсандами**

Як видно з прикладу, при першому виконанні запиту змінна NUM ще була не визначена, тому з'явився запит на введення відповідного значення. При повторному виконанні ця операція була відсутня.

Змінні підстановки можна використовувати в будь-якому місці в командах SQL*Plus і SQL, крім першого слова, що вводиться після командної підказки.

Використання змінних підстановки дозволяє, створивши шаблон запиту, багаторазово його виконувати з різними значеннями імен стовпців, таблиць, умов пошуку тощо.

Не можна використовувати підстановочні змінні у командах редагування буфера (APPEND, CHANGE, DEL, INPUT) та в інших командах, де підстановка ніщо не означає (таких, як REMARK і TIMING).

Команди редагування буфера, APPEND, CHANGE, DEL, INPUT обробляють текст, що починається з "&" та "&&" так само, як і будь-який інший текстовий рядок.

Системні змінні та змінні підстановки

Деякі системні змінні, що обумовлені командою SET, впливають на змінні підстановки:

- SET SCAN – включення / виключення підстановки;
- SET DEFINE – визначення символу підстановки (за замовчуванням – "&");
- SET ESCAPE – визначення символу перемикання (ESC), який можна використовувати перед символом підстановки. Символ ESC вказує SQL*Plus на те, що символ підстановки треба сприймати як звичайний символ. За замовчуванням, ESC є зворотною косою рисою (зворотним слешем) \);
- SET VERIFY ON – друкування кожного рядка командного файла до і після підстановки;
- SET CONCAT – визначення символу, що відділяє ім'я змінної підстановки або параметр від іншого тексту, який розташовується безпосередньо за змінною або параметром, за замовчуванням – це крапка (.).

Запитання і завдання

1. Яке призначення має утиліта SQL*Plus?

2. Які команди утиліти SQL*Plus вам відомі? Опишіть їх призначення.

3. За допомогою яких засобів можна вести облік команд, які виконав користувач у середовищі SQL*Plus?

1.4. Командні файли

1.4.1. Створення командного файла

Якщо та чи інша група команд SQL, блоків PL/SQL, команд SQL*Plus виконується часто, їх можна не вводити в інтерактивному режимі, а запам'ятати у командних файлах і потім виконувати багаторазово. Створити командний файл в рамках SQL*Plus можна такими способами:

- за допомогою команди SAVE, що зберігає вміст буфера;
- за допомогою команди INPUT, що вводить рядки у буфер, а потім збереження вмісту буфера командою SAVE;
- за допомогою команди EDIT для створення файла системним редактором;
- за допомогою будь-якого іншого текстового редактора операційної системи.

Приклад 1.27. Створити командний файл з іменем INIT шляхом збереження вмісту буфера командою SAVE. Команді необхідно вказати ім'я створюваного файла. За замовчуванням, командний файл має розширення.SQL.

Роздруківка вмісту буфера та його збереження командою SAVE у файл з іменем INIT.SQL наведена на рис. 1.43

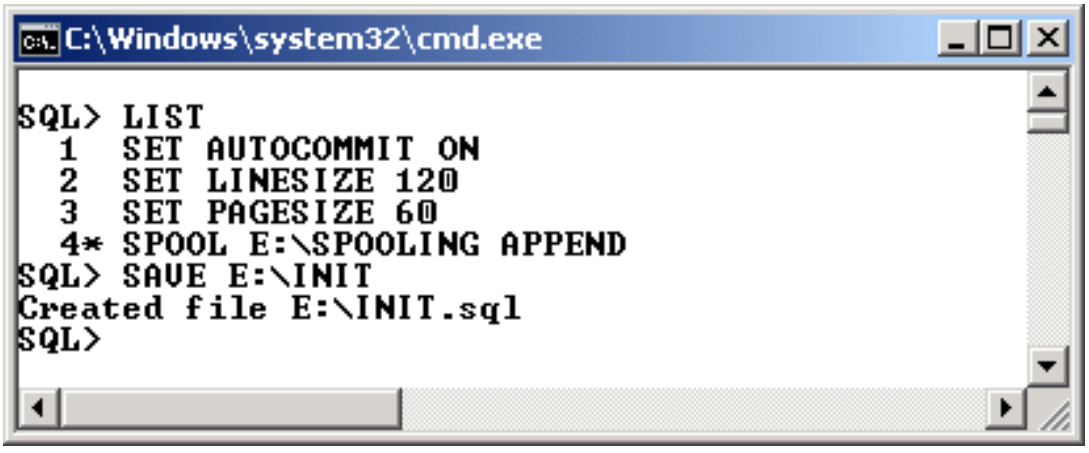


Рис. 1.43. **Збереження вмісту буфера SQL у файлі**

1.4.2. Вибір та виконання командного файла

Для завантаження командного файла в буфер треба звернутися до нього за іменем. Це можна зробити, використовуючи команду GET утиліти SQL*Plus. Вона має такий формат: **GET <ім'я_файла>**

Команда **START**, на відміну від **GET**, не тільки завантажує командний файл але й виконує команди, які у ньому містяться. У команді START вказується ім'я файла: **START <ім'я_файла>**

Для запуску командного файла можна також використати команду @, наприклад: @ **SALES**

Команда @ виводить на екран і виконує команди із зазначеного командного файла аналогічно команді START.

Команди START і @ залишають у буфері останню команду SQL або блок PL/SQL.

Приклад 1.28. Запустити на виконання командний файл INIT, створений у попередньому прикладі (рис. 1.44)

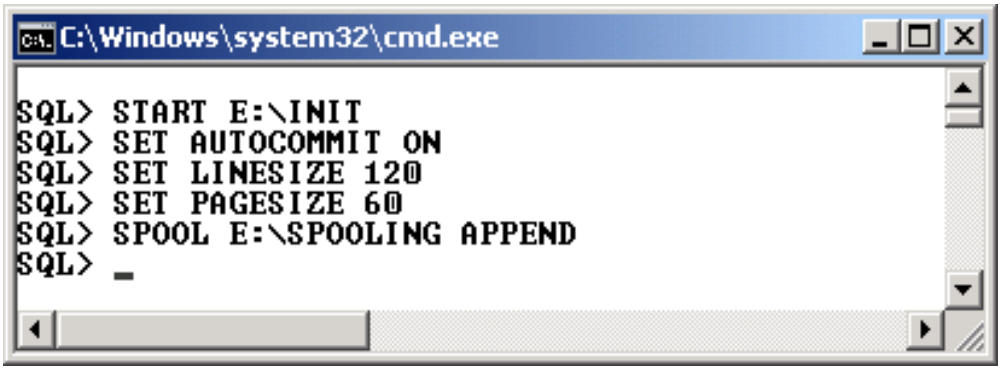


Рис. 1.44. **Запуск командного файла на виконання**

Після виконання командного файла значення відповідних системних змінних, а саме – AUTOCOMMIT, LINESIZE та PAGESIZE, будуть змінені на нові, а також активується дописування у файл E:\SPOOLING.LST усіх команд та результатів їх виконання.

1.4.3. Виконання командного файла у момент старту SQL*Plus

Якщо під час старту SQL*Plus треба постійно виконувати ті чи інші команди, наприклад, налаштування роботи утиліти за допомогою значень системних змінних, то таку операцію можна зробити автоматичною, виконуючи командний файл під час запуску SQL*Plus. Для цього потрібно вказати після SQLPLUS ім'я, похилу риску, пароль, пробіл, @, ім'я файла:

SQLPLUS SCOTT/TIGER @E:\INIT

Утиліта SQLPLUS стартує і виконує командний файл E:\INIT.SQL. Якщо вказати після SQLPLUS ім'я, пробіл, @ та ім'я файла, наприклад:

SQLPLUS SCOTT @ E:\INIT

У цьому разі утиліта SQLPLUS попросить ввести пароль, стартує та виконає командний файл E:\INIT.SQL.

1.4.4. Розміщення коментарів у командних файлах

Коментарі у командні файли можна вводити трьома способами:

- використовуючи команду REMARK утиліти SQL*Plus;
- використовуючи обмежники коментаря /*...*/ мови SQL;
- використовуючи коментарі "--" мови PL/SQL.

Команда REMARK вводиться на окремому рядку в командному файлі з наступним коментарем на тому ж рядку. Не можна розривати одну команду SQL командами REMARK.

Обмежники коментарю SQL /*...***/** можна вводити на окремих рядках командного файла або в рядку, що містить команду SQL чи блок PL/SQL. Коментар може містити кілька рядків:

Якщо ввести коментар SQL безпосередньо після командної підказки, SQL*Plus не збереже коментар у буфері. Коментарі PL/SQL "--" закінчуються на тому ж рядку, де записані. Тобто вони діють до кінця рядка і для них не потрібен кінцевий обмежник.

Обмежники коментарю SQL /*...***/** та коментарі PL/SQL "--" можуть зустрічатися усередині запиту SQL.

Приклад 1.29. Створити командний файл з використанням усіх трьох способів розміщення коментарів.

Результат може виглядати таким чином:

```
/* This is an example script that
shows, how to use comments */
REMARK  Employees of the department 10
SELECT * FROM EMP
WHERE DEPTNO=10 -- Condition for dept 10
```

1.4.5. Вкладеність командних файлів

Щоб виконати декілька командних файлів поспіль, спочатку необхідно створити головний командний файл, що містить кілька команд START із зазначенням імен інших командних файлів, які необхідно запустити. Потім виконати головний командний файл, який складається з команд START.

Приклад 1.30. Створити три командні файли, кожний з яких виводить інформацію про структуру та вміст окремої таблиці БД (EMP, DEPT та SALGRADE). Назвемо ці файли відповідно TABLE_EMP, TABLE_DEPT та TABLE_SALGRADE. Крім того, створимо командний файл на ім'я TABLES_INFO.SQL, який, у свою чергу, буде складатися з виклику цих трьох командних файлів.

Кожен з трьох початкових файлів містить дві команди DESC та SELECT, відповідно, для отримання інформації про структуру конкретної таблиці та її вміст (рис. 1.45).

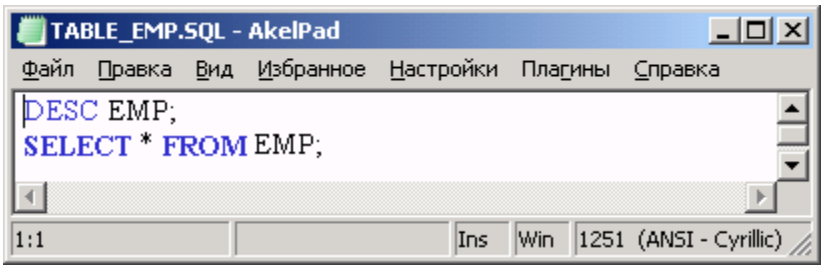


Рис. 1.45. **Файл TABLE_EMP.SQL у редакторі**

Головний командний файл TABLES_INFO буде складатися з таких рядків:

```
START E:\TABLE_EMP
START E:\TABLE_DEPT
START E:\TABLE_SALGRADE
```

Після запуску файла TABLES_INFO на виконання, на екрані з'явиться необхідна інформація (рис. 1.46).

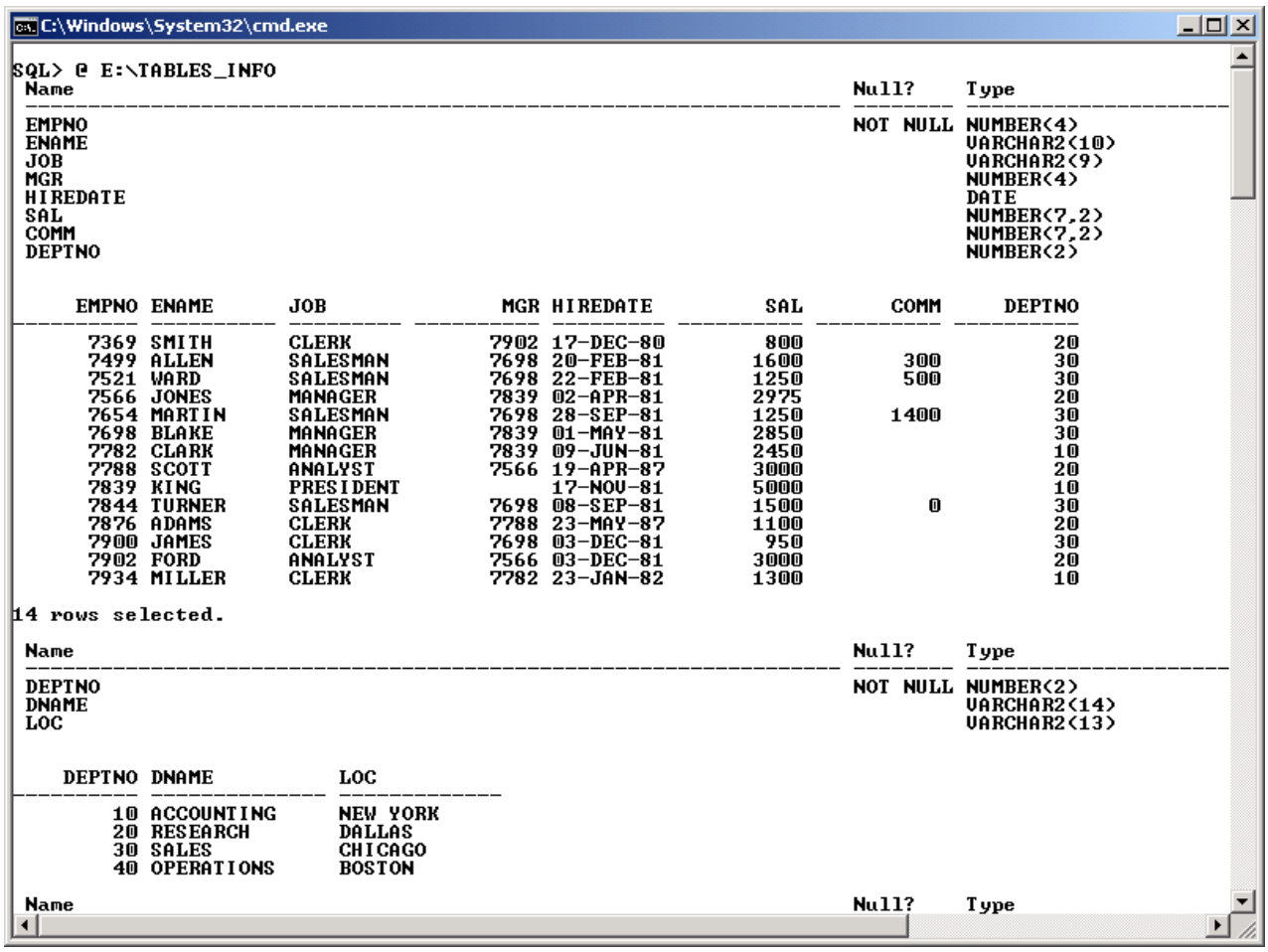


Рис. 1.46. **Виконання вкладених командних файлів** (початок)

1.4.6. Вихід з командного файла з кодом повернення

Якщо командний файл у процесі виконання генерує помилку операційної системи або SQL, можна перервати виконання командного файла і вийти з нього з кодом повернення або обробити помилку. Для цього використовуються команди SQL*Plus **WHENEVER OSERROR** або **WHENEVER SQLERROR,** наприклад:

```
SQL> WHENEVER OSERROR EXIT 7
SQL> WHENEVER SQLERROR EXIT SQL.SQLCODE
```

Приклад 1.31. Навести вміст командного файла з помилковою командою SQL та передбачити у файлі реакцію на подібні помилки.

Результат може виглядати таким чином:

```
WHENEVER SQLERROR EXIT SQL.SQLCODE
DESC EMP;
SELECT * FROM EMPT;
```

Якщо командний файл з таким вмістом запустити на виконання, то SQL*Plus завершує свою роботу, бо під час виконання команди SELECT виникне помилка, оскільки таблиці з ім'ям EMPT не існує.

1.4.7. Написання діалогових програм

Три команди SQL*Plus – PROMPT, ACCEPT, PAUSE допомагають спілкуватися з кінцевим користувачем, тобто користувачем утиліти SQL*Plus. Ці команди дозволяють посилати повідомлення та приймати введення даних від користувача, включаючи просте натискання [Enter]. Також можна використовувати PROMPT і ACCEPT для зміни підказки під час введення значень, які автоматично генерує SQL*Plus для змінних підстановки. Використання цих команд у командних файлах фактично дозволяє перетворити їх на невеликі діалогові програми, які в інтерактивному режимі дозволяють виконати відповідні скрипти.

Підказка та введення значень користувача

Використовуючи команди PROMPT і ACCEPT, можна посилати повідомлення користувачу і приймати дані від нього. Команда PROMPT лише виводить вказане повідомлення на екран. Цю команду слід використовувати для передачі інструкцій або інформації користувачеві. Команда ACCEPT видає підказку користувачу для введення значення і запам'ятовує його у зазначеній змінній користувача.

Приклад 1.32. Створити командний файл на ім'я SALARY, який після запуску у діалоговому режимі запитує значення мінімальної та максимальної заробітної плати, а потім виводить відомості про співробітників, чия заробітна плата знаходиться у вказаному діапазоні.

Вміст файла SALARY з командами PROMPT та ACCEPT буде таким:

```
PROMPT Enter a title
ACCEPT MYTITLE PROMPT 'Title:'
PROMPT Enter  the range of salaries
ACCEPT SALMIN NUMBER  PROMPT 'Min salary: '
ACCEPT SALMAX NUMBER  PROMPT 'Max salary: '
TTITLE CENTER MYTITLE SKIP 2
SELECT * FROM EMP
WHERE SAL >= &SALMIN AND SAL <=&SALMAX;
```

Створити його можна будь-яким шляхом, описаним у п. 1.4.1.

Команда TITLE у цьому прикладі визначає текст заголовка звіту, після якого буде пропущено два рядки. Ця команда більш детально буде розглянута у п. 1.6.5.

Запустивши командний файл на виконання, отримується такий результат (рис. 1.47).

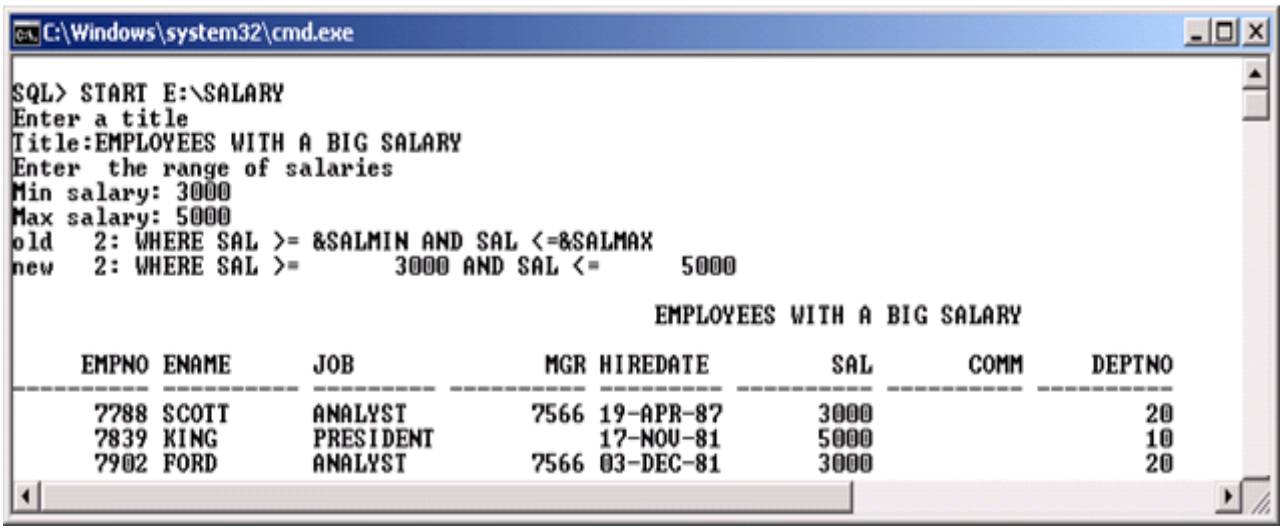


Рис. 1.47. **Результат виконання командного файла SALARY.SQL**

Як видно з отриманого результату, спочатку було запрошено назву звіту, а потім – значення мінімальної та максимальної заробітної плати. Після введення відповідних значень вони були підставлені у конкретні команди, і на екрані з'явився перелік співробітників, у яких заробітна платня знаходиться у діапазоні від 3000 до 5000.

У наведеному прикладі усі значення, що вводилися з клавіатури, були правильними. А що станеться при введенні помилкових даних?

Приклад 1.33. Виконати командний файл SALARY, але замість значення заробітної плати ввести певний текст (рис. 1.48).

Під час виконання скрипта спроба ввести символьне значення для числової змінної (SALMIN) викликала повідомлення про помилку. Це повідомлення з'являється, доки не буде введене правильне числове значення.

Щоб значення введеного заголовка не з'являлося у подальших прикладах, його бажано відключити командою

SQL> TTITLE OFF

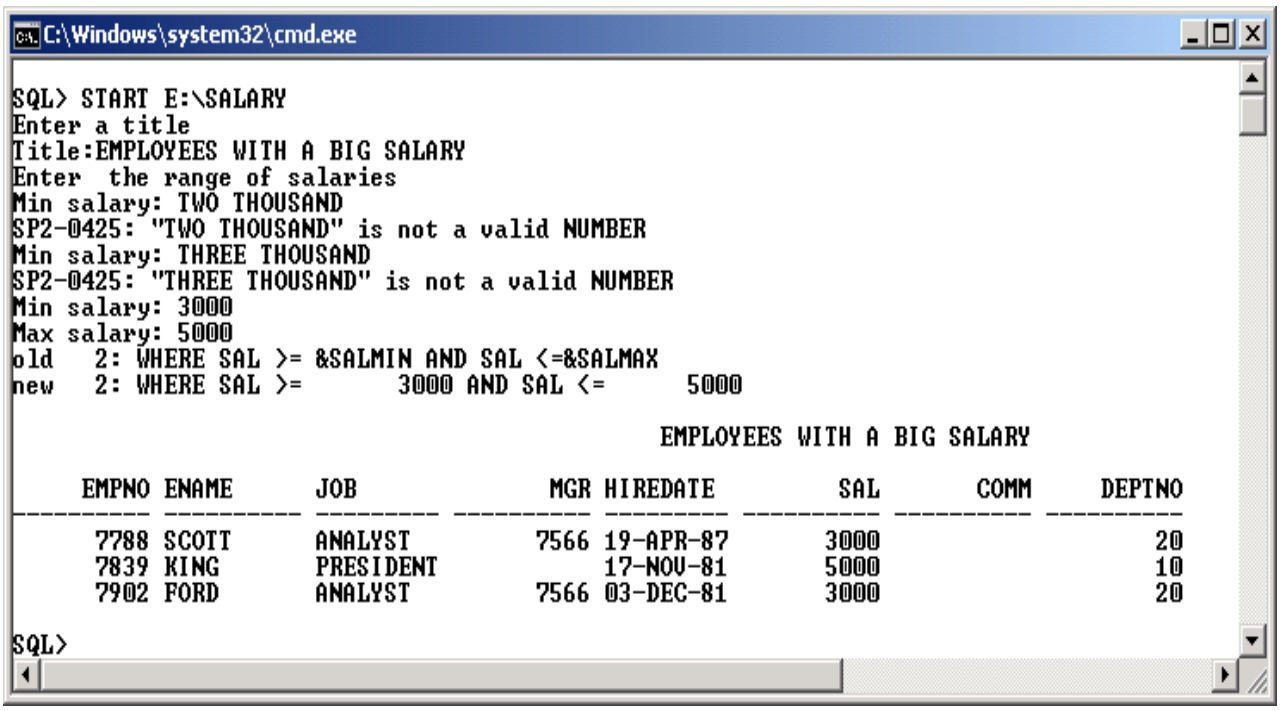


Рис. 1.48. **Повідомлення про помилку під час введення даних**

Важливо

Команди, які знаходяться у командних файлах, можуть мати змінні підстановки. У цьому випадку при виконанні такого командного файла для введення значень змінних, які ще не були визначені, з'явиться відповідний рядок. Це було продемонстровано у попередніх прикладах (див. рис. 1.41, 1.42).

Виведення повідомлення і пауза у командних файлах

Для виведення повідомлення на екран користувача і очікування натискання [Enter] після прочитання повідомлення слід використовувати команду PAUSE.

Приклад 1.34. Створити командний файл, який виводить на екран вміст таблиць EMP та DEPT. Результат з другої таблиці виводиться тільки після натискання клавіші [Enter]. Новому файлу дати ім'я PAUSEDEMO.

```
PROMPT  TABLE EMP
SELECT * FROM EMP;
PAUSE Press ENTER for continue
PROMPT  TABLE DEPT
SELECT * FROM DEPT;
```

Результат виконання наведено на рис. 1.49.

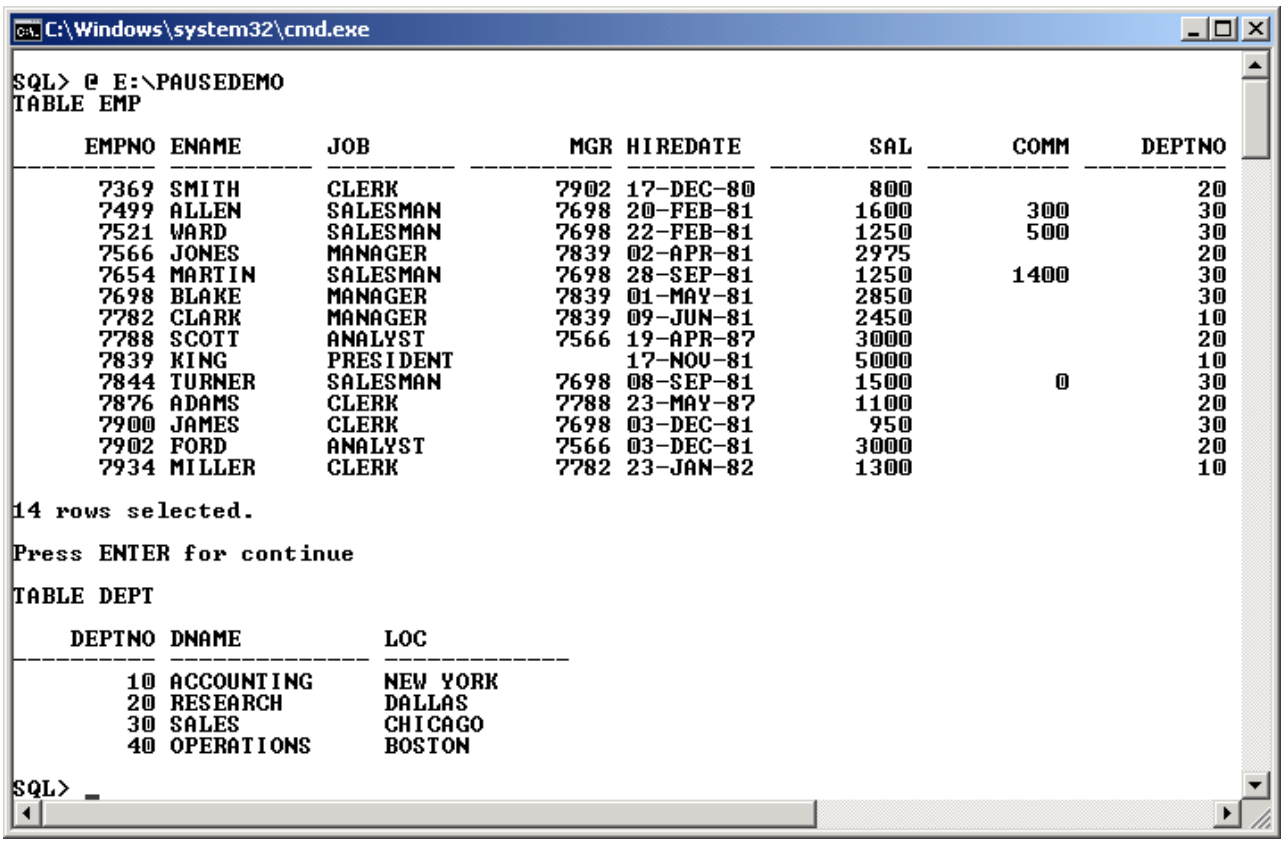


Рис. 1.49. **Виконання командного файла з паузою**

Пояснення. Між командами SQL, що виводять на екран вміст таблиць EMP та DEPT, додано рядок з командою PAUSE, яка призупиняє виконання командного файла, щоб користувач міг ознайомитися з рядками таблиці EMP і після натискання клавіші [Enter] виконання буде продовжено.

1.4.8. Передача параметрів у команді START

Можна обійти підказку для введення відповідних значень для змінних підстановки за допомогою передачі значень через параметри командного файла під час запуску його на виконання командою START. Для цього у командному файлі замість підстановочної змінної потрібно помістити амперсанд, за яким іде число. Щоразу, коли буде виконуватися командний файл, START заміщає кожне входження &1 у файлі на перше значення (що називається аргументом) після START <ім'я файла>, потім заміщає кожне &2 на другий аргумент і т. д.

У прикладі 1.30, що наводився для ілюстрації вкладеності командних файлів, основний файл містив три рядки:

START E:\TABLE_EMP
START E:\TABLE_DEPT
START E:\SALGRADE

Кожен з цих файлів також містив два рядки з командами DESC та SELECT. Слід зазначити, що усі ці три файли були, по суті, однакові, а відрізнялися тільки іменами таблиць бази даних. А що потрібно зробити, якщо у базі даних не три – чотири таблиці, а сто – двісті? Невже потрібно писати для кожної таблиці свій файл? На щастя відповідь на це питання – ні! І допоможе у цьому саме використання параметрів у командних файлах.

Приклад 1.35. Створити командний файл на ім'я TABLE_INFO.SQL, який виводить на екран інформацію про структуру та вміст конкретної таблиці БД, ім'я якої задається як параметр до цього файлу.

Створюваний файл буде містити такі рядки:

DESC &1
SELECT * FROM &1;

Якщо запустити такий файл на виконання командою **START TABLE_INFO DEPT,** то першим параметром у рядку запуску буде текст DEPT і саме він підставиться замість &1 у командному файлі. У результаті команди будуть перетворені на такі:

DESC DEPT
SELECT * FROM DEPT;

Приклад 1.36. Виконати модифікацію командного файла TABLES_INFO.SQL (див. приклад 1.30) таким чином, щоб замість трьох різних файлів з нього викликався один TABLE_INFO, але з параметром – назвою таблиці. Змінений файл буде виглядати таким чином:

START E:\TABLE_INFO &1
START E:\TABLE_INFO &2
START E:\TABLE_INFO &3

Запустити його на виконання, передаючи три параметри з іменами таблиць, а саме: EMP, DEPT, SALGRADE. Результат виконання наведений на рис. 1.50.

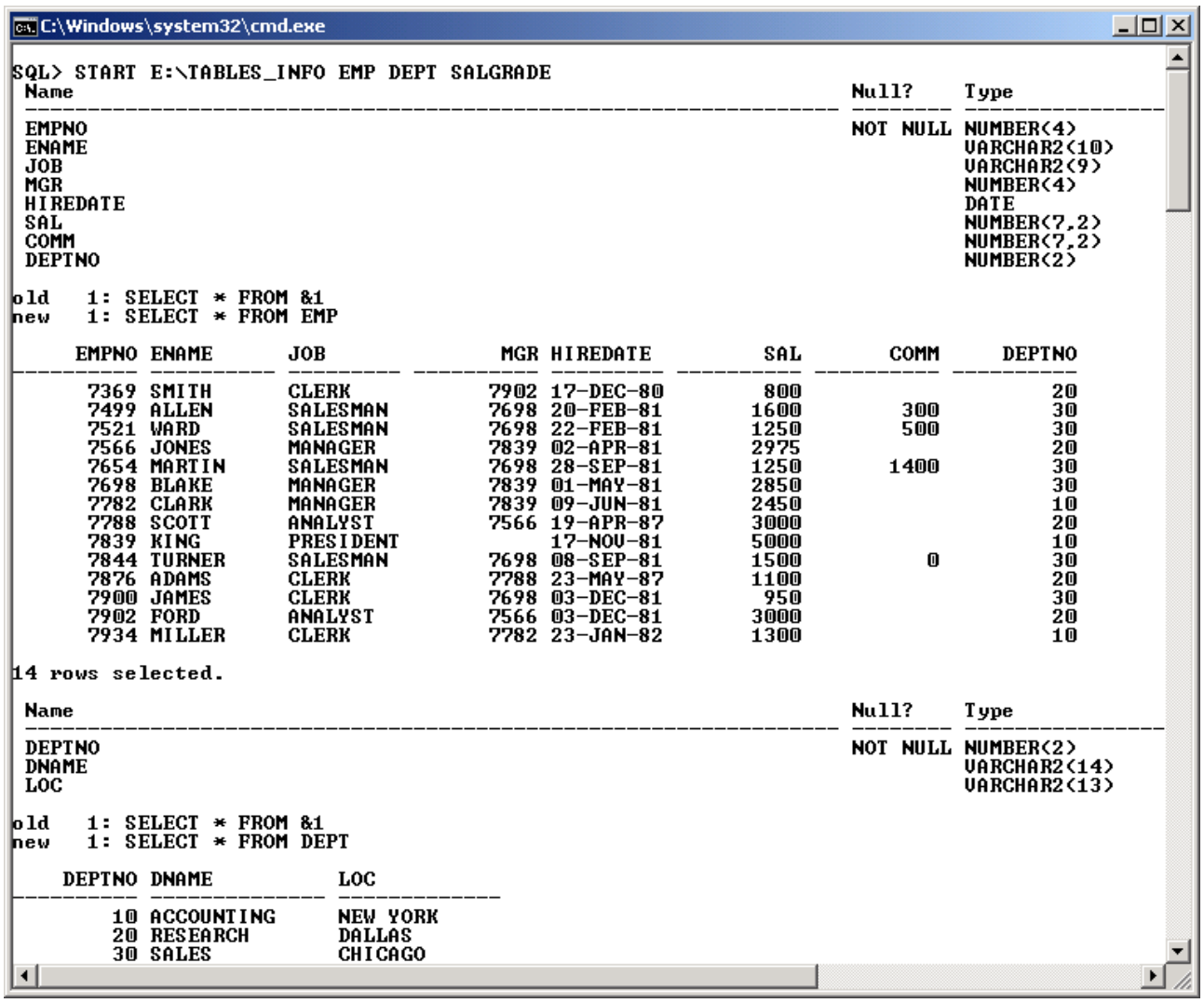


Рис. 1.50. **Виконання командного файла TABLES_INFO з параметрами**

Якщо виникне необхідність в отриманні інформації про інші таблиці, можна буде використати ту саму команду, замінивши тільки параметри, тобто задавши інші імена таблиць.

Приклад 1.37. Створити командний файл на ім'я MYFILE, який відбирає співробітників, що працюють на певній посаді та отримують заробітну плату, більшу за означену. Значення посади та заробітної плати передавати у командний файл як параметри. Вміст командного файлу такий:

```
SELECT * FROM EMP
WHERE JOB = '&1'
AND SAL> & 2;
```

Приклад його виконання наведено на рис. 1.51.

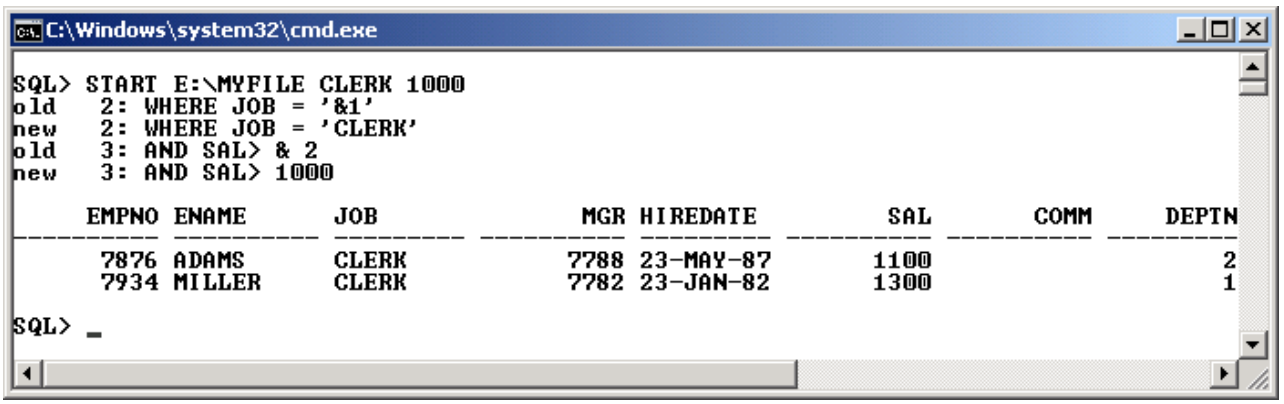


Рис. 1.51. **Виконання командного файла MYFILE.SQL з параметрами**

Пояснення. Файл запускається на виконання командою **START** з параметрами **CLERK** та **1000.** Відповідно до цього SQL*Plus замінить у командному файлі MYFILE &1 на CLERK, а &2 – на 1000, що призведе до виконання такого запиту:

```
SELECT * FROM EMP
WHERE JOB = 'CLERK'
AND SAL> 1000;
```

Запитання і завдання

1. Яке призначення мають командні файли в середовищі утиліти SQL*Plus?
2. Опишіть механізм функціонування параметрів у командних файлах.

1.5. Копіювання даних з однієї БД в іншу

Команда COPY дозволяє виконувати копіювання даних між БД і між таблицями однієї БД. За допомогою команди COPY можна копіювати дані між БД такими способами:

- копіювати дані з віддаленої БД у локальну БД;
- копіювати (за замовчуванням) дані з локальної БД у віддалену БД;
- копіювати дані з віддаленої БД в іншу віддалену БД.

Команда COPY підтримує типи даних: CHAR, DATE, LONG, NUMBER та VARCHAR2.

Синтаксис команди COPY такий:

COPY {FROM database | TO database | FROM database TO database}
{APPEND|CREATE|INSERT|REPLACE} destination_table
[(column, column, column,...)] USING query,

де FROM database – БД, звідки буде здійснюватись копіювання;

TO database – БД, у яку буде здійснюватись копіювання;

{APPEND|CREATE|INSERT|REPLACE} – дії, які необхідно виконати з таблицею призначення (цільовою таблицею);

destination_table – імя таблиці призначення;

[(column, column, column,...)] – імена стовпців у новій таблиці призначення (необов'язкові);

query – запит SELECT, за допомогою якого здійснюється копіювання даних.

Значення database має синтаксис подібний до команди CONNECT, а саме: **username[/password]@connect_identifier,** тобто логін користувача, його пароль і специфікація БД.

Керування зверненням до таблиці призначення

Керувати зверненням до таблиці призначення можна за допомогою фраз, що починаються ключовими словами – REPLACE, CREATE, INSERT або APPEND.

Фраза ***REPLACE*** вказує таблицю, яку необхідно створити у цільовій базі даних, і задає такі дії:

- якщо цільова таблиця вже існує, COPY видаляє існуючу таблицю та заміщає її таблицею, що містить дані, які копіюються;
- якщо цільова таблиця не існує, COPY створює її, використовуючи дані, які копіюються.

Фразу ***CREATE*** використовують, щоб уникнути записування даних у вже існуючу таблицю. CREATE вказує такі дії:

- якщо цільова таблиця вже існує, COPY повідомляє про помилку і зупиняє виконання команди;
- якщо цільова таблиця не існує, COPY створює таблицю, використовуючи дані, які копіюються.

Фраза ***INSERT*** використовується для додавання даних до вже існуючої таблиці. INSERT визначає такі дії:

- якщо цільова таблиця вже існує, COPY вставляє копійовані дані в цільову таблицю;

• якщо цільова таблиця не існує, COPY повідомляє про помилку та зупиняє виконання команди.

Фраза ***APPEND*** використовується в разі необхідності вставити дані в існуючу таблицю або створити нову таблицю, якщо цільова таблиця не існує. APPEND визначає такі дії:

• якщо цільова таблиця вже існує, COPY вставляє дані у цільову таблицю;

• якщо цільова таблиця не існує, COPY створює таблицю і потім заносить у неї дані, які копіюються.

Приклад 1.38. Здійснити копіювання даних з однієї віддаленої БД, яка розміщується на сервері з адресою 192.168.0.155, в іншу віддалену за адресою 192.168.0.166. У процесі копіювання реалізувати різні операції – CREATE, REPLACE, INSERT (рис. 1.52 – 1.54).

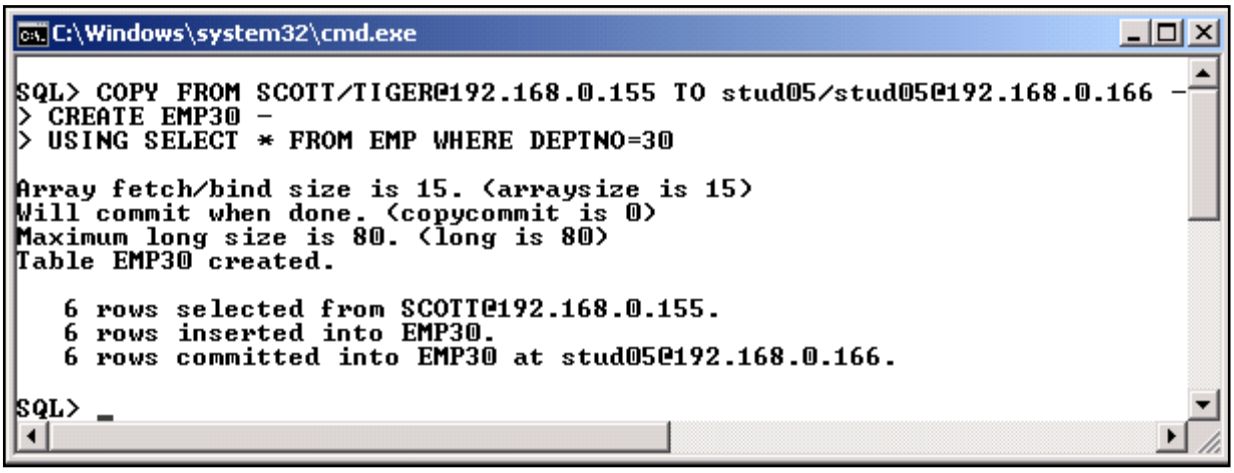


Рис. 1.52. **Приклад реалізації операції COPY** (Create)

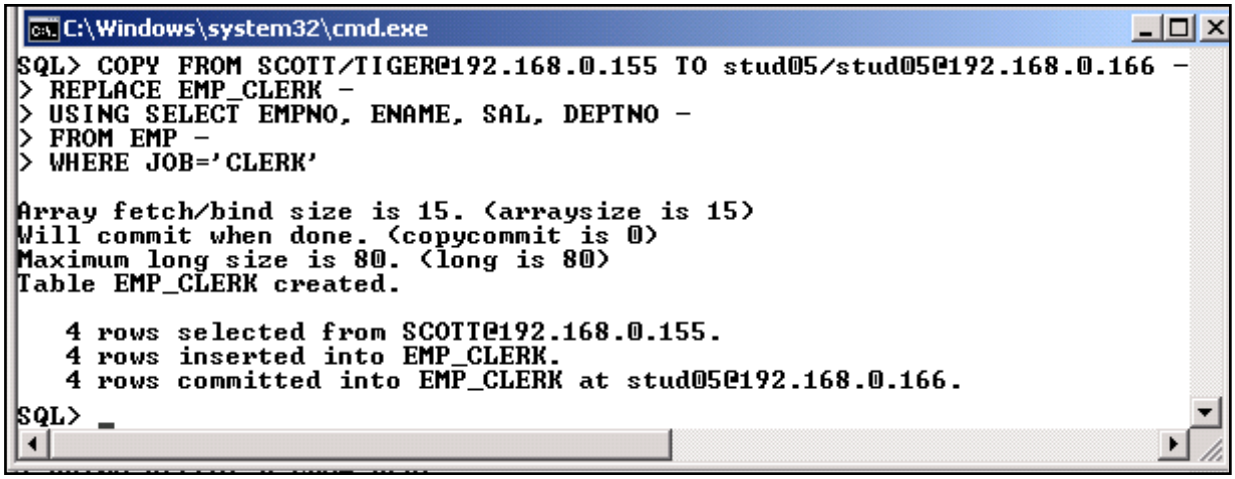


Рис. 1.53. **Приклад реалізації операції COPY** (Replace)

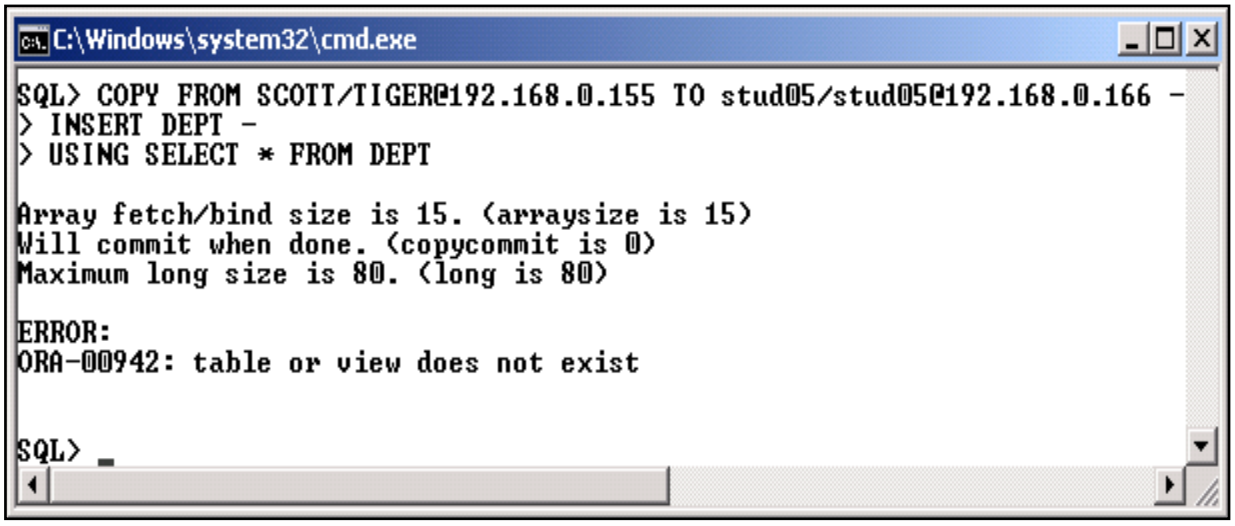


Рис. 1.54. **Приклад реалізації операції COPY (Insert) з помилкою**

Пояснення. У першому прикладі (з опцією CREATE) створюється нова таблиця на ім'я EMP30, у яку заносяться шість рядків з відомостями про співробітників відділу 30.

У другому прикладі (з опцією REPLACE) створюється нова таблиця на ім'я EMP_CLERK, у яку заносяться чотири рядки з відомостями про співробітників, працюючих на посаді CLERK. Таблиця нова, оскільки до цього її не існувало у цільовій БД.

Нарешті, у третьому прикладі (з опцією INSERT) робиться спроба додати рядки до таблиці DEPT цільової БД. Однак оскільки такої таблиці не існує, генерується повідомлення про помилку.

Запитання і завдання

1. Опишіть призначення елементів команди COPY.
2. Які можливості керування зверненням до таблиці призначення має команда COPY?

1.6. Форматування результатів запитів у SQL*Plus

Результати запитів в SQL*Plus можна додатково форматувати. Утиліта SQL*Plus дозволяє керувати форматом стовпців, кількістю рядків на сторінці та додатковими порожніми рядками, заголовками сторінок тощо. Для цього використовують спеціальні команди форматування, які застосовують, якщо стандартний формат подання стовпців не підходить.

Стандартний формат стовпців

Стандартний формат виведення стовпців визначається такими правилами.

• Ширина числових стовпців відповідає максимуму з ширини заголовка стовпця, ширини, заданої за допомогою опції COLUMN FORMAT плюс один символ для знака, та ширини, заданої командою SET NUMWIDTH (за замовчуванням, десять символів). Якщо кількість значущих цифр у числі більша, ніж допускається шириною числового стовпця, утиліта SQL*Plus округлює число.

• Для стовпців інших типів ширина стовпця відповідає його ширині в базі даних. Усі ці стовпці, за замовчуванням, вирівнюються вліво. Для стовпців типу DATE формат визначається відповідними NLS-параметрами. Якщо вони не задані, передбачається формат A9.

•

1.6.1. Команда COLUMN

Команда COLUMN використовується для керування форматом виведення стовпця та має такий синтаксис:

COL [UMN] [<посилання на стовпець> {<опція>}]

<посилання на стовпець> :: = <ім'я стовпця> | <псевдонім> | <вираз>

Опції команди **COLUMN** наведені у табл. 1.4.

Таблиця 1.4.

Основні опції команди COLUMN

Опція	Призначення
ALIAS	Надає стовпцю вказаний псевдонім. З цього псевдоніму на стовпець можна посилатися в подальшому у командах BREAK, COMPUTE і COLUMN
CLEAR	Скидає атрибути зазначеного стовпця у стандартні значення
ENTMAP	Дозволяє включати і відключати форматування значень стовпця для HTML-звіту. Якщо ця опція для стовпця включена, в значеннях будуть замінюватися символи, що мають керуюче значення в HTML (<,>, & тощо)
FOLD_AFTER	Вставляє перехідна новий рядок після заголовка стовпця і кожного значення в даному стовпці. Перехід на новий рядок не вставляється, якщо зазначений стовпець – останній у списку вибору

Продовження табл. 1.4

Опція	Призначення
FOLD_BEFORE	Вставляє перехід на новий рядок перед заголовком стовпця та кожним значенням у даному стовпці. Перехід на новий рядок не вставляється, якщо зазначений стовпець перший у списку вибору
FORMAT	Задає формат виведення значень стовпця
HEADING	Задає заголовок стовпця. Якщо ця опція не використовується, то заголовком слугують початкові символи (до ширини стовпця) імені стовпця чи виразу, що формує значення стовпця. Якщо у тексті є пробіли або символи пунктуації, його необхідно брати в одинарні або подвійні лапки. Замість кожного входження символу HEADSEP (за замовчуванням – "|") в тексті заголовка вставляється символ переходу на новий рядок
JUSTIFY	Задає вирівнювання стовпця. За замовчуванням стовпці типу NUMBER вирівнюються вправо, усі інші – вліво
LIKE	Копіює особливості форматування вказаного стовпця, що не задані для поточного стовпця явно
NEWLINE	Вставляє перехід на новий рядок перед виведенням значення стовпця аналогічно до FOLD_BEFORE
NEW_VALUE	Задає змінну, у якій буде зберігатися значення стовпця. Цю зміну можна використовувати в команді TTITLE як елемент верхнього колонтитула. Сам стовпець необхідно при цьому зазначити в команді BREAK з дією SKIP PAGE
NOPRINT PRINT	Керує виведенням стовпця. Опція NOPRINT відключає виведення стовпця на екран і в звіт. Опція PRINT відновлює виведення стовпця
NULL	Задає текст, що видається утилітою SQL*Plus замість порожніх значень у стовпці. За замовчуванням використовується пробіл
OLD_VALUE	Задає змінну, у якій буде зберігатися значення стовпця. Цю зміну можна використовувати в команді BTITLE як елемент нижнього колонтитула. Сам стовпець необхідно при цьому зазначити у команді BREAK з дією SKIP PAGE
ON OFF	Керує застосуванням особливостей (атрибутів) форматування стовпця. Значення OFF відключає застосування особливостей форматування не скасовуючи їх. Значення ON знову включає застосування заданих особливостей форматування

Закінчення табл. 1.4

Опція	Призначення
WRAPPED WORD_WRAPPED TRUNCATED	Задає правила роботи зі значеннями, що перевищують ширину стовпця. Допускається перенесення на наступний рядок по межі стовпця, по межі слова або усікання по межі стовпця

Зазвичай найбільш уживаним є використання опції FORMAT команди COLUMN, адже саме вона дозволяє задавати форму виведення тих чи інших значень.

Приклад 1.39. Використовуючи команду COLUMN, змінити формат відображення стовпця SAL таким чином, щоб значення виводилися з двома знаками після десяткової крапки і відокремлювались комою три знаки у цілій частині. Також змінити назву стовпця для поля ENAME на EMPLOYEE SURNAME, розташувавши нову назву на двох рядках.

Перевірити результат командою SELECT. Очистити формат стовпців та виконати команду SELECT повторно (див. рис. 1.53).

```
COL SAL FORMAT 9,999.99
COL ENAME HEADING "EMPLOYEE |SURNAME"
SELECT ENAME, SAL FROM EMP;
COL SAL CLEAR
```

Пояснення. У першому рядку задається формат виведення стовпців, які мають назву SAL, з двома знаками після крапки для відображення десяткових значень і відокремлення перших трьох розрядів комою.

Другий рядок змінює назву, яка буде з'являтися над стовпцями бази даних, що мають назву ENAME. Нова назва тепер буде розташовуватись на двох рядках і містити слова EMPLOYEE та SURNAME.

Третій рядок виконує запит до БД, у якому будуть застосовані результати попередніх двох команд.

Четвертий рядок "очищує" встановлений формат виведення поля SAL і повертає його до стану за замовчуванням.

Результати виконання цього прикладу наведені на рис. 1.55.

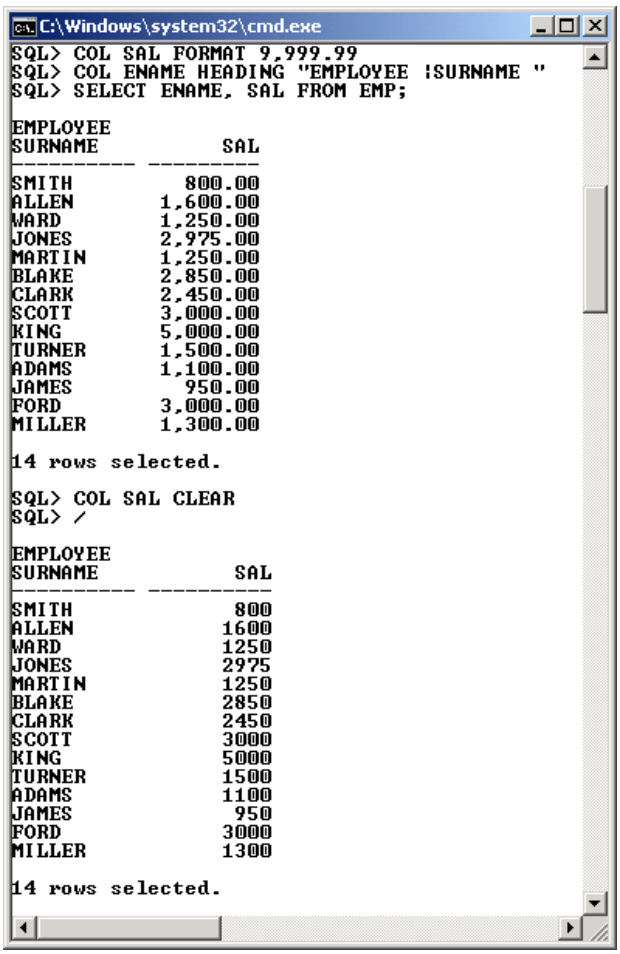


Рис. 1.55. **Результати запиту SELECT з форматуванням стовпців**

Команда COLUMN з єдиним параметром – посиланням на стовпець, видає усі атрибути форматування вказаного стовпця.

Команда COLUMN без параметрів видає всі атрибути форматування для усіх стовпців, для яких вони явно встановлювалися.

Приклад 1.40. Використовуючи команду COLUMN без параметрів, отримати відомості про поточні формати для усіх стовпців (рис. 1.56).

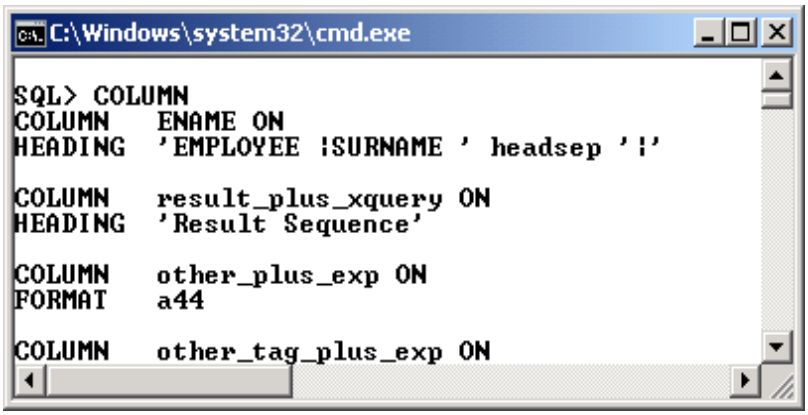


Рис. 1.56. **Отримання інформації про форматування стовпців**

Приклад 1.41. Використовуючи команду COLUMN, виконати такі дії:

• задати аліасне ім'я (псевдонім) SALARY для поля SAL;

• встановити формат виведення поля SALARY з чотирма знаками у цілій частині числа та двома – після крапки;

• встановити новий заголовок для поля ENAME як SURNAME та задати ширину його виведення у п'ять стовпців.

```
COLUMN SAL  ALIAS SALARY
COLUMN SALARY FORMAT $9999.99
COLUMN ENAME HEADING "SURNAME" FORMAT A5
SELECT ENAME, HIREDATE, SAL FROM EMP;
```

Результат виконання наведено на рис. 1.57.

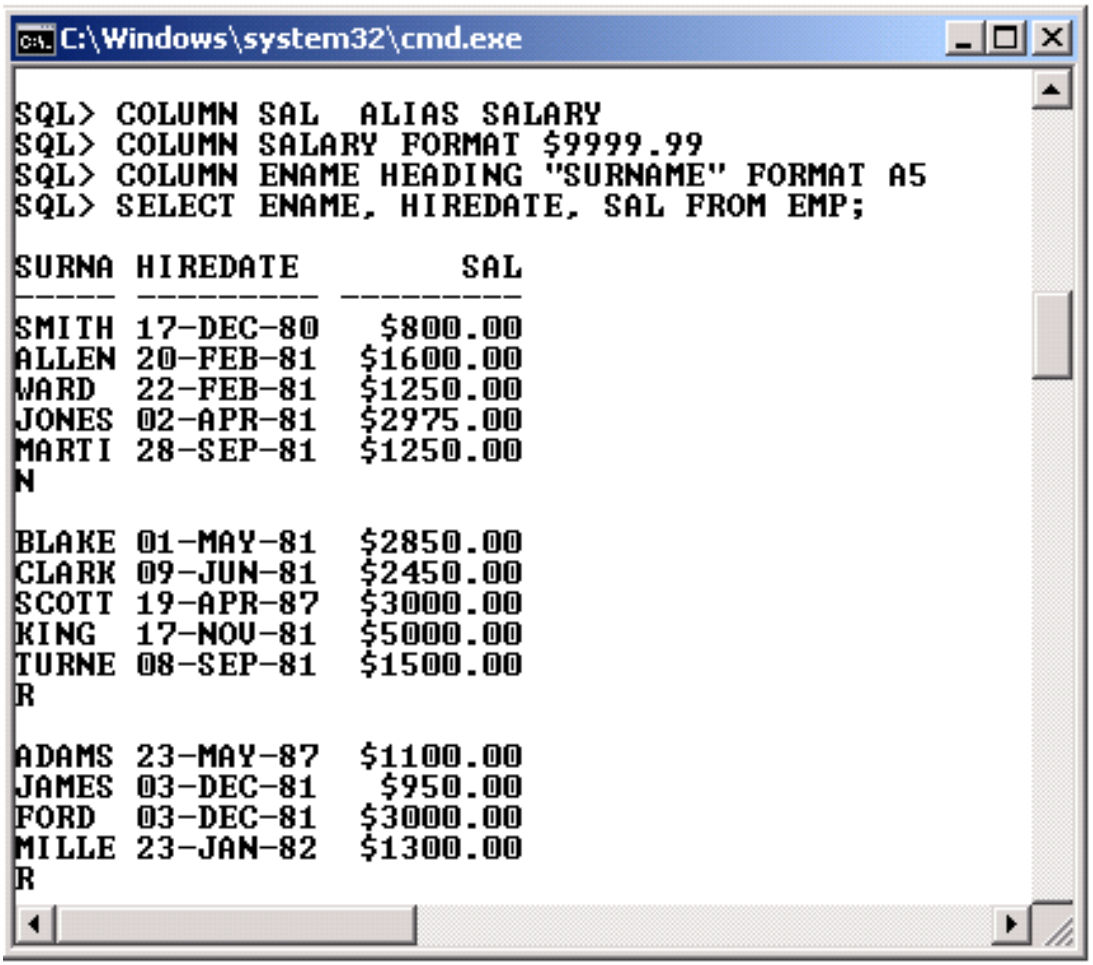


Рис. 1.57. **Результат форматування трьох стовпців різного типу**

Основні елементи, що можуть використовуватися в опції FORMAT команди COLUMN наведені у табл. 1.5.

Таблиця 1.5.

Основні елементи опції FORMAT команди COLUMN

Елемент	Приклад	Опис
A <ширина>	A20	Цей елемент формату дозволяє змінити стандартну ширину стовпця рядкового типу і типу DATE. Якщо значення стовпця не поміщається у задану <ширину>, воно буде скорочуватися або переноситися, залежно від відповідних установок (SET TRUNC або SET WRAP)
9	9999	Визначає значущу цифру у форматі виведення числової величини. Замість початкових нулів видаються пробіли. Нульове значення позначається цифрою 0
0	0999	Виводить початковий нуль
$	$ 999	Виводить перед числовим значенням символ долара
L	9999L	Виводить символ локальної грошової одиниці у даній позиції
. (крапка)	9999.99	Виводить роздільник цілої та дробової частини (десяткову кому) у даній позиці
, (кома)	9,999	Виводить кому (роздільник розрядів) у даній позиції.
DATE	DATE	Виводить числове значення (яке є датою у юліанському форматі) як дату в форматі MM/DD/YY
EEEE	9.999EEEE	Виводить значення в експоненційному форматі (обов'язково вказувати рівно чотири E)

Розширений опис форматів, які можуть використовуватися для виведення числових значень, наведені у табл. 1.6.

Таблиця 1.6.

Числові формати.

Елемент	Приклад	Опис
9	9999	Кількість цифр, що визначають ширину виводу значення
0	0999	Виведення провідних нулів
$	$ 9999	Префіксне значення зі знаком долара
B	B9999	Виведення нульових значень як пробілів, а не як нулів
MI	9999MI	Виведення знака "–" після від'ємних чисел
PR	9999PR	Виведення від'ємних чисел у <кутових дужках>

Закінчення табл. 1.6

Елемент	Приклад	Опис
, (кома)	9,999	Виведення коми у зазначеній позиції
. (крапка)	99.99	Налаштування десяткової крапки на зазначену позицію
V	999V99	Множення на 10 у N-му ступені, де N – кількість дев'яток після 'V'
E	9 & 999EEEE	Виведення у експоненційній формі (формат повинен містити рівно чотири літери 'E')

Для зміни форми подання значень стовпців, які мають тип DATE, слід використовувати команду

ALTER SESSION SET NLS_DATE_FORMAT = 'формат';

наприклад, так:

ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY';

або використати у команді SELECT функцію TO_CHAR (див. п. 2.17.4) і вибрати відповідну форматну маску [44].

Приклад 1.42. Проілюструвати різні формати виведення поля HIREDATE за допомогою функції TO_CHAR (рис. 1.58).

```
SELECT ENAME, HIREDATE,
       TO_CHAR(HIREDATE,'DD/MM/YY') AS HDATE1,
       TO_CHAR(HIREDATE,'DD-MONTH-YYYY') AS HDATE2
FROM EMP;
```

У другому стовпцю дата виводиться у форматі за замовчуванням, а у третьому та четвертому згідно з форматними масками перетворення дати для функції TO_CHAR.

Слід зазначити також, що формат виведення першого стовпця відповідає попередньо встановленому формату (див. приклад 1.41).

Якщо ж формат подання дати за замовчуванням буде змінений командою ***ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY',*** то у цьому випадку до кінця сеансу буде діяти новий формат виведення дати за замовчуванням.

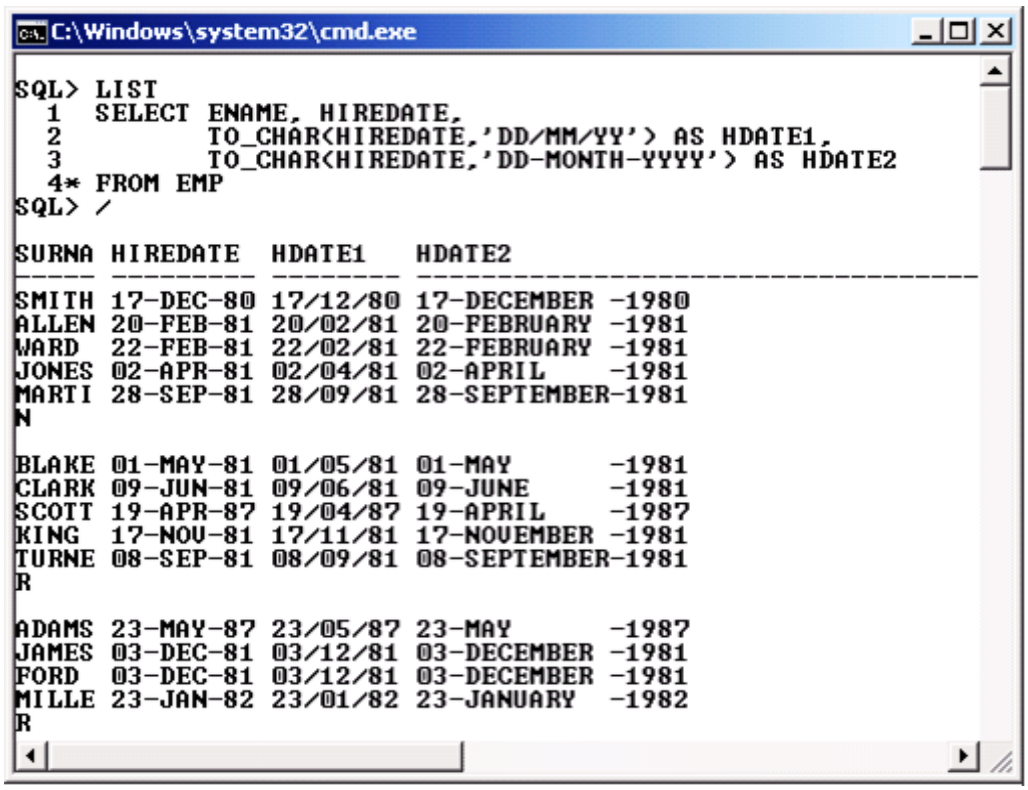


Рис. 1.58. **Форматування HIREDATE різними форматними масками**

Приклад 1.43. Змінити формат відображення дат у поточному сеансі за допомогою команди ALTER SESSION SET NLS_DATE_FORMAT та перевірити результат запитом SELECT (рис. 1.59).

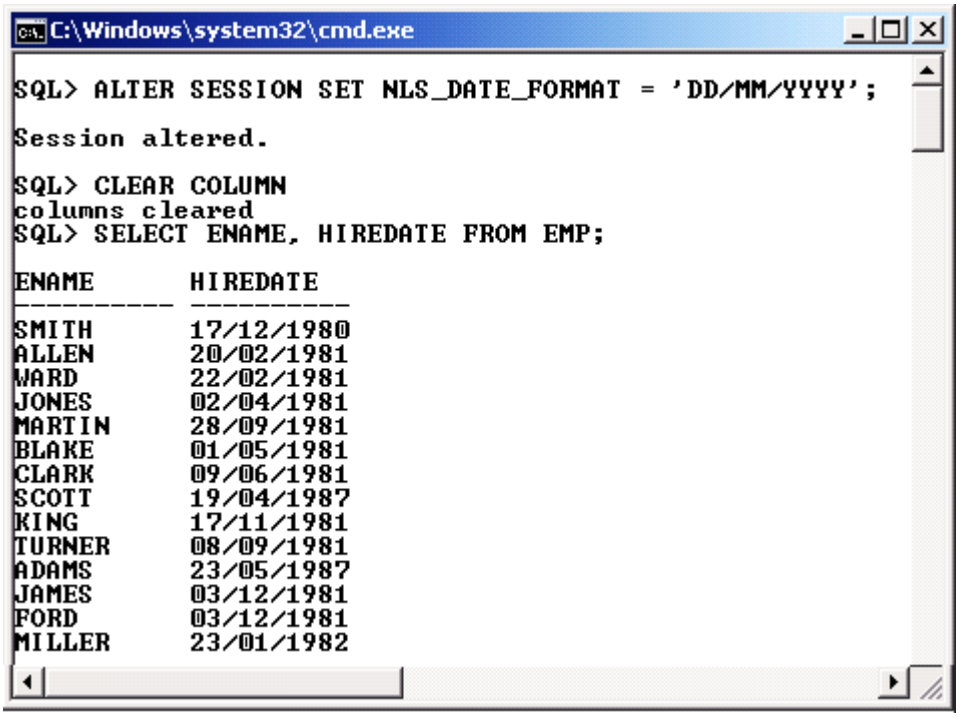


Рис. 1.59. **Зміна формату виведення дати для поточного сеансу**

1.6.2. Команда BREAK

Команда BREAK дозволяє розбити рядки результату виконання запиту SELECT на групи за значеннями стовпця, розділяючи групи порожніми рядками, а також керувати виведенням дубльованих значень і значень, підрахованих за допомогою команди COMPUTE. Вона має наступний синтаксис:

<**команда BREAK**> :: = BRE [AK] {ON <елемент звіту> {<дія>}}

<**елемент звіту**> :: = <стовпець> | <вираз> | ROW | REPORT

<**дія**> :: = SKI [P] <кількість рядків> [<дублікати>]

| SKI [P] PAGE [<дублікати>]

<**дублікати**> :: = NODUP [LICATES] | DUP [LICATES]

Опис параметрів команди BREAK наведений у табл. 1.7.

Команда BREAK без параметрів видає значення своїх поточних параметрів (параметрів розриву). Кожен наступний виклик BREAK з параметрами скасовує попередній. Для відміни параметрів розриву використовується команда CLEAR BREAKS.

Таблиця 1.7.

Основні варіанти використання команди BREAK

Конструкція	Опис
ON <стовпець> {<дія>}	Задає дії, які виконуються у разі зміни значення зазначеного стовпця. Стовпець задається за іменем або псевдонімом, без уточнення імені об'єкта. Якщо дія не вказана, виведення повторюваних значень не відбувається та відзначається місце, де виконується обчислення, що задане у відповідній команді COMPUTE
ON <вираз> {<дія>}	Задає дії, які виконуються при зміні значення виразу зі списку вибору. Вираз треба задавати буквально, як у списку вибору. Якщо дія не вказана, виведення повторюваних значень не відбувається і відзначається місце, де виконується обчислення, задане у відповідній команді COMPUTE
ON ROW <дія> {<дія>}	Задає дії, які виконуються при виведенні кожного рядка. Необхідно задати хоча б одну дію

Закінчення табл. 1.7

Конструкція	Опис
ON REPORT {<дія>}	Відзначає місце у звіті, де SQL*Plus виконає обчислення, задане у відповідній команді COMPUTE. Дозволяє видавати сумарні значення
SKIP <кількість рядків>	Пропускає вказану кількість рядків (вставляє стільки порожніх рядків) перед рядком, у якому відбувається розрив
SKIP PAGE	Пропускає стільки рядків, скільки задано в якості розміру сторінки (задається за допомогою SET PAGESIZE). Після останнього рядка даних рядки не пропускаються
NODUPLICATES	Видає пробіли замість значення у стовпці, якщо воно збігається зі значенням у попередньому рядку
DUPLICATES	Видає значення стовпця в кожному рядку, незалежно від дублювання

Конструкцію ON <стовпець> можна задавати в одній команді BREAK кілька разів. При цьому стовпці перевіряються в порядку запису. Дії ж виконуються у зворотному порядку від самого внутрішнього розриву. Якщо під час введення необхідно перенести опції на наступний рядок, треба задати дефіс (-) наприкінці першого рядка.

Зазвичай команда BREAK використовується з операторами SELECT, що містять конструкцію ORDER BY.

Приклад 1.44. Використовуючи команду BREAK встановити пропуск рядка у результаті команди SELECT при зміні значення заробітної плати та номера відділу. Не виводити значення номерів відділів та заробітної плати, якщо вони повторюють попереднє значення (рис. 1.60).

```
BREAK ON DEPTNO SKIP ON SAL SKIP 1
BREAK ON DEPTNO NODUP ON SAL SKIP 1 NODUP
SELECT DEPTNO, ENAME, SAL
FROM EMP
ORDER BY DEPTNO, SAL DESC;
```

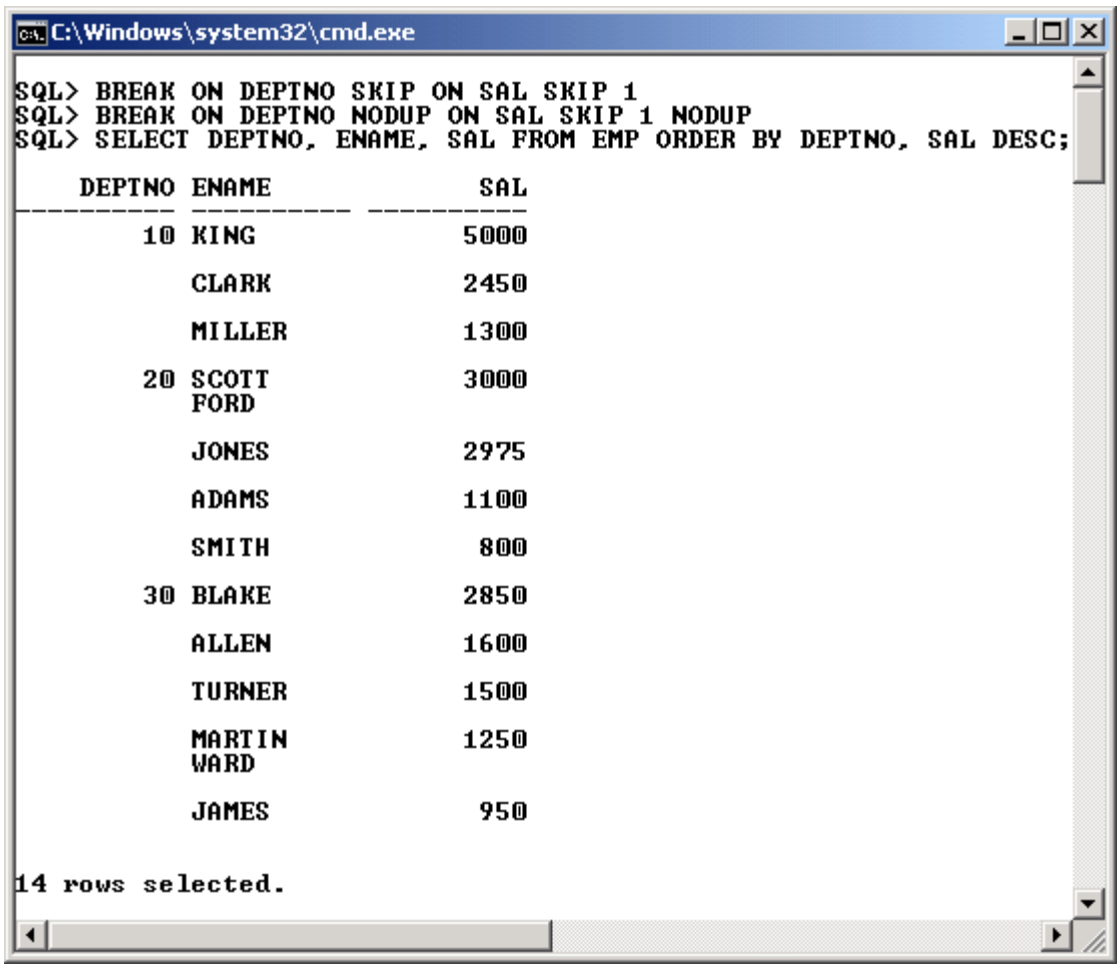


Рис. 1.60. **Ілюстрація використання команди BREAK**

Як видно з отриманого результату, якщо заробітна плата співробітників співпадає (SCOTT, FORD – 3000 і MARTIN,WARD – 1250), пропуск рядка не здійснюється.

Виконання команди BREAK після попередніх дій без параметрів сповіщає про поточні установки.

Приклад 1.45. Отримати поточні параметри команди BREAK (рис. 1.61).

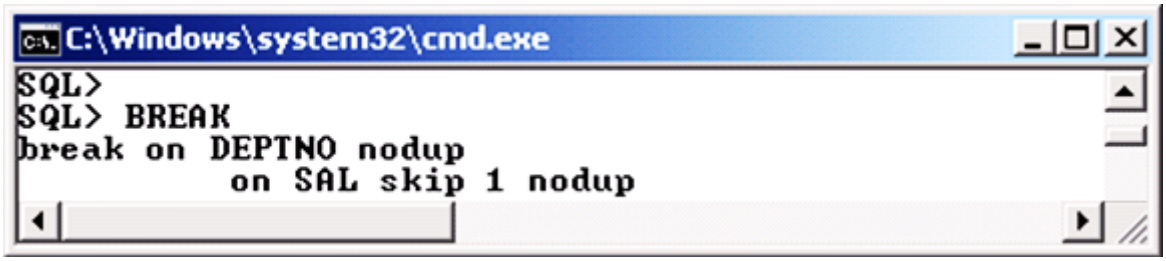


Рис. 1.61. **Відомості про поточні установки команди BREAK**

1.6.3. Команда COMPUTE

Команда COMPUTE дозволяє обчислювати та виводити підсумкові значення. У разі виклику без параметрів виводить усі задані обчислення.

Команда COMPUTE має такий синтаксис:

<**команда COMPUTE**> :: =
COMP [UTE] {<функція> [LAB [EL] <текст>]}
OF <посилання на стовпець> {<посилання на стовпець>}
ON <посилання на місце> {<посилання на місце>}
<**посилання на місце**> :: = <посилання на стовпець> | REPORT | ROW

Функції, які можна використовувати у команді COMPUTE, наведені у табл. 1.8, а призначення основних конструкції LABEL, ON, OFF – у табл. 1.9.

Для видалення усіх визначень **COMPUTE** використовується команда **CLEAR COMPUTES**.

Таблиця 1.8.

Функції у команді COMPUTE

Функція	Призначення	Допустимі типи даних
AVG	Середнє серед непорожніх значень	Числові
COU [NT]	Кількість непорожніх значень	Усі
MIN [IMUM]	Мінімальне значення	Числові та рядкові
MAX [IMUM]	Максимальне значення	Числові та рядкові
NUM [BER]	Кількість рядків	Усі
SUM	Сума непорожніх значень	Числові
STD	Середньоквадратичне відхилення непорожніх значень	Числові
VAR [IANCE]	Дисперсія непорожніх значень	Числові

Таблиця 1.9.

Основні конструкції команди COMPUTE

Конструкція	Опис
LABEL <текст>	Задає позначку обчислюваного значення. Якщо ця конструкція не вказана, видається повне ім'я функції. Максимальна довжина тексту – 500 символів. Якщо текст містить пробіли і символи пунктуації, його треба брати в одинарні лапки. Позначка вирівнюється вліво та буде скорочуватися до меншого зі значень ширини стовпчика або довжини рядка.

Закінчення табл. 1.9

Конструкція	Опис
	Позначка для обчислюваного значення видається у стовпці, за яким виконується BREAK. Щоб позначка не видавалась, необхідно задати опцію NOPRINT у команді COLUMN для цього стовпця. Якщо обчислення виконуються за ON ROW або ON REPORT, значення, яке обчислюється, видається у першому стовпці та позначка не видається. Щоб позначка була видана, необхідно включити у список вибору першим фіктивний стовпець
OF <посилання на стовпець>	Задає стовпчики чи вирази, функція від яких обчислюється. У конструкції OF можна послатися на вираз у списку вибору, взявши його у подвійні лапки; ім'я або псевдонім стовпчика у лапки брати не треба
ON <посилання на місце>	Задає подію, яку утиліта SQL*Plus буде вважати місцем для обчислення. У разі посилання на стовпець його ім'я не можна уточнювати, при необхідності треба використовувати псевдоніми. При досягненні місця обчислення (коли змінюється значення у стовпці чи виразі, формується новий рядок або досягається кінець звіту) команда COMPUTE видає розраховане значення та починає обчислення спочатку. Якщо для одного і того ж стовпця задано декілька команд COMPUTE, застосовується остання з них. У конструкції ON можна послатися на вираз у списку вибору, взявши його в подвійні лапки. Ім'я або псевдонім стовпчика у лапки брати не треба. Якщо у якості події було задано ON ROW або ON REPORT, необхідно, щоб в останній команді BREAK також використовувався критерій розриву ROW або REPORT

Приклад 1.46. Використовуючи команду COMPUTE, сформувати звіт, у якому відображується середня заробітна плата працівників по кожному відділу (рис. 1.62).

```
COLUMN DEPTNO FORMAT 999999999999
SET PAGESIZE 16
BREAK ON DEPTNO SKIP 1
COMPUTE AVG LABEL 'СЕРЕДНЯ З/П:' OF SAL ON DEPTNO
SELECT DEPTNO, ENAME, SAL FROM EMP ORDER BY DEPTNO;
```

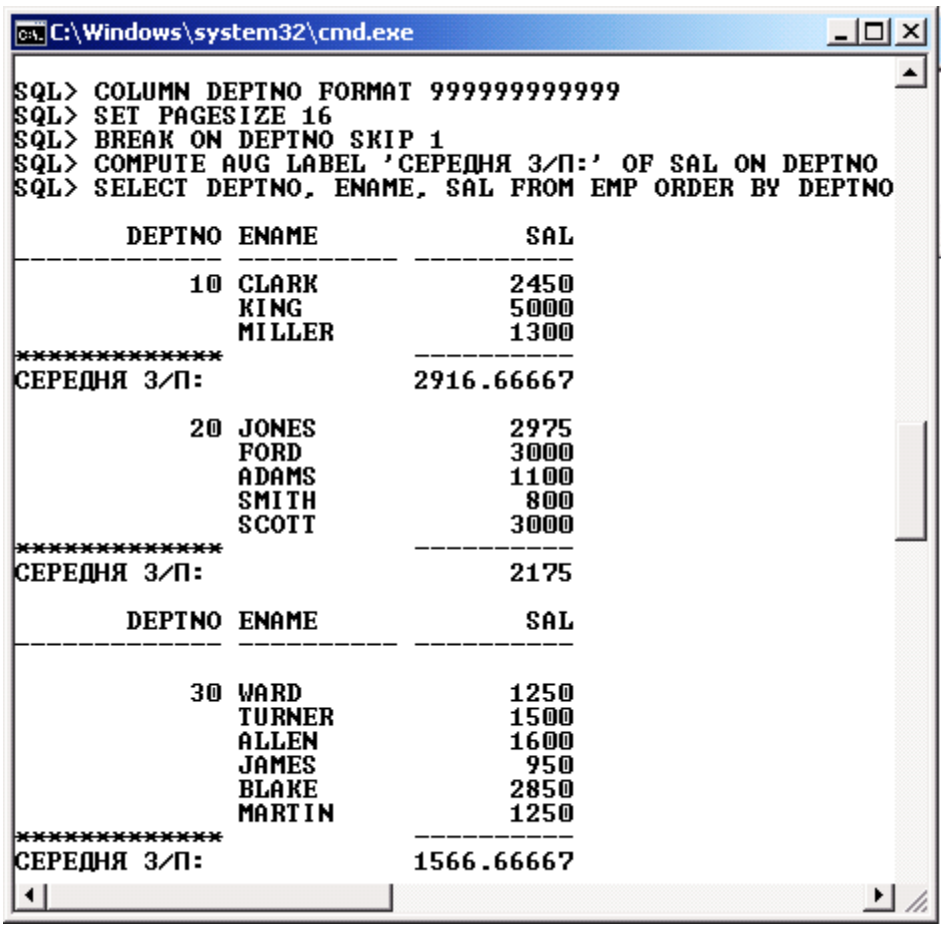


Рис. 1.62. **Обчислення середньої заробітної плати командою COMPUTE**

Пояснення. Перша команда задає формат виведення поля DEPTNO у дванадцять позицій. Це зроблено для того, щоб текст 'СЕРЕДНЯ З/П:' у цьому стовпці розташувався повністю.

Друга команда зменшує кількість рядків на сторінці до десяти (тільки для прикладу).

Третя команда встановлює виведення порожнього рядка в разі зміни значення поля DEPTNO.

Нарешті, четверта команда задає обчислення середньої заробітної плати для кожного відділу, оскільки обчислення функції AVG відбувається при зміні значення поля DEPTNO, а наступна команда SELECT задає впорядкування саме по цьому полю.

З отриманого результату також видно, що "шапка" таблиці повторюється саме через шістнадцять рядків, тобто значення параметра PAGESIZE.

Приклад 1.47. Не відміняючи попередніх налаштувань, зроблених командами COMPUTE та COLUMN, додатково підрахувати суму усіх заробітних плат і видати її у кінці звіту (рис. 1.63).

Для вирішення задачі слід виконати такі команди:

```
COMPUTE SUM LABEL 'УСЬОГО:' OF SAL ON REPORT
BREAK ON DEPTNO SKIP 1 ON REPORT
SELECT DEPTNO, ENAME, SAL FROM EMP ORDER BY DEPTNO;
```

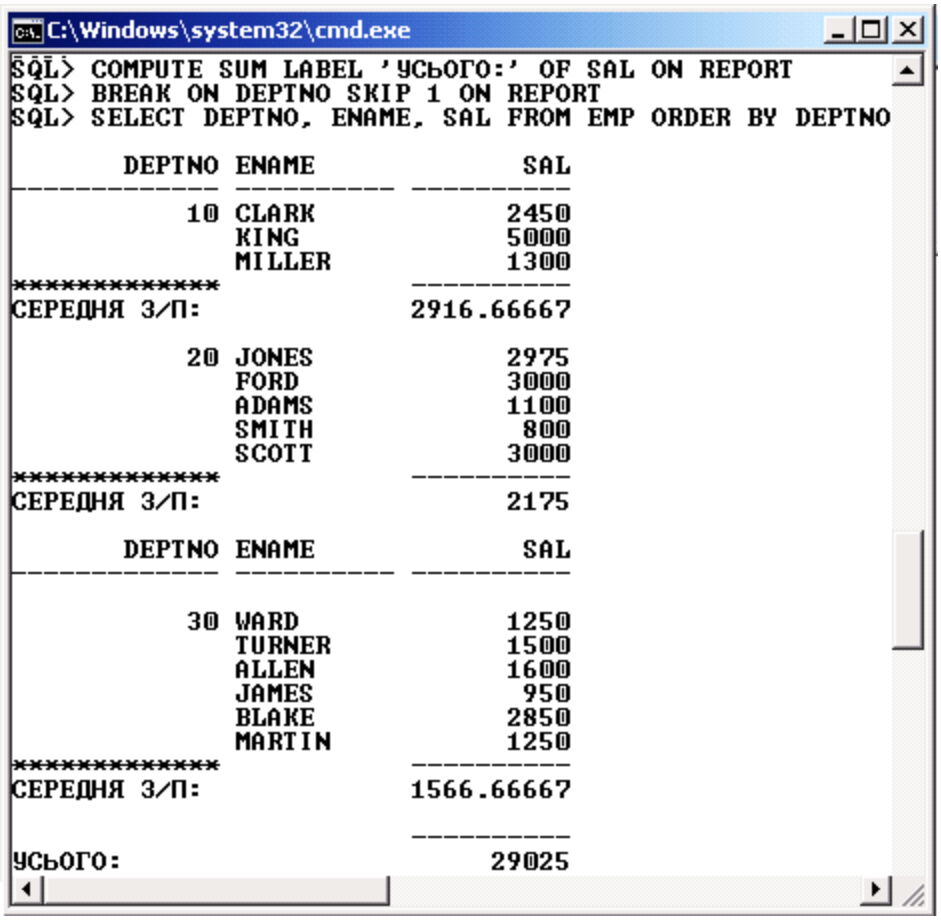


Рис. 1.63. **Додаткове обчислення загальної заробітної плати**

Пояснення. Перша команда безпосередньо вказує на необхідність підрахувати суму усіх заробітних плат і видати її у кінці звіту. Друга – задає виведення порожнього рядка в кінці звіту.

1.6.4. Команда CLEAR

Команда CLEAR дозволяє "скинути" значення низки опцій утиліти SQL*Plus, зокрема, пов'язаних з форматуванням результатів. Вона має наступний синтаксис:

<**команда CLEAR**> :: =
BRE [AKS] | BUFF [ER] | COL [UMNS] | COMP [UTES] | SCR [EEN] | SQL | TIMI [NG].

Характеристика параметрів команди наведена у табл. 1.10.

Таблиця 1.10.

Характеристика параметрів команди CLEAR

Опція	Призначення
BREAKS	"Скидає" визначення груп, встановлені командою BREAK
BUFFER	Очищує буфер SQL
COLUMNS	"Скидає" у стандартні значення атрибути подання даних усіх стовпців, що встановлені командами COLUMN
COMPUTES	Видаляє усі визначення підсумкових функцій, встановлені командою COMPUTE
SCREEN	Очищує екран SQL*Plus
SQL	Очищує буфер SQL. Аналогічно до CLEAR BUFFER, якщо тільки не використовується кілька буферів (SET BUFFER)
TIMING	Видаляє усі таймери, що створені командою TIMING

1.6.5. Команди BTITLE та TTITLE

Утиліта SQL*Plus дозволяє задавати заголовок для показу в якості верхнього (команда TTITLE) або нижнього (команда BTITLE) колонтитула на кожній сторінці звіту. Ці команди мають такий синтаксис:

<**команда BTITLE**> :: = BTI [TLE] [<специфікації друку> <текст або змінна> {<текст або змінна>}] [<вкл-викл>]

<**команда TTITLE**> :: = TTI [TLE] [<специфікації друку> <текст або змінна> {<текст або змінна>}] [<вкл-викл>]
<**специфікації друку**> :: = <специфікація друку> {<специфікація друку>}
<**специфікація друку**> :: = COL <позиція> | S [KIP] [<кількість рядків>] | TAB <кількість табуляцій> | LE [FT] | CE [NTER] | R [IGHT] | BOLD | FORMAT <формат>
<**текст або змінна**> :: = <текст> | SQL.LNO | SQL.PNO | SQL.RELEASE | SQL.SQLCODE | SQL.USER

Текст колонтитула необхідно брати в одиночні лапки, якщо він складається з декількох слів. При виведені однієї із вбудованих змінних SQL.* можна вказувати конструкцію FORMAT. Конструкція <вкл-викл>

(ON | OFF) дозволяє включати та відключати колонтитули, не впливаючи на їх визначення [41; 84].

1.6.6. Команди REPHEADER та REPFOOTER

Команди REPHEADER і REPFOOTER дозволяють задати текст, який видається на початку та у кінці кожного звіту, відповідно. При цьому використовуються ті ж самі специфікації друкування, що і в командах BTITLE і TTITLE. Команди мають такий синтаксис:

<**команда REPHEADER**> :: =
REP [HEADER] [PAGE] [<специфікації друкування>
<текст або змінна> {<текст або змінна>}] [<вкл-викл>]
<**команда REPFOOTER**> :: =
REP [FOOTER] [PAGE] [<специфікації друкування>
<текст або змінна> {<текст або змінна>}] [<вкл-викл>]

Якщо вказана опція PAGE, то відповідний текст видається на окремій сторінці. При виклику без параметрів ці команди видають поточний формат і ознаку виведення тексту на початку і в кінці звіту.

Специфікації друкування у командах BTITLE, TTITLE, REPHEADER та REPFOOTER наведені в табл. 1.11 [41; 83].

Таблиця 1.11.

Специфікації друкування у командах BTITLE, TTITLE, REPHEADER та REPFOOTER

Специфікація	Призначення
COL	Вирівнює текст по вказаній символьній позиції
SKIP	Виводить вказану кількість порожніх рядків. За замовчуванням один порожній рядок. Значення 0 означає повернення до початку рядка
TAB	Пропускає вказану кількість стовпців (позицій табуляції). Якщо вказано від'ємне значення, зміщує поточну позицію на відповідну кількість стовпців вліво
LEFT CENTER RIGHT	Задає вирівнювання поточного рядка (відповідно вліво, по центру, вправо). Усі наступні елементи колонтитула (до кінця специфікації або до наступної специфікації LEFT, CENTER, RIGHT або COL) вирівнюються разом, як група. При цьому використовується довжина рядка, задана за допомогою команди SET LINESIZE

Закінчення табл. 1.11

Специфікація	Призначення
BOLD	Виділяє дані. На терміналі для цього однакові дані друкуються в трьох послідовних рядках. При виведенні на друкування рядок зазвичай видається жирним шрифтом
FORMAT	Задає модель формату для наступних даних

Приклад 1.48. Використовуючи команди BTITLE, TTITLE, REPHEADER та REPFOOTER, сформувати звіт відділу кадрів, що містить інформацію про співробітників.

Для цього треба ввести та виконати такі команди (рис. 1.64):

```
TTITLE CENTER 'LIST OF EMPLOYEES'
REPHEADER LEFT ' Human Resources Department Report' –
RIGHT '02 January 2015'
BTITLE RIGHT 'Page №' FORMAT 9 SQL.PNO
REPFOOTER LEFT ' Human ResourcesManager' –
RIGHT ' /signature/'
BREAK ON REPORT SKIP 2

SET LINESIZE 50
SET PAGESIZE 15
COLUMN No FORMAT 99
SELECT ROWNUM AS No, EMPNO, ENAME, JOB, DEPTNO
FROM EMP ORDER BY EMPNO;
```

Пояснення. Перша команда (TTITLE) формує верхній колонтитул кожної сторінки з текстом **'LIST OF EMPLOYEES'**, розташованим по центру.

Друга команда (REPHEADER), що міститься на двох рядках, формує текст заголовку звіту **' Human Resources Department Report'** – ліворуч і дату формування – праворуч.

Третя команда (BTITLE) формує нижній колонтитул кожної сторінки звіту з номером цієї сторінки, що розташована праворуч.

Четверта команда (REPFOOTER), що також розташована на двох рядках, формує місце у кінці звіту для підпису керівником відділу.

П'ята команда (BREAK) задає пропуск двох рядків у кінці звіту перед підписом керівника.

Шоста та сьома команди (SET LINESIZE та SET PAGESIZE) задають розміри сторінки звіту у символах та рядках.

Восьма команда (COLUMN) визначає формат виведення поля на ім'я NO.

І нарешті, команда SELECT отримує з БД інформацію про співробітників, яка форматується згідно встановленим параметрам.

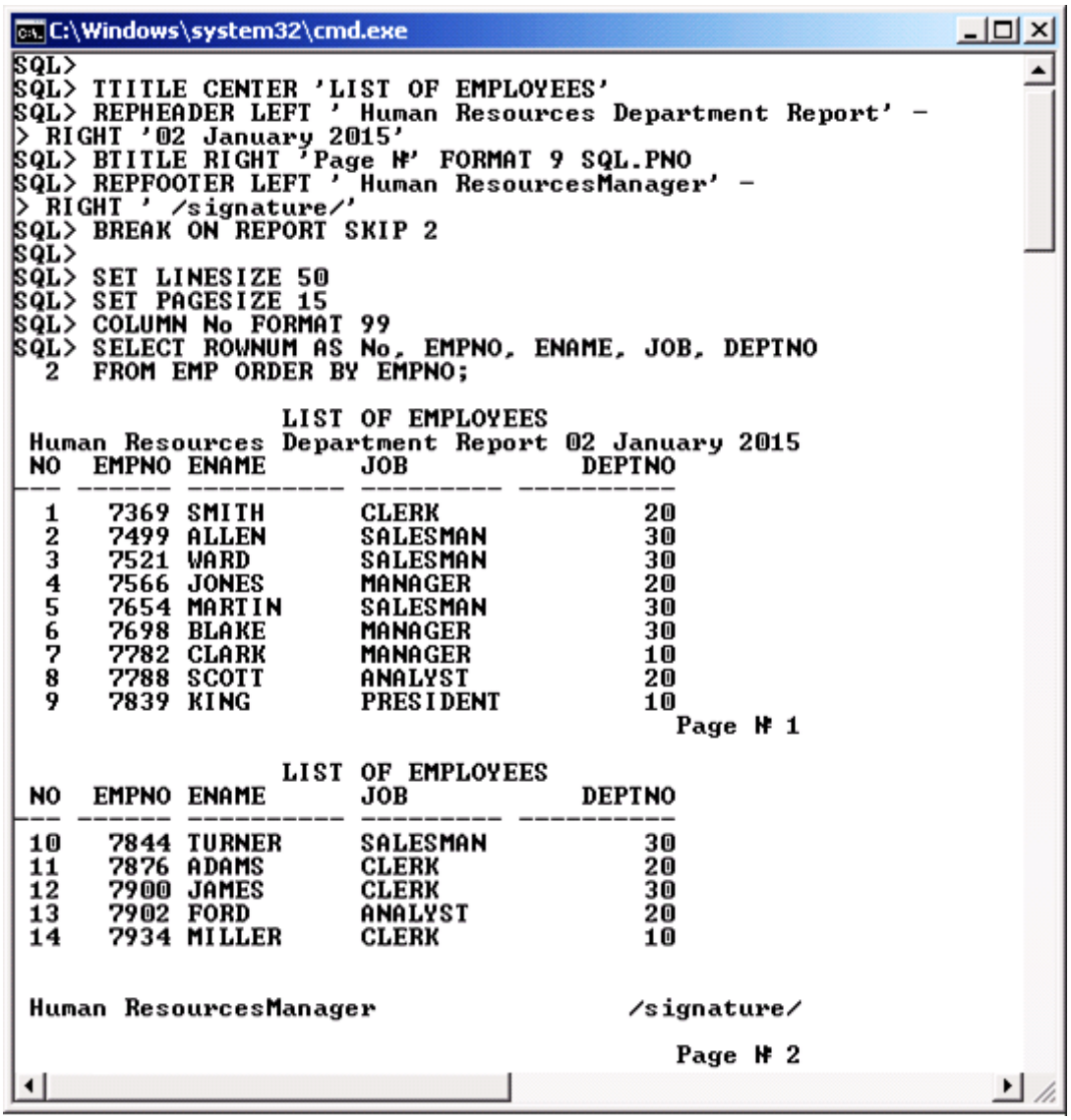


Рис. 1.64. **Результат формування звіту з кадрів**

Запитання і завдання

1. За допомогою яких команд можна сформувати звіт у середовищі SQL*Plus?
2. Опишіть основні опції команди COLUMN.

3. За допомогою якої команди можна обчислювати та виводити підсумкові значення?

4. За допомогою яких засобів можна керувати шириною і довжиною сторінки?

1.7. Інтегроване середовище розробки мовами SQL і PL/SQL – Oracle SQL Developer

1.7.1. Oracle SQL Developer – загальна інформація

Oracle SQL Developer – програмний засіб з графічним інтерфейсом для розробки програм мовами SQL та PL/SQL, і орієнтований на застосування у середовищі Oracle Database. Крім того, використовуючи плагіни до програми, можна підключатися до баз даних IBM DB2, Microsoft Access, Microsoft SQL Server, MySQL та інших. Корпорація Oracle надає продукт безкоштовно. Сама програма написана мовою програмування Java, працює на всіх платформах, де доступне середовище виконання Java SE.

Завантажити Oracle SQL Developer можна з офіційного з сайту корпорації Oracle за адресою [77] і встановити собі на комп'ютер. Працює програма без установки, потрібно лише розпакувати каталог SqlDeveloper із завантаженого zip-архіву.

Підтримує практично усі команди SQL*Plus окрім деяких, наприклад пов'язаних з форматуванням звітів.

1.7.2. Робота з базою даних Oracle за допомогою програми Oracle SQL Developer.

Запустити на виконання файл SqlDeveloper.exe (рис. 1.65.)

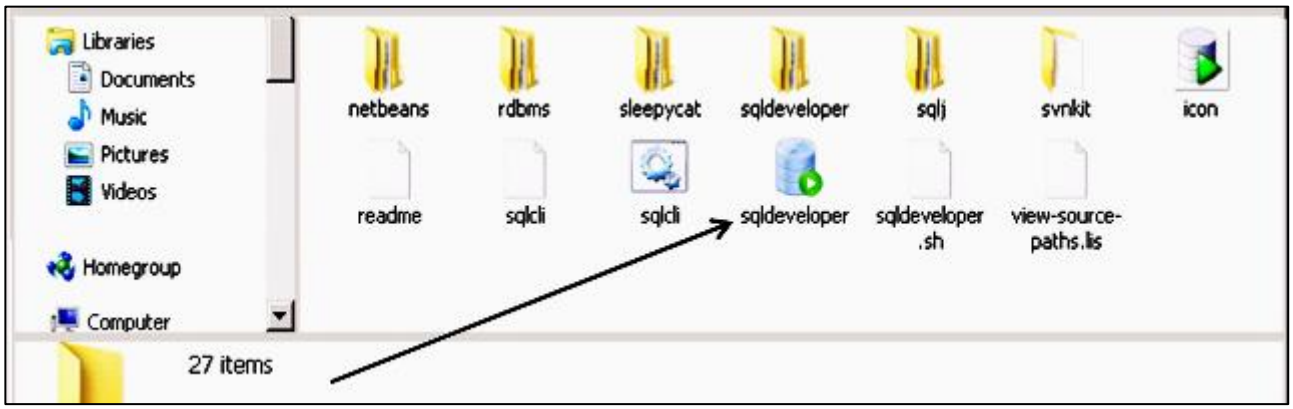


Рис. 1.65. **Запуск файла SqlDeveloper.exe**

Для створення нового з'єднання з базою даних слід натиснути значок "+" на вкладці "Connections" (рис. 1.66.), а далі ввести відповідні дані у вікно "New/Select Database Connection", що з'явилося, і натиснути кнопку "Connect" (рис. 1.67).

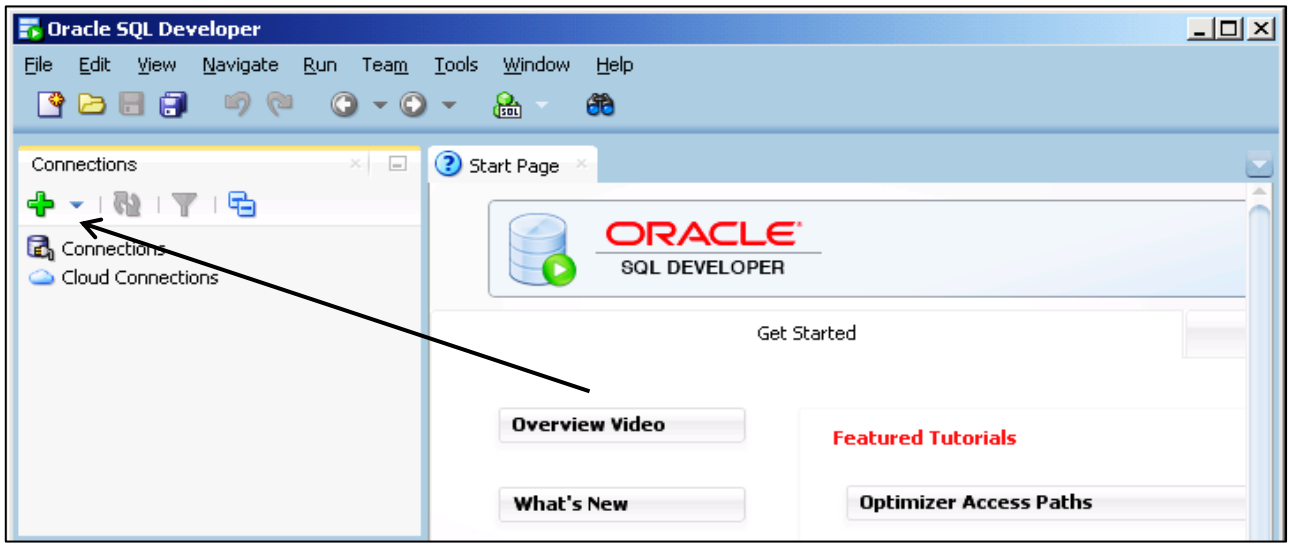


Рис. 1.66. **Створення нового з'єднання**

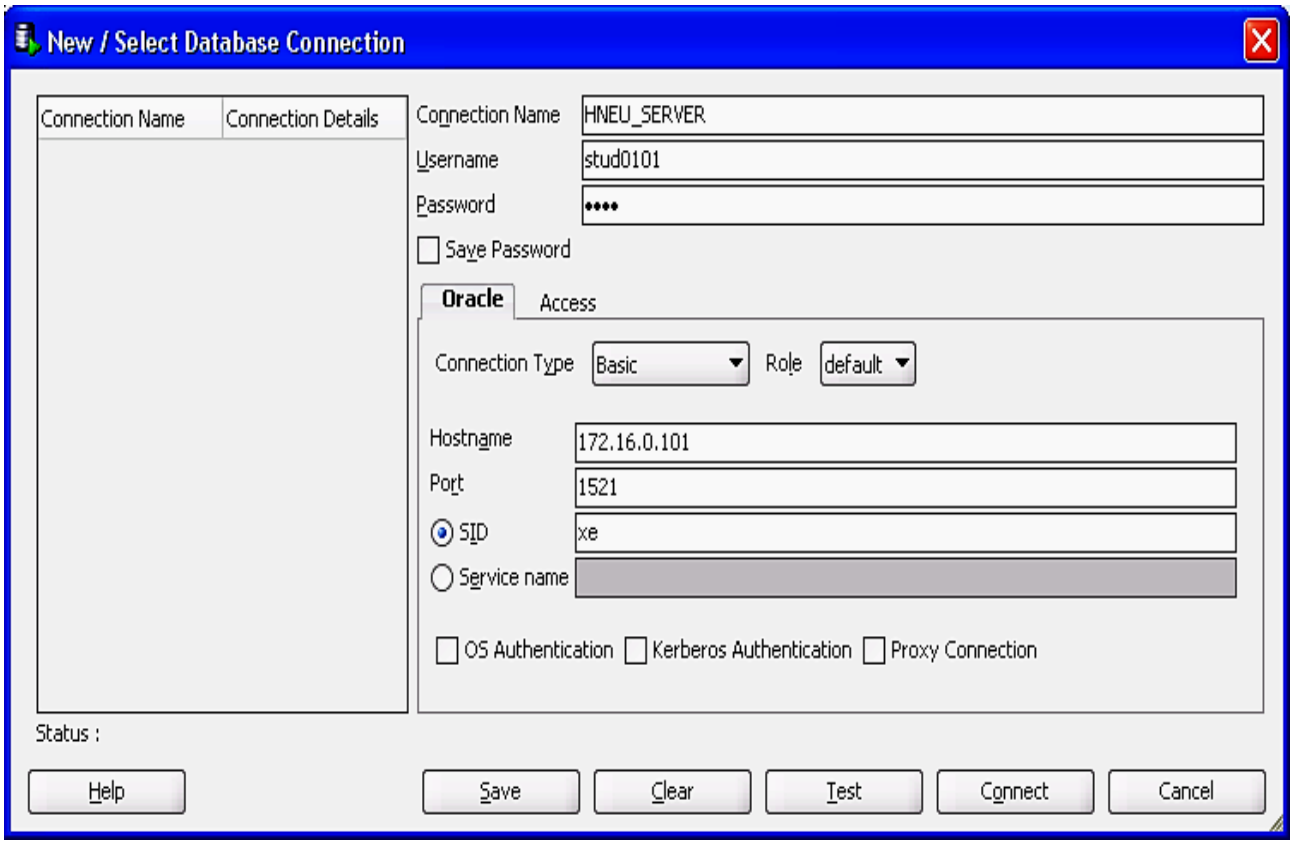


Рис. 1.67. **Введення даних для нового з'єднання.**

На вкладці "Connections" з'явиться нове з'єднання (рис. 1.68).

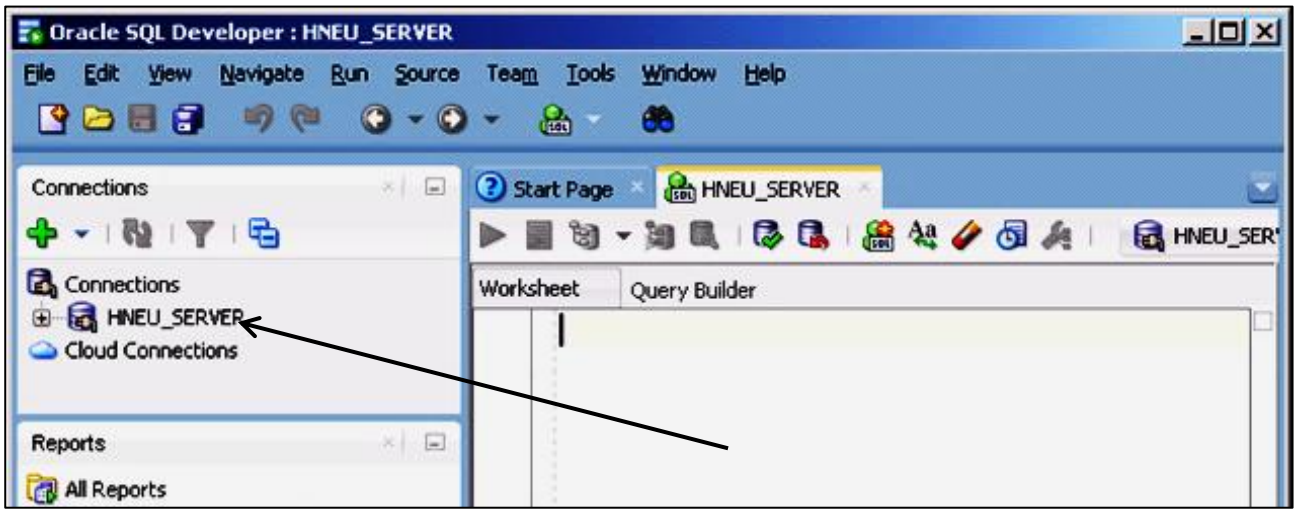


Рис. 1.68. **Нове з'єднання у вікні Oracle SQL Developer**

Розкривши дерево нового з'єднання, можна отримати доступ до об'єктів бази даних (рис. 1.69).

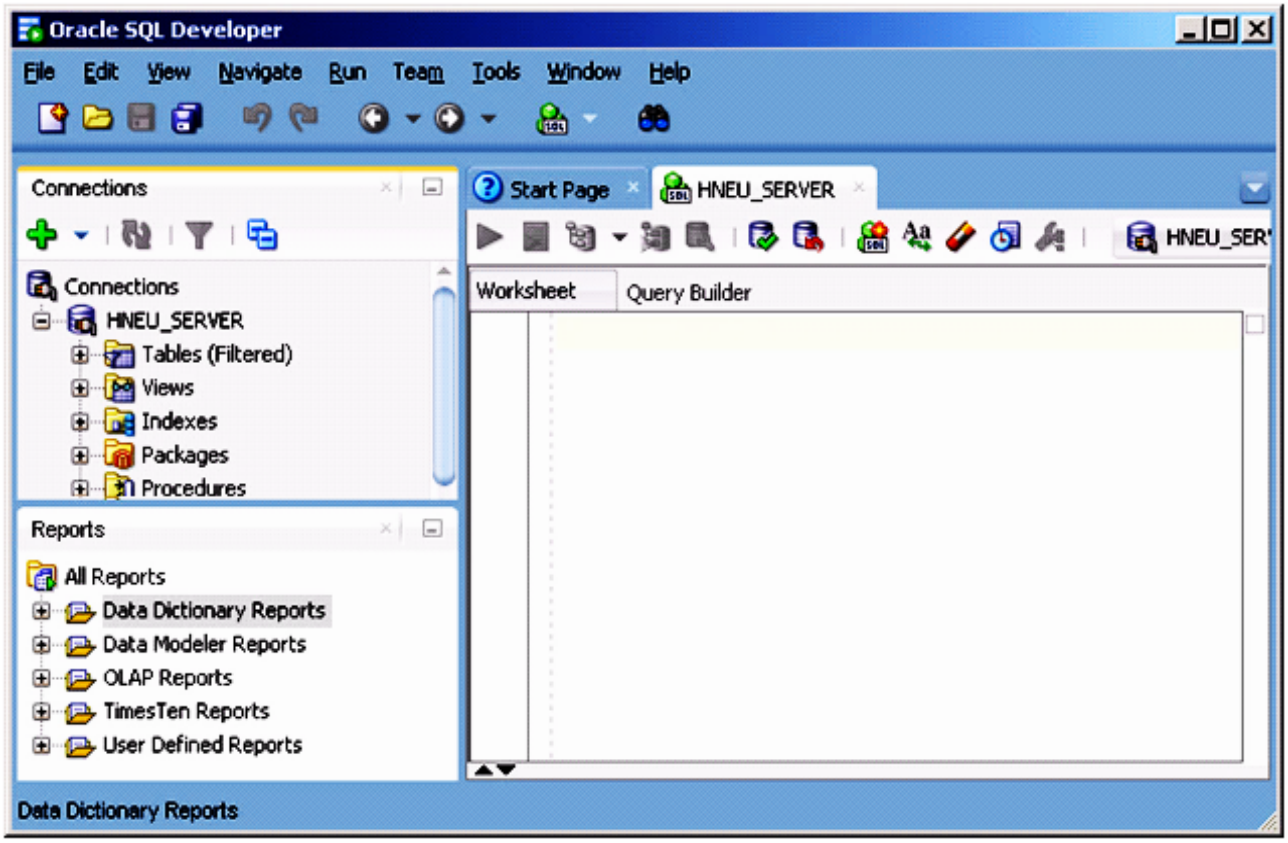


Рис. 1.69. **Дерево об'єктів бази даних**

На вкладці "WorkSheet" для поточного підключення можна ввести будь-який запит та виконати його, натиснувши кнопку "Run Script (F5)" (рис. 1.70).

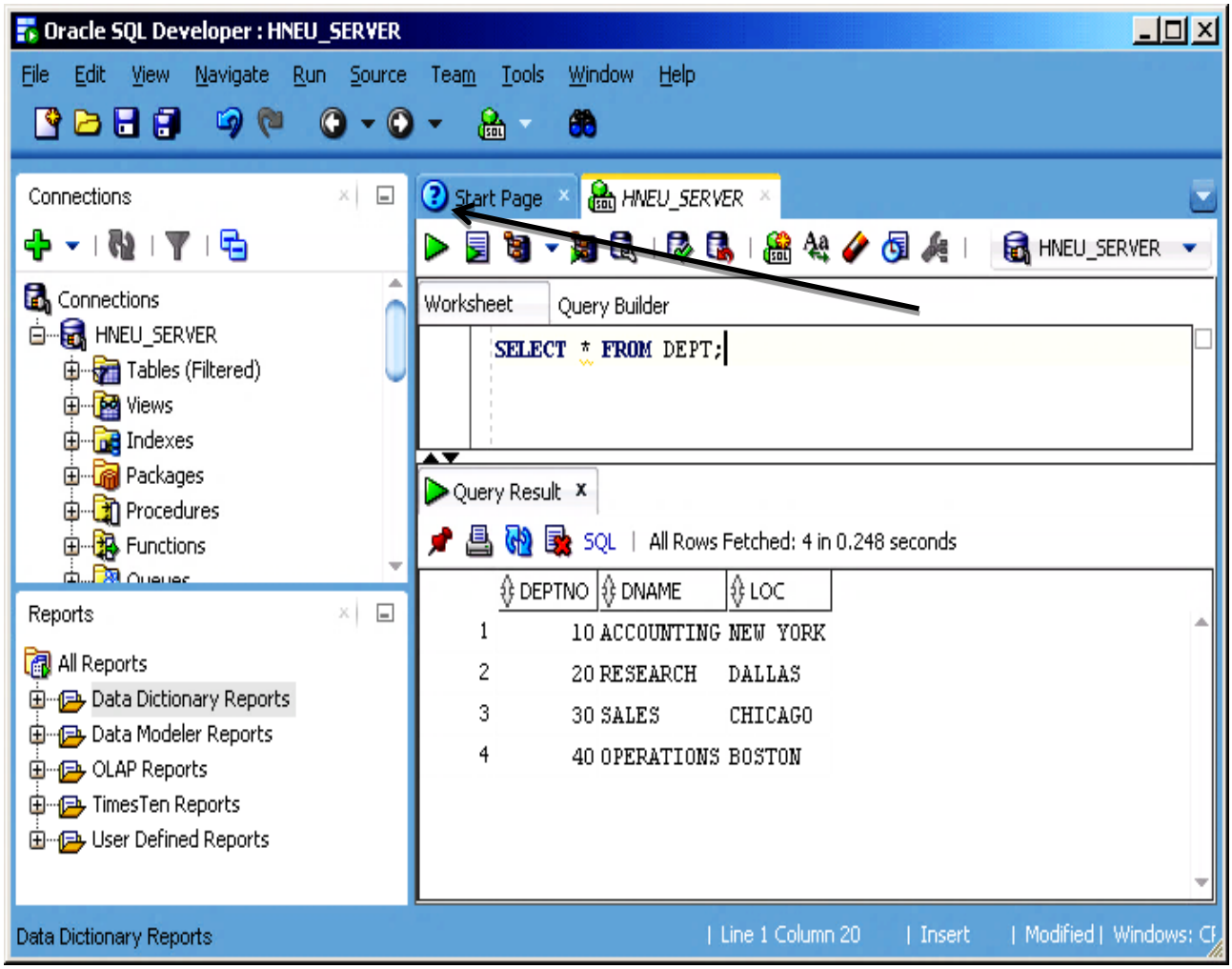


Рис. 1.70. **Виконання скрипта у вікні Oracle SQL Developer**

Можна також виконати будь-який скрипт мовою PL/SQL (рис. 1.71).

Користувач має можливість створити функцію або процедуру та використати їх у своїх запитах (рис. 1.72, 1.73) тощо.

Протокол результатів роботи можна зберегти у файлі, використавши панель інструментів і натиснувши кнопку "Save" (Зберегти) (рис. 1.74). Можна також використовувати команду SPOOL, однак, на відміну від SQL*Plus, вона тут працює тільки в межах поточного скрипта.

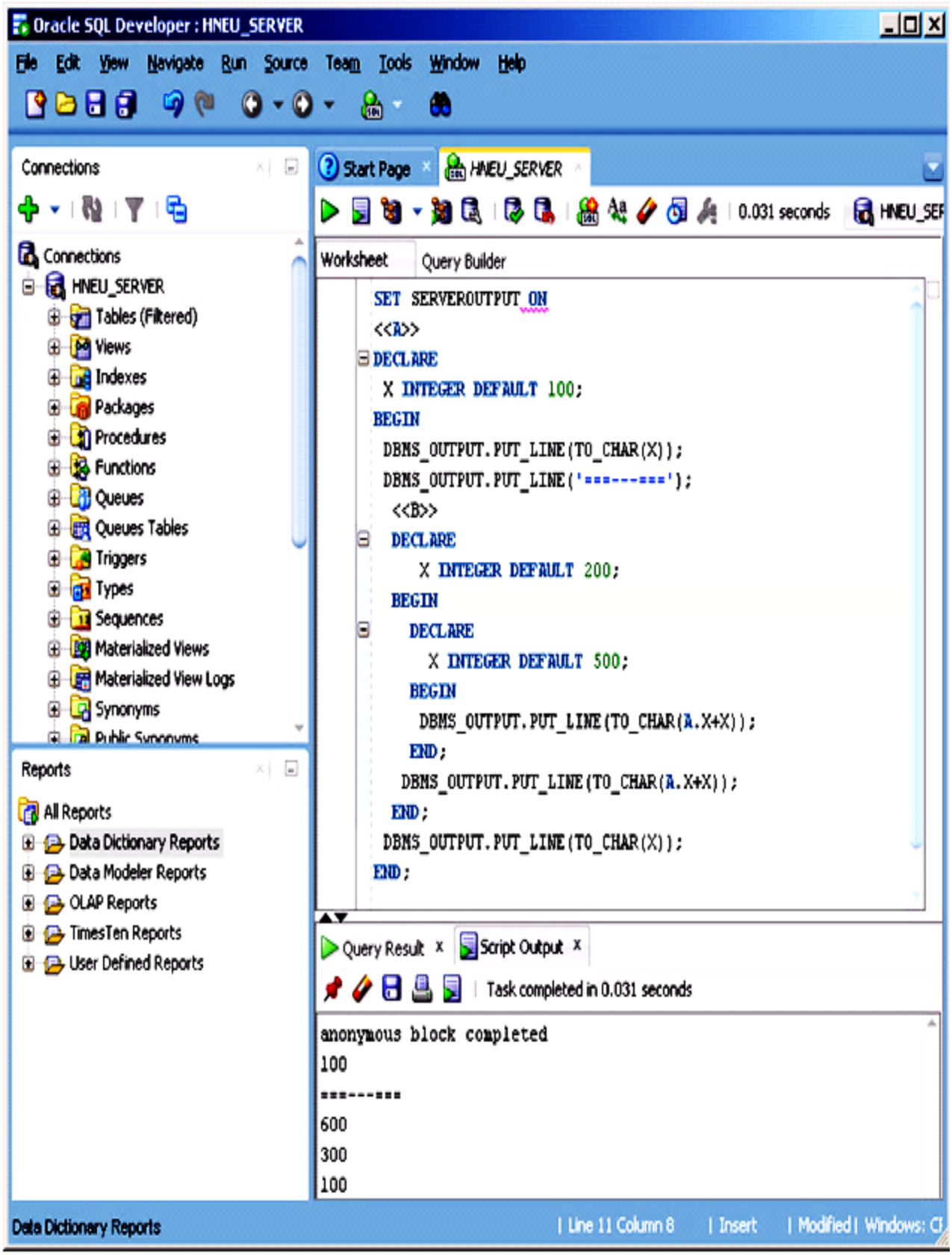


Рис. 1.71. **Виконання скрипта мовою PL/SQL**

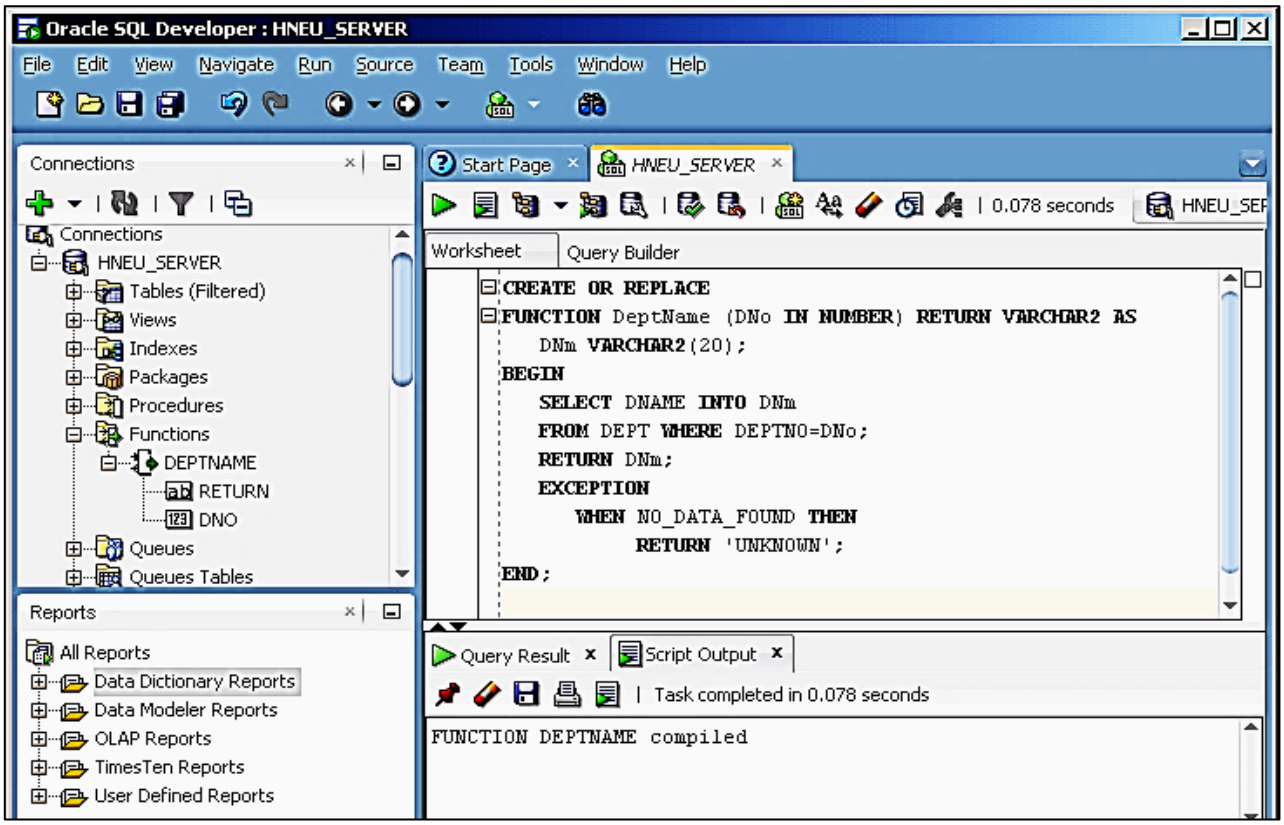


Рис. 1.72. **Створення функції користувача**

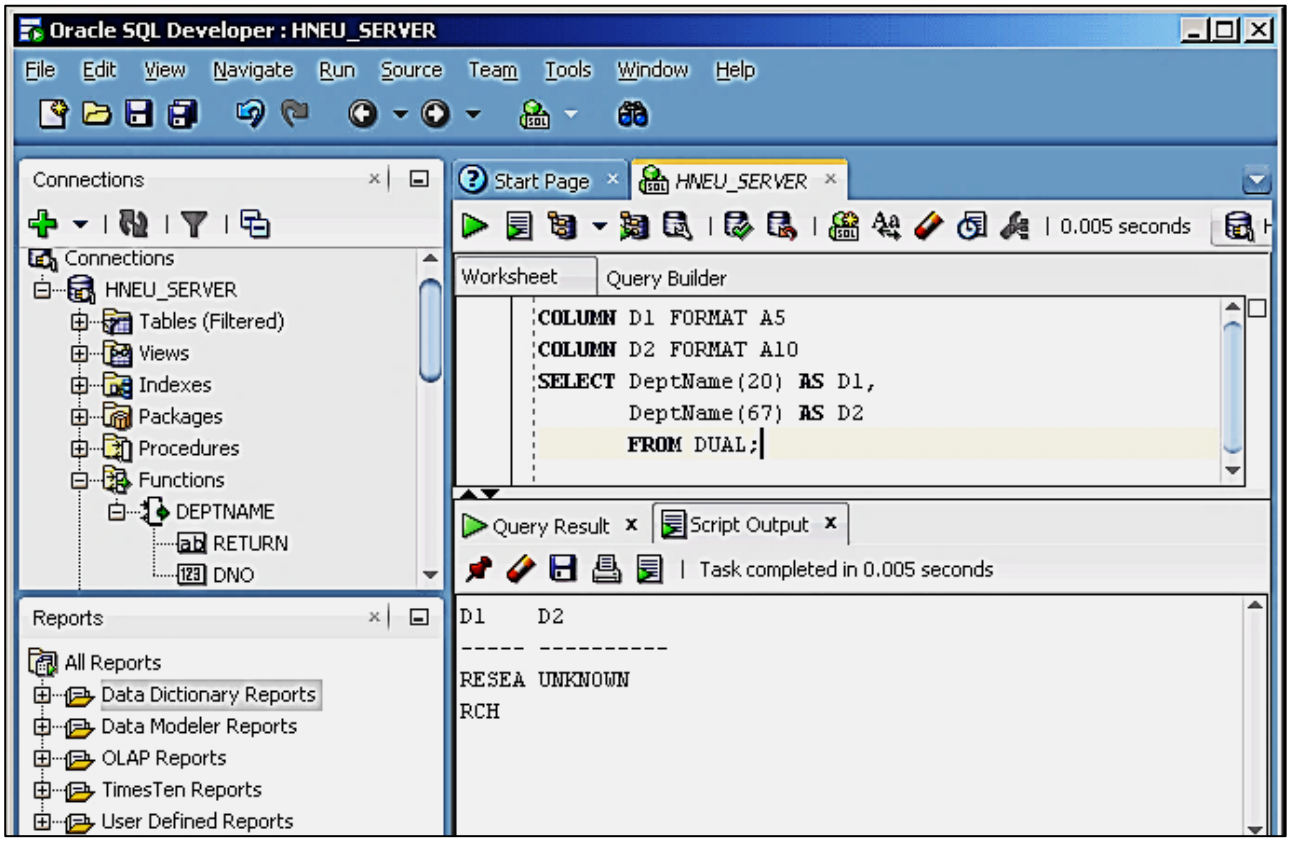


Рис. 1.73. **Використання функції користувача у запиті**

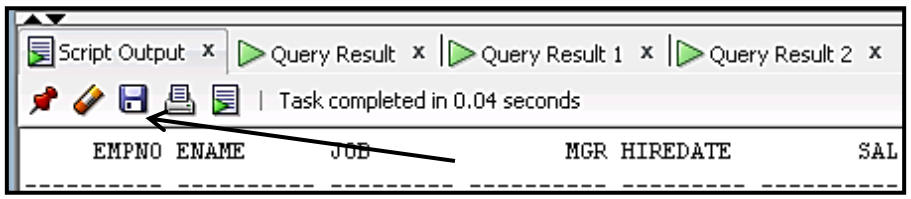


Рис. 1.74. **Збереження протоколу роботи у файл.**

Важливо Oracle SQL Developer не підтримує такі команди SQL*Plus, як ATTRIBUTE, BREAK, BTITLE, CLEAR, COMPUTE, OSERROR, RECOVER, REPFOOTER, REPHEADER, SQLERROR, STORE, TTITLE.

Запитання і завдання

1. Яке призначення має програма Oracle SQL Developer?
2. У чому полягає різниця між програмами SQL*Plus та SQL Developer?
3. Які програми для роботи з базами даних, що мають графічний інтерфейс, вам відомі?

Лабораторна робота № 1
Використання утиліти SQL*Plus для роботи з базою даних Oracle

Цілі лабораторної роботи:

1. Набуття практичних навичок підключення до баз даних Oracle за допомогою утиліти SQL*Plus.
2. Формування вмінь та навичок роботи з основними командами SQL*Plus та сумісного використання команд SQL*Plus та SQL.
3. Оволодіння засобами створення та використання командних файлів для автоматизації процесу розв'язання задач при роботі з БД.
4. Набуття практичних навичок форматування звітів на основі даних, збережених у базі.

Перед виконанням лабораторної роботи студент повинен знати:

1. Функціональне призначення утиліти SQL*Plus та її основних команд.
2. Правила узгодження синтаксису команд SQL*Plus та SQL.
3. Поняття "буферу SQL" та засоби його редагування.

4. Поняття змінних користувача та змінних підстановки та їх призначення.

5. Призначення командних файлів та правила використання параметрів.

6. Основні команди, що призначені для форматування звітів.

Після виконання лабораторної роботи студент повинен уміти:

1. Самостійно підключатися та відключатися від бази даних Oracle засобами SQL*Plus.

2. Використовувати команди SQL*Plus для налаштування режимів роботи утиліти.

3. Виконувати запити до БД мовою SQL з форматуванням отриманого результату.

4. Формувати звіти за даними, збереженими у БД.

Порядок виконання роботи:

1.Підготовчий етап до виконання лабораторної роботи.

2.Отримання відомостей про структуру та вміст таблиць БД.

3. Виконання команд SQL.

4. Виконання блоків PL/SQL.

5. Виконання команд SQL*Plus.

6. Управління значеннями системних змінних SQL*Plus.

7. Використання додаткових можливостей SQL*Plus.

8. Команди редагування вмісту буфера SQL.

9. Створення та використання командних файлів.

10. Написання діалогових програм.

11. Форматування звітів.

1. Підготовка до виконання лабораторної роботи

Для виконання лабораторної роботи спочатку треба виконати певні підготовчі кроки:

1. Завантажити на свій носій (флешку) zip-архів з лабораторною роботою з сайту дистанційного навчання ХНЕУ та розпакувати архів.

2. Ознайомитися зі вмістом архіву. Архів містить опис поточної лабораторної роботи, каталог XEClient з програмою SQL*Plus для підключення до сервера БД Oracle, скрипт SCOTT_XE.SQL для створення у БД тестових таблиць.

Каталог XEClient перемістіть до кореня флеш-диску (тобто D:\XEClient)!

3. Для запуску SQL*Plus та підключення до сервера Oracle необхідно увійти до каталога XEClient та запустити на виконання пакетний файл **StartSqlPlus.bat.** Після запуску з'явиться консольне вікно SQL*Plus (рис. 1.75).

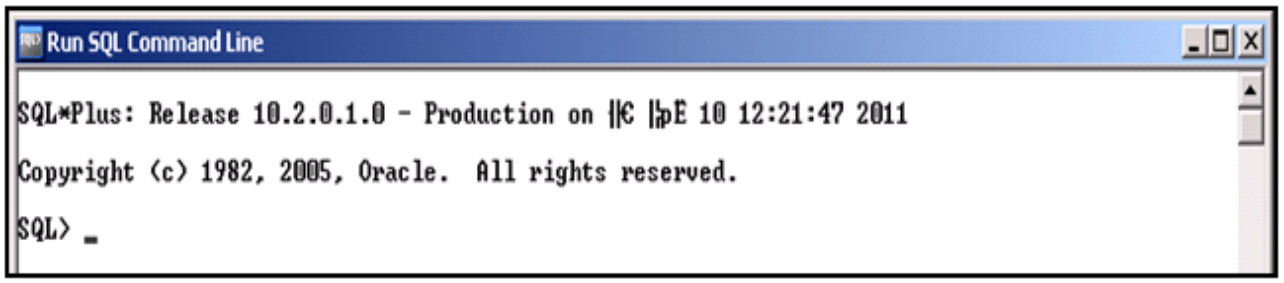


Рис. 1.75. **Вигляд консольного вікна SQL*Plus**

Розмір вікна, що відкриється за замовчуванням, бажано змінити. Для цього потрібно клацнути правою кнопкою миші на піктограмі у верхньому лівому куті, вибрати у контекстному меню пункт "Свойства" і далі у вікні, що з'явиться вибрати вкладку "Расположение" (рис. 1.76) та встановити значення розміру вікна, наприклад, 120 x 50, або інше за бажанням

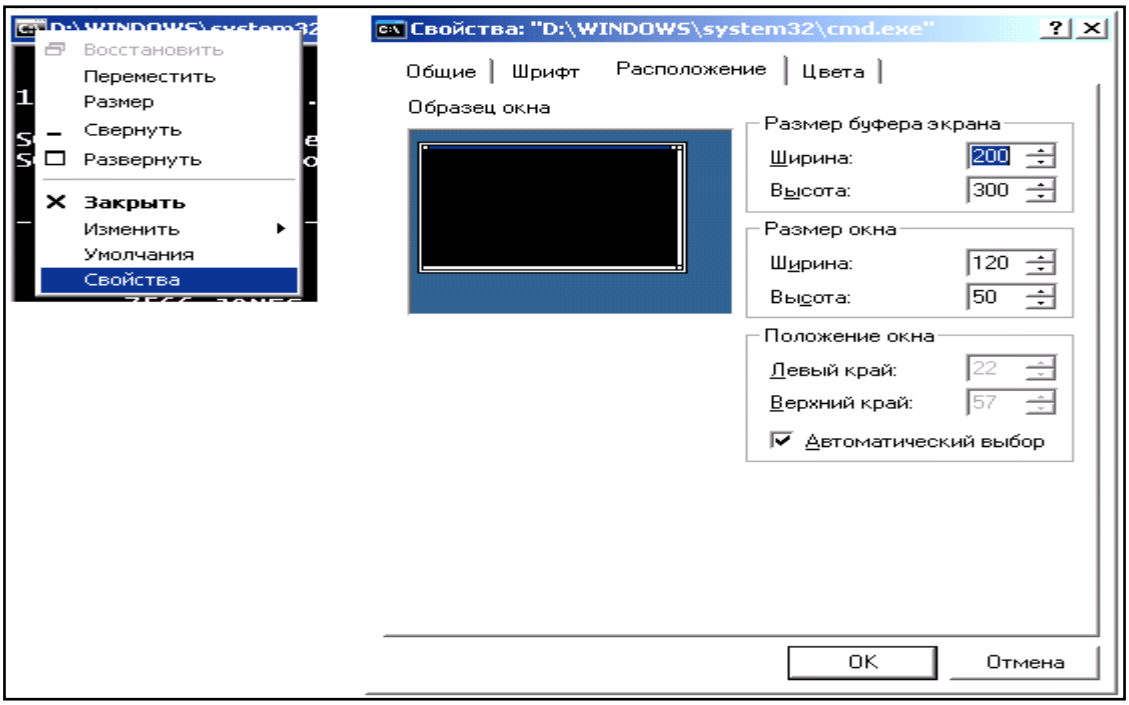


Рис. 1.76. **Встановлення розмірів вікна SQL*Plus**

Повторіть призначення команди SET LINESIZE.

4. Для підключення до сервера введіть наступну команду:

CONNECT studGGNN/stud@172.16.0.101/XE,

де studGGNN – ім'я користувача (допустимі імена **stud0101, stud0102,…stud0330),**

де **GG**– номер відповідний номеру групи;

NN – номер студента у журналі групи.

Після введення команди повинно з'явитися слово Connect, що сповіщає, про успішне підключення до сервера.

5. Для ознайомлення зі вмістом скрипта SCOTT_XE.SQL відкрити його у програмі "Блокнот", та визначити, які таблиці будуть створені у тестовій БД, та які дані до них будуть занесені.

6. Для створення тестової БД введіть команду:

START D:\path\scott_xe.sql,

де path – це шлях до скрипта scott_xe.sql на вашому флеш-носії.

При виконанні роботи, якщо повідомлення про помилки виводяться незрозумілими символами, можна спробувати змінити поточну кодову сторінку сеансу за допомогою команди

HOST CHCP 1251

7. Введіть та виконайте команду **SET AUTOCOMMIT ON** для встановлення режиму автоматичного збереження змін у базі даних.

8. Введіть команду **SPOOL D:\filename**, де filename – ім'я файла з розширенням LST, для зберігання у файлі протоколу роботи з базою даних.

Після закінчення виконання роботи перед виходом із SQL*Plus необхідно ввести команду **SPOOL OFF**. По закінченні лабораторної роботи, слід завершити роботу з SQL*Plus командою **EXIT** або **QUIT**.

2. Отримання відомостей про структуру та вміст таблиць БД

Завдання 1. Вивести на екран відомості про структуру таблиць тестової бази даних, що була створена за допомогою скрипта SCOTT_XE.SQL, та отримати дані з них.

Виконання

У SQL*Plus для виведення структури таблиці використовується команда DESCRIBE. Ця команда видає імена стовпців і типи даних, а також відомості про те, чи повинен стовпець містити дані.

Для отримання даних з таблиці DEPT, створеної бази даних після підказки ***SQL>*** ввести команду DESCRIBE DEPT та натиснути клавішу [Enter]. Результат отримаємо на екрані:

```
SQL> DESCRIBE DEPT
 Name                      Null?    Type
 ------------------------- -------- -------
 DEPTNO                    NOT NULL NUMBER(2)
 DNAME                              VARCHAR2(14)
 LOC                                VARCHAR2(13)
```

Щоб переглянути вміст таблиці DEPT, потрібно ввести команду:

```
SQL> SELECT * FROM DEPT;
```

Результат виконання буде такий:

```
    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Для інших таблиць, що створені у БД, самостійно виконайте такі дії:

1. Визначте їх імена, аналізуючи текст скрипта SCOTT_XE.SQL.

2. Отримайте відомості по структуру кожної таблиці.
3. Отримайте відомості про дані, що зберігаються у кожній таблиці.

3. Виконання команд SQL

Мова команд SQL дозволяє маніпулювати даними в БД, і більш докладно її можливості будуть розглянуті під час вивчення команд SQL у другій лабораторній роботі.

Завдання 2. За допомогою команди SQL отримати відомості про номери співробітників, їх імена, посади та оклади, що містяться у таблиці EMP, для тих співробітників, у яких оклад менше за 2500.

Виконання

1. Після командної підказки ***SQL>*** ввести перший рядок команди

SELECT EMPNO, ENAME, JOB, SAL

Якщо при введенні були допущені помилки, то можна використовувати [Backspace] для стирання помилки, а надалі – повторного уведення правильного рядка. По закінченні введення команди, натиснути [Enter] для переходу до наступного рядка.

2. SQL*Plus покаже "2"– підказку для введення другого рядка. Ввести другий рядок команди

FROM EMP WHERE SAL < 2500;

Крапка з комою (;) означає кінець команди. Натиснути [Enter]. SQL*Plus обробить команду й видасть результати на екрані:

```
SQL> SELECT EMPNO, ENAME, JOB, SAL
  2  FROM EMP WHERE SAL < 2500;

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7369 SMITH      CLERK            800
      7499 ALLEN      SALESMAN        1600
      7521 WARD       SALESMAN        1250
      7654 MARTIN     SALESMAN        1250
```

```
7782 CLARK      MANAGER        2450
7844 TURNER     SALESMAN       1500
7900 JAMES      CLERK           950
7934 MILLER     CLERK          1300

8 rows selected.
```

Після показу результатів і кількості обраних записів SQL*Plus знову виводить командну підказку. Якщо при введенні команди була зроблена помилка й не отримано результатів, потрібно знов ввести команду, виправивши помилку.

Розподіл команд SQL на окремі рядки

Команду SQL можна ділити на декілька рядків у будь-яких місцях, але не можна розривати окремі слова між рядками. Таким чином, можна вводити запит прикладу, наведеного вище, на одному рядку:

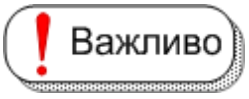
SELECT EMPNO, ENAME, JOB, SAL FROM EMP WHERE SAL < 2500;

або його можна вводити на декількох рядках:

```
SQL>  SELECT
2 EMPNO, ENAME, JOB, SAL
3 FROM EMP
4 WHERE SAL<2500;
```

Більшість команд SQL найчастіше розділяється на пункти – по одному пункту на кожному рядку.

Завершення команди SQL



Можна завершити команду SQL одним із трьох способів:

- за допомогою крапки з комою (;);
- за допомогою похилої риски (слеш) (/);
- за допомогою порожнього рядка.

Крапка з комою повідомляє SQL*Plus, що команда закінчилась і її необхідно виконати. Після введення крапки з комою треба натиснути [Enter]. SQL*Plus обробить команду. Якщо помилково натиснено [Enter]

до введення крапки з комою, SQL*Plus виведе новий номер рядка для команди. Щоб виконати команду, введіть крапку з комою й натисніть [Enter] знову.

Похила риска (/) у рядку також повідомляє SQL*Plus, що необхідно виконати команду. Натисніть [Enter] наприкінці останнього рядка команди. SQL*Plus виведе новий номер рядка. Введіть похилу рису й натисніть [Enter] знову. SQL*Plus виконає команду.

Порожній рядок повідомляє SQL*Plus, що введення команди було закінчено, але її поки не потрібно виконувати. Натисніть [Enter] наприкінці останнього рядка команди. SQL*Plus підкаже новий номер рядка.

Натисніть [Enter] знову, зараз SQL*Plus виведе вже свою підказку. SQL*Plus не виконає команду, але запам'ятає її в пам'яті, що називається буфером SQL.

Самостійно виконайте команду зі завдання 2 декількома способами: розташовуючи її на різній кількості рядків (від одного до чотирьох); закінчуючи команду різними способами.

Виконання поточної команди SQL

Завдання 3. Виконати повторно поточну команду SQL, що зберігається у буфері SQL, не вводячи її знову.

Виконання

Після командної підказки ***SQL>*** уведіть команди **RUN** або похилу риску "/" і натиснути [Enter]. У результаті поточна команда SQL, тобто та, яка останньою виконувалась до цього моменту, знов буде виконана.

```
SQL> RUN
  1  SELECT EMPNO, ENAME, JOB, SAL
  2* FROM EMP WHERE SAL < 2500

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7369 SMITH      CLERK            800
      7499 ALLEN      SALESMAN        1600
      7521 WARD       SALESMAN        1250
      7654 MARTIN     SALESMAN        1250
```

```
7782 CLARK      MANAGER          2450
   7844 TURNER     SALESMAN         1500
   7900 JAMES      CLERK             950
   7934 MILLER     CLERK            1300
```

8 rows selected.

4. Виконання блоків PL/SQL

Працюючи у SQL*Plus, можна, також використовувати програми мовою PL/SQL (часто вони називаються блоками) для маніпулювання даними у БД. Блоки PL/SQL починаються з ключових слів DECLARE, BEGIN або імені блоку SQL*Plus.

Працювати із блоками PL/SQL можна таким же чином, як і з командами SQL, крім крапки з комою або порожнього рядка, які не закінчують і не виконують блок.

Блок PL/SQL завершується крапкою (.) на новому рядку. SQL*Plus зберігає введені блоки у буфері SQL. Виконуючи поточний блок за допомогою команди RUN або "/", SQL*Plus посилає весь блок PL/SQL до ORACLE RDBMS для обробки.

Завдання 4. Ввести та виконати тестовий блок команд мовою PL/SQL, за наведеним прикладом.

Виконання

1. Для виконання цього завдання потрібно створити додаткову таблицю на ім'я TEMP – самостійно визначте її структуру, проаналізувавши команду INSERT у наведеному скрипті, що розташований далі, і попередньо створіть таблицю командою CREATE TABLE.

2. Після створення таблиці TEMP – перевірте її структуру за допомогою команди DESCRIBE.

3. Введіть команду SET SERVEROUTPUT ON для активізації виведення результатів роботи скрипта на екран.

4. Введіть та виконайте блок PL/SQL.

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
  2     x   NUMBER :=100;
  3  BEGIN
  4     FOR i IN 1..10 LOOP
  5       IF TRUNC(i / 2) = i / 2 THEN   -- ?  is  even
  6         INSERT INTO temp VALUES(i, x, ' i is  even');
  7       ELSE
  8         INSERT INTO temp VALUES(i, x, ' i is odd');
  9       END IF;
 10       DBMS_OUTPUT.PUT_LINE ('I= '||i||' X='||x);
 11       x := x+100;
 12     END LOOP;
 13  END;
 14 .
SQL> /
I= 1 X=100
I= 2 X=200
I= 3 X=300
I= 4 X=400
I= 5 X=500
I= 6 X=600
I= 7 X=700
I= 8 X=800
I= 9 X=900
I= 10 X=1000
PL/SQL procedure successfully completed.
```

5. Після успішного виконання блоку PL/SQL отримати за допомогою команди SELECT вміст таблиці TEMP.

5. Виконання команд SQL*Plus

Команди SQL*Plus можна використовувати для маніпулювання командами SQL та блоками PL/SQL для форматування та друкування результатів запитів. SQL*Plus обробляє команди SQL*Plus інакше, ніж команди SQL і блоки PL/SQL. Для збільшення

швидкості введення команд можна скорочувати більшість команд SQL*Plus до однієї або декількох букв.

Завдання 5. Використовуючи команду SQL*Plus – COLUMN, змінити формат виведення стовпця SAL з таблиці EMP.

Виконання

1. Уведіть та виконайте команду

```
SELECT EMPNO, ENAME, JOB, SAL FROM EMP WHERE SAL < 2500;
```

та проаналізуйте отриманий результат, звертаючи особливу увагу на відображення значень у стовпці SAL.

2. Уведіть команду SQL*Plus:

```
COLUMN SAL FORMAT $99,999 HEADING SALARY
```

Якщо в процесі введення були допущені помилки, використовуючи [Backspace], виправте команду. По закінченні натиснути клавішу [Enter]. SQL*Plus запам'ятає новий формат і знову виведе командну підказку SQL*Plus, тобто він знову буде готовий до прийому команд.

3. Уведіть команду RUN для повторного виконання останнього запиту (вищеприведений приклад). SQL*Plus знову обробить запит і виведе результати:

```
SQL> COLUMN SAL FORMAT $99,999 HEADING SALARY
SQL> RUN
  1* SELECT EMPNO, ENAME, JOB, SAL FROM EMP WHERE SAL<2500
     EMPNO ENAME      JOB          SALARY
---------- ---------- --------- --------
      7369 SMITH      CLERK          $800
      7499 ALLEN      SALESMAN     $1,600
      7521 WARD       SALESMAN     $1,250
      7654 MARTIN     SALESMAN     $1,250
      7782 CLARK      MANAGER      $2,450
      7844 TURNER     SALESMAN     $1,500
      7900 JAMES      CLERK          $950
      7934 MILLER     CLERK        $1,300

8 rows selected.
```

Команда COLUMN форматувала колонку SAL за допомогою знака долар ($) і коми (,) і присвоїла їй новий заголовок. Команда RUN знову виконала запит із прикладу, що зберігається в буфері. SQL*Plus не запам'ятовує команди SQL*Plus у буфері.

Узгодження синтаксису команд SQL*Plus з командами SQL

Команди SQL*Plus мають синтаксис, відмінний від синтаксису команд SQL і блоків PL/SQL. Продовжити довгі команди SQL*Plus на інших рядках за допомогою введення дефіса (сполучна риска) наприкінці рядка й натискання [Enter]. SQL*Plus виведе праву кутову дужку (>) як підказку для кожного додаткового рядка.

Закінчувати команду SQL*Plus за допомогою крапки з комою необов'язково Після закінчення введення команди можна одразу натиснути [Enter]. Однак, за бажанням, можна ввести крапку з комою наприкінці команди SQL*Plus.

Змінні, що впливають на виконання команд

Команда SET утиліти SQL*Plus управляє більшістю змінних, які називаються системними змінними й установка яких впливає на спосіб виконання SQL*Plus команд, що вводяться.

Системні змінні управляють безліччю параметрів усередині SQL*Plus, включаючи ширину стовпців за замовчуванням для виведення на екран, показ кількості оброблених записів або довжину сторінки. Системні змінні також називають змінними команди SET.

Залежно від установки системних змінних виведені дані в прикладах можуть відрізнятися від наведених в даній роботі.

6. Управління значеннями системних змінних SQL*Plus

Завдання 6. Отримати на екрані поточні значення усіх системних змінних.

Виконання

Щоб роздрукувати поточні установки змінних, уведіть команду SHOW ALL

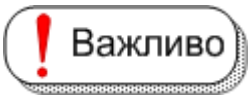
```
SQL> SHOW ALL
appinfo is OFF and set to "SQL*Plus"
arraysize 15
autocommit IMMEDIATE
autoprint OFF
autorecovery OFF
autotrace OFF
blockterminator "." (hex 2e)
btitle OFF and is the first few characters of the next
       SELECT statement
cmdsep OFF
colsep " "
.................
```

Завдання 7. Отримати на екрані поточні значення системних змінних LINESIZE та PAGESIZE.

Виконання

Для отримання поточних установок указаних змінних уведіть команди SHOW LINESIZE та SHOW PAGESIZE.

```
SQL> SHOW LINESIZE
linesize 120
SQL> SHOW PAGESIZE
pagesize 60
```



Автоматичне збереження змін у БД регулюється значенням системної змінної AUTOCOMMIT. Для активізації цього режиму слід виконати команду **SET AUTOCOMMIT ON.**

Цей режим був активізований на підготовчому етапі поточної лабораторної роботи. Для скасування режиму автоматичного збереження змін слід виконати команду **SET AUTOCOMMIT OFF**.

Щоб результати лабораторної роботи зберігались у БД постійно, режим автоматичного зберігання треба включати одразу після підключення до бази даних.

7. Використання додаткових можливостей SQL*Plus

Завдання 8. Проілюструвати можливості збору статистичних даних про час виконання певних команд при роботі у SQL*Plus. Для ілюстрації завдання виконати такі дії.

1. Створити та запустити на виконання перший таймер.
2. Ввести команду на отримання даних з таблиці DEPT.
3. Створити та запустити на виконання другий таймер.
4. Повторно отримати дані з таблиці DEPT за допомогою команди RUN.
5. Вивести на екран кількість активованих таймерів та значення поточного таймеру.
6. Зупинити другий таймер.
7. Вивести на екран кількість активованих таймерів та значення поточного таймеру.
8. Зупинити перший таймер.

Виконання

Введіть рядок за рядком та запустіть на виконання наступні команди у вікні SQL*Plus:

```
TIMING START TIMER_1
SELECT * FROM DEPT;
TIMING START TIMER_2
RUN
TIMING
TIMING SHOW
TIMING STOP
TIMING
TIMING SHOW
TIMING STOP
```

Після завершення виконання завдання порівняйте отримані результати часу виконання команд від наведених нижче. Спробуйте пояснити розбіжність у часі виконання команд.

```
SQL> TIMING START TIMER_1
SQL> SELECT * FROM DEPT;

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> TIMING START TIMER_2
SQL> RUN
  1* SELECT * FROM DEPT

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

SQL> TIMING
2 timing elements in use
SQL> TIMING SHOW
timing for: TIMER_2
Elapsed: 00:00:00.01
SQL> TIMING STOP
timing for: TIMER_2
Elapsed: 00:00:00.01
SQL> TIMING
1 timing element in use
SQL> TIMING SHOW
timing for: TIMER_1
Elapsed: 00:00:00.04
SQL> TIMING STOP
timing for: TIMER_1
Elapsed: 00:00:00.04
```

Завдання 9. Проілюструвати можливість виконання команд операційної системи (ОС), не покидаючи SQL*Plus. Щоб виконати команду ОС, слід ввести команду SQL*Plus HOST з наступною командою операційної системи.

Для ілюстрації отримати за допомогою команд операційної системи перелік файлів з розширенням.SQL на диску D:\ та вміст файла INIT.SQL на диску D:\.

Виконання

Для першого завдання слід ввести та виконати команду HOST DIR D:*.SQL

```
SQL> HOST DIR D:\*.SQL
 Volume in drive D has no label.
 Volume Serial Number is 90C4-1EB5

 Directory of D:\
21.09.2014  22:39               190 BONUS.SQL
08.09.2014  10:00               849 DEPT.SQL
08.09.2014  10:22             3 116 EMP.SQL
08.09.2014  11:33                90 index.sql
14.09.2014  16:36               129 INIT.sql
               5 File(s)          4 374 bytes
               0 Dir(s)   4 671 997 952 bytes free
```

Для другого завдання – команду HOST TYPE D:\INIT.SQL

```
SQL> HOST TYPE D:\INIT.SQL
SET AUTOCOMMIT ON
SET LINESIZE 120
SET PAGESIZE 60
SPOOL E:\SPOOLING APPEND
ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY';
```

Після завершення виконання команди ОС на екрані знову з'явиться командна підказка SQL*Plus.

Якщо ввести команду HOST без параметрів, то SQL*Plus переходить до командного рядка операційної системи.

У цьому випадку команди ОС можна виконувати безпосередньо. Для повернення до SQL*Plus слід ввести команду EXIT.

```
SQL> HOST
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\XEClient\bin>EXIT
SQL>
```

Завдання 10. Проілюструвати виконання команд, що є помилковими. Для цього отримати відомості про структуру неіснуючої таблиці (команда SQL*Plus) та відомості про співробітників певного відділу з помилкою в умові пошуку.

Виконання

Для першого завдання ввести команду **DESCRIBE DPT.**

```
SQL> DESCRIBE DPT
ERROR:
ORA-04043: object DPT does not exist
```

Система сповіщає, що об'єкт на ім'я DPT не існує у БД.

Для другого завдання ввести запит SELECT * FROM EMP WHER DEPTNO=20;

```
SQL> SELECT * FROM EMP WHER DEPTNO=20;
SELECT * FROM EMP WHER DEPTNO=20
                         *
ERROR at line 1:
ORA-00933: SQL command not properly ended
```

Система видає повідомлення про помилку і зірочкою вказує на місце її виявлення. У даному випадку помилка була у слові WHERE.

Якщо необхідні подальші пояснення, то потрібно зробити один із наступних кроків, щоб визначити причину помилки та спосіб її виправлення:

• якщо помилка має номер, що починається з "ORA", потрібно шукати повідомлення у "Довіднику з кодів і повідомлень про помилки ORACLE" або "Керівництві з інсталяції для користувача";

• якщо помилка без номера, необхідно знайти правильний синтаксис команди, що призвела до помилки, у "Довіднику команд SQL*Plus", у "Довіднику з мови SQL" або у " Керівництві користувача з мови PL/SQL" для блоків PL/SQL.

8. Команди редагування вмісту буфера SQL

Команди редагування дозволяють виправити команду SQL чи блок команд PL/SQL, що вводиться, без повторного її введення. Для цього можна використати номери рядків для редагування команд SQL або блоків PL/SQL, збережених у цей момент у буфері. Також утиліта SQL*Plus дозволяє редагувати вміст буфера за допомогою редактора ОС.

У табл. 1.3. було наведено перелік команд SQL*Plus, що дозволяють переглядати та змінювати команду в буфері без її повторного введення.

Ці команди дуже корисні, якщо ви помилилися при введенні або побажали змінити вже введену команду при роботі з консолі оператора. Однак наразі вони рідко використовуються, оскільки більш вживаним є використання системного текстового редактора (п. 1.3.10).

8.1. Друкування вмісту буфера

Будь-які команди редагування (крім LIST) впливають тільки на один рядок у буфері. Цей рядок називається поточним рядком. Він позначається зірочкою, коли друкується поточна команда або блок.

Завдання 11. Роздрукувати (вивести на екран) поточну команду SQL.

Виконання

Спочатку ввести та виконати таку команду:

```
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP WHERE SAL < 2500;
```

Потім за допомогою команди LIST отримати вміст буфера SQL.

```
SQL> SELECT EMPNO, ENAME, JOB, SAL
  2   FROM EMP WHERE SAL < 2500;

     EMPNO ENAME      JOB              SAL
---------- ---------- --------- ----------
      7369 SMITH      CLERK            800
      7499 ALLEN      SALESMAN        1600
      7521 WARD       SALESMAN        1250
      7654 MARTIN     SALESMAN        1250
      7782 CLARK      MANAGER         2450
      7844 TURNER     SALESMAN        1500
      7900 JAMES      CLERK            950
      7934 MILLER     CLERK           1300
8 rows selected.

SQL> LIST
  1  SELECT EMPNO, ENAME, JOB, SAL
  2*  FROM EMP WHERE SAL < 2500
```

Крапка з комою, яка була введена наприкінці команди SELECT, не друкується. Крапка з комою необхідна при введенні команди, але вона не зберігається в самому буфері SQL. Це робить редагування більш зручним, тому що є можливість додавати нові рядки у кінець буфера без видалення крапки з комою.

8.2. Редагування поточного рядка

Команда SQL*Plus CHANGE дозволяє редагувати поточний рядок. Певні дії визначають, який рядок є поточним:

- LIST даного рядка робить його поточним;
- при виконанні або друкуванні команди з буфера останній рядок команди стає поточним (проте використання символу "/" для запуску команди не впливає на положення поточного рядка);

• якщо отримано повідомлення про помилку, рядок з помилкою автоматично стає поточним.

Завдання 12. Ввести наступну команду зі спеціально зробленою помилкою у першому рядку та виконати її.

```
SELECT DPTNO, ENAME, SAL
FROM EMP
WHERE DEPTNO=10;
```

Результат виконання буде таким

```
SQL> SELECT DPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO=10;
SELECT DPTNO, ENAME, SAL
       *
ERROR at line 1:
ORA-00904: "DPTNO": invalid identifier
```

Повідомлення про помилку вказує на неправильне ім'я стовпця у першому рядку запиту. Зірочка показує місце помилки – неправильно введений стовпець DPTNO.

Замість повторного введення команди можна виправити помилку, редагуючи команду в буфері. Рядок, що містить помилку, стає поточним рядком. Використовуючи команду CHANGE, можна виправити помилку. Ця команда складається з трьох частин, розділених похилою рискою або іншим не алфавітно-цифровим символом:

- *слово CHANGE;*
- *послідовність символів, які треба змінити;*
- *послідовність символів, на які треба замінити.*

Команда CHANGE знаходить перше входження послідовності символів у рядку, який необхідно змінити, змінює її на нову послідовність символів.

Якщо є бажання ввести рядок знову, то не слід використовувати команду CHANGE. Наберіть рядок знову, вводячи номер рядка і, через пробіл, новий текст і натисніть [Enter].

Виконання

Для заміни DPTNO на DEPTNO командою CHANGE введіть таку команду:

CHANGE /DPTNO/DEPTNO

Виправлений рядок з'явиться на екрані:

```
SQL> CHANGE /DPTNO/DEPTNO
  1* SELECT DEPTNO, ENAME, SAL
```

Після виправлення можна використати команду RUN:

```
SQL> RUN
  1  SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3* WHERE DEPTNO=10

    DEPTNO ENAME                SAL
---------- ---------- ----------
        10 CLARK              2450
        10 KING               5000
        10 MILLER             1300
```

8.3. Додавання нового рядка

Щоб вставити новий рядок після поточного, слід використовувати команду INPUT.

Завдання 13. З метою упорядкування результатів вибірки додати 4-ий рядок до останньої команди SQL.

Виконання

Оскільки третій рядок уже є поточним (після виконання команди SELECT), уведіть INPUT (можна скорочувати: I) і натисніть [Enter]. SQL*Plus видасть підказку для введення нового рядка:

SQL> INPUT
4

Введіть новий рядок. Потім натисніть [Enter]. SQL*Plus знову видасть підказку для нового рядка:

```
4  ORDER BY SAL
5
```

Натисніть [Enter] знову, щоб сповістити SQL*Plus про припинення введення нових рядків, потім використайте команду RUN для перевірки та виконання нового запиту.

```
SQL> INPUT
  4  ORDER BY SAL
  5
SQL> RUN
  1  SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO=10
  4* ORDER BY SAL

    DEPTNO ENAME                SAL
---------- ---------- ----------
        10 MILLER             1300
        10 CLARK              2450
        10 KING               5000
```

8.4. Додавання тексту в кінець рядка

Щоб додати текст у кінець рядка у буфері, слід використовувати команду APPEND. Для цього зазвичай спочатку виконують команду LIST (і безпосередньо номер рядка) для рядка, який необхідно змінити. Потім виконується команда APPEND і текст, котрий необхідно додати. Якщо текст починається з пробілу, необхідно відокремити слово APPEND від першого символу тексту двома пробілами (перший пробіл –роздільник; а другий – заноситься до буфера).

Завдання 14. Змінити у буфері SQL поточну команду, додавши додаткову умову на номер відділа та задавши сортування результату за спаданням окладів.

Виконання

Оскільки поточним рядком після виконання попереднього запиту є четвертий (останній) рядок, то у кінець цього рядка додати ключове слово DESC, змінюючи порядок впорядкування результату:

APPEND DESC

Далі необхідно зробити поточним третій рядок і до нього додати додаткову умову відбору (після APPEND ввести два пробіли).

LIST 3
APPEND OR DEPTNO=20

Після модифікації запита слід використати команду RUN для його перевірки та виконання.

```
SQL> APPEND  DESC
  4* ORDER BY SAL DESC
SQL> LIST 3
  3* WHERE DEPTNO=10
SQL> APPEND  OR DEPTNO=20
  3* WHERE DEPTNO=10 OR DEPTNO=20
SQL> RUN
  1  SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO=10 OR DEPTNO=20
  4* ORDER BY SAL DESC

    DEPTNO ENAME                SAL
---------- ---------- ----------
        10 KING                5000
        20 FORD                3000
        20 JONES               2975
        10 CLARK               2450
        10 MILLER              1300
        20 SMITH                800

6 rows selected.
```

8.5. Видалення рядка

Щоб видалити рядок з буфера, слід застосовувати команду DEL. Для цього спочатку необхідний рядок роблять поточним за допомогою команди LIST (і номер рядка), а потім виконують команду DEL.

DEL робить наступний рядок у буфері поточним. У такий спосіб послідовно можна видаляти кілька рядків, спочатку роблячи рядок поточним, а потім вводячи команду DEL.

Завдання 15. Видалити з поточного запиту рядок з умовою і повторно виконати запит.

Виконання

Для знаходження необхідного рядка спочатку виконати команду LIST для роздрукування всього буфера SQL. Потім за допомогою команди LIST <номер рядка> – зробити поточним рядок, який потрібно видалити, і, на останок, командою DEL видалити його з буфера.

Для перевірки результату виконати запит командою RUN:

```
SQL> LIST
  1  SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3  WHERE DEPTNO=10 OR DEPTNO=20
  4* ORDER BY SAL DESC
SQL> LIST 3
  3* WHERE DEPTNO=10 OR DEPTNO=20
SQL> DEL
SQL> RUN
  1  SELECT DEPTNO, ENAME, SAL
  2  FROM EMP
  3* ORDER BY SAL DESC

    DEPTNO ENAME              SAL
---------- ---------- ----------
        10 KING             5000
        20 FORD             3000
        20 JONES            2975
        30 BLAKE            2850
        10 CLARK            2450
        30 ALLEN            1600
        30 TURNER           1500
        10 MILLER           1300
        30 MARTIN           1250
        30 WARD             1250
        30 JAMES             950
        20 SMITH             800
12 rows selected.
```

8.6. Редагування команди системним текстовим редактором

Звичайно редагувати текст команд або скрипт на PL/SQL зручніше текстовим редактором. Текстовий редактор виконує ті ж функції, що й команди редагування SQL*Plus.

Системний текстовий редактор можна запускати без виходу із SQL*Plus за допомогою введення команди EDIT:

SQL> EDIT

При цьому, за замовчуванням, викликається стандартний системний редактор Notepad операційної системи Windows. По закінченні модифікації тексту буфера натиснути комбінацію клавіш Alt + F4 і відповісти "Так" на питання про збереження результатів редагування.

Якщо каталог програми SQL*Plus захищений від запису, спроба зберегти результати редагування може викликати помилку. Одним із рішень у цьому випадку є запуск SQL*Plus з правами адміністратора (Windows 7,8).

Щоб завантажити вміст буфера в інший текстовий редактор, а не в описаний за замовчуванням, слід використати команду SQL*Plus DEFINE для визначення змінної, _EDITOR, у якій міститься ім'я редактора.

Завдання 16. Використавши системний редактор, виправити текст останнього запиту таким чином, щоб отримати впорядковані дані з таблиці DEPT.

Виконання

Ввести команду EDIT і у вікні редактора виправити текст останнього запиту таким чином (рис. 1.77).

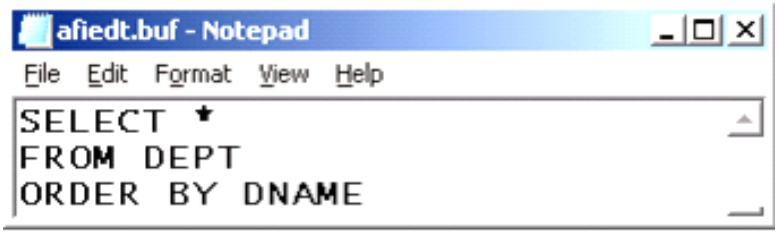


Рис. 1.77. **Запит у системному редакторі**

Крапку з комою у кінці не ставлять. Далі натиснути ALT+F4 і повернутися у SQL*Plus. Ввести команду RUN або "/" для виконання запиту.

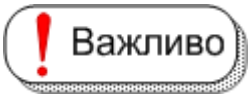
```
SQL> EDIT
Wrote file afiedt.buf
  1  SELECT *
  2  FROM DEPT
  3* ORDER BY DNAME
SQL> /

    DEPTNO DNAME          LOC
---------- -------------- ----------
        10 ACCOUNTING     NEW YORK
        40 OPERATIONS     BOSTON
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
```

9. Створення та використання командних файлів

Завдяки SQL*Plus можна зберігати одну або більше команд у файлі, що називається командним. Після створення командного файла можна вибирати, редагувати та виконувати його.

Створити командний файл, що може зберігати одну й більше команд SQL, блоків PL/SQL, команд SQL*Plus у командних файлах, можна одним із трьох способів:

- ввести команду й зберегти вміст її буфера;
- використати INPUT для введення команд, а потім зберегти вміст буфера;
- використати EDIT для створення тимчасового файла, використовуючи системний редактор.



Оскільки команди SQL*Plus не запам'ятовуються в буфері, для їх збереження необхідно використати один із двох останніх методів, причому, використовуючи команду INPUT, не слід закінчувати крапкою з комою останню команду SQL.

Якщо буде включена в команду крапка з комою, SQL*Plus спробує виконати вміст буфера, а команди SQL*Plus у буфері призведуть до помилок тому, що ***SQL*Plus очікує в буфері тільки команди SQL*** або ***блоки PL/SQL*.**

9.1. Створення командних файлів командою INPUT

Для створення командного файла за допомогою збереження вмісту буфера треба ввести команду SAVE <ім'я файла>: SQL*Plus додає розширення SQL до імені файла, щоб ідентифікувати його як файл запитів SQL. За бажанням, можна задати інше розширення в імені файла через крапку.

Завдання 17. Ввести у буфер команду SQL*Plus, яка форматує значення стовпця SAL, та команду SELECT на впорядковану вибірку інформації з таблиці EMP для співробітників десятого відділу. Введення здійснити за допомогою команди INPUT.

```
COLUMN SAL FORMAT $9,999 HEADING SALARY
SELECT DEPTNO, ENAME, SAL
FROM EMP
WHERE DEPTNO = 10
ORDER BY SAL DESC
```

Зберегти введені рядки у файлі INFO_10 на диску D:\.

Отримати на екрані вміст збереженого файла за допомогою команди операційної системи TYPE.

Виконання

```
SQL> INPUT
  1  COLUMN SAL FORMAT $9,999 HEADING SALARY
  2  SELECT DEPTNO, ENAME, SAL
  3  FROM EMP
  4  WHERE DEPTNO = 10
  5  ORDER BY SAL DESC
  6
SQL> SAVE D:\INFO_10
Created file D:\INFO_10.sql
```

```
SQL> HOST TYPE D:\INFO_10.SQL
COLUMN SAL FORMAT $9,999 HEADING SALARY
SELECT DEPTNO, ENAME, SAL
FROM EMP
WHERE DEPTNO = 10
ORDER BY SAL DESC
/
```

Перед тим як вводити команди до буферу, слід спочатку його очистити командою CLEAR BUFFER.

9.2. Створення командних файлів системним редактором

Можна створювати командний файл за допомогою текстового редактора ОС, що, звісно, є більш доцільним. Для цього необхідно ввести команду EDIT з іменем файла, наприклад: EDIT D:\SALES

Подібно команді SAVE, EDIT також додає до імені файла розширення SQL, якщо не вказано інше розширення.

Наприкінці кожної команди необхідно ставити крапку з комою та рядок із крапкою після кожного блоку PL/SQL у створюваному файлі. (Допускається введення безлічі команд SQL і блоків PL/SQL).

Коли створюється командний файл командою EDIT, можна включати команди SQL*Plus у кінець файла. Цього не можна зробити при створенні командного файла командою SAVE, тому що SAVE додає похилу риску в кінець файла. Ця похила риска буде наказувати SQL виконати командний файл двічі: при зустрічі крапки з комою наприкінці останньої команди SQL (або похилої риски після останнього блоку PL/SQL) та при виявленні похилої риски наприкінці файла.

Завдання 18. За допомогою текстового редактора створити командний файл на ім'я SALES та зберегти його на диску D:\. Файл повинен мати такі рядки.

```
COLUMN ENAME HEADING SALESMAN
COLUMN SAL HEADING SALARY FORMAT $99,999
COLUMN COMM HEADING COMMISION FORMAT $99,990
```

```
SELECT EMPNO, ENAME, SAL, COMM
FROM EMP
WHERE JOB = 'SALESMAN';
```

Виконання

Ввести команду EDIT D:\SALES і на питання редактора "Чи бажаєте ви створити новий файл?" відповісти ствердно, натиснувши кнопку YES.

У вікно редактора ввести рядки, що наведені у завданні.

Натиснути ALT+F4 і підтвердити збереження файла.

Перевірити наявність та вміст файла за допомогою команди TYPE.

```
SQL> HOST TYPE D:\SALES.SQL
COLUMN ENAME HEADING SALESMAN
COLUMN SAL HEADING SALARY FORMAT $99,999
COLUMN COMM HEADING COMMISION FORMAT $99,990
SELECT EMPNO, ENAME, SAL, COMM
FROM EMP
WHERE JOB = 'SALESMAN';
```

9.3. Розміщення коментарів у командних файлах

Як указувалось у п. 1.4.4, коментарі у командні файли можна вводити трьома способами:

- використовуючи команду SQL*Plus REMARK$
- використовуючи обмежники коментарю SQL, /*...*/$
- використовуючи коментарі PL/SQL "--"/

Завдання 19. Ввести коментарі всіх трьох видів у файл D:\SALES.SQL, що був створений у попередньому завданні.

Виконання

Завантажити файл у текстовий редактор командою EDIT D:\SALES.

Додати до файла коментарі в такий спосіб

```
REMARK Month commision report
/* Set output format
   for three columns */
COLUMN ENAME HEADING SALESMAN
COLUMN SAL HEADING SALARY FORMAT $99,999
COLUMN COMM HEADING COMMISION FORMAT $99,990
```

```
SELECT EMPNO, ENAME, SAL, COMM  --List of columns
FROM EMP
WHERE JOB = 'SALESMAN' ;
```

Вийти з редактора та зберегти зміни. Перевірити вміст файла командою HOST TYPE D:\SALES.SQL, аналогічно до завдання 18.

9.4. Завантаження командного файла у буфер

Виклик командного файла в буфер можна здійснити за допомогою команди SQL*Plus GET <ім'я_файла>. Після цього його можна роздрукувати командою LIST або перейти до редагування, використовуючи команду EDIT.

Завдання 20. Очистити буфер SQL та завантажити до нього командою GET файл D:\SALES.SQL.

Виконання

```
SQL> CLEAR BUFFER
buffer cleared
SQL> GET D:\SALES
  1  REMARK Month commision report
  2  /* Set output format
  3     for three columns */
  4  COLUMN ENAME HEADING SALESMAN
  5  COLUMN SAL HEADING SALARY FORMAT $99,999
  6  COLUMN COMM HEADING COMMISION FORMAT $99,990
  7  SELECT EMPNO, ENAME, SAL, COMM --List of columns
  8  FROM EMP
  9* WHERE JOB = 'SALESMAN';
```

Якщо завантажений файл містить команди SQL*Plus, його не можна виконати командою RUN.

9.5. Виконання командного файла

Команда START (або @) вибирає командний файл і виконує команди, які в ньому містяться. Команду START можна використовувати для

виконання командного файла, що містить команди SQL*Plus, команди SQL, блоки PL/SQL. За командою START вказується ім'я файла:

START ім'я_файла

Якщо файл має розширення SQL, його можна не вказувати в імені файла.

Завдання 21. Виконати командний файл D:\SALES.

Виконання

Для запуску на виконання команд, збережених у файлі SALES.SQL, ввести команду START D:\SALES:

```
SQL> START D:\SALES

     EMPNO SALESMAN     SALARY COMMISION
---------- ---------- -------- ---------
      7499 ALLEN        $1,600      $300
      7521 WARD         $1,250      $500
      7654 MARTIN       $1,250    $1,400
      7844 TURNER       $1,500        $0
```

Завдання 22. Подивитися команди, що вибирає SQL*Plus під час виконання командного файла.

Виконання

Для вирішення поставленого завдання слід встановити змінну ECHO команди SET у стан ON. Змінна ECHO управляє виведенням на екран команд з командного файла, який запускається на виконання командою START або за командою '@'. Установка змінної ECHO у стан OFF забороняє виведення команд.

```
SQL> SET ECHO ON
SQL> @ D:\SALES
SQL> REMARK Month commision report
SQL> /* Set output format
SQL>    for three columns */
SQL> COLUMN ENAME HEADING SALESMAN
SQL> COLUMN SAL HEADING SALARY FORMAT $99,999
SQL> COLUMN COMM HEADING COMMISION FORMAT $99,990
```

```
SQL> SELECT EMPNO, ENAME, SAL, COMM -- List of columns
  2  FROM EMP
  3  WHERE JOB = 'SALESMAN';

     EMPNO SALESMAN     SALARY COMMISION
---------- ---------- -------- ---------
      7499 ALLEN         $1,600      $300
      7521 WARD          $1,250      $500
      7654 MARTIN        $1,250    $1,400
      7844 TURNER        $1,500        $0

SQL> SET ECHO OFF
```

Команди START і '@' залишають у буфері останню команду SQL або блок PL/SQL.

9.6. Вкладеність командних файлів

Щоб виконати кілька командних файлів поспіль, спочатку необхідно створити командний файл, що містить кілька команд START із зазначенням імен файлів, що запускаються. Потім виконується командний файл, який складається з команд START.

Завдання 23. Створити три командних файли на ім'я SEL_EMP, SEL_DEPT та SEL_SALGRADE, кожен з яких виводить певні дані з таблиць EMP, DEPT та SALGRADE відповідно. Запустити на виконання створені командні файли з четвертого, що має ім'я SEL_TOTAL.

Виконання

1. Ввести команду EDIT D:\SEL_EMP та ввести такі рядки:

```
COLUMN SAL FORMAT $99,999.99 HEADING SALARY
SELECT EMPNO, ENAME, JOB, SAL
FROM EMP WHERE DEPTNO<30
ORDER BY SAL;
```

Зберегти зроблені зміни у файлі.

2. Ввести команду EDIT D:\SEL_DEPT та ввести такі рядки:

```
COLUMN DNAME FORMAT A5 WRAP
SELECT * FROM DEPT ORDER BY DNAME;
```

Зберегти зроблені зміни у файлі.

3. Ввести команду EDIT D:\SEL_SALGRADE та ввести такий рядок:

SELECT * FROM SALGRADE;

Зберегти зроблені зміни у файлі.

4. Ввести команду EDIT D:\SEL_TOTAL та ввести такі рядки:

START D:\SEL_EMP
START D:\SEL_DEPT
START D:\SEL_SALGRADE

Зберегти зроблені зміни у файлі.

5. Запустити на виконання командний файл D:\SEL_TOTAL.

```
SQL> EDIT D:\SEL_EMP
SQL> EDIT D:\SEL_DEPT
SQL> EDIT D:\SEL_SALGRADE
SQL> EDIT D:\SEL_TOTAL
SQL> START D:\SEL_TOTAL
     EMPNO ENAME      JOB              SALARY
---------- ---------- --------- -----------
      7369 SMITH      CLERK          $800.00
      7934 MILLER     CLERK        $1,300.00
      7782 CLARK      MANAGER      $2,450.00
      7566 JONES      MANAGER      $2,975.00
      7902 FORD       ANALYST      $3,000.00
      7839 KING       PRESIDENT    $5,000.00

6 rows selected.

    DEPTNO DNAME LOC
---------- ----- -------------
        10 ACCOU NEW YORK
           NTING
        40 OPERA BOSTON
           TIONS
        20 RESEA DALLAS
           RCH
        30 SALES CHICAGO
```

```
     GRADE      LOSAL      HISAL
---------- ---------- ----------
         1        700       1200
         2       1201       1400
         3       1401       2000
         4       2001       3000
         5       3001       9999
```

10. Написання діалогових програм

Утиліта SQL*Plus має спеціальні можливості, що дозволяють писати командні файли, які взаємодіють з кінцевим користувачем. Для цього зазвичай використовують так звані змінні користувача.

10.1. Визначення змінних користувача

Кожний користувач може визначати змінні, що називаються змінними користувача, для повторного використання в одному командному файлі, використовуючи команду SQL*Plus DEFINE. Такі змінні можна використовувати і в заголовках звітів.

Завдання 24. Визначити змінну користувача EMPLOEE і присвоїти їй значення "SMITH".

Виконання

Введіть команду DEFINE EMPLOYEE = SMITH та натисніть [Enter.]

Для перевірки опису змінної введіть DEFINE <ім'я змінної>:

```
SQL> DEFINE EMPLOYEE = SMITH
SQL> DEFINE EMPLOYEE
DEFINE EMPLOYEE         = "SMITH" (CHAR)
```

Щоб роздрукувати опис всіх змінних користувача, потрібно ввести DEFINE після командної підказки, не вказуючи ім'я змінної. У цьому випадку на екрані з'являться не тільки змінні користувача, а й ще перелік системних змінних. Змінні, явно визначені командою DEFINE, отримують значення CHAR. Для визначення змінної корис-

тувача з типом NUMBER слід використовувати команду ACCEPT (завдання 31).

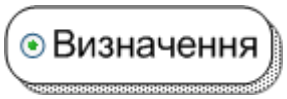
Для видалення змінної треба використовувати команду SQL*Plus UNDEFINE зі вказівкою імені змінної.

```
SQL> DEFINE
DEFINE _DATE            = "10-OCT-14" (CHAR)
DEFINE _CONNECT_IDENTIFIER = "192.168.0.170" (CHAR)
DEFINE _USER            = "STUD06" (CHAR)
DEFINE _PRIVILEGE       = "" (CHAR)
DEFINE _SQLPLUS_RELEASE = "1002000100" (CHAR)
DEFINE _EDITOR          = "Notepad" (CHAR)
DEFINE _O_VERSION       = "Oracle Database 11g Express…
DEFINE _O_RELEASE       = "1002000100" (CHAR)
DEFINE _RC              = "1" (CHAR)
DEFINE EMPLOYEE         = "SMITH" (CHAR)
SQL> UNDEFINE EMPLOYEE
```

10.2. Використання змінних підстановки

Якщо бажано написати запит, подібний до запиту завдання 18, для друкування службовців з різними професіями, а не тільки продавців (SALESMAN), це можна зробити, змінюючи відповідне значення в умові WHERE щоразу, при виконанні команди.

Використовуючи змінну підстановки замість значення SALESMAN в умові WHERE, можна отримати ті ж результати, які б були отримані у результаті вказівки значення у самій команді.



Підстановочна змінна – це ім'я змінної користувача з одним або двома амперсандами (&), що записуються перед іменем змінної.

Коли SQL*Plus зустрічає підстановочну змінну у команді, то команда виконується таким чином, наче вона містить значення підстановочної змінної, а не саму змінну.

Завдання 25. Визначити змінні користувача SORTCOL та MYTABLE зі значенням "ENAME" та "EMP", відповідно, та використати їх у команді SELECT.

Виконання

1. Створити змінні користувача командами

```
DEFINE SORTCOL=ENAME
DEFINE MYTABLE=EMP
```

2. Ввести та запустити на виконання такий запит

```
SELECT &SORTCOL, DEPTNO
FROM &MYTABLE WHERE DEPTNO <30
ORDER BY &SORTCOL;
```

3. Отримати результат

```
SQL> DEFINE SORTCOL=ENAME
SQL> DEFINE MYTABLE=EMP
SQL> SELECT &SORTCOL, DEPTNO
  2  FROM &MYTABLE WHERE DEPTNO <30
  3  ORDER BY &SORTCOL;
old   1: SELECT &SORTCOL, DEPTNO
new   1: SELECT ENAME, DEPTNO
old   2: FROM &MYTABLE WHERE DEPTNO <30
new   2: FROM EMP WHERE DEPTNO <30
old   3: ORDER BY &SORTCOL
new   3: ORDER BY ENAME

ENAME           DEPTNO
---------- ----------
CLARK               10
FORD                20
JONES               20
KING                10
MILLER              10
SMITH               20
6 rows selected.
```

Використовувати змінні підстановки дозволяється у будь-якому місці в командах SQL*Plus і SQL, крім першого слова, що вводиться після командної підказки.

Коли SQL*Plus зустрічає невизначену змінну підстановки у команді, він просить ввести для неї значення. Можна ввести кілька рядків після підказки, навіть рядки, що містять пробіли й пунктуації. Якщо команда

SQL, що містить посилання, повинна мати лапки навколо змінної, а вони там відсутні, користувач повинен включити їх при введенні.

Після введення значення SQL*Plus друкує рядок, що містить підстановочну змінну двічі: до підстановки з позначкою ***old***, після підстановки з позначкою ***new***.

Друкування скасувати за допомогою установки змінної VERIFY (OFF) команди SET.

Завдання 26. Створити командний файл із ім'ям GROUP_AVG для обчислення середнього та максимального значень деякого числового стовпця по певній групі. Стовпець, що вказує на групу, слід визначити як змінну.

Виконання

Ввести такі команди для створення командного файла:

```
CLEAR BUFFER
INPUT
SELECT &GROUP_COL,
AVG(&NUMBER_COL) AVERAGE
MAX(&NUMBER_COL) MAXIMUM
FROM   &TABLE
GROUP BY &GROUP_COL

SAVE D:\GROUP_AVG
```

У результаті отримано:

```
SQL> CLEAR BUFFER
buffer cleared
SQL> INPUT
  1  SELECT &GROUP_COL,
  2  AVG(&NUMBER_COL) AVERAGE,
  3  MAX(&NUMBER_COL) MAXIMUM
  4  FROM   &TABLE
  5  GROUP BY &GROUP_COL
  6
SQL> SAVE D:\GROUP_AVG
Created file D:\GROUP_AVG.sql
```

Запустити на виконання створений файл D:\GROUP_AVG і дати відповіді, як показано нижче:

```
SQL> @ D:\GROUP_AVG
Enter value for group_col: JOB
old   1: SELECT &GROUP_COL,
new   1: SELECT JOB,
Enter value for number_col: SAL
old   2: AVG(&NUMBER_COL) AVERAGE,
new   2: AVG(SAL) AVERAGE,
Enter value for number_col: SAL
old   3: MAX(&NUMBER_COL) MAXIMUM
new   3: MAX(SAL) MAXIMUM
Enter value for table: EMP
old   4: FROM   &TABLE
new   4: FROM   EMP
Enter value for group_col: JOB
old   5: GROUP BY &GROUP_COL
new   5: GROUP BY JOB

JOB           AVERAGE    MAXIMUM
--------- ---------- ----------
CLERK     1016.66667       1300
SALESMAN        1400       1600
PRESIDENT       5000       5000
MANAGER   2758.33333       2975
ANALYST         3000       3000
```

Якщо необхідно додати символи безпосередньо після підстановочної змінної, слід використовувати крапку для розмежування змінної та символів. Наприклад:

```
SQL> SELECT * FROM EMP WHERE EMPNO='&NUM.03';
Enter value for NUM:  75
Буде інтерпретуватися так:
SQL> SELECT * FROM EMP WHERE EMPNO='7503';
```

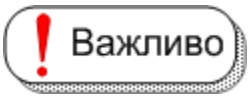
10.3. Усунення необов'язкової підказки для введення значення

Які недоліки, з точки зору здорового глузду, були при виконанні попереднього завдання?

Основний недолік заключався у тому, що одна і та ж змінна визначалась у діалоговому режимі декілька разів, тобто стільки, скільки вона використовувалась у запиті. А це, по-перше, потребує вводити більше даних, а по-друге, при повторному введенні значення змінної можна помилитися, що призведе до отримання неправильного результату, або до невиконання запиту.

Можна уникнути повторного введення імені групового та числового стовпців за допомогою приєднання другого амперсанда перед GROUP_COL і NUMBER_COL у файлі D:\GROUP_AVG. У цьому випадку SQL*Plus автоматично визначить підстановочну змінну з двома амперсандами, але не зробить цього для змінної з одним амперсандом.

Таким чином, коли SQL*Plus зустрічає підстановочну змінну протягом сеансу більше одного раза, він використовує певне значення для змінних із двома амперсандами і попросить введення значення для змінних з одним амперсандом. Ця можливість також корисна, якщо є необхідність виконувати файл для різних таблиць із тими ж значеннями GROUP_COL і NUMBER_COL.



Фактично, якщо перед ім'ям змінної поставити два амперсанда, то після введення значення для неї буде неявно виконана команда DEFINE і повторне введення значення не буде потрібно.

Завдання 27. Поліпшити командний файл D:\GROUP_AVG за допомогою подвійних амперсандів, а потім виконати файл.

Виконання

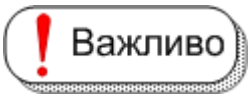
Відредагувати командний файл D:\GROUP_AVG, ввівши команду: EDIT D:\GROUP_AVG та змінивши перед змінними GROUP_COL та NUMBER_COL один амперсанд на два:

```
SELECT &&GROUP_COL,
AVG(&&NUMBER_COL) AVERAGE,
MAX(&&NUMBER_COL) MAXIMUM
FROM   &TABLE
GROUP BY &&GROUP_COL
```

і виконати командний файл D:\GROUP_AVG.

```
SQL> START D:\GROUP_AVG
Enter value for group_col: JOB
old   1: SELECT &&GROUP_COL,
new   1: SELECT JOB,
Enter value for number_col: SAL
old   2: AVG(&&NUMBER_COL) AVERAGE,
new   2: AVG(SAL) AVERAGE,
old   3: MAX(&&NUMBER_COL) MAXIMUM
new   3: MAX(SAL) MAXIMUM
Enter value for table: EMP
old   4: FROM   &TABLE
new   4: FROM   EMP
old   5: GROUP BY &&GROUP_COL
new   5: GROUP BY JOB

JOB           AVERAGE    MAXIMUM
--------- ---------- ----------
CLERK     1016.66667       1300
SALESMAN        1400       1600
PRESIDENT       5000       5000
MANAGER   2758.33333       2975
ANALYST         3000       3000
```

Щоб SQL*Plus не виводив старе та нове значення змінних, потрібно ввести команду SET VERIFY OFF. А для повернення у попередній режим – SET VERIFY ON.



Не можна використовувати підстановочні змінні у командах редагування буфера (APPEND, CHANGE, DEL, INPUT) і в інших командах, де підстановка нічого не означає (таких, як REMARK і TIMING). Команди редагування буфера APPEND, CHANGE, DEL, INPUT обробляють текст, що починається з "&" і "&&" буквально, тобто як будь-який інший текстовий рядок.

Слід також звернути увагу на системні змінні, обумовлені командою SET, що впливають на змінні підстановки. Вони вказані у п. 1.3.11 теоретичної частини.

10.4. Передача параметрів у команді START

Обійти підказку для введення відповідних значень для підстановочних змінних можна використовуючи параметри, що передаються командному файлу в команді START. Це можна зробити, вказавши амперсанд із наступним числом у командному файлі замість підстановочної змінної. Щоразу, під час виконання командного файла, кожне входження &1 у файлі заміняється на перше значення (називається аргументом або параметром) після START <ім'я_файла>. Аналогічно, текст &2 заміняється на другий аргумент, і т. д.

Завдання 28. Зробити копію командного файла D:\GROUP_AVG і замінити у ньому підстановочні змінні на параметри. Виконати новий командний файл, задаючи параметри у команді START.

Виконання.

1. Зробити копію файла D:\GROUP_AVG і на диску D:\ і дати йому ім'я PGROUP_AVG.

```
HOST  COPY  D:\GROUP_AVG.SQL    D:\PGROUP_AVG.SQL
```

2. Відредагувати файл D:\PGROUP_AVG.SQL, замінивши в ньому підстановочні змінні на параметри. Зберегти зміни.

```
EDIT  D:\PGROUP_AVG.SQL
```

Файл буде мати такий вигляд:

```
SELECT &1,
AVG(&2) AVERAGE,
MAX(&2) MAXIMUM
FROM   &3
GROUP BY &1;
```

3. Запустити файл на виконання командою START і передати йому під час запуску відповідні параметри.

```
START D:\PGROUP_AVG   JOB  SAL  EMP
```

4. Результат виконання виглядає так:

```
SQL> HOST  COPY  D:\GROUP_AVG.SQL    D:\PGROUP_AVG.SQL
        1 file(s) copied.
SQL> EDIT  D:\PGROUP_AVG.SQL
SQL> START D:\PGROUP_AVG   JOB  SAL  EMP
old   1: SELECT &1,
new   1: SELECT JOB,
old   2: AVG(&2) AVERAGE,
new   2: AVG(SAL) AVERAGE,
old   3: MAX(&2) MAXIMUM
new   3: MAX(SAL) MAXIMUM
old   4: FROM   &3
new   4: FROM   EMP
old   5: GROUP BY &1
new   5: GROUP BY JOB

JOB           AVERAGE    MAXIMUM
--------- ---------- ----------
CLERK     1016.66667        1300
SALESMAN         1400        1600
PRESIDENT        5000        5000
MANAGER   2758.33333        2975
ANALYST          3000        3000
```

Завдання 29. Створити та виконати командний файл для вибору інформації з таблиці EMP, у якому значення умови пошуку різних типів даних задаються у параметрах.

Виконання

1. Створити за допомогою команди EDIT D:\PWHERE_EMP командний файл з такими рядками:

```
SELECT EMPNO, ENAME, DEPTNO, JOB, SAL, HIREDATE
FROM EMP
WHERE JOB='&1' AND SAL>&2 AND HIREDATE > '&3';
```

2. Запустити файл на виконання командою

```
START D:\PWHERE_EMP   MANAGER   1500   01/01/1960
```

```
SQL> START D:\PWHERE_EMP   MANAGER   1500   01/01/1960
old   3: WHERE JOB='&1' AND SAL>&2 AND HIREDATE > '&3'
new   3: WHERE JOB='MANAGER' AND SAL>1500 AND HIREDATE >
'01/01/1960'

EMPNO ENAME     DEPTNO JOB           SAL HIREDATE
------ -------------- -------------- ----------
  7566 JONES          20 MANAGER     2975 02/04/1981
  7698 BLAKE          30 MANAGER     2850 01/05/1981
  7782 CLARK          10 MANAGER     2450 09/06/1981
```

Завдання 30. Зробити модифікацію командного файла з завдання 23 таким чином, щоб у якості вкладених командних файлів виступав тільки один файл, якому у якості параметра передається ім'я таблиці БД.

Виконання

1. Ввести команду EDIT D:\SEL_TABLE та ввести у створюваний файл такий рядок:

SELECT * FROM &1;

2. Ввести команду EDIT D:\SEL_TOTAL_PAR та ввести такі рядки:

START D:\SEL_TABLE &1
START D:\SEL_TABLE &2
START D:\SEL_TABLE &3

3. Запустити на виконання командний файл D:\SEL_TOTAL_PAR таким чином

START D:\SEL_TOTAL_PAR DEPT SALGRADE EMP

```
SQL> START   D:\SEL_TOTAL_PAR   DEPT   SALGRADE   EMP
old   1: SELECT  *  FROM  &1
new   1: SELECT  *  FROM  DEPT

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
```

```
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
old   1: SELECT  *  FROM  &1
new   1: SELECT  *  FROM  SALGRADE
     GRADE      LOSAL      HISAL
---------- ---------- ----------
         1        700       1200
         2       1201       1400
         3       1401       2000
         4       2001       3000
         5       3001       9999

old   1: SELECT  *  FROM  &1
new   1: SELECT  *  FROM  EMP
EMPNO ENAME      JOB          MGR HIREDATE       SAL
----- ---------- ------------- ---------- ------
 7369 SMITH      CLERK       7902 17/12/1980     800
 7499 ALLEN      SALESMAN    7698 20/02/1981    1600
 7521 WARD       SALESMAN    7698 22/02/1981    1250
 7566 JONES      MANAGER     7839 02/04/1981    2975
 7654 MARTIN     SALESMAN    7698 28/09/1981    1250
 7698 BLAKE      MANAGER     7839 01/05/1981    2850
 7782 CLARK      MANAGER     7839 09/06/1981    2450
 7839 KING       PRESIDENT        17/11/1981    5000
 7844 TURNER     SALESMAN    7698 08/09/1981    1500
 7900 JAMES      CLERK       7698 03/12/1981     950
 7902 FORD       ANALYST     7566 03/12/1981    3000
 7934 MILLER     CLERK       7782 23/01/1982    1300

12 rows selected.
```

10.5. Зв'язок з користувачем

Підказка та введення значень для користувача

Три команди SQL*Plus – PROMPT, ACCEPT та PAUSE допомагають спілкуватися з кінцевим користувачем. Ці команди дозволяють посилати повідомлення та приймати введення даних від користувача, включаючи й просте натискання [Enter]. PROMPT і

ACCEPT також можна використовувати для зміни підказки при введенні значень, що автоматично генерує SQL*Plus для підстановочних змінних.

Команда PROMPT просто виводить зазначене користувачем повідомлення на екран. Цю команду бажано використовувати для передачі інструкцій або інформації користувачеві.

ACCEPT видає підказку користувачеві для введення значення й запам'ятовує його в зазначеній змінній користувача. Рекомендується вживати PROMPT разом з ACCEPT, коли підказка містить більше одного рядка.

Завдання 31. Ввести у діалоговому режимі назву звіту, запам'ятати введене значення у змінній REPTITLE, відобразити введене значення перед вибіркою даних з таблиці DEPT.

Виконання

1. Очистити буфер командою CLEAR BUFFER та за допомогою команди INPUT додати у буфер такі рядки (останній рядок порожній):

```
SET LINESIZE 40
PROMPT  Input report title, please  (up to 30 characters)
ACCEPT MYTITLE PROMPT 'Title:  '
TTITLE CENTER REPTITLE SKIP 2
SELECT * FROM DEPT
```

2. Зберегти файл командою SAVE D:\PROMPT_TEST.

3. Запустити на виконання створений командний файл ввівши START D:\PROMPT_TEST

```
SQL> CLEAR BUFFER
buffer cleared
SQL> INPUT
  1  SET LINESIZE 40
  2  PROMPT  Input report title, please  (up to 30
characters)
  3  ACCEPT MYTITLE PROMPT 'Title:  '
  4  TTITLE CENTER REPTITLE SKIP 2
  5  SELECT * FROM DEPT
  6
SQL> SAVE D:\PROMPT_TEST
```

```
Created file D:\PROMPT_TEST.sql
SQL> START D:\PROMPT_TEST
Input report title, please  (up to 30 characters)
Title:  Department  Report

                  REPTITLE

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Перед тим як продовжити, треба відключіти створений верхній колонтитул командою ***TTITLE OFF*** та повернути значення LINESIZE до 120 командою ***SET LINESIZE 120***.

Зміна підказки в процесі введення значення підстановочної змінної

Для зміни підказки в процесі введення значення підстановочної змінної, рекомендується спільно використовувати команди PROMPT та ACCEPT.

Завдання 32. Здійснити спільне використання у одній команді PROMPT та ACCEPT для введення умови на пошук співробітника за його номером.

Виконання

Аналогічно до попереднього завдання занести до буфера командою INPUT та зберегти у файлі D:\PROMPT_ACCEPT_TEST такі рядки, попередньо очистивши буфер.

```
PROMPT Please, enter a valid employee number
PROMPT For example: 7369, 7499, 7521
ACCEPT EMPNUM NUMBER PROMPT 'Emp. number= '
SELECT EMPNO, ENAME, MGR, JOB, SAL
FROM EMP
WHERE EMPNO = &EMPNUM
```

Після збереження виконати командний файл і спочатку ввести на запрошення не число, а деякий текст, щоб виникла помилка. Повторно ввести правильний номер.

```
SQL> CLEAR BUFFER
buffer cleared
SQL> INPUT
  1   PROMPT Please, enter a valid employee number
  2   PROMPT For example: 7369, 7499, 7521
  3   ACCEPT EMPNUM NUMBER PROMPT 'Emp. number= '
  4   SELECT EMPNO, ENAME, MGR, JOB, SAL
  5   FROM EMP
  6   WHERE EMPNO = &EMPNUM
  7
SQL> SAVE D:\PROMPT_ACCEPT_TEST
Created file D:\PROMPT_ACCEPT_TEST.sql
SQL> START D:\PROMPT_ACCEPT_TEST
Please, enter a valid employee number
For example: 7369, 7499, 7521
Emp. number= ABCD
SP2-0425: "ABCD" is not a valid NUMBER
Emp. number= 7844
old   3:  WHERE EMPNO = &EMPNUM
new   3:  WHERE EMPNO =       7844

  EMPNO ENAME           MGR JOB                SAL
------- ---------- ------- --------- ----------
   7844 TURNER         7698 SALESMAN         1500
```

Надсилання повідомлення та приймання [Enter] у відповідь

Якщо є необхідність вивести повідомлення на екран користувача і зажадати, щоб після прочитання повідомлення користувач натиснув [Enter], слід використовувати команду PAUSE.

Завдання 33.

Створити командний файл для перегляду інформації з декількох різних таблиць. Після виведення результату з будь-якої таблиці зробити паузу та чекати на натиснення клавіші [Enter].

Виконання

Командою EDIT D:\TEST_PAUSE створити командний файл з такими рядками:

```
PROMPT This is example file for show a using PAUSE
PROMPT Please, press Enter after each table
PAUSE Press ENTER to continue
SELECT * FROM SALGRADE;
PAUSE Press  ENTER again
SELECT * FROM DEPT;
PAUSE That's all  :)
```

Після виконання отримано такий результат:

```
SQL> START D:\TEST_PAUSE
This is example file for show a using PAUSE
Please, press Enter after each table
Press ENTER to continue

     GRADE      LOSAL      HISAL
---------- ---------- ----------
         1        700       1200
         2       1201       1400
         3       1401       2000
         4       2001       3000
         5       3001       9999

Press ENTER again

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON

That's all :)
```

11. Форматування звітів

Як було зазначено в п. 1.6 теоретичної частини, результати запитів у SQL*Plus можна додатково форматувати. Для цього зазвичай використовуються команди COLUMN, BREAK, COMPUTE, BTITLE та TTITLE, REPHEADER та REPFOOTER.

Завдання 34. Створити звіт за даними таблиць EMP та DEPT про загальну заробітну плату по відділах та загалом по організації.

Виконання

1. Аналогічно попереднім завданням створити командний файл на ім'я D:\REPORT_TEST та занести до нього такі рядки:

```
TTITLE LEFT '   DEPARTMENT SALARY' SKIP 2
BTITLE CENTER 'Page No ' FORMAT 9 SQL.PNO

SET LINESIZE 60
SET PAGESIZE 18

COLUMN DNAME FORMAT A20
BREAK ON DNAME SKIP 1
COMPUTE SUM LABEL 'Total dept salary:' OF SAL ON DNAME

COMPUTE SUM LABEL 'Total salary:' OF SAL ON REPORT
BREAK ON DNAME SKIP 1 ON REPORT

SELECT DNAME, DEPT.DEPTNO, ENAME, SAL FROM DEPT INNER JOIN EMP
ON DEPT.DEPTNO = EMP.DEPTNO
ORDER BY DNAME;
```

2. Виконати створений файл командою START.

```
SQL> @ D:\REPORT_TEST
```

```
   DEPARTMENT SALARY

DNAME                DEPTNO ENAME                SAL
-------------------- ---------- ---------- ----------
ACCOUNTING               10 KING                5000
                         10 CLARK               2450
                         10 MILLER              1300
**********                                      ----------
Total dept salary:                              8750

RESEARCH                 20 FORD                3000
                         20 SMITH                800
                         20 JONES               2975
**********                                      ----------
Total dept salary:                              6775
                        Page No  1

   DEPARTMENT SALARY

DNAME                DEPTNO ENAME                SAL
-------------------- ---------- ---------- ----------
SALES                    30 JAMES                950
                         30 TURNER              1500
                         30 MARTIN              1250
                         30 WARD                1250
                         30 ALLEN               1600
                         30 BLAKE               2850
**********                                      ----------
Total dept salary:                              9400

                                                ----------
Total salary:                                  24925
                        Page No  2
```

Поясніть, результати, отримані при виконанні командного файла і формуванні звіту.

Завдання для самостійного виконання

1. Створіть скрипт для формування таблиці у базі даних за прикладом:

```
CREATE TABLE GOODS_NXX (
GOODS_ID NUMBER(4) NOT NULL PRIMARY KEY,
NAME VARCHAR2(20),
VOLUME NUMBER(6,2),
STORAGE_NEED VARCHAR2(20), PRICE NUMBER(6,2));
```

де **_NXX – № групи + №студента у списку (дві цифри)**

2. Перетворіть скрипт таким чином, щоб ім'я таблиці можна було передавати як параметр. Виконайте скрипт. Перевірте результат.

3. Створіть скрипт на заповнення даними створеної таблиці (не менш ніж десять рядків).

4. Створіть скрипт, який вводить дані у таблицю у діалоговому режимі.

5. Створіть вкладені скрипти на вибірку даних з таблиць бази за конкретними умовами. Умови передайте як параметри до головного командного файла.

6. За даними тестової бази сформуйте звіт, виконавши самостійно постановку завдання.

7. Підключіться до бази даних за допомогою програми Oracle SQL Developer, розкрийте дерево об'єктів бази даних та ознайомтесь з переліком та вмістом існуючих таблиць.

Висновки

1. На світовому ринку корпоративних систем керування базами даних домінує традиційна трійка продуктів: IBM DB2, Microsoft SQL Server і Oracle, які охоплюють понад 80% продажів на світовому ринку СКБД.

2. Під час встановлення СКБД Oracle на комп'ютер дуже важливим є екран Specify Database Passwords, оскільки в ньому заноситься

пароль адміністратора бази даних. Його потрібно запам'ятати, бо інакше буде неможливо розпочати роботу з системою.

3. Утиліта SQL*Plus дозволяє виконувати команди SQL і блоки PL/SQL, а також вирішувати ряд інших завдань. За допомогою SQL*Plus можна:

вводити, редагувати, запам'ятовувати, завантажувати та виконувати команди SQL і блоки PL/SQL;

виконувати налагодження SQL-операторів, блоків PL/SQL, збережених процедур і функцій перед їх використанням в розроблюваних додатках;

форматувати, створювати, зберігати, друкувати та публікувати у Web результати виконання запитів (звіти);

отримувати опис (імена та типи стовпців) будь-якої таблиці та подання;

звертатися до віддалених баз даних і копіювати з них дані;

посилати та приймати повідомлення від кінцевих користувачів;

виконувати вбудовані команди;

адмініструвати базу даних.

4. Під час роботи з SQL*Plus використовуються такі базові поняття:

команда – команда SQL*Plus або оператор SQL Oracle;

блок PL/SQL – група взаємопов'язаних операторів PL/SQL, що оформлена у вигляді анонімного блоку;

таблиця – базова одиниця зберігання даних в Oracle;

запит SQL – оператор мови SQL, що створює нові об'єкти у базі даних або вибирає, модифікує чи видаляє інформацію з існуючих об'єктів бази даних;

результати запиту – дані, що повернуті запитом (для вибірки), чи зміна стану БД – для інших команд.

звіт – результати запиту на вибірку даних, що відформатовані за допомогою команд SQL*Plus.

5. У середовищі SQL*Plus можна задавати команди мови структурованих запитів SQL для обробки на сервері баз даних, які легко читати та редагувати.

6. Для використання програм мовою PL/SQL (їх часто називають блоками) потрібно дотримуватися таких правил:

блоки PL/SQL починаються з ключових слів DECLARE, BEGIN або імені блоку SQL*Plus;

на відміну від команд SQL, у блоках PL/SQL крапка з комою або порожній рядок не закінчують і не виконують блок;

блок PL/SQL завершується крапкою (.) на новому рядку.

7. Системні змінні керують великою кількістю параметрів всередині SQL*Plus, включаючи ширину стовпців за замовчуванням для виведення на екран, довжину та ширину сторінки для виведення результатів, автоматичним збереженням змін у базі даних тощо. Їх значення встановлюються за допомогою команди SET.

8. Змінна підстановки – це ім'я змінної користувача з одним або двома попередніми амперсандами (&). Коли SQL*Plus зустрічає змінну підстановки у команді, він виконує команду так, ніби вона містить значення цієї змінної, а не саму змінну.

9. Якщо та чи інша група команд SQL, блоків PL/SQL, команд SQL*Plus виконується часто, їх можна не вводити в інтерактивному режимі, а запам'ятати у командних файлах і потім виконувати багаторазово.

10. Команди PROMPT, ACCEPT, PAUSE допомагають спілкуватися з користувачем утиліти SQL*Plus. Ці команди дозволяють надсилати повідомлення та приймати введення даних від користувача, включаючи просте натискання [Enter].

11. Можна обійти підказку для введення відповідних значень для змінних підстановки за допомогою передачі значень через параметри командного файла під час запуску його на виконання командою START. Для цього потрібно помістити амперсанд, за яким іде число, у командному файлі замість підстановочної змінної.

12. Команда COPY дозволяє виконувати копіювання даних між БД і між таблицями однієї БД.

13. Результати запитів у SQL*Plus можна додатково форматувати. Утиліта SQL*Plus дозволяє керувати форматом стовпців, кількістю рядків на сторінці, додатковими порожніми рядками, заголовками сторінок тощо.

14. Oracle SQL Developer – це програмний засіб з графічним інтерфейсом для розробки програм мовами SQL та PL/SQL і орієнтований на застосування у сёередовищі Oracle Database. Він підтримує практично усі команди SQL*Plus, окрім деяких, наприклад, пов'язаних з форматуванням звітів.

Тема 2. Особливості мови DDL і DML у СКБД Oracle. Використання стандартних функцій у СКБД Oracle

Мета розділу – ознайомитися з призначенням та прикладами використання команд мови SQL Oracle для створення, модифікації та вибірки інформації з бази даних, проілюструвати використання розширення мовних конструкції SQL для здійснення агрегації у сховищах даних, а також використання стандартних функцій СКБД Oracle для розв'язання практичних завдань.

Основні питання

Команди створення та модифікації об'єктів у БД – описується формат та наводяться приклади створення основних об'єктів у базі даних: таблиць, індексів, послідовностей, синонімів тощо.

Вибірка інформації з БД – розглядаються основні мовні конструкції команди SELECT на вибірку даних з однієї або декількох таблиць. Наводяться приклади формування умов пошуку та впорядкування вислідного набору даних.

Використання агрегатних функції – пояснюється сенс агрегатних функції та на прикладах ілюструється їх використання.

SQL та сховища даних – розглядаються спеціальні конструкції мови SQL, призначені для формування OLAP – кубів та сховищ даних.

Стандартні функції СКБД Oracle – подається класифікація та наводяться приклади використання функції СКБД Oracle для вирішення практичних завдань та форматування результатів вибірки.

2.1. Мова SQL та її Oracle-діалект

Невід'ємна та найважливіша частина будь-якої системи, яка застосовує бази даних, – мовні засоби, що забезпечують можливість доступу і дій над даними, визначення їх структур, способів використання та інтерпретації. Стандартом такої мови для усіх реляційних систем керування базами даних є мова SQL (Structured Query Language – мова структурованих запитів). Вона є універсальною комп'ютерною мовою для створення, модифікації та керування даними у реляційних базах даних. SQL використовує усі переваги реляційної моделі, (зокрема, її математичного

апарату – реляційної алгебри та реляційного числення), використовуючи при цьому досить невелику кількість команд та відносно простий синтаксис.

Спочатку SQL створювалася як проста мова запитів до реляційної бази даних [11], яка близька до природної мови. При цьому часто використовувався термін very high level language (VHL-language) – мова високого рівня, яка має можливість формулювати нескладними мовними конструкціями побажання виконати якісь операції з базою даних. Тобто за своєю суттністю SQL є непроцедурною мовою. Термін "***непроцедурна***" означає, що цією мовою можна сформулювати завдання на обробку даних, але алгоритм реалізації безпосередньо не задається.

Зазвичай, усі команди SQL підрозділяють на три основні групи [24]:

1. **Data Definition Language (DDL)** – мова визначення даних, призначена для створення, модифікації та видалення таблиць і усієї бази даних.

2. **Data Manipulation Language (DML)** – мова маніпулювання даними, призначена для виконання основних операцій під час роботи з даними.

3. **Data Control Language (DCL)** – мова керування даними, призначена для забезпечення захисту бази даних; застосовується для здійснення адміністративних функцій, що надають або відміняють право (привілей) використовувати базу даних, таблиці у базі даних, а також виконувати ті або інші оператори SQL.

Крім того, часто виділяють ще дві групи команд:

4. **Transaction Control Language (TCL)** – спеціальні оператори, які застосовуються для керування модифікацією бази даних, що здійснюють оператори DML (команди керування транзакціями).

5. **Cursor Control Language (CCL)** – оператори CCL використовуються для визначення курсору, підготовки SQL-речення для виконання, а також для деяких інших операторів [57]. Ці групи команд найчастіше використовують у процедурних розширеннях мови SQL, тому їх опис виходить за межі цього посібника.

Незважаючи на те, що для мови SQL існують певні стандарти [24; 42], кожна комерційна СКБД має певні особливості (і Oracle у цьому випадку не виключення), які безпосередньо впливають як на функціональні можливості системи, так і на реалізацію тих чи інших запитів до бази даних.

Стосовно Oracle можна казати про такі особливості, як: "рідні" типи даних; стандартні функції системи; використання підзапитів для оновлення у команді UPDATE чи скорочена форма команди DELETE; особливості використання зовнішніх з'єднань; деревоподібні запити тощо. У процесі викладення матеріалу ці особливості будуть поступово розглянуті.

Приступаючи до роботи з базою даних, на першому етапі зазвичай створюють об'єкти бази даних з використанням команд DDL мови SQL. DDL – це певна підмножина команд мови SQL, призначена саме для створення, модифікації та видалення об'єктів БД. Основними командами DDL є команди: CREATE (створити), ALTER (змінити), DROP (видалити).

За допомогою цих команд користувачі можуть створювати, видаляти та модифікувати такі об'єкти БД, як: зв'язок з базою даних (DATABASE LINK); індекси (INDEX); кластери (CLUSTER); подання (VIEW); послідовності (SEQUENCE); синоніми (SYNONYM) та, звісно, таблиці (TABLE), які є основними об'єктами бази даних.

Ілюстрація використання команд DDL SQL Oracle, (а у подальшому і команд DML) буде проводитися на основі розширеної та дещо модифікованої бази даних SCOTT/TIGER, яка використовувалась у попередньому розділі посібника. Схема навчальної БД наведена на рис. 2.1.

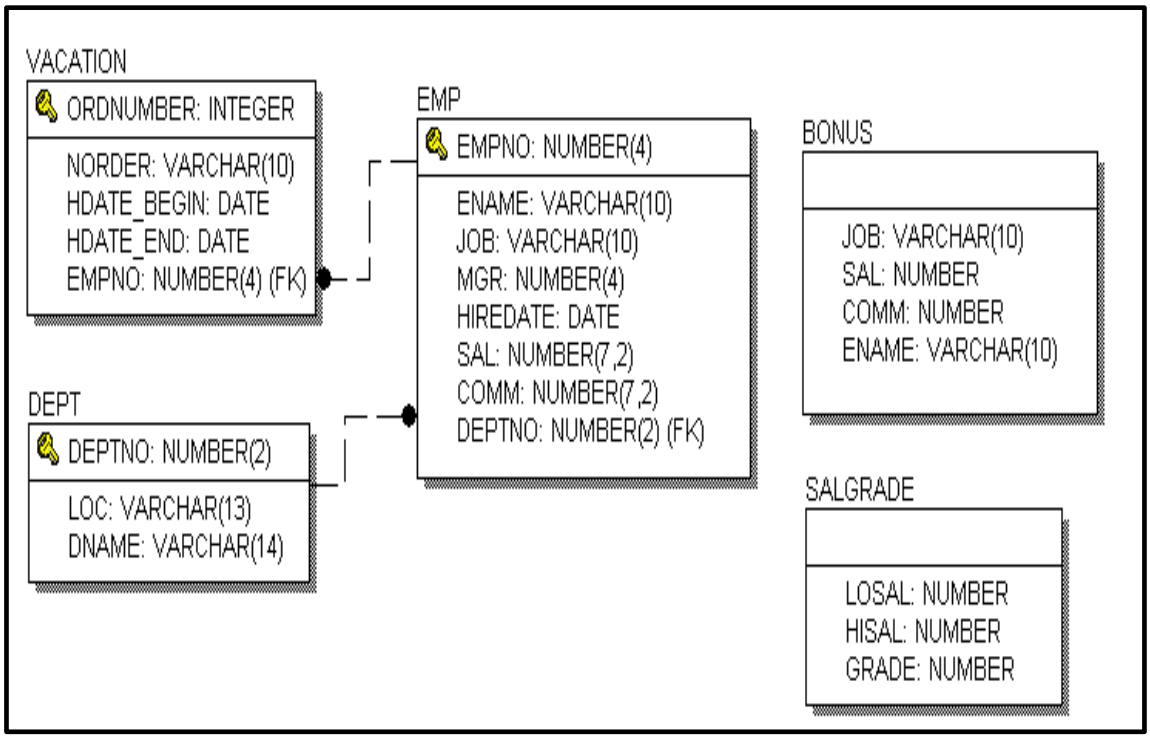


Рис. 2.1. **Схема модифікованої навчальної БД.**

Скрипт SCOTT_XE2.SQL на створення та заповнення навчальної БД, а також пояснення сенсу основних атрибутів наведено у додатку В. Крім того, для самостійного опрацювання наведених прикладів самі скрипти можна завантажити з сайта дистанційного навчання ХНЕУ ім. С. Кузнеця [51].

Запитання і завдання

1. Яке призначення має мова SQL?
2. Які основні групи команд SQL вам відомі? Охарактеризуйте їх.
3. Які особливості мови SQL реалізовані в СКБД Oracle? Перелічіть їх.

2.2. Створення таблиць (загальні положення)

Приступаючи до створення таблиці, необхідно мати відповіді на низку запитань [24]:

- Як називатиметься таблиця?
- Як називатимуться стовпці (поля) таблиці?
- Які типи даних будуть закріплені за кожним стовпцем?
- Який розмір пам'яті потрібно виділити для зберігання кожного стовпця?
- У які стовпці таблиці обов'язково вводити дані?
- З яких стовпців складатиметься первинний ключ?
- Значення якого стовпця або стовпців є унікальними?
- Значення яких стовпців можуть мати значення "за замовчуванням"?
- Які існують обмеження на дані, що зберігаються у стовпцях та усій таблиці?

Щоб створити нову таблицю БД, необхідно використати команду CREATE TABLE, яка має такий формат:

CREATE TABLE [схема.]таблиця
({ім'я_стовпця тип_даних [значення_за_ замовчуванням] [обмеження_стовпця[,...]]} | {обмеження_таблиці}[,...])
[{[PCTFREE цілочислове_значення |
PCTUSED цілочислове_значення |
INITRANS цілочислове_значення |

MAXTRANS цілочислове_значення |
TABLESPACE ім'я | *STORAGE характеристики_збереження* |
RECOVERABLE | *UNRECOVERABLE]} [,...]* |
{[CLUSTER кластер (список_стовпців)]}]
[ENABLE обмеження] [DISABLE обмеження]
[AS підзапит]
[CACHE | *NOCACHE]*

Ім'я таблиці та ім'я стовпця

Як видно із синтаксису команди, **схема** є необов'язковим параметром і необхідна лише у тому випадку, коли таблиця створюється у схемі іншого користувача, на що повинні існувати певні повноваження. За замовчуванням використовується схема того користувача, який виконує команду CREATE TABLE. З іншого боку, **таблиця** є обов'язковим параметром і задає ім'я створюваної таблиці. Імена стовпців є також обов'язковими параметрами, вони визначають структуру створюваної таблиці.

Максимальна довжина імені таблиці або стовпця не повинна перевищувати тридцяти символів. Імена таблиць і стовпців можуть містити літери, цифри та символ підкреслення і повинні починатися з літерного символу. Символи верхнього та нижнього регістрів в іменах таблиць і стовпців вважаються однаковими.

В Oracle таблиці за замовчуванням присвоюються тому користувачеві, який їх створив. Кожна з таблиць повинна мати ім'я, відмінне від імен інших таблиць користувача, тобто у користувача не може бути двох таблиць з однаковими іменами. Однак різні користувачі можуть створювати таблиці, що мають одне і теж ім'я. Усі стовпці в межах таблиці повинні мати унікальні імена.

Тип даних

Параметр **тип_даних** є обов'язковим для кожного стовпця в таблиці. Число стовпців у таблиці не повинно перевищувати 1000. Характеристики кожного стовпця відокремлюються від іншого комами.

Специфікація типу даних може доповнюватися одним або двома числами (залежно від типу) у круглих дужках, що визначає розмір стовпця, тобто задає максимальну довжину, яку можуть приймати його значення.

Найбільш часто використовувані типи даних – це NUMBER, CHAR, VARCHAR2 (або VARCHAR) та DATE. У стовпцях DATE розмір не зада-

ється, у стовпцях VARCHAR2 завдання розміру необхідно. Крім того в Oracle існують і інші типи даних, менш використовувані [55].

Перелік основних типів даних у Oracle та їх призначення наведено у табл. 2.1.

Таблиця 2.1

Найбільш вживані типи даних Oracle

Тип даних	Характеристика
CHAR (n)	Символьні рядки постійної довжини (до 2 000 байт)
VARCHAR2 (n) VARCHAR (n)	Символьні рядки змінної довжини (до 4 000 байт)
NCHAR (n) NVARRCHAR2 (n)	Символьні рядки у національному кодуванні (Unicode) (2 000 та 4 000 символів відповідно)
NUMBER	Числа з плаваючою комою до 38-ми розрядів
NUMBER (p)	Цілі числа до <p> розрядів
NUMBER (p, s)	Числа мають до <p> розрядів, <s> з яких – дробові, тобто праворуч від десяткової коми
DATE	Зберігає дату та час у діапазоні від 00:00:00 01/01/4712 до н. е. до 23:59:59 31/12/9999.
TIMESTAMP (n)	Значення повної дати та часу, де n – кількість цифр для часток секунди у полі секунд (допустимі значення 0 – 9, за замовчуванням – 6).
INTERVAL	Зберігає проміжок часу, вимірюваний у днях, годинах, хвилинах і секундах

Примітка. У стовпцях типу CHAR значення можуть бути задані за замовчуванням, у цьому випадку вони будуть мати максимальну довжину. Дробові значення округляються до розміру подання числа, тобто до знака <s> після десяткової коми

Приклади задавання типів даних:

NUMBER (5) –- цілі числа до п'яти розрядів;

NUMBER (7, 3) – числа до семи розрядів, три з яких – після десяткової коми;

VARCHAR2(65) – значення можуть містити до 65-ти символів;

CHAR(80) – рядки фіксованої довжини до 80-ти символів, більш короткі значення доповнюються пробілами праворуч.

Слід зазначити, що замість стандартних типів СКБД Oracle можна використовувати і типи даних, що визначені стандартом мови SQL, (DECIMAL, REAL тощо [60]), які Oracle все одно перетворить на власні

типи даних. Тому рекомендується використовувати власні типи системи замість стандартних.

Обмеження

Вираз **значення_за_замовчуванням,** а також обмеження **обмеження_стовпця** та **обмеження_таблиці** є необов'язковими.

Значення за замовчуванням використовується для присвоєння стовпцю значення, яке йому буде автоматично задане, якщо оператор вставки не виконує явного присвоювання.

Обмеження стовпця використовується для визначення обмеження цілісності, такого, як **NOT NULL,** або конструкцією **CHECK** для завдання умов приналежності до певного діапазона або переліку значень.

Обмеження таблиці використовується для визначення обмеження цілісності, наприклад, такої, як первинний ключ **PRIMARY KEY**, унікальне значення **UNIQUE** або зовнішній ключ **REFERENCES**.

Крім зазначених можуть також задаватися обмеження, розташовані після переліку усіх стовпців створюваної таблиці і які починаються ключовим словом **CONSTRAINT**. Ці обмеження можуть визначати обмеження кортежу (або рядка) разом з конструкцією **CHECK**, складений первинний ключ або унікальне значення – **PRIMARY KEY** та **UNIQUE**, а також зовнішній ключ – **FOREIGN KEY**.

Ілюстрація та пояснення основних конструкцій, які використовуються при створенні таблиць, буде наведено на прикладі навчальної бази даних (підрозділ 2.3).

Параметри PCTFREE, PCTUSED, NITRANS, MAXTRANS, TABLESPACE, STORAGE, RECOVERABLE, UNRECOVERABLE, CLUSTER, CACHE та NOCACHE є необов'язковими, звичайними користувачами бази даних як правило не використовуються і є прерогативою адміністратора бази даних. Сенс цих параметрів дається у додатку Г, а детальний опис наведено у [81].

Запитання і завдання

1. Які аспекти таблиць потрібно враховувати під час їх створення?
2. Назвіть обов'язкові параметри команди CREATE TABLE.
3. Які обмеження на таблиці можна задавати під час їх створення?

2.3. Створення навчальної бази даних

Аналіз схеми навчальної бази даних (див. рис.2.1) показує, що таблиця DEPT, яка містить інформацію про відділи є незалежною від інших таблиць, тобто це довідник першого рівня. Тому створення слід починати саме з цієї таблиці. Команда на створення таблиці є такою:

Таблиця DEPT (Відділи)

```
CREATE TABLE DEPT
    (DEPTNO NUMBER(2) PRIMARY KEY
                CHECK (MOD(DEPTNO,10)=0 AND DEPTNO > 0),
     DNAME VARCHAR2(14) UNIQUE,
     LOC VARCHAR2(13) NOT NULL);
```

Доцільно проаналізувати цю команду. Перший рядок є безпосередньо командою SQL на створення таблиці DEPT. Інші рядки описують три стовпці цієї таблиці, а саме: DEPTNO – номер відділу, DNAME – назва відділу та LOC – місто розташування відділу.

Поле DEPTNO є цілим числом і може мати у значенні не більше двох розрядів – на це вказує тип даних NUMBER(2). Це поле є також первинним ключем, про що говорить PRIMARY KEY. Крім того, у характеристиці цього атрибуту є конструкція CHECK, яка задає певні обмеження (умову) на допустимі значення цього атрибуту.

Загальна умова (обмеження) складається з двох елементарних умов, які з'єднані булевою операцією AND. Це означає, що для того щоб значення номера відділу було коректним з точки зору предметної області, потрібне одночасне виконання обох елементарних умов. Перша з них використовує стандартну функцію MOD, яка обчислює залишок від ділення номера відділу на 10 і потребує, щоб він дорівнював нулю. Друга умова вимагає, щоб значення відділу були більше нуля. Таким чином, загальне обмеження на номер відділу дозволяє додати тільки значення 10, 20, 30, 40, 50, 60, 70, 80 та 90. Усі інші будуть сприйняті як помилкові.

Поле DNAME є рядком символів загальною довжиною, що не перевищує чотирнадцять. Крім того, ознака UNIQUE вимагає, щоб усі значення були унікальними, тобто відрізнялися один від одного. Однак, оскільки для поля DNAME не вказано обмеження NOT NULL, значення цього поля можуть у СКБД Oracle бути NULL.

І нарешті, поле LOC також є рядком символів з максимальною довжиною тринадцять і обмеженням, що не дозволяє вводити неіснуючі значення, тобто новий відділ, навіть якщо його назва ще не визначена, має знаходитись у певному місті.

Таблиця EMP (Співробітники)

У таблиці EMP зберігаються відомості про співробітників. Команда на створення таблиці мовою SQL така:

```
CREATE TABLE EMP
    (EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
     ENAME VARCHAR2(10),
     JOB VARCHAR2(10) DEFAULT 'CLERK',
     MGR NUMBER(4),
     HIREDATE DATE DEFAULT SYSDATE,
     SAL NUMBER(7,2) NOT NULL
                CHECK (SAL > 500 AND SAL <= 6000),
     COMM NUMBER(7,2),
     DEPTNO NUMBER(2)
                CONSTRAINT FK_DEPTNO REFERENCES DEPT,
     CONSTRAINT CH_COMM CHECK (COMM<= 2*SAL));
```

Таблиця має вісім стовпців (полів або атрибутів).

Перший стовпець EMPNO (номер співробітника) визначає первинний ключ таблиці, що є числом, яке не перевищує 9999, оскільки тип даних є NUMBER(4). На відміну від попередньої таблиці, перед фразою PRIMARY KEY записана конструкція CONSTRAINT PK_EMP, якої не було при створенні таблиці DEPT. У першому випадку система сама давала назву обмеженню щось на зразок SYS_C004175, а другому назва обмеження PK_EMP задається безпосередньо у команді CREATE TABLE.

Другий стовпець таблиці ENAME містить прізвище співробітника та не може перевищувати десяти символів. Треба звернути увагу на те, що обмеження NOT NULL тут не вказано, тому теоретично у БД можна занести відомості про нового співробітника не вказуючи його ім'я. З точки зору здорового глузду це сенсу не має, однак може мати сенс з точки зору питань безпеки чи конфіденційності.

Трете поле JOB визначає посаду співробітника, значення якої не може перевищувати десяти символів, причому, ключове слово DEFAULT

вказує, що якщо при введенні даних посада не буде вказана, то, за замовчуванням, вона буде введена як 'CLERK'.

Четверте поле таблиці MGR характеризує номер співробітника який є безпосереднім керівником поточного співробітника. Це поле допускає значення NULL, оскільки особа, яка займає найвищу посаду, не має безпосереднього керівника.

П'яте поле HIREDATE характеризує дату прийому на роботу та має значення за замовчуванням, яке визначається стандартною функцією Oracle – SYSDATE, тобто поточну дату.

Шосте поле SAL характеризує оклад (місячну платню) співробітника. Поле числове з двома знаками після коми, не може мати порожнє значення і повинно знаходитися у діапазоні від 500 до 6000.

Сьоме поле COMM визначає розмір річної премії, що отримав співробітник. Це поле може мати значення NULL, оскільки не усі співробітники нагороджуються премією.

І останнє поле – DEPTNO вказує на номер відділу, у якому працює співробітник. Оскільки номера відділів визначені та знаходяться у таблиці DEPT, то фраза CONSTRAINT FK_DEPTNO REFERENCES DEPT формує обмеження зовнішнього ключа до таблиці DEPT. Тобто значення, які будуть вводитися для поля DEPTNO у таблицю EMP, будуть перевірятися системою на наявність у таблиці DEPT. Якщо такого відділу не існує, станеться помилка.

Останньою фразою команди є визначення додаткового обмеження, що накладається на кожний рядок таблиці та зв'язує два її стовпця (поля) SAL та COMM. Обмеження потребує, щоб премія співробітнику не перевищувала подвійного значення окладу.

Таблиця SALGRADE (Рівень оплати).

Таблиця SALGRADE досить проста і містить дані про диференціацію заробітних плат по рівнях. Таблиця має лише три стовпці. Команда на її створення така:

```
CREATE TABLE SALGRADE
    (GRADE NUMBER NOT NULL,
      LOSAL NUMBER  NOT NULL,
      HISAL NUMBER  NOT NULL);
```

Перше поле задає значення рівня заробітної платні (перший рівень найнижчий), а друге та третє, відповідно, мінімальне та максимальне значення заробітної плати для цього рівня. Як видно з наведеної команди, первинний ключ у таблиці не визначений, тому можливі певні помилки при введенні даних.

Спробуйте самостійно визначити, які колізії можуть виникнути при роботі з цією таблицею, і перетворити команду на створення таблиці таким чином, щоб уникнути можливих помилок.

Таблиця VACATION (Відпустки)

Таблиця VACATION містить інформацію про дати, коли той чи інший співробітник знаходився у відпустці. Команда на її створення така:

```
CREATE TABLE VACATION
      (ORDNUMBER  NUMBER PRIMARY KEY,
       NORDER       VARCHAR2(10) NOT NULL,
       EMPNO              NUMBER(4) NOT NULL,
       HDATE_BEGIN      DATE NOT NULL,
       HDATE_END   DATE NOT NULL,
       CONSTRAINT     FK_EMPNO FOREIGN KEY (EMPNO)
                      REFERENCES EMP(EMPNO),
       CONSTRAINT     VK_UN  UNIQUE (EMPNO, HDATE_BEGIN),
       CONSTRAINT     DT_CHK
                      CHECK (HDATE_BEGIN < HDATE_END));
```

Таблиця має п'ять стовпців.

Перший стовпець ORDNUMBER – первинний ключ таблиці і по суті є порядковим номером рядка у таблиці. Значення цього поля будуть автоматично формуватися за допомогою послідовності (див. п. 2.5.2).

Другий стовпець NORDER – номер наказу по організації щодо надання відпустки. Він повинен вказуватися обов'язково.

Третій стовпець EMPNO – це номер співробітника, якому надається відпустка. Природно, що цей номер повинен належати довіднику співробітників, яким є таблиця EMP. Тобто поле EMPNO таблиці VACATION є зовнішнім ключем до таблиці EMP. Це можна було б вказати безпосередньо при описі поля, але для ілюстрації додаткових можливостей винесено окремо.

Четвертий та п'ятий стовпці визначають початкову та кінцеву дати відпустки, відповідно.

Після опису полів таблиці додатково задаються ще додаткові обмеження, а саме:

FK_EMPNO – це обмеження зовнішнього ключа EMPNO, який посилається на таблицю EMP.

VK_UN – обмеження унікальності на парі атрибутів EMPNO та HDATE_BEGIN. Тобто не можна додати до таблиці рядки зі значеннями цих полів, які вже є у таблиці. Причому унікальність перевіряється саме на двох значеннях одночасно.

DT_CHK – обмеження кортежу (рядка). Визначає той факт, що дата початку відпустки повинна бути менша за дату закінчення.

Таблиця BONUS (Бонус)

Таблиця BONUS є додатковою таблицею й у початковій БД жодних рядків не містить. Вона використовується у подальшому для ілюстрації команди додавання рядків у таблицю на основі вкладеного підзапиту (див приклад 2.88). Стовпці таблиці BONUS є підмножиною стовпців таблиці EMP і фактично вона буде містити дані про імена, посади, заробітну плату та премію співробітників. Команда на її створення така:

```
CREATE TABLE BONUS
      (ENAME VARCHAR2(10),
       JOB VARCHAR2(10),
       SAL NUMBER,
       COMM NUMBER);
```

Запитання і завдання

1. Опишіть призначення кожної таблиці навчальної бази даних.
2. Якими засобами встановлюють обмеження у таблиці DEPT?
3. Яким чином встановлюють відношення між таблицями DEPT та EMP?
4. Опишіть обмеження, які встановлюють у таблиці VACATION.
5. Як співвідносяться таблиці EMP та BONUS?

2.4. Підтримка посилальної цілісності

2.4.1. Зовнішні ключі та посилальна цілісність

З наведених прикладів створення таблиць видно, що таблиці EMP та VACATION мають зовнішні ключі, що посилаються на батьківські таблиці DEPT та EMP, відповідно. На ER-діаграмі (див. рис. 2.1) атрибути зв'язку, що знаходяться у дочірніх таблицях, мають позначення FK (**foreign key)**, тобто **зовнішній ключ**.

Саме із зовнішніми ключами пов'язане одне з основних правил цілісності, а саме – правило посилальної цілісності [23; 24]. База даних не повинна містити неузгоджених значень зовнішнього ключа, тобто значень, яких немає для потенційного ключа у посилальному відношенні.

У визначенні посилальної цілісності беруть участь два відношення – батьківське та дочірнє. Між ними може існувати зв'язок як "один-до-багатьох", так і "один-до-одного". Для кожного з відношень можливі три операції – вставка, оновлення, видалення, в результаті яких посилальна цілісність може порушитися. З аналізу цих шести можливих варіантів видно, що в чотирьох із них в принципі може бути порушена посилальна цілісність, а саме – під час виконання операцій:

- оновлення кортежу у батьківському відношенні;
- видалення кортежу у батьківському відношенні;
- вставка кортежу в дочірнє відношення;
- оновлення кортежу в дочірньому відношенні.

2.4.2. Стратегії підтримки посилальної цілісності

У зв'язку з необхідністю підтримки посилальної цілісності у БД мова SQL передбачає спеціальні конструкції, які описують так звані "стратегії підтримки посилальної цілісності", що визначають дії СКБД у разі, коли під час виконання однієї з чотирьох зазначених операцій може статися порушення посилальної цілісності. У СКБД Oracle передбачені такі дії:

• **RESTRICT** (обмежити) або **NO ACTION** (не дозволяти виконання операції) призводять до порушення посилальної цілісності. Це найпростіша стратегія, що вимагає тільки перевірки, чи є кортежі в дочірньому відношенні, пов'язані з певним кортежем у батьківському відношенні. ***Ця стратегія у Oracle передбачена за замовчуванням і не вказується у визначенні посилальної цілісності***.

• **CASCADE** (каскадувати) – дозволити виконання операції, але внести при цьому необхідні поправки в інші відношення так, щоб не допустити порушення посилальної цілісності та зберегти усі наявні зв'язки. Зміна починається у батьківському відношенні та каскадно виконується в дочірньому. У реалізації цієї стратегії є нюанс, який полягає в тому, що дочірнє відношення саме може бути батьківським для деякого третього відношення. ***У СКБД Oracle ця стратегія дозволяється лише для операції видалення (DELETE).***

• **SET NULL** (встановити в **NULL**) – дозволити виконання необхідної операції, але усі некоректні значення зовнішніх ключів, що з'являються, змінювати на **NULL**-значення.

Деякі інші СКБД можуть також мати додаткові стратегії, зокрема:

• **SET DEFAULT** (встановити за замовчуванням) – дозволити виконання необхідної операції, але усі некоректні значення зовнішніх ключів, що з'являються, змінювати на значення, яке прийняте за замовчуванням. Переваги цієї стратегії порівняно з попередньою у тому, що вона дозволяє не користуватися **NULL**-значеннями. Проте вона має недоліки: у батьківському відношенні має бути кортеж, потенційний ключ якого прийнятий як значення за замовчуванням для зовнішніх ключів; цей кортеж не можна видаляти з батьківського відношення; у цьому кортежі не можна змінювати значення потенційного ключа.

• **IGNORE** (ігнорувати) – виконувати операції, не звертаючи уваги на порушення посилальної цілісності.

У розглянутій навчальній БД для зв'язка між таблицями DEPT та EMP за замовчуванням буде підтримуватись стратегія підтримки посилальної цілісності RESTRICT. Тобто СКБД не дозволить видалити з бази даних будь-який відділ, якщо у ньому зареєстровані співробітники.

Для ілюстрації іншої стратегії – CASCADE, змінимо команду на створення таблиці VACATION.

```
CREATE TABLE VACATION
      (ORDNUMBER  NUMBER PRIMARY KEY,
       NORDER       VARCHAR2(10) NOT NULL,
       EMPNO              NUMBER(4) NOT NULL,
       HDATE_BEGIN      DATE NOT NULL,
       HDATE_END   DATE NOT NULL,
```

```
CONSTRAINT    FK_EMPNO FOREIGN KEY (EMPNO)
              REFERENCES EMP(EMPNO)
              ON DELETE CASCADE,
CONSTRAINT    VK_UN  UNIQUE (EMPNO, HDATE_BEGIN),
CONSTRAINT    DT_CHK
              CHECK (HDATE_BEGIN < HDATE_END));
```

Додаткова опція у визначенні зовнішнього ключа вказує на те, що якщо відомості про певного співробітника будуть видалені з таблиці EMP, то автоматично (каскадно) усі відомості про його відпустки теж будуть видалені.

Із застосуванням цієї стратегії треба буди уважним, щоб безповоротно не видалити необхідні дані. Рекомендується спочатку перенести дані про звільненого працівника і його відпустки до архівної БД, а вже потім видалити їх з оперативної БД.

Запитання і завдання

1. У чому полягає правило посилальної цілісності?
2. Як пов'язані посилальна цілісність та зовнішній ключ?
3. Які дії передбачені у СКБД Oracle для підтримки посилальної цілісності?

2.5. Створення інших об'єктів у базі даних

2.5.1. Індекси

Індекс є спеціальним об'єктом бази даних і створюється для підвищення ефективності виконання пошукових операцій. Індекс може створюватися для одного або декількох стовпців таблиці і забезпечує більш швидкий доступ до БД за рахунок прямих посилань на місце зберігання рядків, які містять необхідні дані.

Зазвичай для первинного або унікального ключа таблиці індекс створюється автоматично.

Якщо в процесі проектування або експлуатації БД виявлено, що за деякими атрибутами (стовпцями), які не входять в унікальні ключі, дуже часто проводять пошукові операції, то заради підвищення ефективності для них теж бажано створити індекси. Такі ключі називають вторинними або інверсними.

Синтаксис для створення індексу зазвичай є таким (ця конструкція не була регламентована ISO/*ANSI* SQL92, тобто це не ANSI-стандарт):

CREATE INDEX <ім'я індексу > ON <ім'я таблиці> (<ім'я стовпця>[ASC|DESC] [,<ім'я стовпця >]...);

Необов'язкові ознаки [ASC|DESC] визначать порядок розташування значень ключів в індексі – за зростанням чи за зменшенням.

Приклад 2.1. Створити індекси до таблиць EMP та VACATION для підвищення швидкості виконання запитів до БД, оскільки при аналізі навчальної БД було виявлено, що до таблиці EMP запити часто здійснюються за значенням номера відділу, а до таблиці VACATION – за номером наказу на відпустку.

Команди на створення відповідних індексів будуть такими.

```
CREATE INDEX I_EMP ON EMP (DEPTNO);
CREATE INDEX I_VACATION ON VACATION (NORDER  DESC);
```

Бажано при створенні індексів давати їм відповідні імена, що починаються з літери I.

2.5.2. Послідовності

Послідовності (Sequences) у Oracle найбільш часто використовуються для автоматичної генерації послідовних чисел для значень полів таблиці (сурогатних ключів) [24; 53; 54], оскільки у системі не передбачена така властивість поля, як **IDENTITY** у Microsoft SQL Server, або тип даних Counter у Microsoft Access.

Для створення послідовності використовується така команда:


```
CREATE SEQUENCE [schema.] Sequence
[INCREMENT BY n]
  [START WITH n]
  [MAXVALUE n | NOMAXVALUE]
  [MINVALUE n | NOMINVALUE]
  [CYCLE | NOCYCLE]
  [CACHE n | NOCACHE]
  [ORDER | NOORDER]
```

де schema – ім'я схеми, до якої належить послідовність;

sequence – ім'я послідовності, яке повинно відповідати стандартним вимогам до імен мови Oracle SQL [80].

Параметри команди означають наступне:

INCREMENT BY – вказує інтервал між порядковими номерами. Це ціле значення може бути як позитивним, так негативним числом, але не може бути нулем. Це значення може мати 28-м або менше цифр для зростаючої послідовності і 27-м або менше цифр для спадної послідовності. Абсолютне значення параметра повинно бути менше ніж (MAXVALUE – MINVALUE). За замовчуванням значення дорівнює 1.

START WITH – вказує на перший новостворений порядковий номер. Для зростаючої послідовності значення за замовчуванням дорівнює мінімальному значенню послідовності. Для спадних послідовностей, значення за замовчуванням дорівнює максимальному значенню послідовності. Це ціле значення може мати 28-м або менше цифр для позитивних значень і 27-м або менше цифр для негативних значень. Цей параметр не є обов'язковим для циклічних послідовностей.

MAXVALUE – вказує на максимальне значення послідовності. Це ціле значення може мати 28-м або менше цифр для позитивних значень і 27-м або менше цифр для негативних значень. Значення MAXVALUE повинно бути більшим або рівним START WITH і більшим за MINVALUE.

NOMAXVALUE – вказує на максимальне значення 10^{28} – 1 для зростаючої послідовності або – 1 для спадної послідовності. Це значення за замовчуванням.

MINVALUE – вказує мінімальне значення послідовності. Це ціле значення може мати 28-м або менше цифр для позитивних значень і 27-м або менше цифр для негативних значень. MINVALUE повинна бути меншою або рівною START WITH і меншою за MAXVALUE.

NOMINVALUE – вказує на мінімальне значення 1 для зростаючої послідовності або 10^{27}– 1 для спадної послідовності. Це значення за замовчуванням.

CYCLE – вказує на те, що послідовність продовжує генерувати значення спочатку після досягнення його максимального або мінімального значення.

NOCYCLE – вказує на те, що послідовність не може виробити більше значень після досягнення свого максимального або мінімального значення (за замовчуванням).

CACHE – характеризує кількість значень послідовності, яке Oracle тримає у пам'яті для швидкого доступу. Це ціле значення може мати 28-м або менше цифр. Мінімальне значення цього параметра дорівнює два. Для циклічних послідовностей це значення повинно бути меншим за кількість значень у циклі. Тобто максимально допустиме значення CACHE повинно бути менше за значення, що визначене за формулою:

$$\frac{(\mathrm{CEIL}(\mathrm{MAXVALUE} - \mathrm{MINVALUE}))}{\mathrm{ABS}(\mathrm{INCREMENT})}$$

Якщо відбувається збій системи, то всі значення у кеші послідовностей, які не були використані, губляться.

NOCACHE – при цьому значенні система за замовчуванням кешує двадцять значень.

ORDER – вимагає видавати значення послідовності суворо відповідно до часу їх запиту. Має сенс тільки при використанні у паралельному режимі з опцією Parallel Server, бо при роботі без використання даної опції (exclusive mode) значення завжди генеруються строго послідовно. За замовчуванням, приймається NOORDER.

Приклад 2.2. Створити послідовність яка генерує числа від одиниці до максимального значення та проілюструвати генерацію нею значень у команді SELECT.

Створимо послідовність командою:

CREATE SEQUENCE NUM_VACATION;

За замовчуванням послідовність буде починатися з одиниці і мати крок теж одиницю. Вона буде зростаючою до максимально допустимого значення.Генерація чергового значення послідовності виконується при зверненні до псевдостовпця NEXTVAL, наприклад:

NUM_VACATION.NEXTVAL.

Для вибору поточного значення послідовності (для даної сесії) використовується псевдостовпець CURRVAL: **NUM_VACATION.CURRVAL.**

Поточне значення в рамках даної сесії вже повинно бути згенеровано зверненням до NEXTVAL.

На рис. 2.2 ілюструється використання послідовності для відображення нового та поточного значеннь.

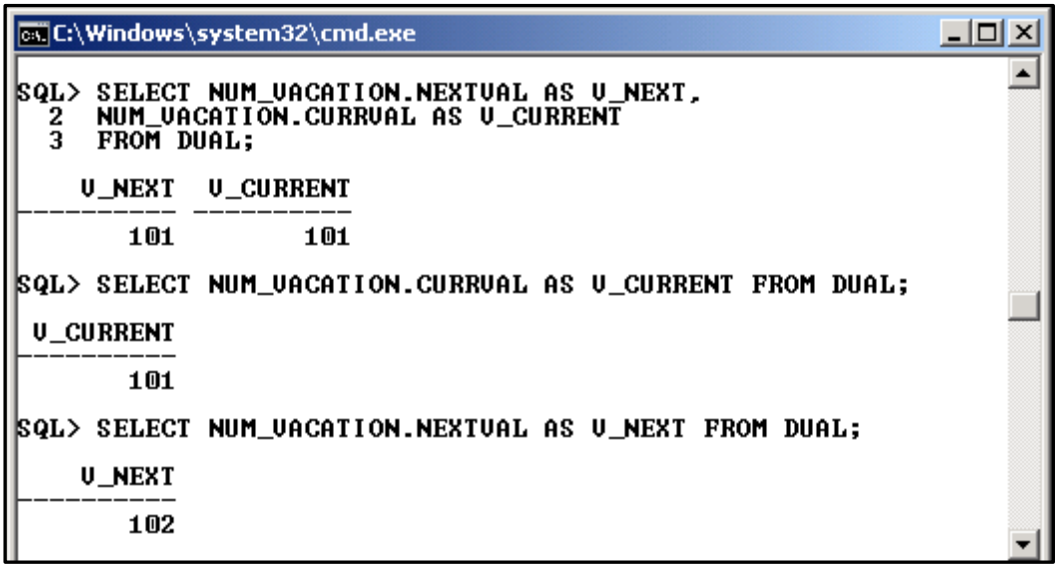


Рис. 2.2. **Виведення поточного та нового значень послідовності**

При заповненні навчальної бази даних створена послідовність знадобиться для генерації значень сурогатного ключа таблиці VACATION, а саме – поля ORDNUMBER. Її використання розглядається у підрозділі 2.7.

2.5.3. Зв'язок з віддаленою базою даних

Зв'язок бази даних (database link) призначений для опису шляху до віддаленої бази даних, що дозволяє звертатися до її таблиць та інших об'єктів у SQL-командах, що виконуються локально. Зв'язок бази даних – це основний спосіб опису шляху до інших баз даних у розподіленому середовищі, що дозволяє виконувати віддалені транзакції [L2-5].

Щоб створити зв'язок з віддаленою базою даних користувач повинен мати привілей (див. підрозділ 3.1) CREATE DATABASE LINK, а сам зв'язок створюється безпосередньо аналогічною командою.

Приклад 2.3. Створити зв'язок з віддаленою базою даних користувача SCOTT, що розташована на сервері за адресою 192.16.0.170 (рис. 2.3).

```
CREATE DATABASE LINK LSCOTT
CONNECT TO SCOTT IDENTIFIED BY TIGER
USING
'(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)
(HOST = 192.168.0.170)(PORT = 1521)))
(CONNECT_DATA = (SERVER = DEDICATED) (SID = XE)))';
```

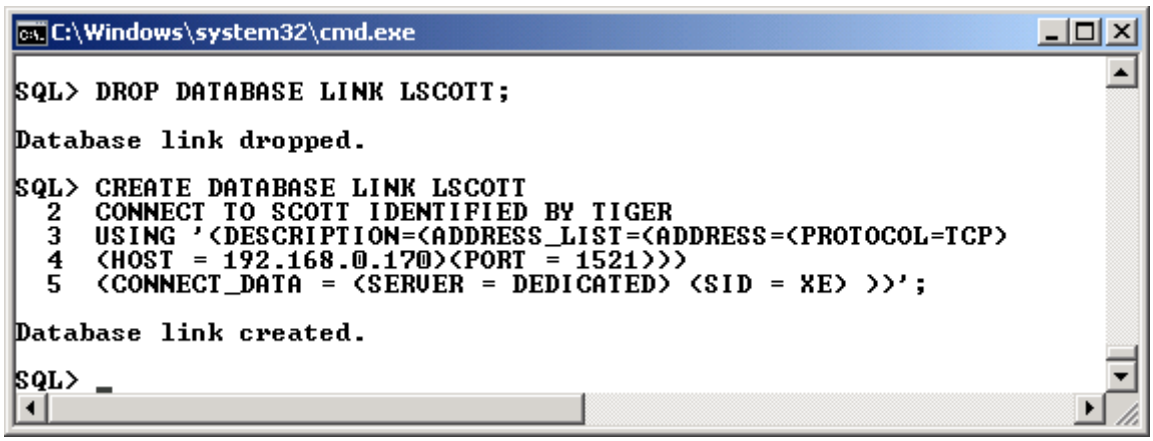


Рис. 2.3. **Приклад створення зв'язку з базою даних**

Приклад 2.4. З використанням створеного зв'язку з базою даних отримати відомості з таблиці DEPT (рис. 2.4).

SELECT * FROM DEPT@LSCOTT;

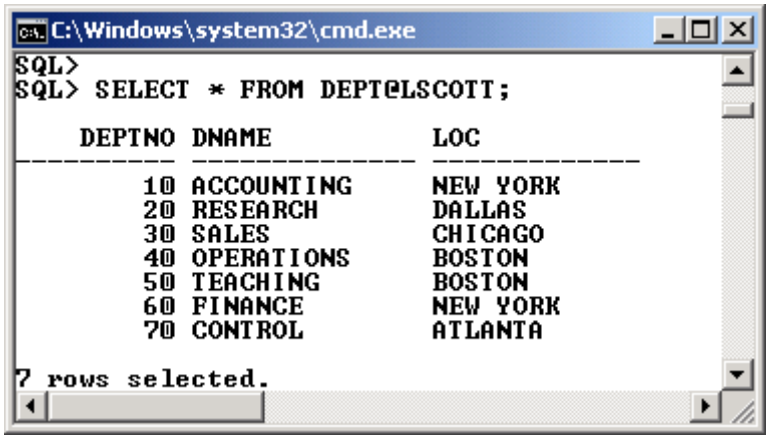


Рис. 2.4. **Вибірка даних за допомогою зв'язку з БД**

Найбільш детально database link описаний у [75].

2.5.4. Синоніми

Синонім (SYNONYM) – це аліасне (додаткове ім'я) для таблиці, подання, послідовності або програмної одиниці. Сам синонім не є конкретним об'єктом бази даних, але він є прямим посиланням на такий об'єкт. Синоніми використовуються для: маскування дійсного імені та власника об'єкта; забезпечення загального доступу до об'єкта; спрощення кодування команд SQL для користувачів бази даних.

Синонім може бути загальним (PUBLIC) чи особистим (PRIVATE). Користувач бази даних зазвичай створює особистий синонім, який

доступний тільки йому. Адміністратори баз даних найчастіше створюють загальні синоніми, завдяки яким об'єкти базових схем стають доступними для загального користування для усіх користувачів бази даних.

Синтаксис команди на створення синоніма такий:

Формат **CREATE [PUBLIC] SYNONYM [SCHEMA.]synonym FOR [SCHEMA.]OBJECT[@DATABASELINK]**,

де ***PUBLIC*** – вказує, що синонім буде доступний усім користувачам (для створення загальних синонімів потрібен привілей CREATE ANY PUBLIC SYNONYM). За замовчуванням, синонім доступний тільки його творцеві:

Synonym –- ім'я синоніма, що відповідає умовам на імена об'єктів;

SCHEMA – схема, якій належить синонім. Якщо вона опущена, оператор CREATE SYNONYM припускає, що поточний користувач є власником об'єкта, якому призначається синонім;

OBJECT – ім'я таблиці, подання, послідовності, збереженої функції, процедури, пакета, знімка чи іншого синоніма

DATABASELINK – зв'язок з віддаленою базою даних.

Приклад 2.5. Створити синонім на ім'я SDEPT для таблиці DEPT, що знаходиться у віддаленій базі даних з іменем зв'язка LSCOTT та отримати за допомогою створеного синоніма дані з таблиці (рис. 2.5).

Команди на створення та вибірку такі:

```
CREATE SYNONYM SDEPT FOR DEPT@LSCOTT;
SELECT * FROM SDEPT;
```

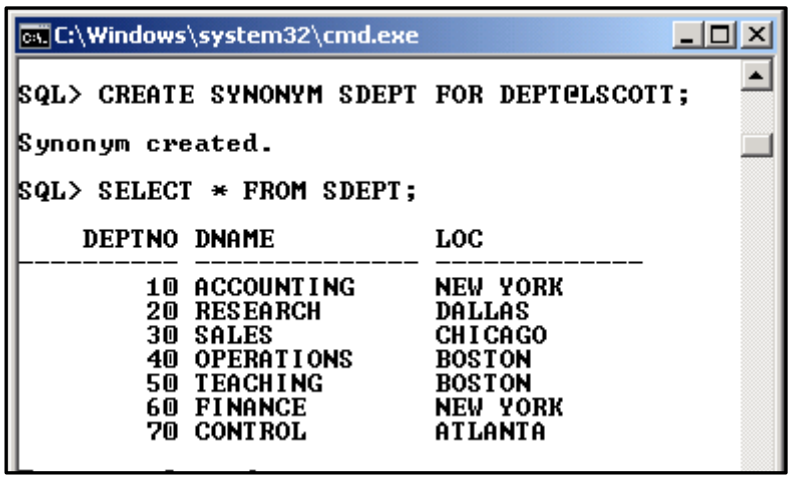


Рис. 2.5. **Створення синоніма та вибірка даних**

У наведеному прикладі використання синоніма сумісно з іменем зв'язка з віддаленою базою дозволило суттєво скоротити текст запиту до бази даних.

Запитання і завдання

1. З якою метою використовують індекси?
2. У яких випадках використовують послідовності?
3. За допомогою яких засобів встановлюють зв'язок з віддаленою базою даних?
4. Опишіть призначення синонімів. За допомогою яких конструкцій мови SQL вони створюються?

2.6 Модифікація створених об'єктів бази даних

Часто трапляється ситуація, коли необхідно внести певні корективи у структуру вже створеної таблиці. З цією метою у стандарті SQL передбачена команда ALTER TABLE [78], яка може виконувати такі дії:

- додавати до таблиці новий стовпець;
- видаляти стовпець з таблиці;
- додавати до визначення таблиці нове обмеження;
- видаляти з визначення таблиці існуюче обмеження;
- задавати для стовпця значення за замовчуванням;
- відміняти для стовпця значення за замовчуванням.

Спрощений синтаксис цієї команди у СКБД Oracle виглядає так:


```
ALTER TABLE [schema.]table
[ADD
  [(]column1 datatype1 [DEFAULT expr1] [col_constraint1]
  [,column2 datatype2 [DEFAULT expr2] [col_constraint2]]
    ...[)]
]
[MODIFY
  [(]column1 datatype1 [DEFAULT expr1] [col_constraint1]
  [,column2 datatype2 [DEFAULT expr2] [col_constraint2]]
    ...[)]
]
```

[DROP {column_clause | constraint_clause}
]
[ENABLE {enable_clause}]
[DISABLE {enable_clause}],

де schema – це схема користувача;
table – ім'я таблиці;
column – назва стовпця таблиці;
datatype – тип даних стовпця;
expr – значення стовпця за замовчуванням;
col_constraint – обмеження на значення стовпця.

Ключові фрази ***ADD*** та ***MODIFY*** вказують, відповідно, на додавання нового стовпця до таблиці, або на модифікацію існуючого стовпця.

DROP – вказує на видалення існуючого стовпця або обмеження.

ENABLE та ***DISABLE*** використовуються для тимчасового відключення або активації певних обмежень без видалення їх зі словника даних.

Приклад 2.6. Додати новий стовпець GRADE_NAME типу CHAR до таблиці SALGRADE для символьного відображення унікальної назви рівня заробітної платні. Наступним кроком змінити тип даних для стовпця GRADE_NAME на VARCHAR2 і додати обмеження NOT NULL:

ALTER TABLE SALGRADE ADD GRADE_NAME CHAR(10) UNIQUE;

ALTER TABLE SALGRADE MODIFY GRADE_NAME VARCHAR2(10) NOT NULL;

Виконання команд для пустої таблиці SALGRADE не викликає ускладнень для системи. Якби таблиця була вже заповнена, то з'явилося б повідомлення про помилку на зразок наведеного на рис. 2.6.

Причина криється у тому, що при додаванні нового стовпця GRADE_NAME до заповненої таблиці значення в усіх рядках для нього були б NULL і, відповідно, спроба встановити обмеження NOT NULL не може виконуватись.

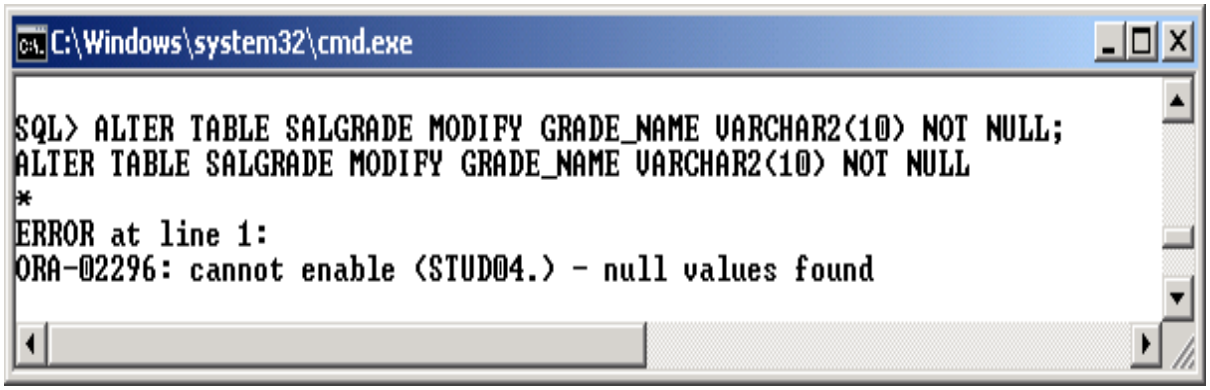


Рис. 2.6. **Помилка при модифікації структури SALGRADE**

У цьому випадку для нового стовпця потрібно спочатку заповнити значеннями вже існуючі рядки, а потім робити модифікацію його типу з обмеженням NOT NULL.

Приклад 2.7. Видалити стовпець GRADE_NAME із таблиці SALGRADE.

```
ALTER TABLE SALGRADE DROP COLUMN GRADE_NAME;
```

Приклад 2.8. Відкоригувати умову для значення заробітної плати у таблиці SAL, зменшивши максимальний розмір з 6000 до 5500.

```
ALTER TABLE EMP MODIFY SAL  CHECK (SAL > 0 AND SAL <= 5500);
```

Приклад 2.9. Видалити з таблиці VACATION існуюче обмеження DT_CHK і замість нього додати нове обмеження, яке встановлює допустиму тривалість відпустки від 4-х до 48-ми календарних днів, згідно з новим наказом керівництва:

```
ALTER TABLE VACATION DROP CONSTRAINT  DT_CHK;

ALTER TABLE VACATION ADD CONSTRAINT  DT_CHK_NEW
                CHECK (HDATE_END – HDATE_BEGIN > 2 AND
                       HDATE_END – HDATE_BEGIN < 48);
```

Слід бути уважним і обережним при використанні команди ALTER TABLE до таблиці, що вже заповнена даними, оскільки запропонована модифікація структури та обмежень може вступити в конфлікт зі вже існуючими даними. Наприклад: спроба додати

обмеження NOT NULL до стовпця, який вже містить значення NULL; додати нову умову CHECK, якій не відповідають окремі значення у таблиці; додати нове обмеження CONSTRAINT (наприклад, зовнішнього ключа), що вступить в протиріччя зі значеннями батьківської таблиці тощо.

Отже, якщо аналіз предметної області та проектування схеми бази даних було виконано правильно, то необхідність у використанні команди ALTER TABLE у більшості випадків не виникає.

Запитання і завдання

1. Які види модифікування структури таблиць передбачені мовою SQL?
2. Опишіть призначення основних параметрів команди ALTER TABLE.

2.7. Заповнення таблиць навчальної бази даних

При створенні таблиць командою CREATE TABLE у базі даних формується тільки її структура, але сама таблиця залишається порожньою, бо дані до неї не додавалися. Щоб наповнити таблицю новими даними, необхідно виконати команду **INSERT** (вставити, додати), яка безпосередньо виконує таку функцію.

Команда **INSERT** має такий формат:

INSERT INTO <table > [(column [,...n])]
{VALUES (value[,...n])|
<SELECT_statement>}

Тут параметр **table** є або ім'ям таблиці бази даних, або ім'ям оновлюваного подання (підрозділ 2.14); **column** – назви стовпців таблиці; **value** – значення стовпців у рядку.

SELECT_statement – команда SELECT мови SQL за допомогою якої здійснюється заповнення таблиці. Цей варіант розглянуто у п. 2.13.5.

Список значень команди INSERT повинен відповідати списку стовпців згідно з правилами [14; 24]:

• кількість елементів в обох списках має бути однаковою;

• повинна існувати пряма відповідність між позицією того самого елементу в обох списках, тому перший елемент списку значень має відноситися до першого стовпця в списку стовпців, другий – до другого стовпця і так далі;

• типи елементів у списку значень мають бути сумісні з типами відповідних стовпців таблиці;

• порожній рядок у Oracle сприймається як значення NULL.

Відповідно до цього, команди додавання рядків у таблиці навчальної бази даних будуть такими:

Формування таблиці DEPT:

```
INSERT INTO DEPT (DEPTNO, DNAME, LOC)
VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
.....
INSERT INTO DEPT VALUES (70,'CONTROL','ATLANTA');
```

Формування таблиці EMP:

```
INSERT INTO EMP VALUES (7369,'SMITH','CLERK',
7902,to_date('17-12-1980','dd-mm-yyyy'), 800,NULL,20);
INSERT INTO EMP VALUES (7499,'ALLEN','SALESMAN',
7698,to_date('20-2-1981','dd-mm-yyyy'), 1600,300,30);
INSERT INTO EMP VALUES (7521,'WARD','SALESMAN',
7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
.....
```

Формування таблиці SALGRADE:

```
INSERT INTO SALGRADE VALUES (1,700,1200);
INSERT INTO SALGRADE VALUES (2,1201,1400);
INSERT INTO SALGRADE VALUES (3,1401,2000);
INSERT INTO SALGRADE VALUES (4,2001,3000);
INSERT INTO SALGRADE VALUES (5,3001,9999);
```

Формування таблиці VACATION:

INSERT INTO VACATION
VALUES (NUM_VACATION.NEXTVAL, '10H-1', 7369,
to_date('02.02.2010', 'dd.mm.yyyy'), to_date(' 23.02.2010', 'dd.mm.yyyy'));

INSERT INTO VACATION
VALUES (NUM_VACATION.NEXTVAL, '10H-1', 7401,
to_date('09.02.2010', 'dd.mm.yyyy'), to_date(' 02.03.2010', 'dd.mm.yyyy'));

.....

З наведених прикладів можна зробити такі висновки:

1. Якщо список значень рядка відповідає всім стовпцям таблиці, то перелік стовпців вказувати не обов'язково. Тобто команда INSERT INTO SALGRADE VALUES (1,700,1200);

еквівалентна команді
INSERT INTO SALGRADE (GRADE, LOSAL, HISAL) VALUES (1,700,1200).

2. Символьні значення записуються в апострофах.

3. Якщо формат значень дат не відповідає прийнятій у системі, то бажано використовувати функцію перетворення to_date, наприклад to_date('15/12/2014','DD/MM/YYYY') (п. 2.17.4). Іншим рішенням буде зміна формату дати для поточного сеансу за допомогою команди ALTER SESSION, наприклад, так:.

ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY';

4. При формуванні стовпця первинного (сурогатного) ключа таблиці VACATION була використана послідовність NUM_VACATION, яка при кожному зверненні до псевдостовпця NEXTVAL генерує нове значення. Тобто при додаванні першого рядка у таблицю NEXVAL повертає значення одиниці, для другого рядка – двійки і т. д.

Скорочена конструкція команди INSERT, що передбачена стандартом SQL/92, дозволяє ввести декілька рядків однією командою:

INSERT INTO SALGRADE (GRADE, LOSAL, HISAL) VALUES
(1,700,1200),
(2,1201,1400),
(3,1401,2000),
(4,2001,3000), (5,3001,9999);

у Oracle генерує помилку.

Запитання і завдання

1. Опишіть призначення основних параметрів команди INSERT.
2. Яким правилам повинен відповідати список значень команди INSERT списку стовпців? Назвіть їх.
3. Як задають значення дат у команді INSERT? Наведіть приклади.

2.8. Вибірка інформації з однієї таблиці

Операція вибірки даних займає найважливіше місце серед усіх команд SQL. Вона є, з одного боку, найбільш поширеною, а з іншого – найбільш складною. У реальних умовах операції вибірки даних, тобто пошуку необхідної інформації у базі даних і подання її в зручному для опрацювання вигляді займають понад 80 % усіх операції, що виконуються з базою даних [24].

Оператор **SELECT** має такий формат:

SELECT [**ALL | DISTINCT**] {*|[ім'я_стовпця [**AS** нове_ім'я стовпця]]} [,...n]
FROM ім'я_таблиці [[**AS**] псевдонім таблиці] [,...n]
[WHERE <умова_пошуку>]
[GROUP BY ім'я_стовпця [,...n]]
[HAVING <критерії вибору груп>]
[ORDER BY ім'я_стовпця [ASC | DESC] [,...n]]

Як видно з опису команди, обов'язковими в операції вибірки є два речення: **SELECT** та **FROM**.

Речення **FROM** визначає ім'я таблиці або таблиць, з якої (яких) необхідно отримати дані. У разі, коли таблиць більше за одну, їх імена записуються через кому.

Речення **SELECT** задає перелік стовпців таблиці або таблиць, дані з яких необхідно вибрати. У разі, коли вибираються усі стовпці, немає потреби їх перераховувати. У цьому випадку ставиться зірочка (*).

Речення **WHERE** задає умову пошуку (умову відбору) тільки тих рядків таблиці, що задовольняють умові. Умова відбору накладається на один або декілька атрибутів таблиці. Кожна така умова є логічним виразом, значення якого можуть бути тільки істина (TRUE) або неправда (FALSE). До окремої умови може застосовуватися булева операція запе-

речення "НІ" (**NOT**). Декілька елементарних умов об'єднуються булевими операціями " І" (**AND**) та "АБО" (**OR**).

У реченні **GROUP BY** вказується ім'я стовпця (або стовпців), за однаковими значеннями у яких проводиться групування (агрегація) рядків результату. Речення **GROUP BY** дає можливість обчислити значення агрегатних функцій для кожної групи окремо.

Аналогічно з реченням **WHERE**, яке використовується для відбору рядків таблиць, що беруть участь у запиті, речення **HAVING** використовується для відбору (фільтрування) тих згрупованих рядків, для яких значення агрегатної функції відповідає певній умові.

Речення **ORDER BY** дозволяє впорядкувати рядки таблиці-результату за значеннями одного або кількох стовпців.

Нове ім'я стовпця змінює тільки його назву у результуючий таблиці. Його не можна використовувати у поточному запиті, наприклад, у виразах чи реченні GROUP BY. На відміну від цього, псевдонім таблиці може бути використаний як у поточному запиті, так і у вкладених у нього (підрозділ 2.13).

Ознайомлення з особливостями використання команди SELECT у СКБД Oracle проведено на конкретних прикладах, починаючи з простих і поступово збільшуючи їх складність.

2.8.1. Вибір усіх або окремих стовпців таблиці

Приклад 2.10. Отримати повну інформацію про відділи.(рис. 2.7)

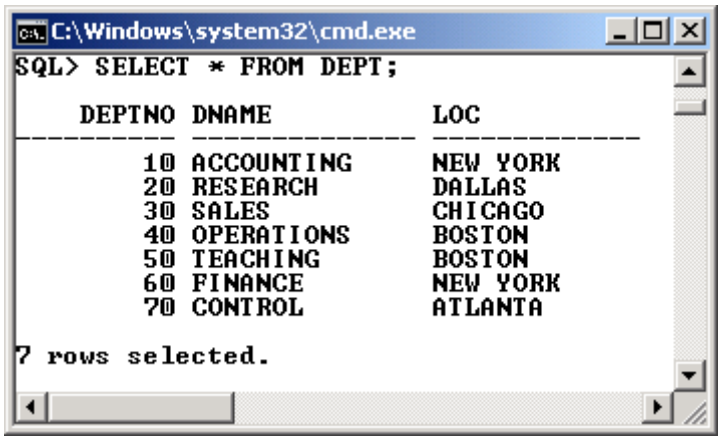


Рис. 2.7. **Інформація про відділи**

Приклад 2.11. Отримати відомості про номер співробітника, його прізвище та оклад (рис. 2.8).

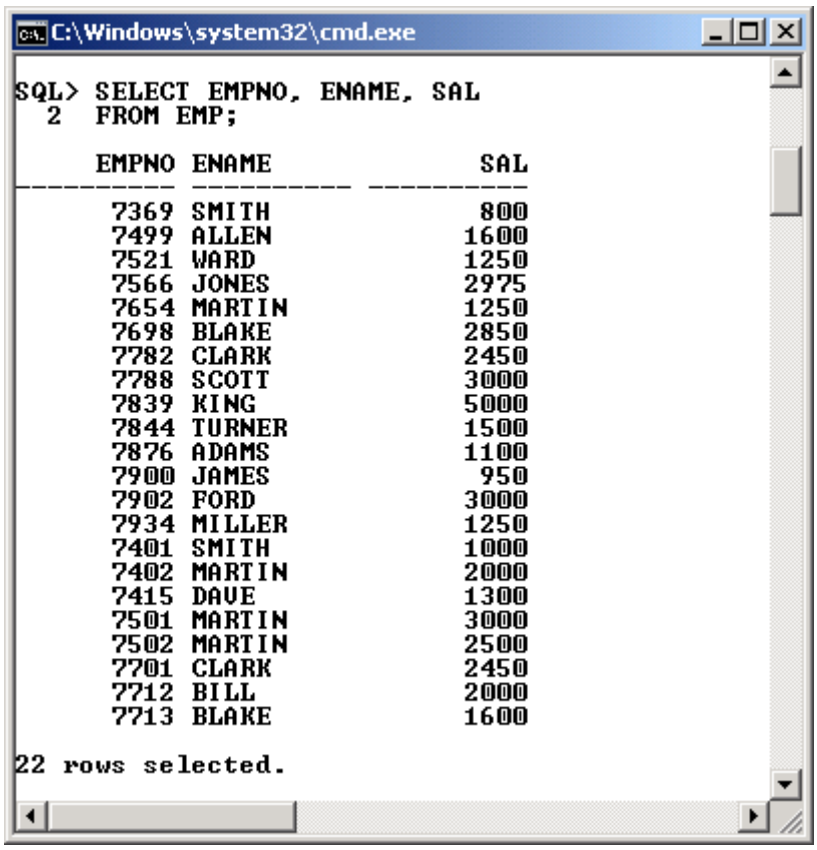


Рис. 2.8. **Номер, прізвище та оклад співробітника**

2.8.2. Вибір рядків без повторень

У реченні **SELECT** часто використовується додаткове ключове слово **DISTINCT,** яке позначає вибір тільки тих рядків, які повністю відрізняються один від одного.

Приклад 2.12. Отримати значення міст розташування відділів без повторювань (рис. 2.9).

```
SELECT DISTINCT LOC FROM DEPT;
```

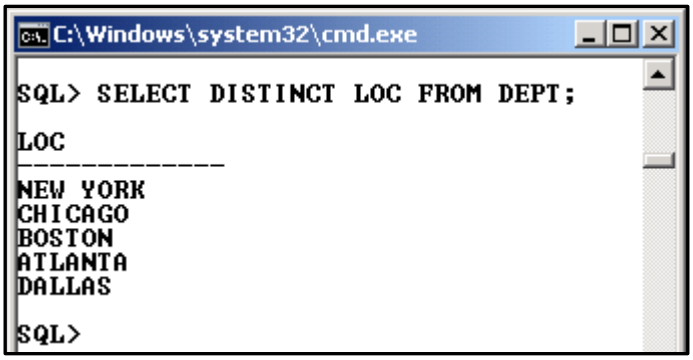


Рис. 2.9. **Перелік міст без повторювань**

2.8.3. Використання замінників імен стовпців

Для наочності подання інформації використовуються замінники імен стовпців, які дають стовпцям альтернативні заголовки у вихідних таблицях. Замінники задаються тільки у реченні SELECT одразу після відповідних імен стовпців через пробіл або за допомогою ключового слова AS. За замовчуванням, замінники переводяться в заголовні букви і не можуть містити пробіли. Рекомендується замінники, що містять пробіли та символи національних мов, включати у подвійні лапки.

Приклад 2.13. Отримати відомості про номер співробітника, його прізвище та оклад, замінивши відповідні назви стовпців на EMP_CODE, SURNAME та SALARY (рис. 2.10).

```
SELECT EMPNO Emp_Code, ENAME AS SurName, SAL SALARY
FROM EMP;
```

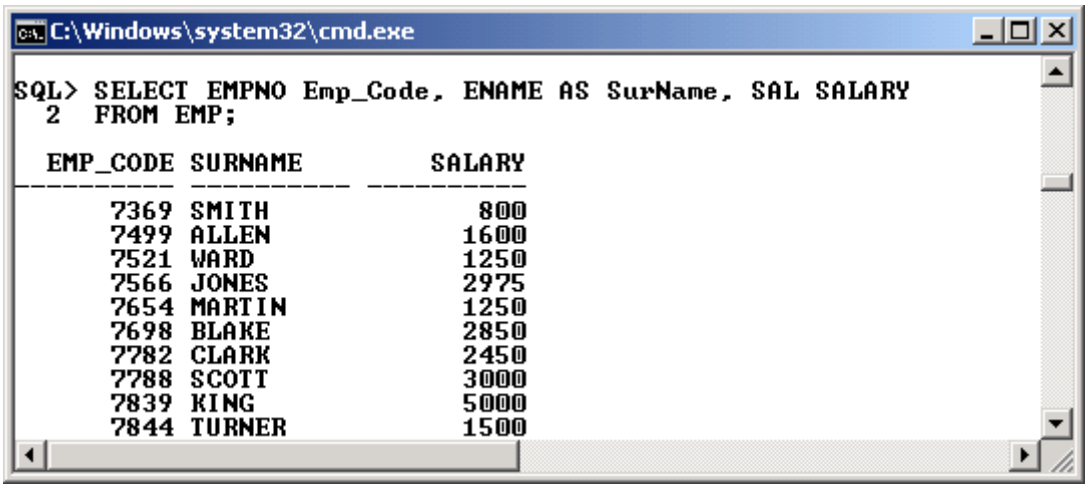


Рис. 2.10. **Приклад перейменування стовпців**

2.8.4. Обчислювані стовпці у запиті та літерали

Під час виконання команди вибірки даних часто виникає необхідність отримати додатково один або декілька стовпців, які явно не присутні у початковій таблиці, а обчислюються за допомогою арифметичних виразів чи функцій. Вираз – це комбінація одного або декількох числових значень, операторів і функцій, результатом якого є числове значення. Арифметичні вирази можуть містити імена полів, числові константи й арифметичні оператори.

До цієї ж групи можна віднести і вирази над символьними значеннями, а саме – операцію конкатенації рядків. Операція конкатенації (||) дозволяє з'єднувати стовпці з іншими стовпцями, арифметичними

виразами або константами для формування символьних рядків. Стовпці з обох сторін знака операції зливаються воєдино.

Приклад 2.14. Отримати відомості про тривалість відпустки співробітників як різницю між датами закінчення та початку відпустки (рис. 2.11).

```
SELECT EMPNO, TO_CHAR(HDATE_BEGIN,'YYYY') YEAR,
    HDATE_END – HDATE_BEGIN+1 AS DURATION
FROM VACATION
```

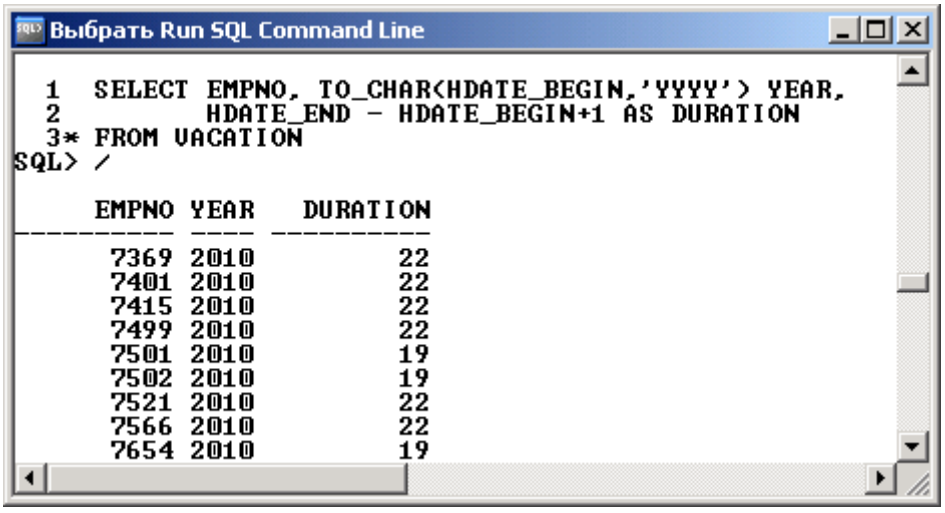


Рис. 2.11. **Приклад обчислюваного стовпця** (різниця дат)

Приклад 2.15. Підрахувати загальну річну заробітну плату співробітників без урахування премії (рис. 2.12).

```
SELECT EMPNO, ENAME, SAL*12 AS YEAR_SAL
FROM EMP;
```

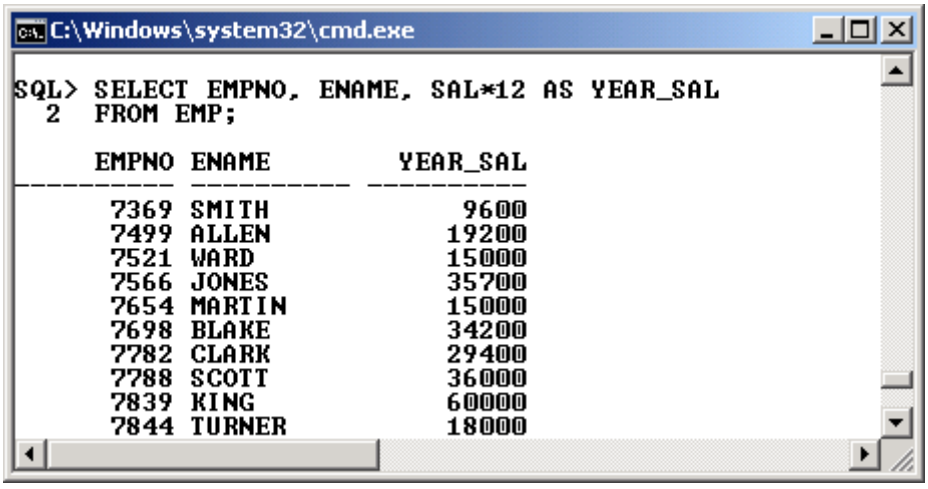


Рис. 2.12. **Приклад обчислюваного стовпця** (множення)

Приклад 2.16. Отримати відомості про номер та назву відділів у одному стовпці запиту з назвою DEPARTMENT (рис. 2.13).

```
SELECT DEPTNO || DNAME AS DEPARTMENT
FROM DEPT;
```

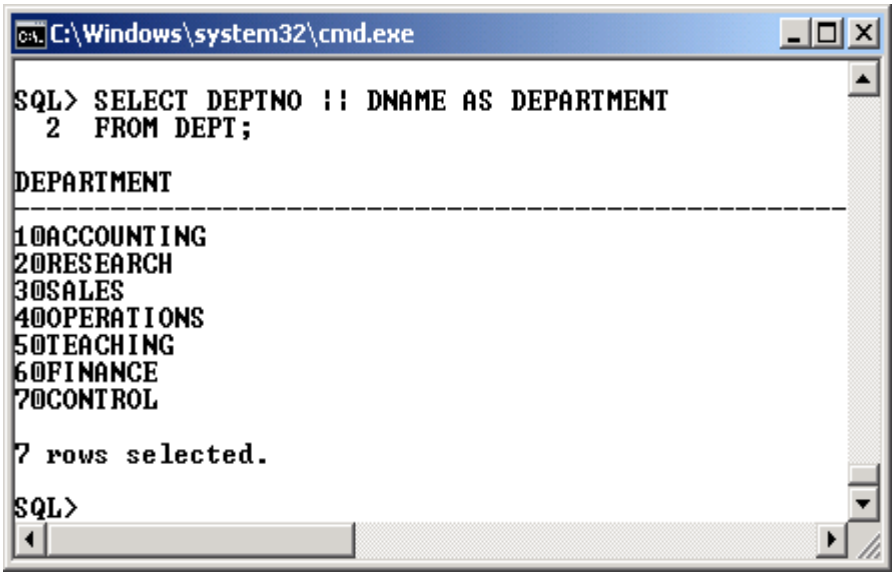


Рис. 2.13. **Приклад обчислюваного стовпця** (конкатенація)

Слід проаналізувати останній результат. По-перше, операція конкатенації виконується над полями двох різних типів: числового та символьного. У цьому випадку, якщо значення дозволяють, Oracle самостійно перетворює числа у символьний рядок, а вже потім виконує конкатенацію. По-друге, отриманий результат не є наочним, бо значення двох стовпців зливаються. Для вирішення цієї проблеми можна використовувати так звані ***літерали*** – додаткові символьні рядки, які зазвичай беруться у апострофи. Якщо літерал записується у переліку стовпців через кому, то його значення з'являється у кожному рядку окремого стовпця таблиці-результату.

Приклад 2.17. Відкоригувати попередній приклад таким чином, щоб отриманий результат став більш наочним. Для цього додати літерал у вигляді пробілу між значеннями номера на назви відділу (рис. 2.14).

```
SELECT DEPTNO || ' ' || DNAME AS DEPARTMENT
FROM DEPT;
```

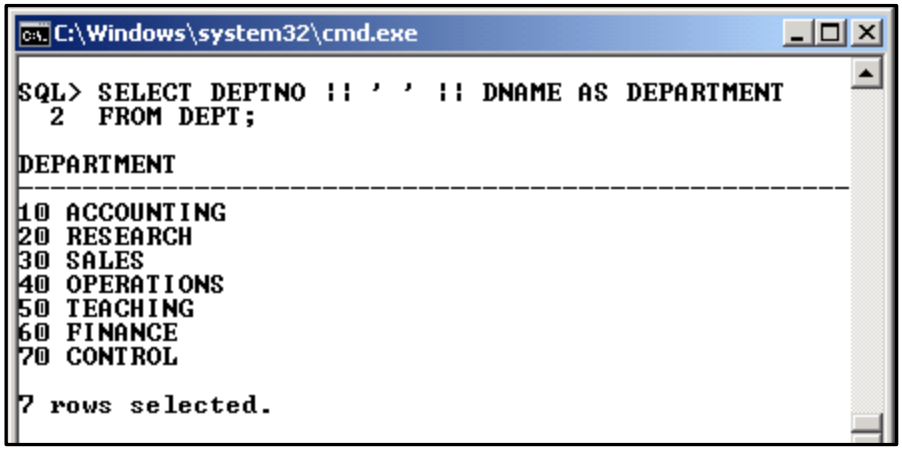


Рис. 2.14. **Використання літералу**

Приклад 2.18. Отримати інформацію про відділи, у яких працюють співробітники у вигляді:

Прізвище Номер працює у XX відділі

ENAME EMPNO works in XX department,

використовуючи окремі літерали для стовпців 'works in' та 'department' (рис. 2.15).

```
SELECT ENAME, EMPNO, 'works in', DEPTNO, 'department'
FROM EMP;
```

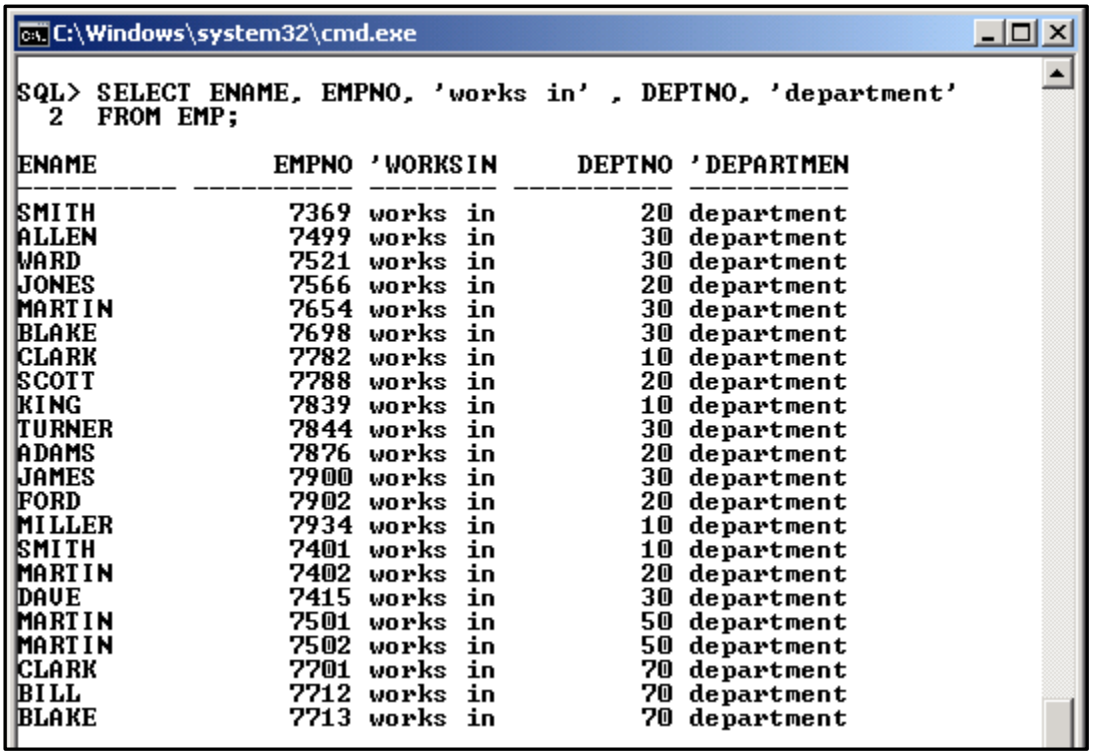


Рис. 2.15. **Приклад літералів-стовпців**

2.8.5. Обробка порожніх значень у виразах.

Якщо в таблиці поле не містить значення, воно вважається порожнім (NULL). Порожнє значення – це значення, яке недоступне, чи не присвоєне, невідоме чи невизначене. Не варто плутати порожнє значення з числом нуль або порожнім рядком. Якщо хоча б одне поле в виразі має порожнє значення, то і результат буде не визначеним.

Щоб результат обчислень, який містить посилання на стовпець, зі значенням NULL, був визначеним, необхідно у Oracle використовувати функцію NVL (функції більш детально розглянуті у підрозділі 2.17).

Функція NVL конвертує порожні значення у непорожні. Вона має вигляд:

NVL (<вираз>, <замінник порожнього виразу>);

Приклади записів функції:
NVL (стовпець_дат, '01.01.2001');
NVL (числовий_стовпець, 9);
NVL (символьний_стовпець, 'рядок');

При конвертуванні порожніх значень різних типів даних тип другого аргументу повинен відповідати типу першого аргументу.

Приклад 2.19. Отримати відомості про сумарний річний дохід співробітника з урахуванням премії. Для ілюстрації додати у запит правильний та неправильний вирази (рис. 2.16).

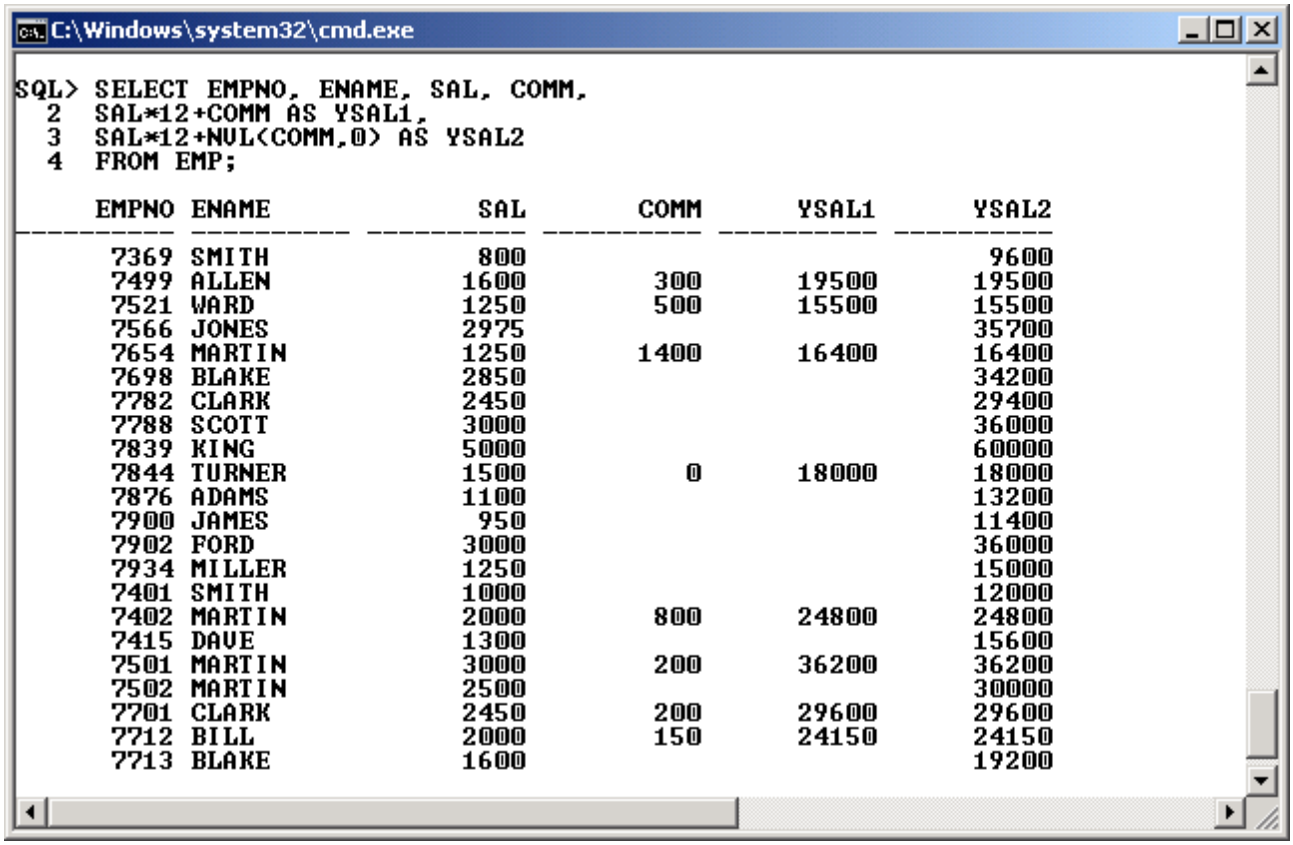
```
SELECT EMPNO, ENAME, SAL, COMM,
SAL*12+COMM AS YSAL1,
SAL*12+NVL(COMM,0) AS YSAL2
FROM EMP;
```

Як видно з отриманого результату, значення річного доходу для співробітників, які не мали премію, буде не визначеним, якщо не використати функцію NVL.

2.8.6. Упорядкування результатів вибірки даних

У всіх прикладах, що були наведені раніще, результівна таблиця містила невпорядковані рядки – вони виводилися у тому порядку, як були занесені у базу. Але часто виникають ситуації, коли необхідно отримати результат, у якому рядки впорядковані за певною умовою.

Наприклад, в алфавітному порядку для назв відділів чи прізвищ співробітників, за зростанням заробітної плати тощо.




```
SQL> SELECT EMPNO, ENAME, SAL, COMM,
  2  SAL*12+COMM AS YSAL1,
  3  SAL*12+NVL(COMM,0) AS YSAL2
  4  FROM EMP;

     EMPNO ENAME             SAL       COMM      YSAL1      YSAL2
---------- ---------- ---------- ---------- ---------- ----------
      7369 SMITH             800                              9600
      7499 ALLEN            1600        300      19500      19500
      7521 WARD             1250        500      15500      15500
      7566 JONES            2975                         35700
      7654 MARTIN           1250       1400      16400      16400
      7698 BLAKE            2850                         34200
      7782 CLARK            2450                         29400
      7788 SCOTT            3000                         36000
      7839 KING             5000                         60000
      7844 TURNER           1500          0      18000      18000
      7876 ADAMS            1100                         13200
      7900 JAMES             950                         11400
      7902 FORD             3000                         36000
      7934 MILLER           1250                         15000
      7401 SMITH            1000                         12000
      7402 MARTIN           2000        800      24800      24800
      7415 DAVE             1300                         15600
      7501 MARTIN           3000        200      36200      36200
      7502 MARTIN           2500                         30000
      7701 CLARK            2450        200      29600      29600
      7712 BILL             2000        150      24150      24150
      7713 BLAKE            1600                         19200
```

Рис. 2.16. **Використання функції NVL**

Такий результат можна отримати, використовуючи спеціальне речення **ORDER BY** (впорядкувати), яке дозволяє визначити стовпці, за значенням яких буде виконуватися впорядкування (сортування) рядків, а також напрям – за зростанням чи спаданням.

Формат речення **ORDER BY** такий:

Формат

[ORDER BY ім'я_стовпця [ASC | DESC] [,...n]],

де ім'я_стовпця вказує на стовпець, за значеннями якого буде виконуватися впорядкування рядків.

Якщо таких стовпців декілька, то вони вказуються через кому. У цьому випадку впорядкування спочатку здійснюється за значеннями першого стовпця, потім для однакових значень першого – по другому стовпцю і т. д.

Ознаки ASC та DESC вказують напрямок. ASC – за зростанням (за замовчуванням), DESC – за спаданням.

Приклад 2.20. Отримати відомості про відділи, впорядковані за зростанням їх назв (рис. 2.17).

SELECT * FROM DEPT ORDER BY DNAME;

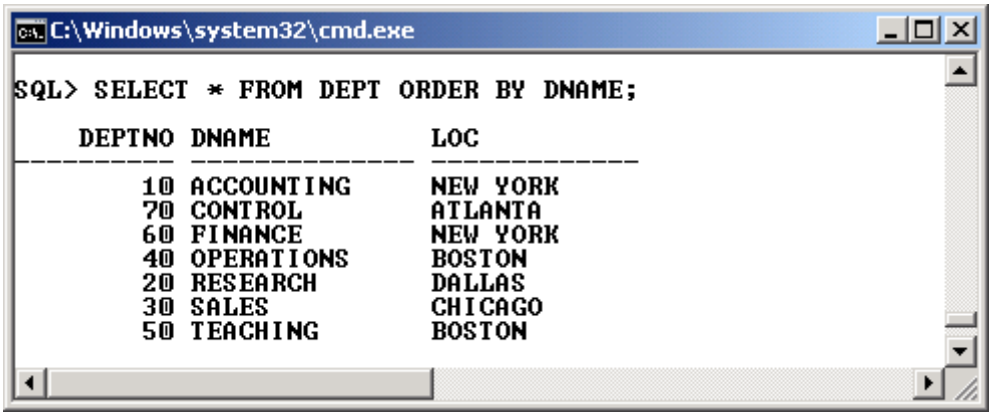


Рис. 2.17. **Впорядкування по одному стовпцю**

Приклад 2.21. Отримати відомості про заробітну плату співробітників та відділи, де вони працюють. Результат впорядкувати за зростанням номеру відділу, а у рамках певного відділу – за спаданням заробітної плати. Для наочності додати порожній рядок у результат при зміні номера відділу (рис. 2.18).

```
BREAK ON DEPTNO DUP SKIP 1
SELECT EMPNO, ENAME, DEPTNO, SAL
FROM EMP
ORDER BY DEPTNO, SAL DESC;
```

Замість назв стовпців речення ORDER BY дозваоляє вказувати порядковий номер стовпця у результаті. Тобто, якщо замість рядка

```
ORDER BY DEPTNO, SAL DESC;
```

попередня команда містила

```
ORDER BY 3, 4 DESC;
```

результат був би аналогічним.

Слід звернути особливу увагу на впорядкування за стовпцями, що містять значення NULL. У Oracle порожні зна-

чення ставляться після непорожніх при сортуванні за зростанням і на початку – при сортуванні за спаданням, тобто вважаються більшими за непорожні.

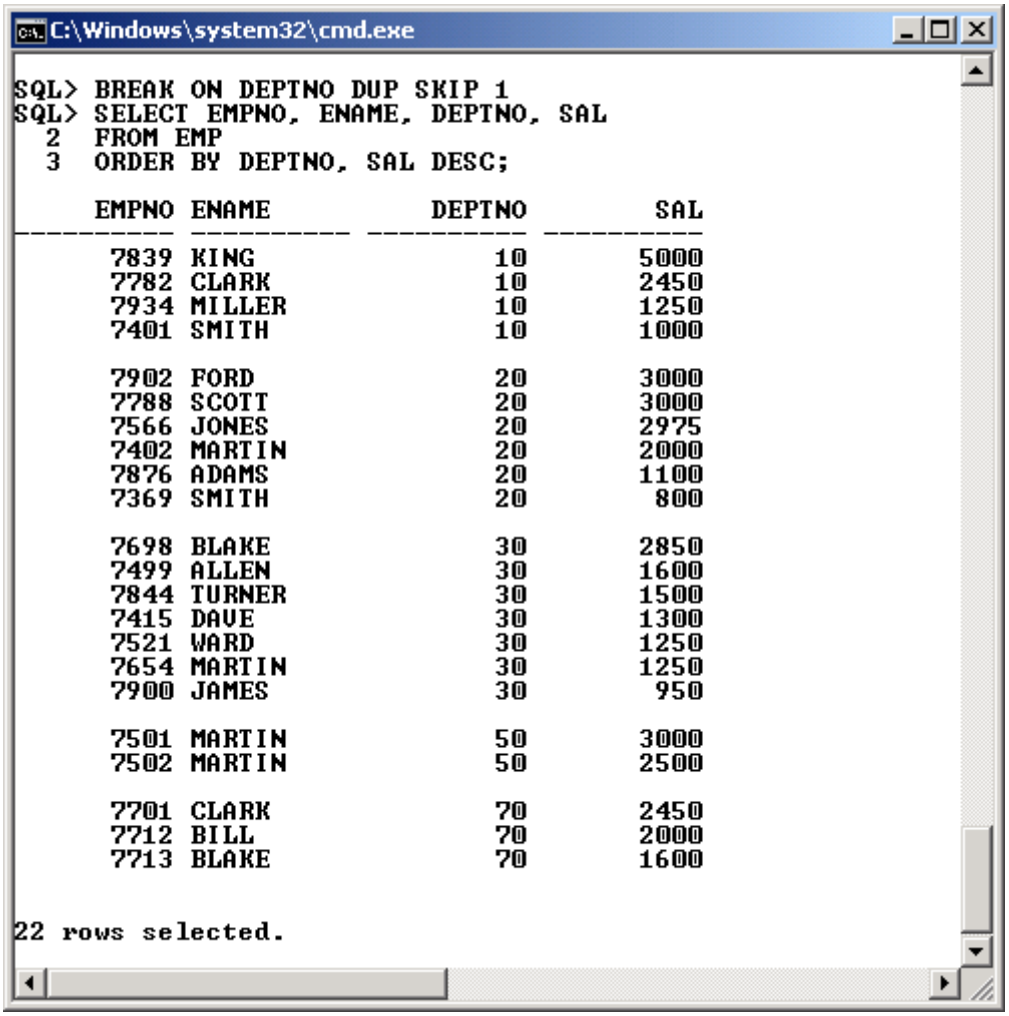


Рис. 2.18. **Впорядкування за двома стовпцями**

Приклад 2.22. Отримати відомості про співробітників за спаданням отриманої премії (рис. 2.19).

```
SELECT EMPNO, ENAME, DEPTNO, COMM
FROM EMP ORDER BY COMM DESC;
```

Слід звернути увагу на те, що для сортування за певним стовпцем, його не обов'язково вказувати у SELECT.

Приклад 2.23. Отримати відомості про співробітників за спаданням їх заробітної плати без включення відповідного стовпця до результату (рис. 2.20).

```
SELECT EMPNO,ENAME FROM EMP
ORDER BY SAL DESC;
```

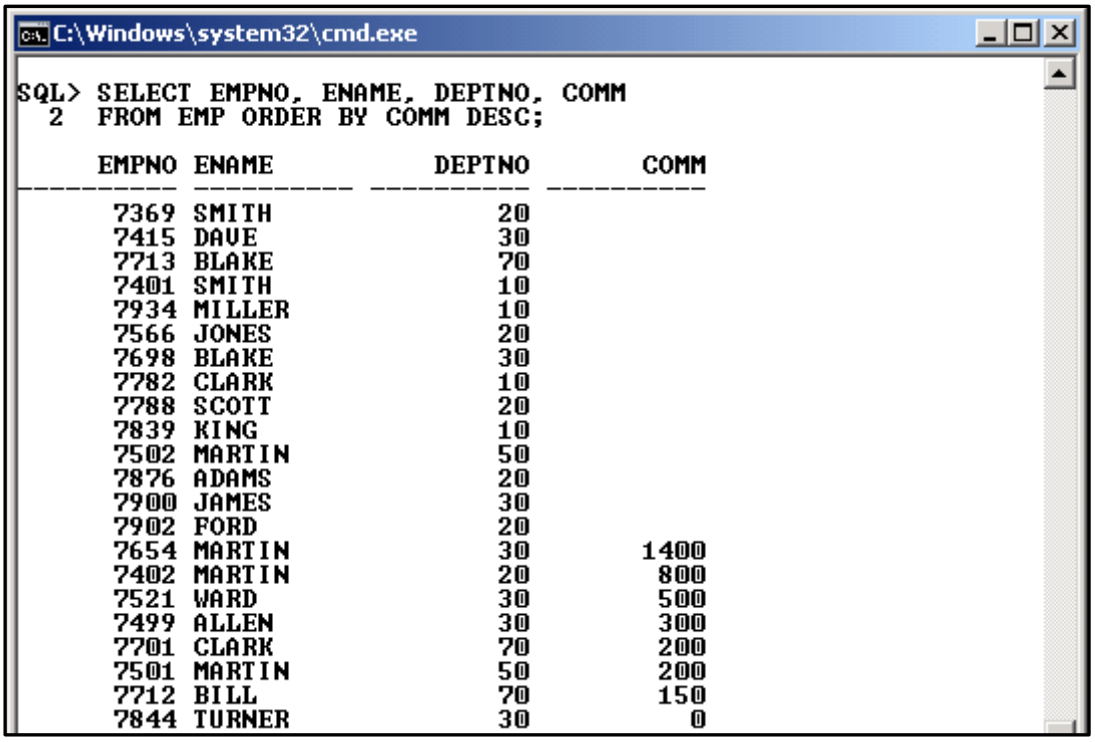


Рис. 2.19. **Сортування за стовпем, що має NULL-значення**

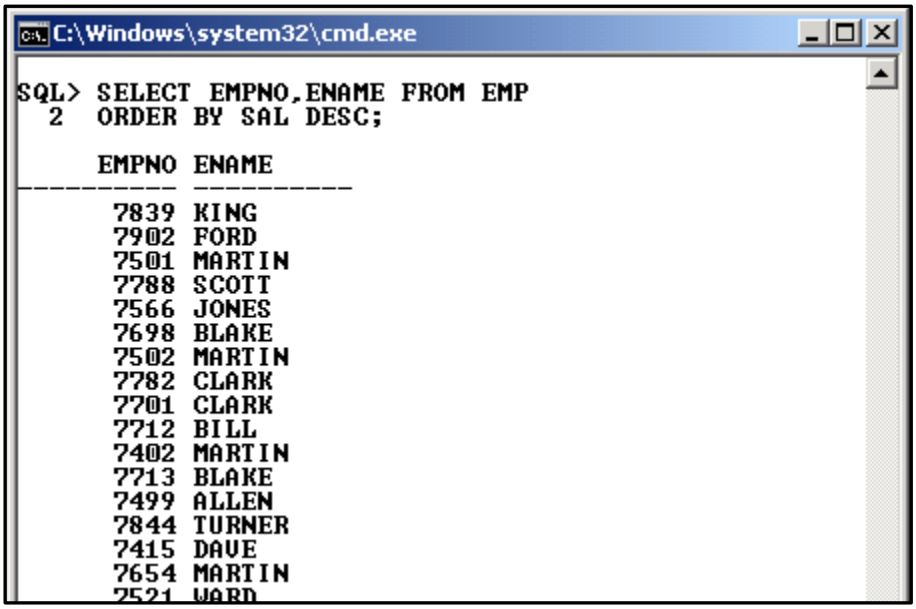


Рис. 2.20. **Сортування за стовпцем, відсутнім у результаті**

2.8.7. Використання умови пошуку. Речення WHERE

Речення **WHERE** є складовою команди **SELECT** і дозволяє задавати умову відбору рядків з таблиць, що беруть участь у вибірці. Саме ця мовна конструкція задає операцію вибірки для реляційних відношень, яка була запропонована Е. Коддом [13; 23].

У реченні WHERE задаються умови порівняння значень полів, літералів, арифметичних виразів, функцій тощо. Символьні рядки та дати в реченні WHERE беруться в одинарні лапки. У символьних рядках суттєва різниця між великими та малими буквами. Тільки спеціальні символьні функції можуть переводити рядки в той чи інший регістр.

Існує п'ять основних типів умов пошуку (чи предикатів), які застосовуються у Oracle. Вони не відрізняються від інших СКБД [14; 24]:

• **Порівняння**: порівнюються результати обчислення одного виразу з результатами обчислення іншого.

• **Діапазон**: перевіряється, чи потрапляє результат обчислення виразу в заданий діапазон значень.

• **Належність до множини**: перевіряється, чи належить результат обчислень виразу до заданої множини значень.

• **Відповідність шаблону**: перевіряється, чи відповідає певне рядкове значення заданому шаблону.

• **Значення NULL**: перевіряється, чи містить цей стовпець визначник **NULL** (невідоме значення).

Більш складні умови можна побудувати за допомогою логічних операторів **AND**, **OR** та **NOT**, а також дужок, які використовуються для визначення порядку обчислення виразу. Обчислення виразу в умовах виконується за такими правилами.

• Вираз обчислюється зліва направо.

• Першими обчислюються підвирази у дужках.

• Оператори **NOT** виконуються до виконання операторів **AND** і **OR**.

• Оператори **AND** виконуються до виконання операторів **OR**.

Порівняння, діапазон та належність до множини

Приклад 2.24. Отримати відомості про відділи, що знаходяться у Бостоні (рис. 2.21).

```
SELECT * FROM DEPT
WHERE LOC='BOSTON';
```

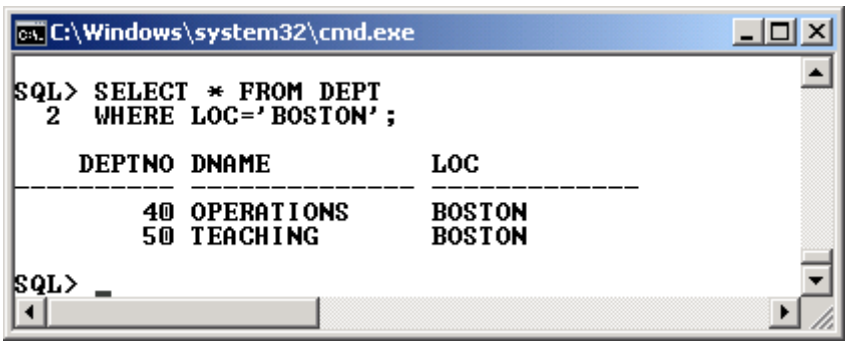


Рис. 2.21. **Вибірка з умовою на порівняння**

Приклад 2.25. Визначити співробітників, чий оклад знаходиться в межах від 1000 до 1500 (рис. 2.22).

```
SELECT * FROM EMP
WHERE SAL BETWEEN 1000 AND 1500;
```

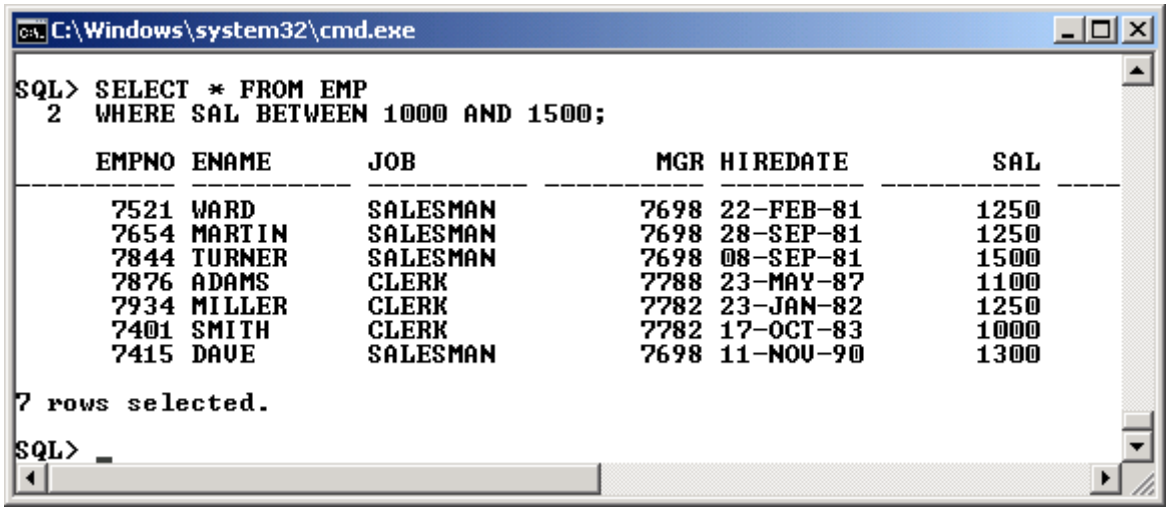


Рис. 2.22. **Вибірка з умовою на діапазон**

Той же результат отримується, якщо умову записано так:
WHERE SAL>= 1000 AND SAL<=1500;

Якщо в умові використати операцію заперечення NOT, то це змінить умову на протилежну. Тобто запис
WHERE NOT (SAL>= 1000 AND SAL<=1500);
означає вибірку співробітників, чий оклад менший за 1 000 або більший за 2 000.

Приклад 2.26. Отримати відомості про співробітників, що працюють у відділах 20, 30 або 40 і мають оклад, вищий за 2 000 (рис. 2.23).

```
SELECT EMPNO, ENAME, JOB, SAL, DEPTNO
FROM EMP
WHERE DEPTNO IN (20,30,40) AND SAL >2000;
```

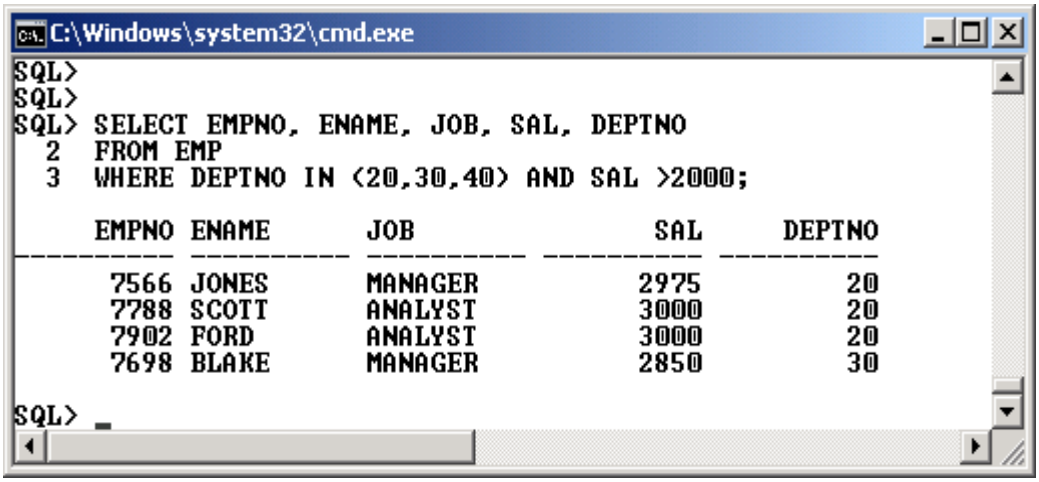


Рис. 2.23. **Вибірка за умовою "належність до множини"**

У цьому прикладі ключове слово IN задає приналежніть до множини значень 20, 30, 40. Аналогічний результат отримується, якщо умову записано таким чином:

```
WHERE (DEPTNO=20 OR DEPTNO=30 OR DEPTNO=40) AND SAL >2000;
```

Відповідність до шаблону

Деякі запити важко реалізувати з використанням операцій порівняння, діапазону чи належності до множини. Досить часто виникають ситуації, які вимагають більш "витончених" способів для формулювання умов пошуку. Наприклад: знайти співробітників, прізвище яких закінчується на **D**, або відділи, у назві яких зустрічається підрядок "**CO**". Вирішенню подібних завдань допомагає використання спеціальної конструкції **LIKE** (подібний, схожий).

За допомогою оператора LIKE можна виконувати порівняння виразу із заданим шаблоном, у якому допускається використання символів-замінників:

Символ ***% (процент)*** – замість цього символу може бути підставлена будь-яка кількість (і навіть нуль) довільних символів.

Символ ***_ (підкреслення)*** – замінює один символ рядка.

Приклад 2.27. Знайти співробітників, чиє прізвище починається на літеру M (рис. 2.24).

```
SELECT EMPNO, ENAME, JOB, SAL, DEPTNO
FROM EMP
WHERE ENAME LIKE 'M%';
```

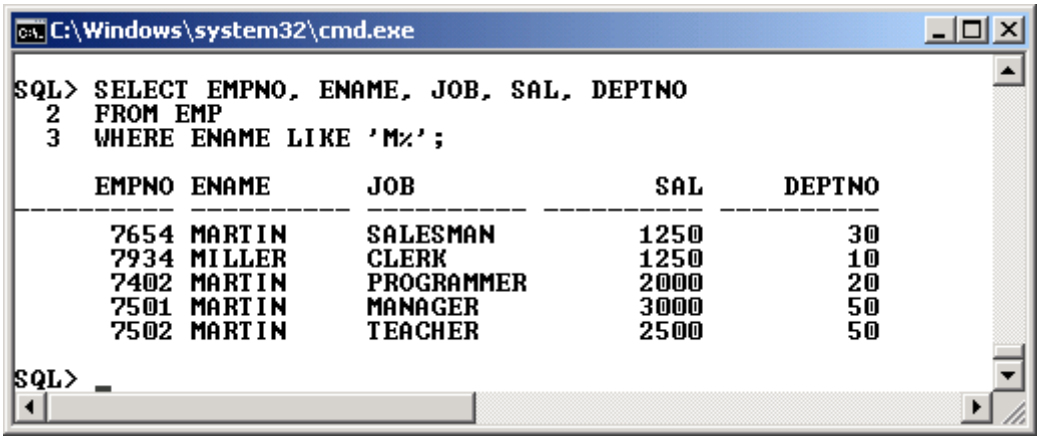


Рис. 2.24. **Використання шаблону для літери на початку слова**

Приклад 2.28. Знайти відділи, у назві яких є підрядок "**CO**" (рис. 2.25)

```
SELECT * FROM DEPT WHERE DNAME LIKE '%CO%';
```

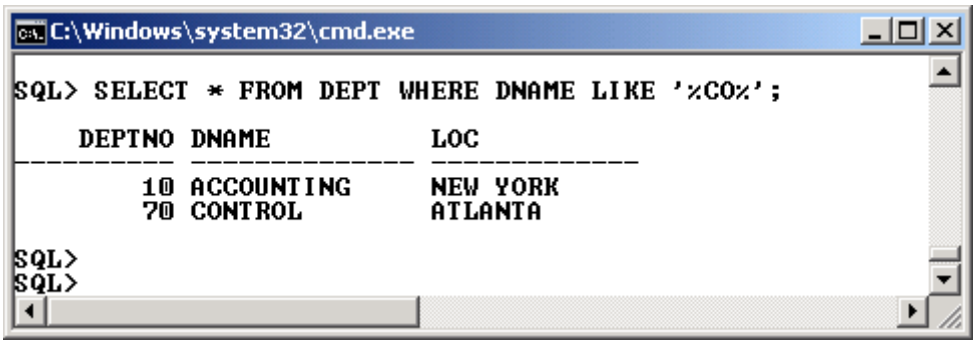


Рис. 2.25. **Використання шаблону для підрядка всередені слова**

Приклад 2.29. Знайти співробітників, які були прийняті на роботу в лютому (рис. 2.26).

```
SELECT EMPNO, ENAME, JOB, HIREDATE, SAL, DEPTNO
FROM EMP
WHERE HIREDATE LIKE '___FEB___';
```

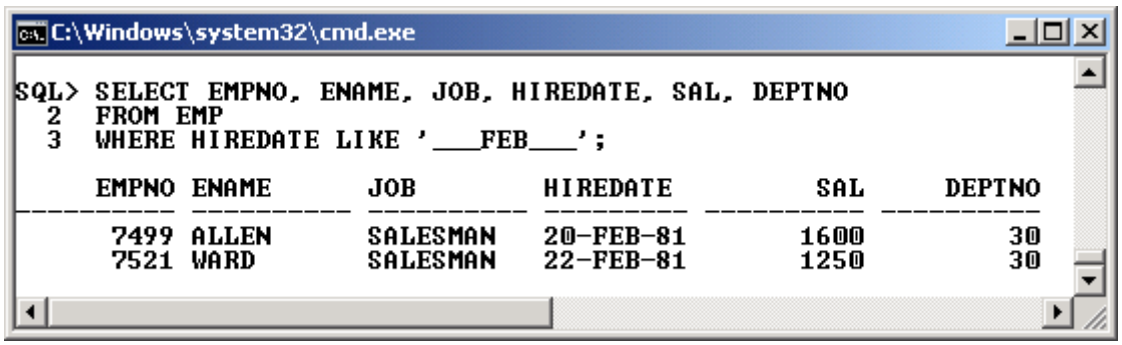


Рис. 2.26. **Використання шаблону для дати**

З використанням шаблонів до дати треба бути особливо уважним, бо він фактично застосовується до поточного формату подання дати. Якщо, наприклад, змінити формат дати командою ALTER SESSION на більш звичний:

ALTER SESSION SET NLS_DATE_FORMAT = 'DD/MM/YYYY';

то повторне виконання того ж запиту не поверне жодного рядка (рис. 2.27).

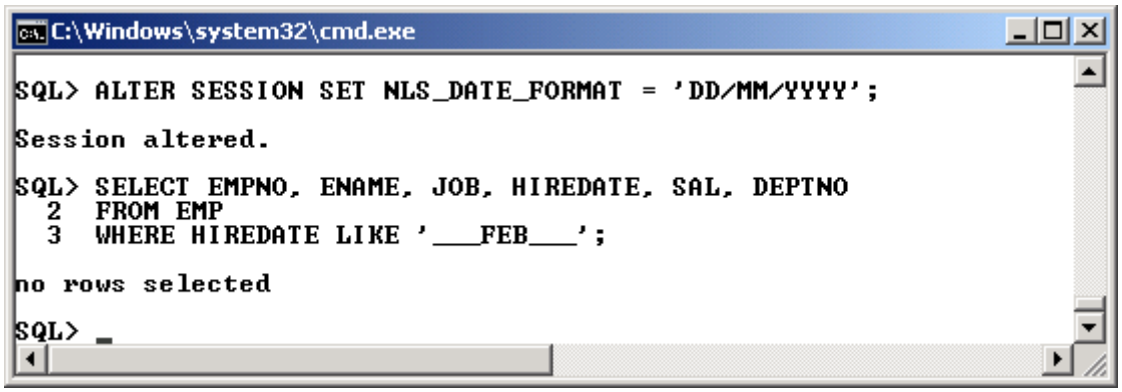


Рис. 2.27. **Результат прикладу 2.29 після зміни формату дати**

Для отримання необхідних даних умову пошуку потрібно змінити на таку: WHERE HIREDATE LIKE '___02_____', тобто три знаки підкреслення до значення місяця і п'ять – після (рис. 2.28);

```
SELECT EMPNO, ENAME, JOB, HIREDATE, SAL, DEPTNO
FROM EMP
WHERE HIREDATE LIKE '___02_____';
```

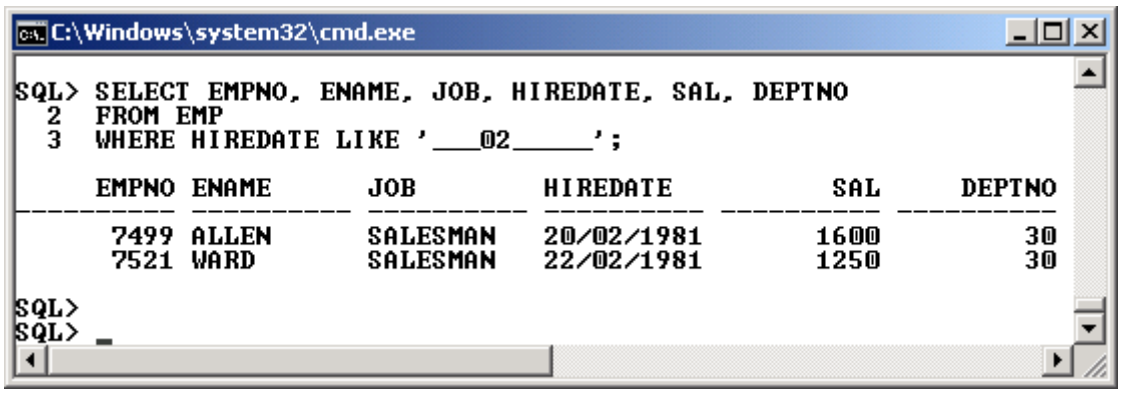


Рис. 2.28. **Результат запиту 2.29 після зміни формату дати та шаблону**

Умовні вирази з використанням значення NULL

Використання в умові пошуку значення **NULL** має свої особливості. Якщо необхідно знайти рядки у таблиці, де значення окремих стовпців не визначені (чи, навпаки, визначені), то задавати умову з використання операції "дорівнює" (=) не можна, бо це викличе непередбачуваний результат. З цією метою в мові SQL використовується особлива конструкція **IS NULL** або **IS NOT NULL**.

Приклад 2.30. Знайти співробітників, які не отримали премію (рис. 2.29).

```
SELECT EMPNO, ENAME, JOB, SAL, COMM
FROM EMP
WHERE COMM IS NULL;
```

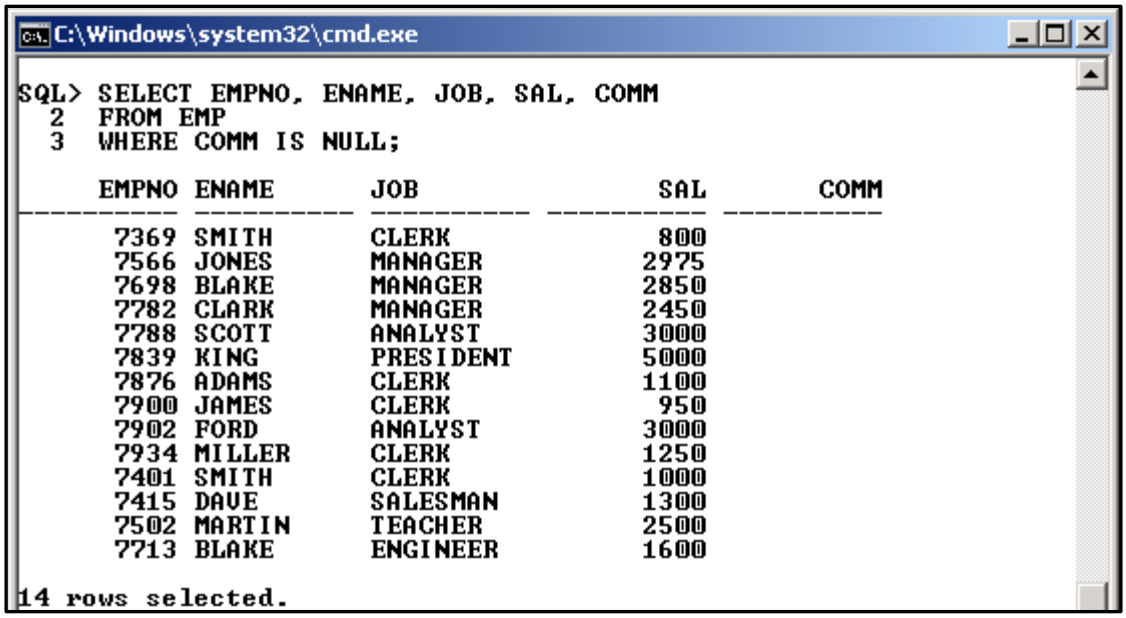


Рис. 2.29. **Правильне використання NULL в умові пошуку**

Той самий запит, але зі знаком "дорівнює", не вибере нікого (рис. 2.30).

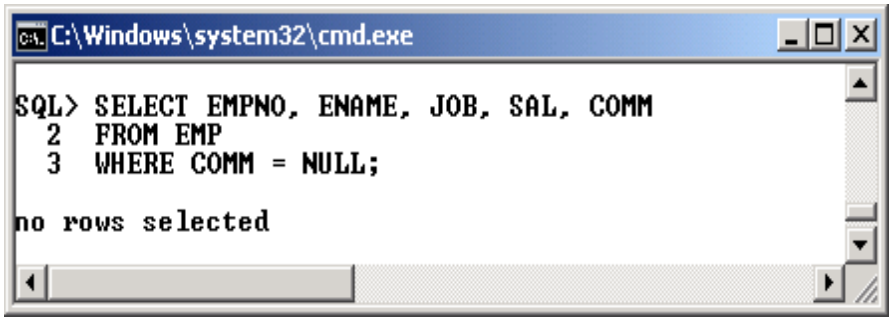


Рис. 2.30. **Помилкове використання NULL в умові пошуку**

Приклад 2.31. Знайти співробітників, що мають безпосереднього керівника та оклад більше за 2000 (рис. 2.31).

```
SELECT EMPNO, ENAME, JOB, MGR, SAL
FROM EMP
WHERE MGR IS NOT NULL AND SAL>2000;
```

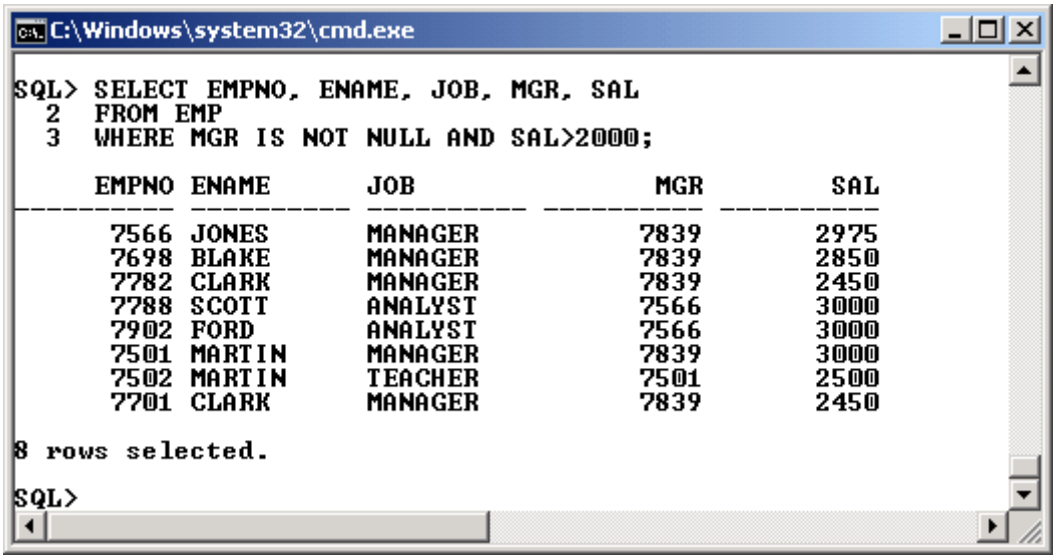


Рис. 2.31. **Правильний запис умови "не дорівнює" NULL**

Особливості використання символьних типів даних і умов у критеріях пошуку

Базові типи даних для бази даних Oracle – це символьні рядки, числа та дати. Слід зазначити, що результати пошуку за умови, що містить символьні стовпці, залежать від типу стовпця. Oracle підтримує тип

CHAR – для символьних рядків фіксованої довжини і VARCHAR2 – для рядків змінної довжини.

Для стовпців типу VARCHAR2 Oracle не додає пробілів до значення поля до довжини його стовпця та веде пошук за точним збігом значень. Для стовпців типу CHAR Oracle доповнює пробілами літерал у критерії пошуку до довжини стовпчика, який порівнюється.

Приклад 2.32. Проілюструвати відмінності у використанні умови пошуку на символьні поля типів VARCHAR2 та CHAR.

Слід зауважити, що тип даних поля ENAME у таблиці EMP є VARCHAR2(10).

Треба виконати поспіль два практично аналогічні запити до таблиці EMP для пошуку співробітників на прізвище MARTIN (рис. 2.32).

SELECT EMPNO,ENAME FROM EMP WHERE ENAME='MARTIN';

та

SELECT EMPNO,ENAME FROM EMP WHERE ENAME='MARTIN ';

Відрізняються вони лише додатковим пробілом у умові пошуку.

Отримані результати відрізняються дуже сильно, бо другий запит зовсім не повернув результат.

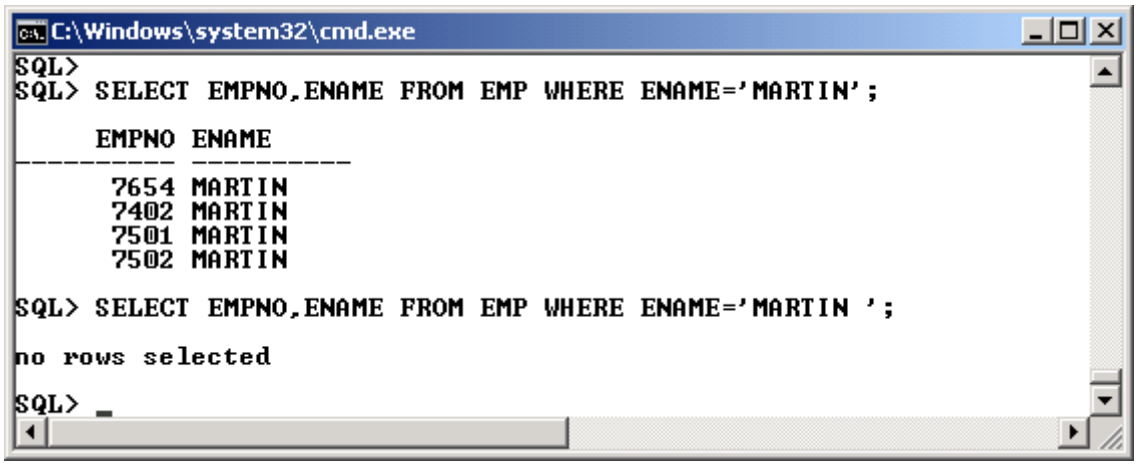


Рис. 2.32. **Умова на поле типу VARCHAR2**

Тепер потрібно змінити тип даних поля ENAME з VARCHAR2 на CHAR

ALTER TABLE EMP MODIFY ENAME CHAR(10);

і виконати ті ж самі запити (рис. 2.33).

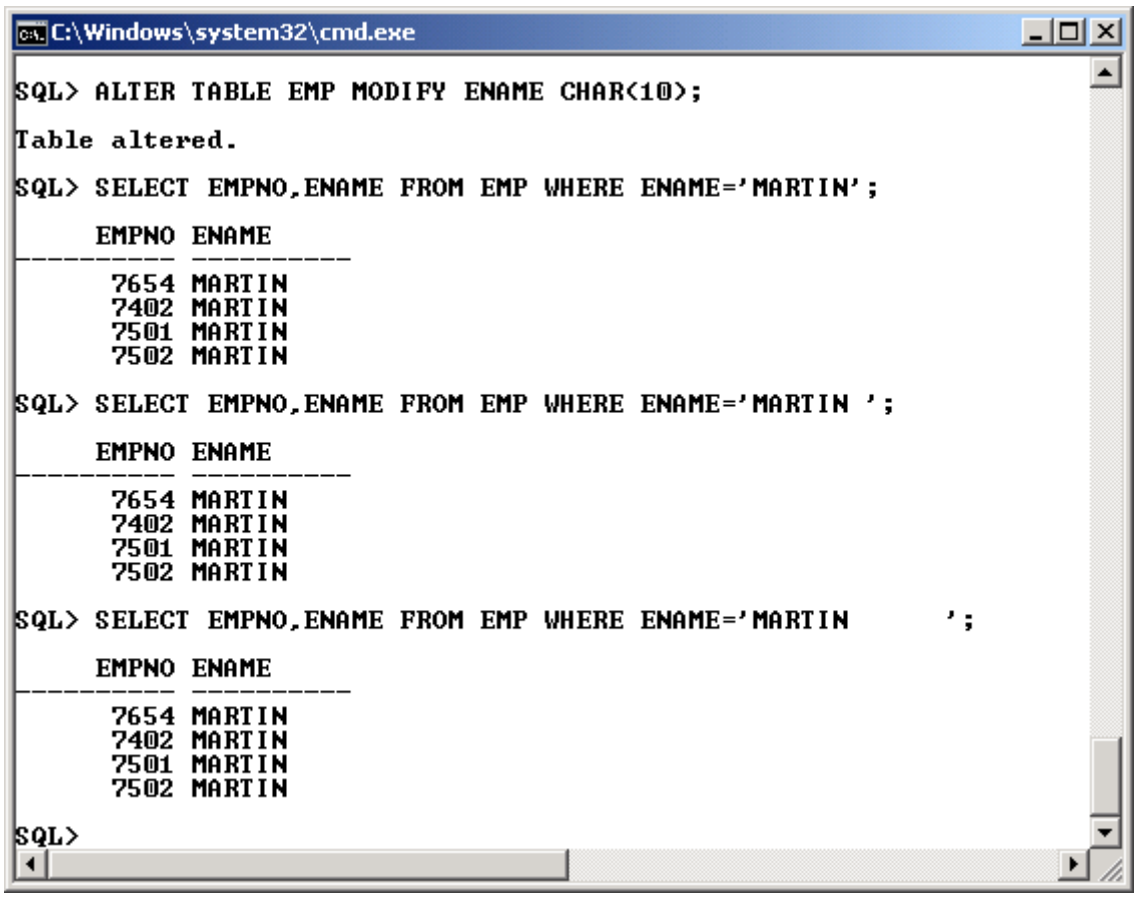


Рис. 2.33. **Умова на поле типу CHAR**

Як видно з результату, навіть, коли було додано значно більше пробілів в умову пошуку, ніж довжина поля, результат все одно отриманий.

Використання псевдостовпця ROWNUM

У мові SQL Oracle дозволяється у запитах SELECT використовувати псевдостовпець ROWNUM, що нумерує рядки результівної таблиці. Цю особливість можна застосовувати, наприклад, для обмеження кількості рядків результівної таблиці.

Приклад 2.33. Отримати інформацію про перші три відділи у таблиці DEPT (рис. 2.34).

```
SELECT  ROWNUM, DEPTNO, DNAME, LOC
FROM DEPT
WHERE ROWNUM < 4;
```

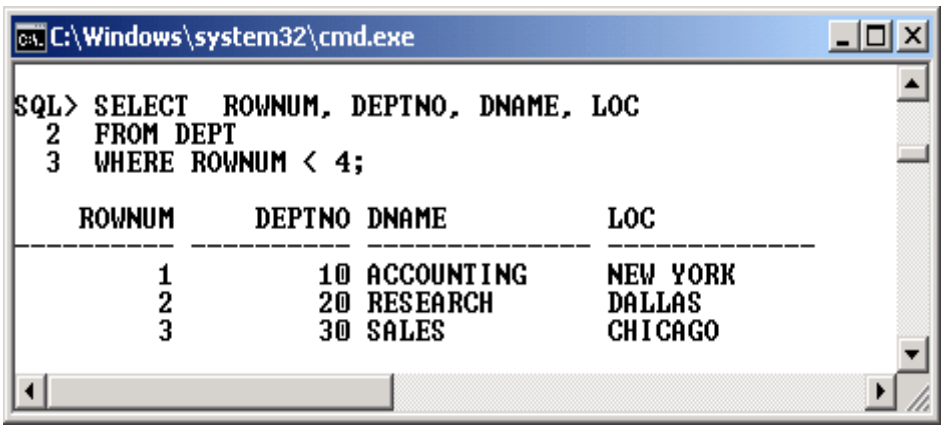


Рис. 2.34. **Використання ROWNUM у умові пошуку**

Приклад 2.34. Отримати інформацію про п'ятьох співробітників організації, що отримують найбільшу заробітну плату (рис. 2.35).

```
SELECT * FROM
      (SELECT EMPNO, ENAME, JOB, SAL, DEPTNO
       FROM EMP ORDER BY 4 DESC) T
WHERE ROWNUM < 6;
```

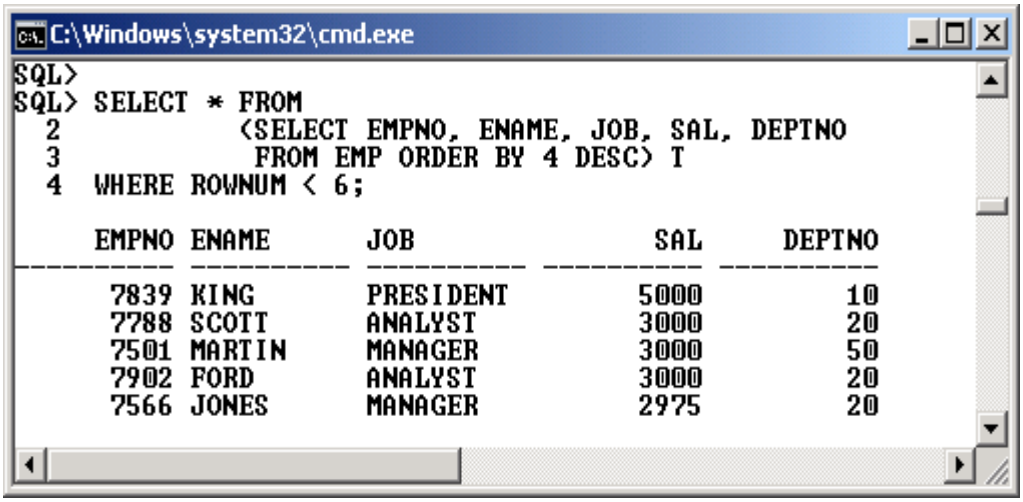


Рис. 2.35. **Використання ROWNUM під час впорядкування результату**

Слід пам'ятати, що рядки результівної таблиці нумеруються до сортування, тобто до того, як буде реалізовано ORDER BY. Саме тому у наведеному прикладі відсортована таблиця використовується як віртуальна (вкладена) для зовнішнього запиту з умовою.

Запитання і завдання

1. Чому операція вибірки даних займає найважливіше місце серед усіх команд SQL? Обґрунтуйте відповідь.

2. Опишіть призначення основних параметрів команди SELECT.

3. Які способи вибору рядків без повторень вам відомі?

4. У яких випадках використовують замінники імен стовпців? Як їх задають?

5. За допомогою яких засобів задають обчислювані стовпці у запиті?

6. Які основні типи умов пошуку застосовуються у Oracle?

2.9. Вибірка з декількох таблиць (з'єднання)

Часто виникає необхідність виконувати вибірку даних не з однієї, а з декількох таблиць. Причому така вибірка виконується навіть частіше, ніж з однієї таблиці. Причина полягає у тому, що правильно спроектована схема БД підкоряється правилам нормалізації [5], і сподіватися, що у базі буде одна або усього кілька таблиць, які містять усі необхідні дані, марна справа.

Зазвичай база даних формується з багатьох таблиць, кожна з яких є найчастіше або довідником (товари, співробітники, види депозитів тощо) або оперативною таблицею (продажі, операції по вкладах, результати змагань тощо). Аналітичні бази даних часто містять також таблиці з агрегованою інформацією за тими чи іншими показниками. Тому операція з'єднання дозволяє виконувати різноманітні запити для вибірки даних з декількох таблиць.

Для з'єднання таблиць обов'язковою є вимога, щоб вони мали один або декілька "однакових" атрибутів.

Поняття "однаковий" слід трактувати як атрибути, значення яких належать до одного домену. Звісно, можна виконати операцію з'єднання і не дотримуючись цієї вимоги, але в цьому разі отримана відповідь часто не має сенсу.

Можна пояснити на простому прикладі необхідність операцій з'єднання. Якщо потрібно отримати для співробітників інформацію щодо міста, у якому вони працюють, то даних однієї таблиці EMP буде недостатньо, оскільки відомості щодо міста розташування зберігаються у таблиці

DEPT. Однаковим атрибутом у цьому випадку буде DEPTNO (код відділу), який є як у першій так і у другій таблиці.

Інший приклад: необхідно скласти звіт про відпустки співробітників за певний період. Зрозуміло, що в разі виведення результату, який базується тільки на таблиці VACATION і містить тільки коди співробітників, мало що буде зрозуміло. Прізвища співробітників зберігаються у таблиці EMP. Звідси випливає, що для отримання результату, як мінімум, потрібні дві таблиці – VACATION та EMP (однаковий атрибут – EMPNO). А якщо у звіті потрібні відомості і про відділ, то додається третя таблиця – DEPT.

Така ситуація зустрічається, коли інформація вибирається з оперативних таблиць, які зазвичай мають тільки посилання на таблиці з довідниковою чи нормативно-довідниковою інформацією, а назви показників, об'єктів, предметів тощо знаходяться у довідниках.

Для запису операції з'єднання можна використовувати два варіанти команди **SELECT**. У першому разі з'єднання за спільним атрибутом або атрибутами описується в реченні **WHERE**, де найчастіше вказується операція порівняння між "однаковими" стовпцями таблиць, а в другому – у реченні **FROM** з використанням спеціального ключового слова – операції **JOIN** (з'єднати).

Класична операція з'єднання у мові SQL позначається як **INNER JOIN** і може мати дві модифікації: за операцією "дорівнює" – еквіз'єднання та операціями відмінними від "дорівнює", наприклад "більше", "не дорівнює" тощо. Це так зване тета-з'єднання. Часто зустрічаються модифікації класичного з'єднання – зовнішні з'єднання або ліве та праве з'єднання (**LEFT JOIN** та **RIGHT JOIN**), повне зовнішнє з'єднання **FULL OUTER JOIN** До цього ж типу операцій можна віднести операцію декартового добутку.

Реалізація еквіз'єднання у Oracle може бути здійснена також додатковими мовними конструкціями з використанням фраз NATURAL JOIN та USING [65; 66] (див п. 2.9.2).

Рядки таблиці із значеннями **NULL** не беруть участь у з'єднанні та не потрапляють до результату вибірки.

2.9.1. З'єднання таблиць у реченні WHERE

Для з'єднання таблиць необхідно виконати, як мінімум, такі кроки: у реченні **SELECT** вказати перелік атрибутів, які необхідно вибрати; у реченні **FROM** через кому вказати імена таблиць, що з'єднуються; у реченні **WHERE** записати умови з'єднання з використанням операції "дорівнює" для спільних стовпців, що об'єднуються булевими операціями AND, якщо таблиць більше за дві; якщо ж імена спільних стовпців у різних таблицях однакові, їх необхідно уточнити, використавши повну форму:

ІМ'Я_ТАБЛИЦІ. ІМ'Я_СТОВПЦЯ.

Приклад 2.35. Отримати відомості про міста, де працюють співробітники. Виконати з'єднання через конструкцію WHERE (рис. 2.36).

```
SELECT EMPNO, ENAME, EMP.DEPTNO, DNAME, LOC
FROM EMP, DEPT
WHERE EMP.DEPTNO=DEPT.DEPTNO;
```

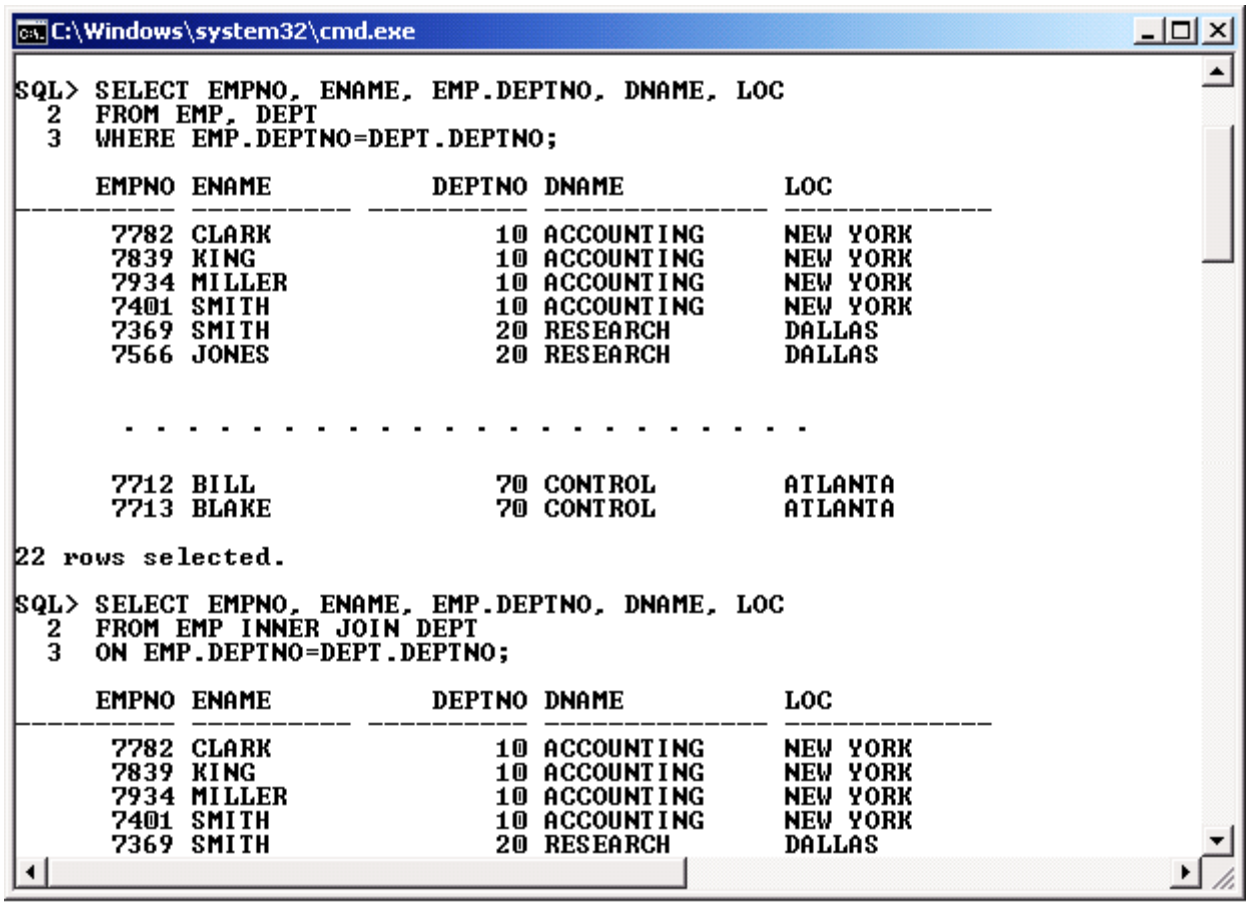


Рис. 2.36. **З'єднання двох таблиць у конструкції WHERE**

Приклад 2.36. Отримати відомості про співробітників, яким були надані відпустки влітку 2014 року. У звіт додати інформацію про відділи та міста, де працюють співробітники. Результат упорядкувати за датою надання початку відпустки (рис. 2.37).

```
SELECT EMP.EMPNO, ENAME, HDATE_BEGIN,
       HDATE_END, DNAME, LOC
FROM EMP, VACATION, DEPT
WHERE  EMP.EMPNO=VACATION.EMPNO AND
        EMP.DEPTNO=DEPT.DEPTNO AND
HDATE_BEGIN  BETWEEN '01/06/2014' AND '31/08/2014'
ORDER BY HDATE_BEGIN;
```

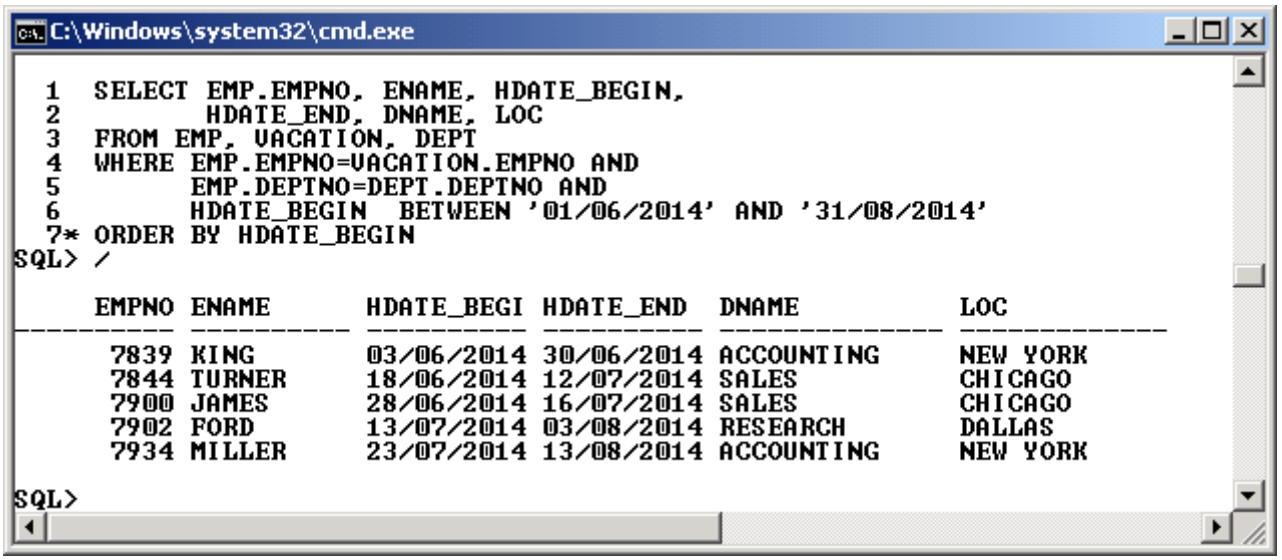


Рис. 2.37. **З'єднання трьох таблиці у конструкції WHERE**

2.9.2. З'єднання таблиць з використанням JOIN

Для здійснення з'єднання таблиць можна також використати ключове слово **JOIN**. Воно задає однойменну операцію реляційної алгебри. У запиті операції порівняння, які були у реченні **WHERE**, переносяться в речення **FROM** з ключовим словом **ON**.

Конструкція **JOIN** має такий загальний вигляд:

Формат

FROM Таблиця1 **[INNER | LEFT | RIGHT] JOIN** Таблиця2 **ON** Таблиця1.Стовпець1=Таблиця2.Стовпець2

Умови пошуку, які не стосуються з'єднання, залишаються у реченні **WHERE**. Для порівняння обох способів виконання з'єднання слід записати попередні приклади за допомогою **JOIN**.

Приклад 2.35j. Отримати відомості про міста, де працюють співробітники. Виконати з'єднання через конструкцію JOIN (рис. 2.38).

```
SELECT EMPNO, ENAME, EMP.DEPTNO, DNAME, LOC
FROM EMP INNER JOIN DEPT
ON EMP.DEPTNO=DEPT.DEPTNO;
```

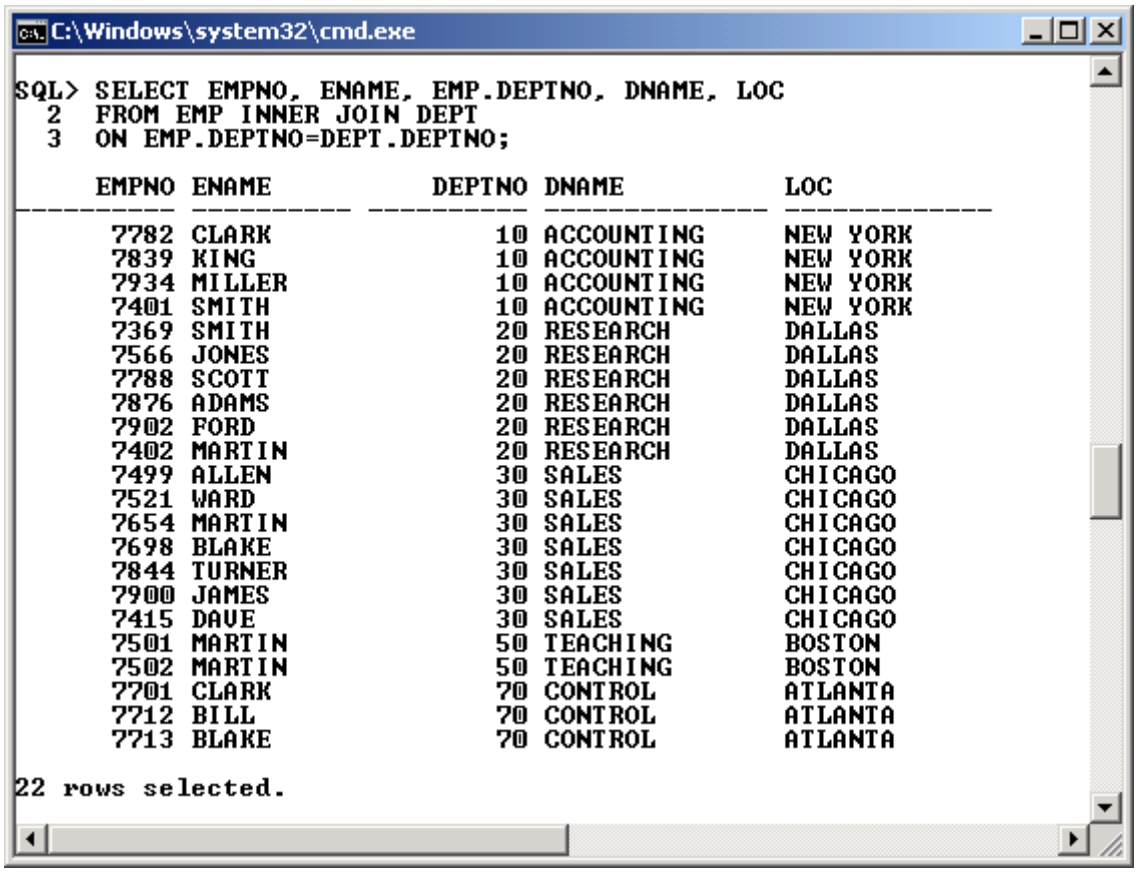


Рис. 2.38. **З'єднання двох таблиць через JOIN**

Приклад 2.36j. Отримати відомості про співробітників, яким були надані відпустки влітку 2014 року. До звіту додати інформацію про відділи та міста, де працюють співробітники. Результат впорядкувати за датою надання початку відпустки. Виконати з'єднання через конструкцію JOIN (рис. 2.39).

```
SELECT EMP.EMPNO, ENAME, HDATE_BEGIN, HDATE_END, DNAME,
LOC
FROM EMP INNER JOIN VACATION
```

```
ON EMP.EMPNO=VACATION.EMPNO
INNER JOIN  DEPT ON EMP.DEPTNO=DEPT.DEPTNO
WHERE HDATE_BEGIN  BETWEEN '01/06/2014' AND '31/08/2014'
ORDER BY HDATE_BEGIN;
```

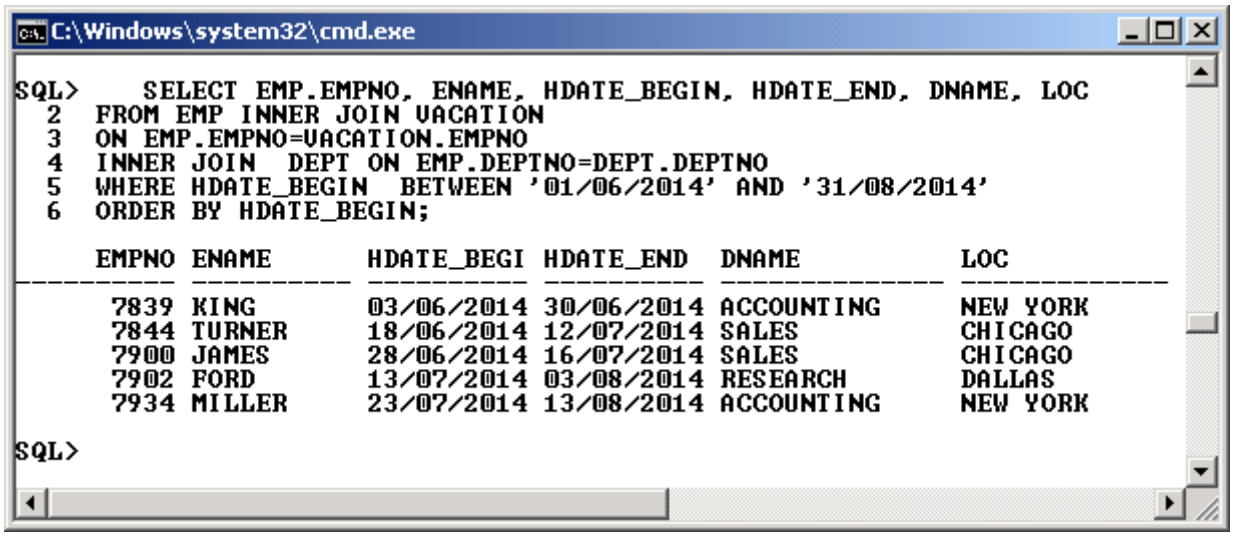


Рис. 2.39. **З'єднання трьох таблиць через JOIN**

Як вже було зазначено, здійснити еквіз'єднання таблиць у Oracle можна додатковими мовними конструкціями. Перша з них – NATURAL JOIN. Ця конструкція реалізує з'єднання неявно, порівнюючи ті імена стовпців в обох таблицях, що мають однакову назву. Результівна таблиця містить тільки один стовпець для кожної пари однойменних стовпців. Використовуючи цю конструкцію, реалізацію приклада 2.35 можна переписати таким чином:

```
SELECT EMPNO, ENAME, DEPTNO, DNAME, LOC
FROM EMP NATURAL JOIN DEPT;
```

Реалізація приклада 2.36 буде така:

```
SELECT EMPNO, ENAME, HDATE_BEGIN, HDATE_END, DNAME, LOC
FROM DEPT NATURAL JOIN EMP NATURAL JOIN  VACATION
WHERE HDATE_BEGIN  BETWEEN '01/06/2014' AND '31/08/2014'
ORDER BY HDATE_BEGIN;
```

Під час використання NATURAL JOIN уточнення імен таблиць перед однойменними стовпцями не вказується.

Важливим також є порядок у якому вказуються імена таблиць, що приймають участь у операції NATURAL JOIN, якщо їх більше двох. Це можна перевірити, якщо для приклада 2.36 записати:

FROM EMP NATURAL JOIN VACATION NATURAL JOIN DEPT.

Результат буде зовсім не той, який очікували, бо фактично таблиці слід записувати у тому порядку, як вони з'єднані у схемі бази даних (див. рис. 2.1) DEPT – EMP – VACATION.

Іншим способом реалізувати еквіз'єднання у Oracle є використання фрази USING з іменем спільного стовпця. Фактично фраза USING замінює операцію порівняння, яка записується у фразі ON.

Реалізація прикладу 2.35 за допомогою такої конструкції матиме вигляд:

```
SELECT EMPNO, ENAME, DEPTNO, DNAME, LOC
FROM EMP INNER JOIN DEPT USING(DEPTNO);
```

а прикладу 2.36 – такий вигляд:

```
SELECT EMPNO, ENAME, HDATE_BEGIN,
          HDATE_END, DNAME, LOC
FROM DEPT INNER JOIN EMP USING(DEPTNO)
               INNER JOIN  VACATION USING(EMPNO)
WHERE HDATE_BEGIN  BETWEEN '01/06/2014' AND '31/08/2014'
ORDER BY HDATE_BEGIN;
```

2.9.3. Зовнішні з'єднання

Приклади з'єднань таблиць, що розглянуті до цього часу, формували результат запиту тільки з тих рядків, які відповідали умові операції з'єднання, а саме – операції "дорівнює". Однак часто трапляються ситуації, коли необхідно, щоб результат містив усі або певні рядки з однієї таблиці, які не задовольняють умові з'єднання. У цьому разі застосовують так звані зовнішні з'єднання, які поділяються на ліві та праві.

Визначення

При лівому з'єднанні у результат запиту включаються усі рядки з лівої таблиці (таблиця є провідною), яка знаходиться ліворуч в умові з'єднання. Ці рядки з'єднуються з рядками правої таблиці у разі виконання умови з'єднання. У іншому разі стовпці правої таблиці будуть мати значення **NULL**.

При правому з'єднанні, навпаки, провідною таблицею є та, ім'я якої вказано в конструкції **JOIN** праворуч.

Приклад 2.37. Отримати інформацію про перелік співробітників, що працюють у кожному відділі (рис. 2.40).

```
SELECT DNAME, DEPT.DEPTNO, EMPNO, ENAME
FROM DEPT LEFT JOIN EMP
ON DEPT.DEPTNO=EMP.DEPTNO;
```

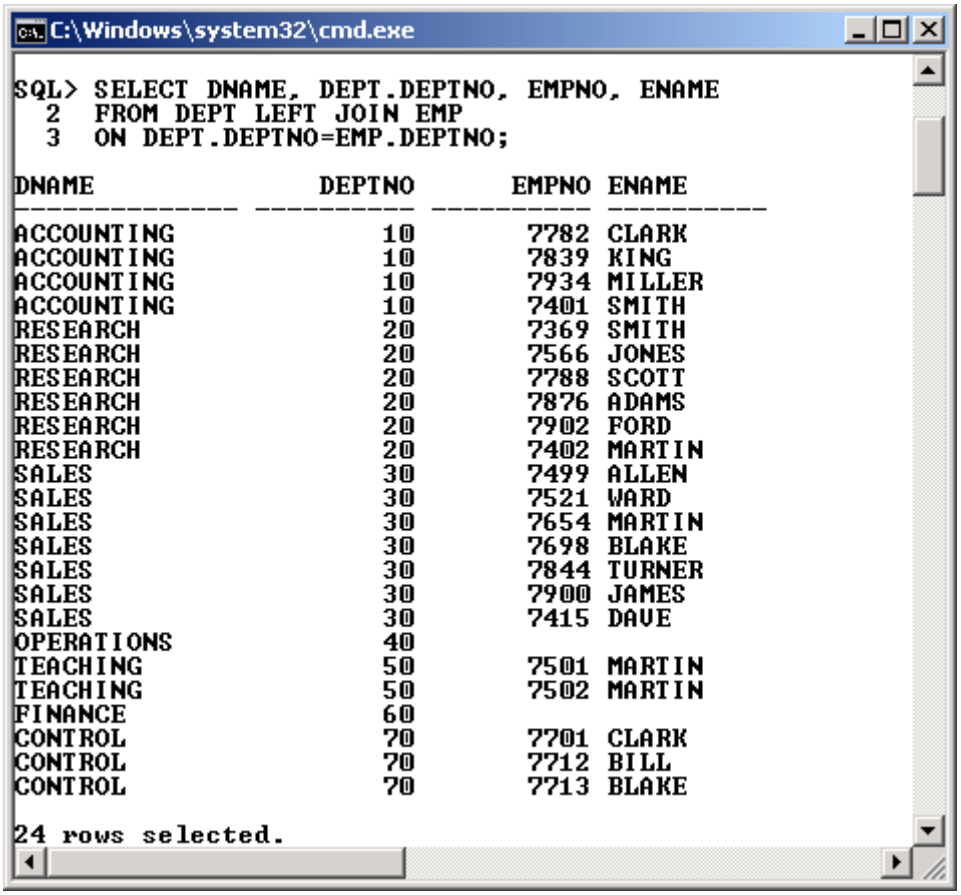


Рис. 2.40. **Результат лівого зовнішнього з'єднання**

Як видно з отриманого результату, у відділах 40-му та 60-му ніхто не працює. Якби виконувалась операція INNER JOIN, то цих відділів не було б у відповіді.

Результат операції RIGHT JOIN у цьому випадку повністю співпадає з INNER JOIN.

Приклад 2.38. Доцільно проілюструвати ще один приклад використання зовнішніх з'єднань – одночасно лівого та правого, у якому необхідно отримати інформацію щодо співробітників кожного відділу, які мали відпустки.

Цілком природно, що можна написати такий запит до БД (рис. 2.41).

```
SELECT DISTINCT DNAME, DEPT.DEPTNO, VACATION.EMPNO, ENAME
FROM EMP LEFT JOIN VACATION ON EMP.EMPNO=VACATION.EMPNO
RIGHT JOIN DEPT ON DEPT.DEPTNO=EMP.DEPTNO
ORDER BY 2;
```

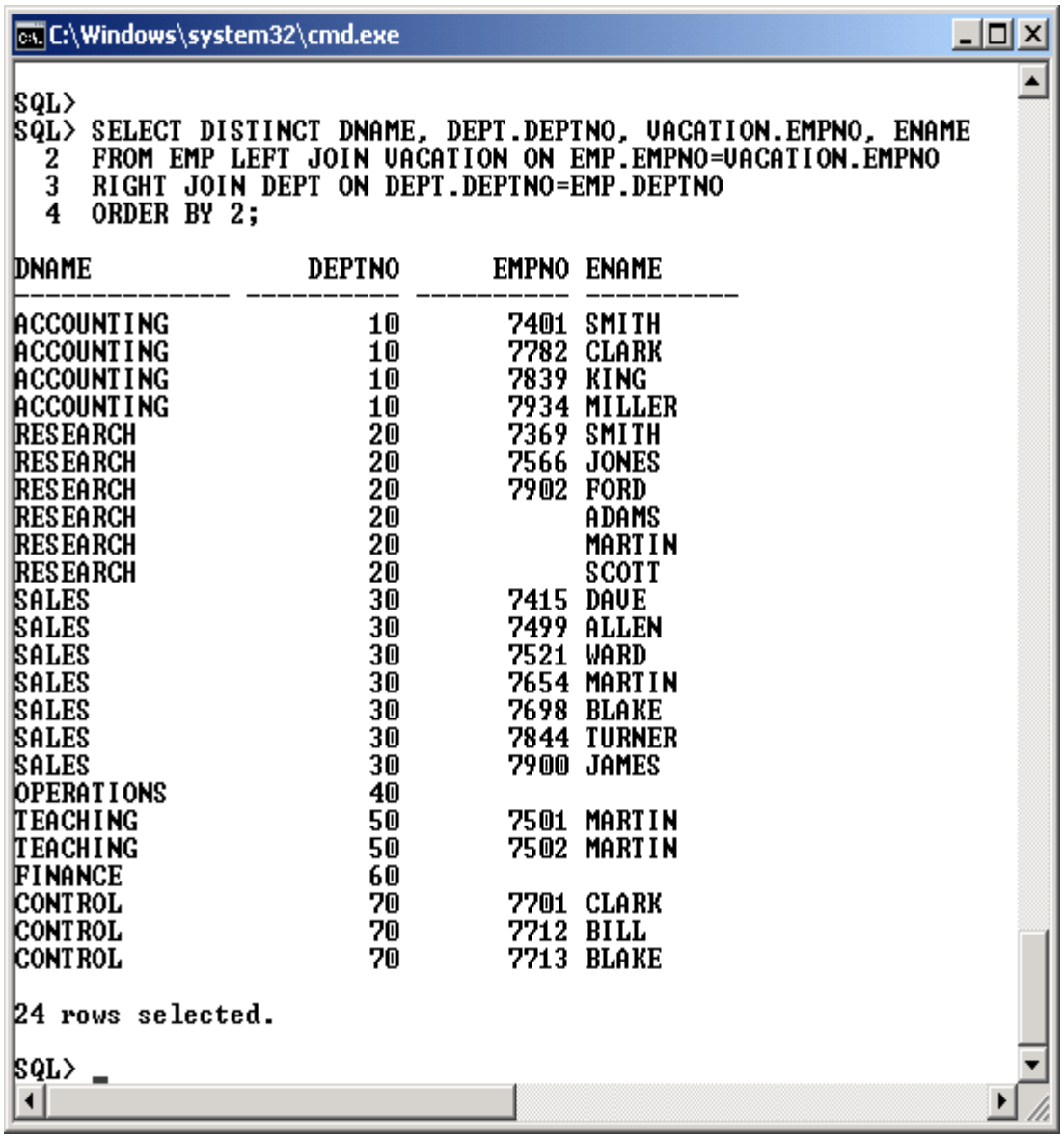


Рис. 2.41. **Результат одночасного лівого та правого з'єднання**

Однак отриманий результат дещо дивує. Якщо відсутність співробітників у відділах 40-му та 60-му вже не новина, то відсутність значень кодів деяких співробітників, а саме для ADAMS'a, MARTIN'a та SCOTT'a з 20-го відділу, викликає подив. Причина такого результату криється у тому, що, по-перше, ці співробітники не

мали відпусток (принаймні відомості про це відсутні у таблиці VACATION), а по-друге, значення поля EMPNO, береться з таблиці VACATION, а не EMP.

Якщо ж замість VACATION.EMPNO написати у запиті EMP.EMPNO, то коди співробітників з'являться у результаті і не можна буде дізнатись, хто не був у відпустці.

Таким чином, наведений приклад вказує на те, що для отримання певної інформації з БД необхідно використовувати зовнішні з'єднання й особливу увагу звертати на те, які саме (ліві чи праві) повинні використовуватись у кожному випадку.

Для зовнішніх з'єднань СКБД Oracle має особливу мовну конструкцію (оператор з'єднання), яка поміщається в умову з'єднання і позначається символом плюс, який береться в круглі дужки (+). Він знаходиться на тому боці з'єднувальної умови, що відповідає таблиці з відсутніми даними. У оператора є ефект створення одного або більшої кількості порожніх рядків, до яких будуть приєднуватись рядки другої таблиці. Тобто, якщо з'єднання ліве, знак плюс ставиться справа, якщо з'єднання праве, то знак плюс ставиться зліва.

Отже приклад 2.37 можна переписати таким чином:

```
SELECT DNAME, DEPT.DEPTNO, EMPNO, ENAME
FROM DEPT, EMP
WHERE DEPT.DEPTNO=EMP.DEPTNO(+);
```

Отриманий результат повністю співпадає з попереднім (рис. 2.42).

Необхідно також навести приклад запиту з так званим повним зовнішнім з'єднанням. Повне зовнішнє з єднання (FULL OUTER JOIN) є одночасно комбінацією лівого та правого з'єднання. Наприклад, запит:

```
SELECT DISTINCT DNAME, DEPT.DEPTNO, VACATION.EMPNO, ENAME
FROM DEPT FULL OUTER JOIN EMP ON DEPT.DEPTNO=EMP.DEPTNO
FULL OUTER JOIN VACATION ON EMP.EMPNO=VACATION.EMPNO
ORDER BY 2;
```

дасть таку ж відповідь, як і у прикладі 2.38.

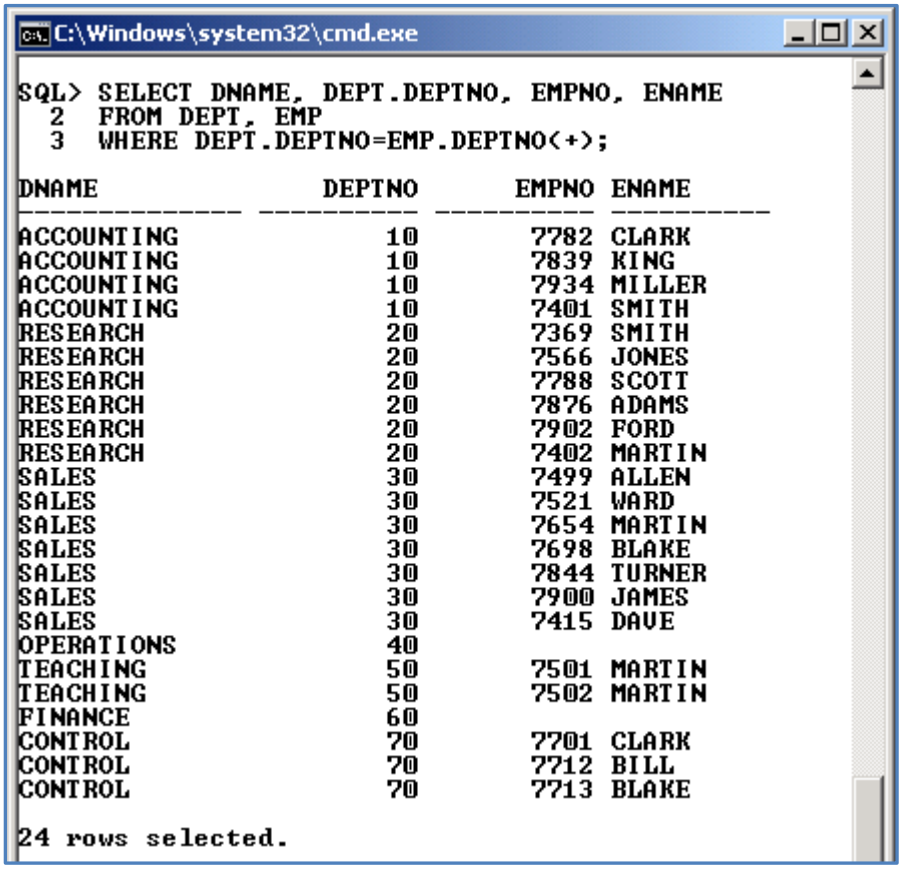


Рис. 2.42. **Результат з'єднання через (+)**

2.9.4. З'єднання за умовами, відмінними від рівності

Часто виникають ситуації, в яких потрібно з'єднати таблиці за умовами, що містять не тільки операцію порівняння, а й будь-які інші, наприклад: "менше", "не дорівнює", "більше або дорівнює" тощо. З точки зору синтаксису запиту на мові SQL ніяких особливих відмінностей не відбувається. Відрізняються запити тільки тим, що замість операції "дорівнює", що записувалась у конструкції WHERE або у конструкції ON (JOIN), записується операція порівняння, яка необхідна для вирішення того чи іншого завдання.

Приклад 2.39. Для кожного співробітника визначити рівень його заробітної плати. Для вирішення завдання слід додатково задіяти таблицю SALGRADE, яка містить інформацію щодо діапазону (інакше – мінімальної та максимальної) заробітної плати для кожного рівня (рис. 2.43).

```
SELECT EMPNO, ENAME, SAL, GRADE
FROM EMP JOIN SALGRADE ON SAL>=LOSAL AND SAL<=HISAL;
```

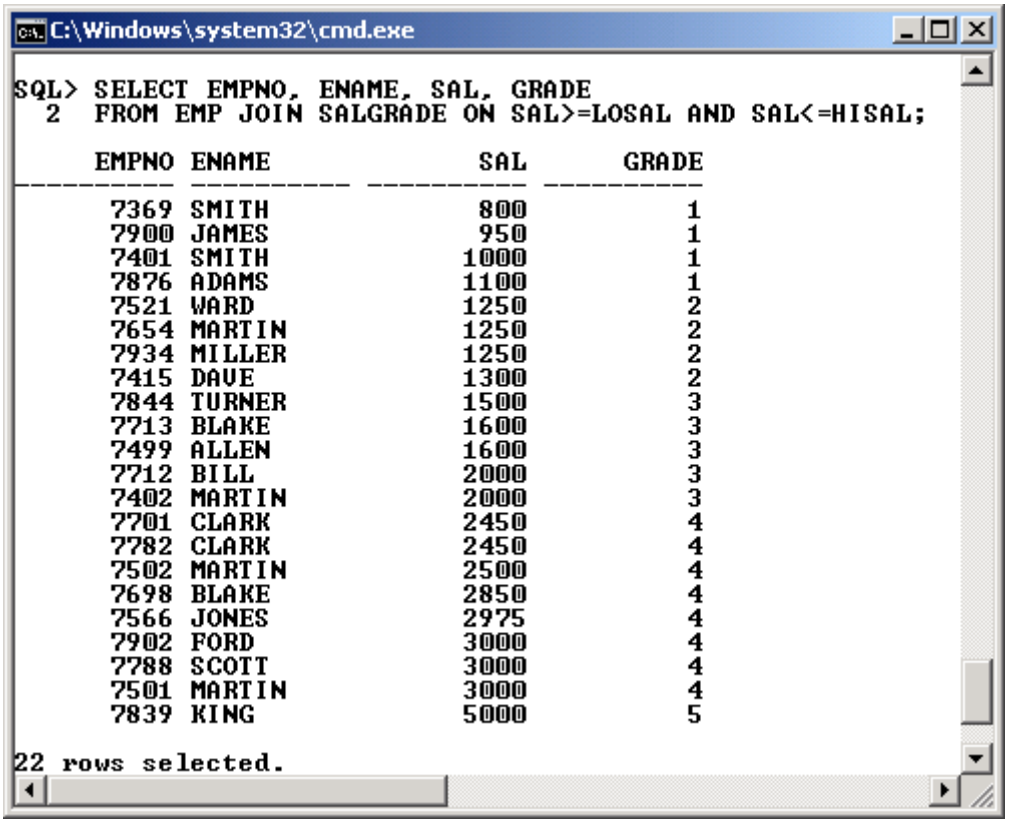


Рис. 2.43. **З'єднання за діапазоном значень**

Аналогічний результат отримується, якщо переписати запит таким чином:

```
SELECT EMPNO, ENAME, SAL, GRADE
FROM EMP,SALGRADE
WHERE SAL>=LOSAL AND SAL<=HISAL;
```

2.9.5. З'єднання таблиці з самою собою

Приклад 2.40. Отримати відомості про заробітну плату співробітників та назви відділів, де вони працюють, які отримують менше, ніж співробітник з номером 7713.

Запит, який дає відповідь на поставлене завдання, містить дві операції з'єднання: одну – між таблицями EMP та DEPT (за операцією "дорівнює" по полю DEPTNO), а іншу – між таблицею EMP з нею ж (за операцією "менше" по полю SAL). Однак, оскільки таблиця EMP у базі даних одна, то у запиті вказуються дві таблиці EMP, але кожна з них отримує своє аліасне ім'я (псевдонім) E1 та E2, відповідно. Це призводить до того, що з точки зору запиту існують дві таблиці E1 та E2.

```
SELECT  E1.EMPNO, E1.ENAME, DNAME, E1.SAL
FROM EMP E1, DEPT, EMP E2
WHERE E1.DEPTNO=DEPT.DEPTNO AND
        E1.SAL < E2.SAL  AND
        E2.EMPNO = 7713
        ORDER BY 4;
```

Результат виконання наведений на рис. 2.44

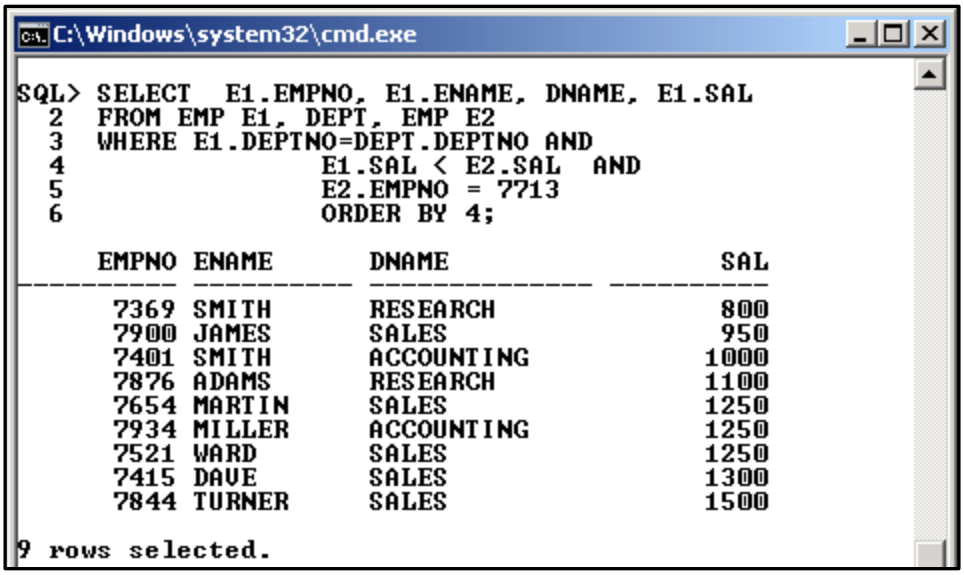


Рис. 2.44. **З'єднання таблиці з самою собою**

Аналогічний результат отримується, якщо переписати запит у такому вигляді:

```
SELECT  E1.EMPNO, E1.ENAME, DNAME, E1.SAL
FROM EMP E1 INNER JOIN DEPT ON E1.DEPTNO=DEPT.DEPTNO
INNER JOIN EMP E2 ON E1.SAL < E2.SAL
WHERE E2.EMPNO = 7713
        ORDER BY 4;
```

Запитання і завдання

1. Які вимоги накладаються на таблиці, щоб виконати їхнє з'єднання?
2. Які засоби мови SQL використовують для реалізації з'єднання таблиць?
3. У чому полягає різниця між внутрішнім та зовнішнім з'єднаннями?
4. Які особливості має СКБД Oracle під час побудови з'єднань?

2.10. Реалізація операцій реляційної алгебри мовою SQL

Як відомо, Е. Кодд у своїх роботах сформулював сім основних та одну додаткову операцію реляційної алгебри [13; 23]. До основних відносяться такі операції: об'єднання, перетин, віднімання (різниці), декартовий добуток, вибірка, проекція та з'єднання. Додаткова операція – ділення.

Результат кожної з цих операцій можна отримати, використовуючи синтаксис мови SQL.

Операції проекції, вибірки та з'єднання відносяться до спеціальних операцій і були розглянуті у підрозділах 2.8 та 2.9.

Операція вибірки формується за допомогою команди **SELECT**, яка містить речення **WHERE**. У ньому вказують умову пошуку (відбору) рядків результівної таблиці, для яких значення стовпців дають результат "істина"

Операція проекції реалізується також командою **SELECT**, яка відбирає декілька стовпців із таблиці.

Операція з'єднання формує результат на основі декількох інших таблиць.

Інші основні операції відносяться до традиційних операцій на множинах.

2.10.1. Операція декартового добутку

Визначення

Декартовим добутком двох відношень (таблиць) T1 та T2 є нове відношення T12, що містить усі можливі кортежі (рядки), які є поєднанням двох кортежів (рядків), що належать відповідно до відношень T1 і T2.

Потужність результівного відношення (тобто загальна кількість рядків) дорівнює добутку потужностей початкових відношень T1 і T2, а ступінь (кількість стовпців) – сумі ступенів T1 і T2.

Реалізація декартового добутку мовою SQL досить проста і від попередніх прикладів операції з'єднання відрізняється тим, що не потрібно вказувати умову з'єднання. Тобто у реченні **FROM** необхідно тільки вказати перелік таблиць через кому.

Цікаво

Незважаючи на те, що цю операцію можна виконати для будь-яких таблиць без обмежень, її результат, зазвичай, може не мати жодного сенсу для поточної предметної області.

Приклад 2.41. Отримати декартовий добуток для таблиць DEPT та SALGRADE (рис. 2.45).

SELECT * FROM DEPT, SALGRADE;

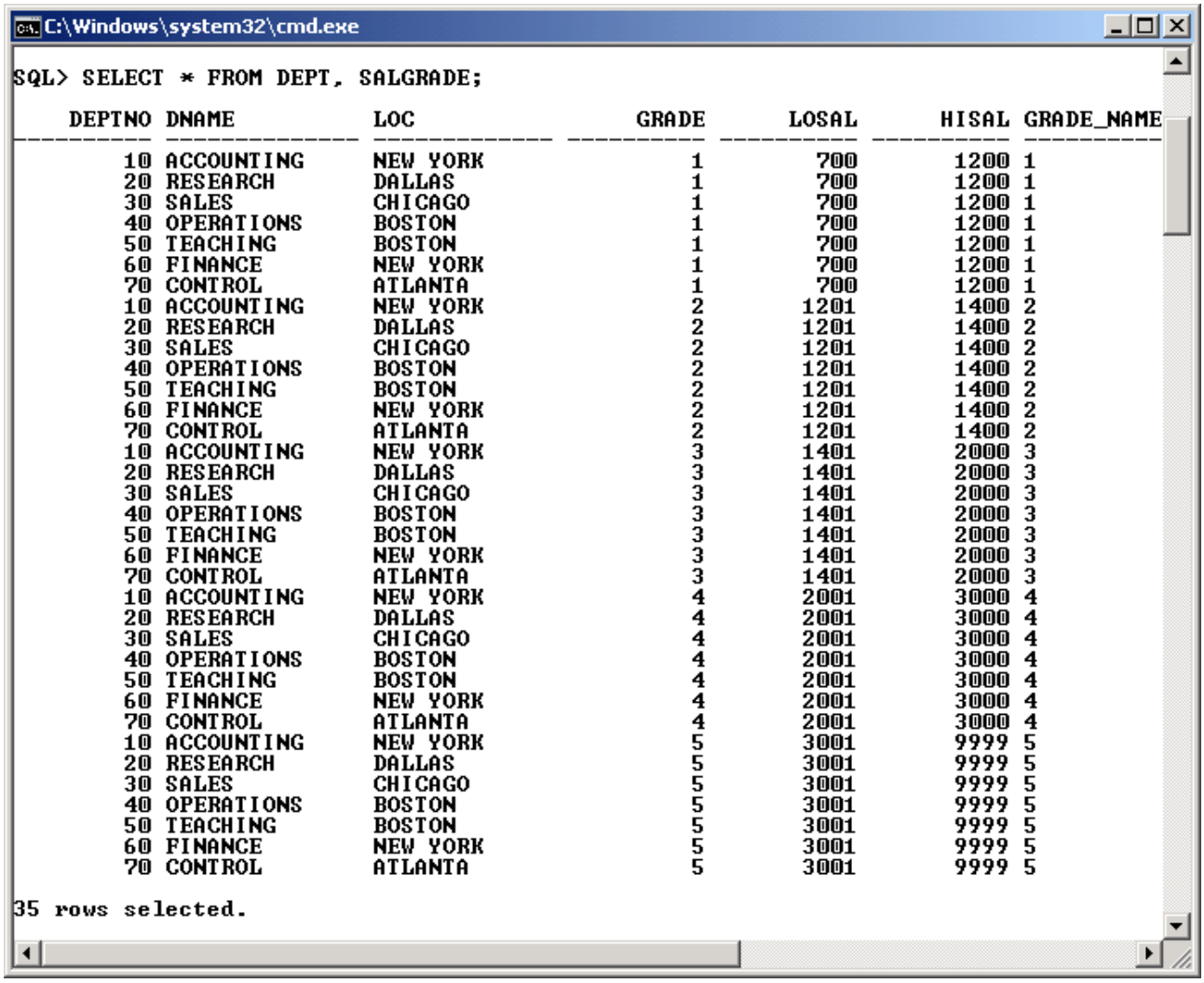


Рис. 2.45. **Результат декартового добутку DEPT та SALGRADE**

Для операції декартового добутку можна використати додаткову мовну конструкцію **CROSS JOIN**, тобто записати

SELECT * FROM DEPT CROSS JOIN SALGRADE;

що повністю аналогічно до попереднього запиту. Але такий варіант є дещо складнішим, і його можна не використовувати.

2.10.2. Операція об'єднання

Визначення

Об'єднання двох відношень R1 і R2 – це нове відношення, що містить усі кортежі, які належать до хоча б одного з R1 і R2.

Якщо виконується, наприклад, об'єднання двох таблиць або проекцій у базі даних, то результатом є нова таблиця, яка містить усі рядки (без дублювання) як з першої, так і з другої таблиці (проекції).

Важливо

Основна вимога до виконання об'єднання таблиць є такою: обидві таблиці (проекції) повинні мати однакову кількість стовпців із сумісними типами даних. Ця умова повинна виконуватись і для операцій віднімання та перетину.

У мові SQL є стандартна команда **UNION**. Вона об'єднує результати декількох команд **SELECT**, автоматично видаляючи рядки, що дублюються.

Приклад 2.42. Отримати перелік співробітників, що працюють у 10-му та 30-му відділах, використовуючи операцію UNION (рис. 2.46).

```
SELECT EMPNO, ENAME, JOB, DEPTNO FROM EMP WHERE DEPTNO = 10
    UNION
SELECT EMPNO, ENAME, JOB, DEPTNO FROM EMP WHERE DEPTNO = 30;
```

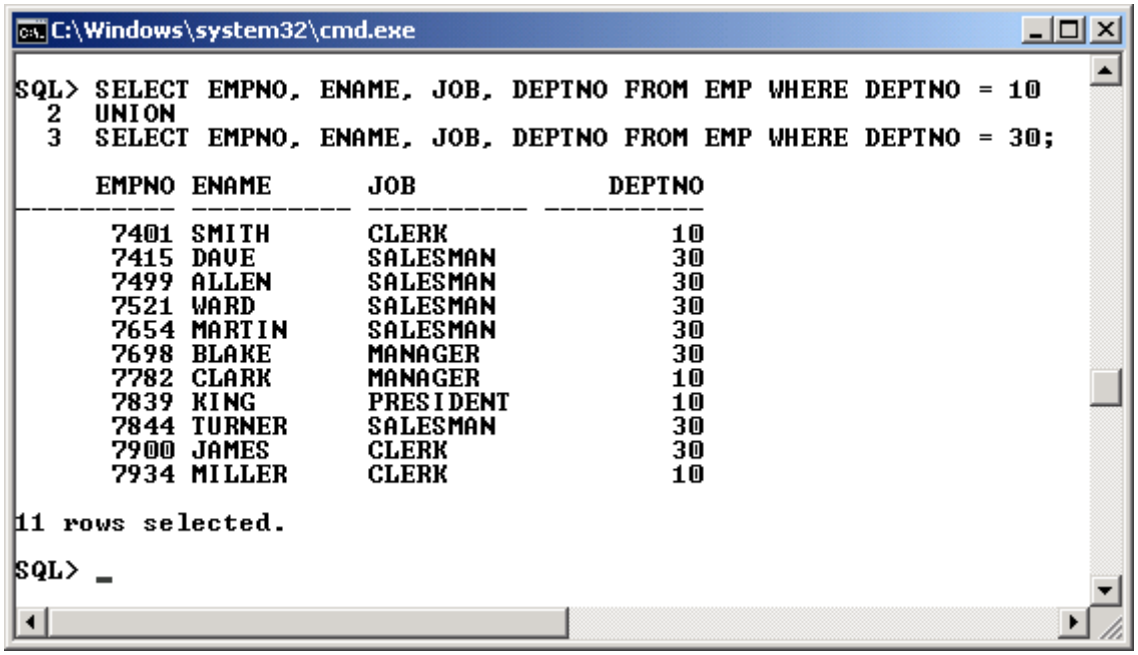


Рис. 2.46. **Результат об'єднання для співробітників**

Приклад 2.43. Отримати перелік посад, які займають співробітники, що працюють у 10-му та 30-му відділах, використовуючи операцію UNION (рис. 2.47).

```
SELECT  JOB FROM EMP WHERE DEPTNO = 10
UNION
SELECT JOB FROM EMP WHERE DEPTNO = 30;
```

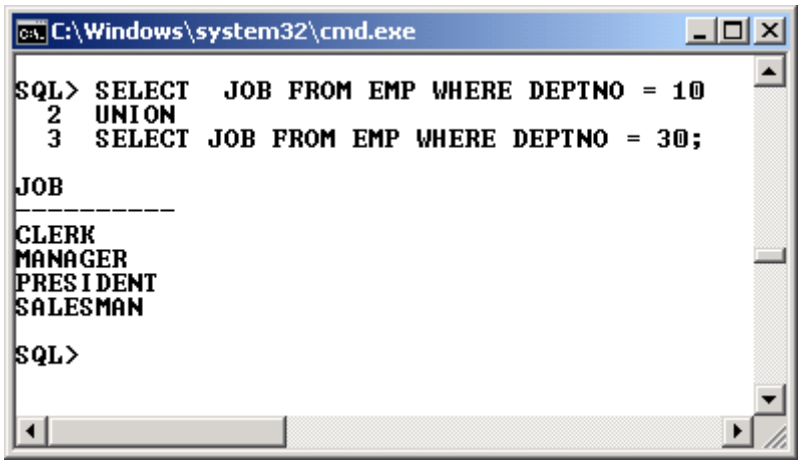


Рис. 2.47. **Результат об'єднання для поля JOB**

Як видно із отриманого результату, усі дублювання значень посад були видалені.

Важливо Якщо потрібно отримати усі рядки з об'єднання таблиць, замість операції **UNION** використовують операцію **UNION ALL**. У ній не видаляються з результату рядки, що дублюються (рис. 2.48).

```
SELECT  JOB FROM EMP WHERE DEPTNO = 10
UNION ALL
SELECT JOB FROM EMP WHERE DEPTNO = 30;
```

Корисно Операцію **UNION** можна використовувати як різновид оператора IF у мовах програмування для відображення різноманітних значень одного поля залежно від значень у інших полях [5].

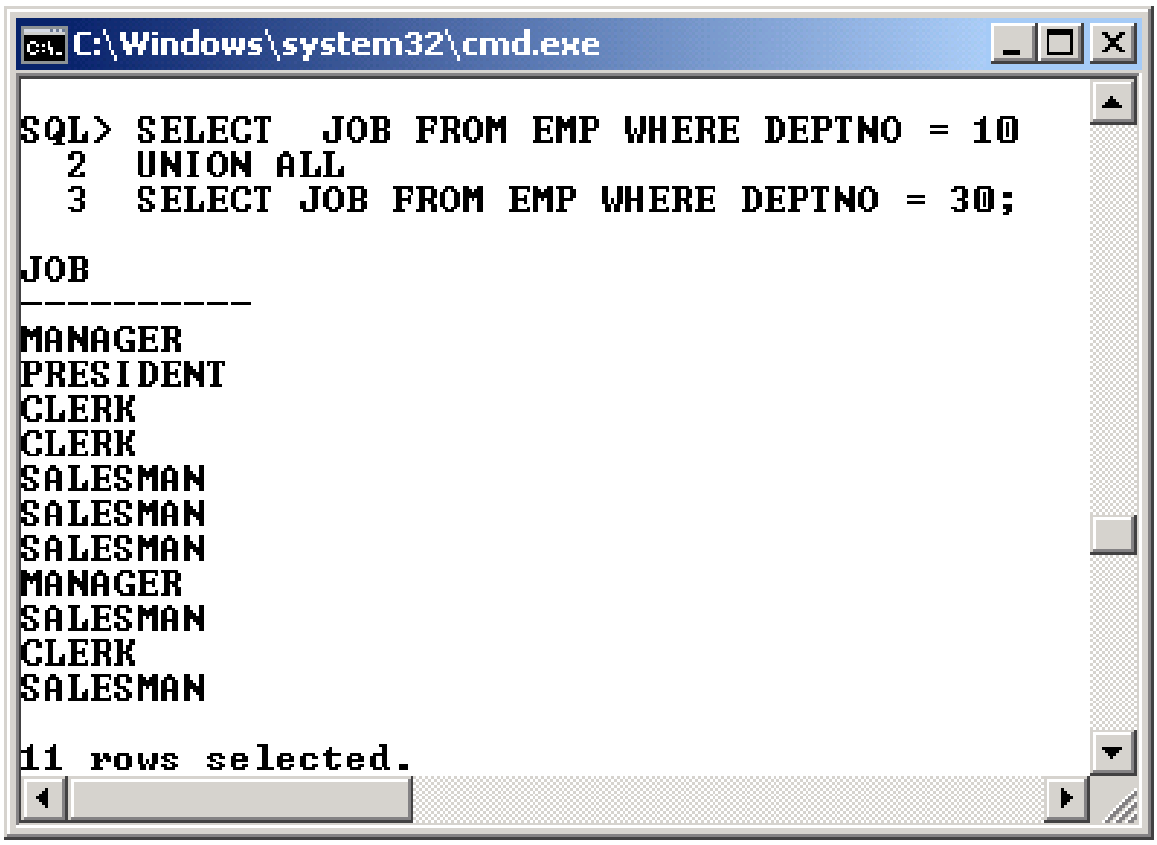


Рис. 2.48. **Результат повного об'єднання для поля JOB**

Приклад 2.44. Визначити розмір матеріальної допомоги співробітникам організації, якщо відомо, що для 10-го та 20-го відділів вона становить 50 % окладу, для 30-го відділа – 40 %, а для 50-го – 35 % (рис. 2.49).

```
COLUMN SAL_HELP FORMAT 9999.99
SELECT EMPNO, ENAME, DEPTNO, SAL*0.5 AS SAL_HELP
FROM EMP  WHERE DEPTNO IN (10,20)
   UNION
SELECT EMPNO, ENAME, DEPTNO, SAL*0.4 AS SAL_HELP
FROM EMP  WHERE DEPTNO = 30
   UNION
SELECT EMPNO, ENAME, DEPTNO, SAL*0.35 AS SAL_HELP
FROM EMP WHERE DEPTNO = 50
ORDER BY DEPTNO;
```

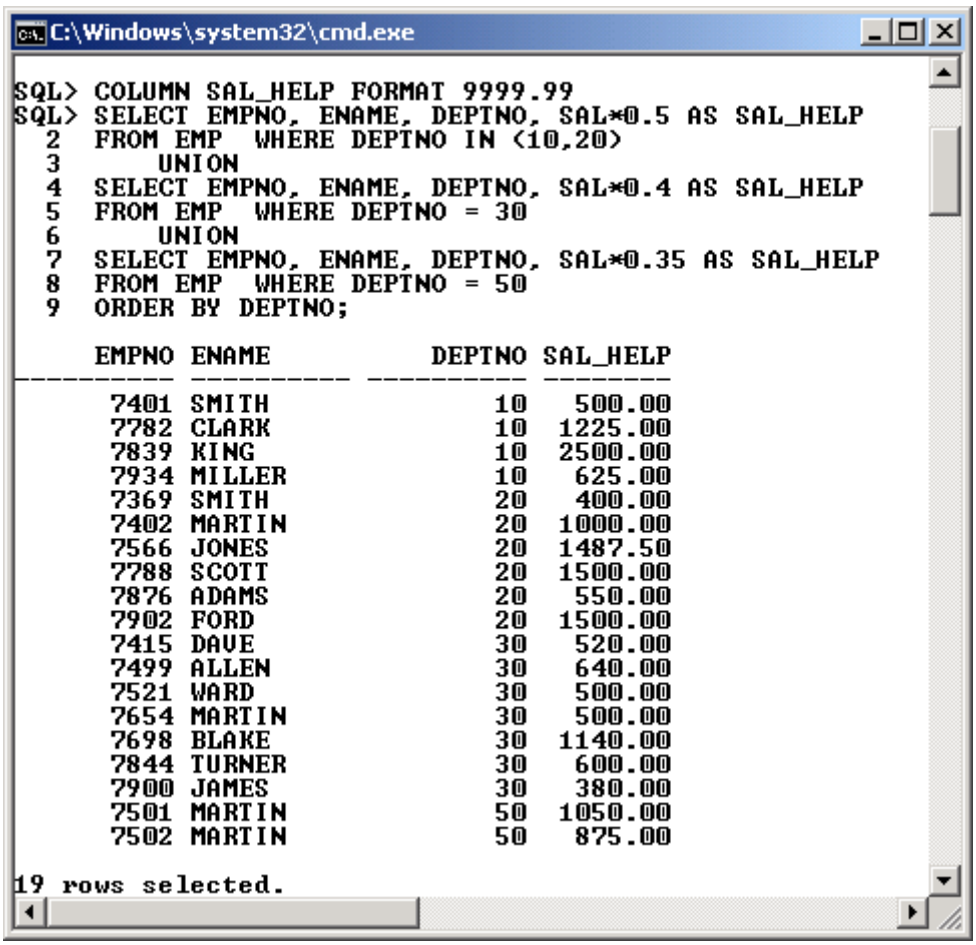


Рис. 2.49. **Результат об'єднання за різними умовами**

2.10.3. Операція віднімання

Визначення

Віднімання (або різниця) двох відношень R1 і R2 – це нове відношення, що містить усі кортежі, які належать першому відношенню R1 і не належать другому R2.

Якщо, наприклад, віднімають з першої таблиці другу, то результатом є нова таблиця, яка містить усі рядки першої за винятком тих рядків, які є також і у другій.

Для виконання такої операції у СКБД Oracle введено ключове слово MINUS. Його використання аналогічне UNION.

Приклад 2.45. Визначити посади співробітників, які є в 10-му відділі, але відсутні у 30-му (рис. 2.50).

```
SELECT JOB FROM EMP WHERE DEPTNO = 10
    MINUS
SELECT JOB FROM EMP WHERE DEPTNO = 30;
```

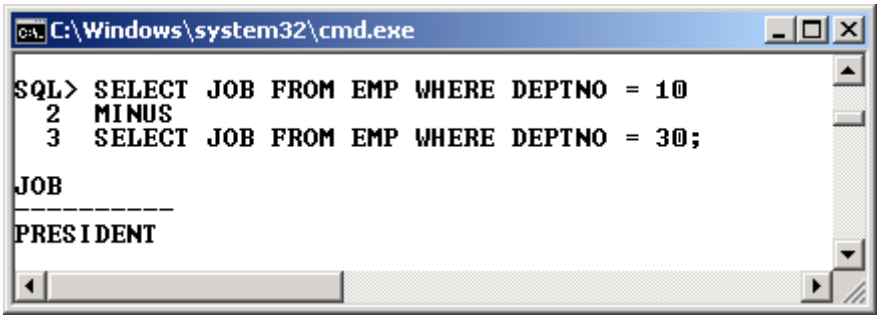


Рис. 2.50. **Результат різниці для поля JOB**

Приклад 2.46. Визначити номери відділів, у яких ще не працюють співробітники (рис. 2.51).

```
SELECT DEPTNO FROM  DEPT
        MINUS
SELECT DEPTNO FROM EMP;
```

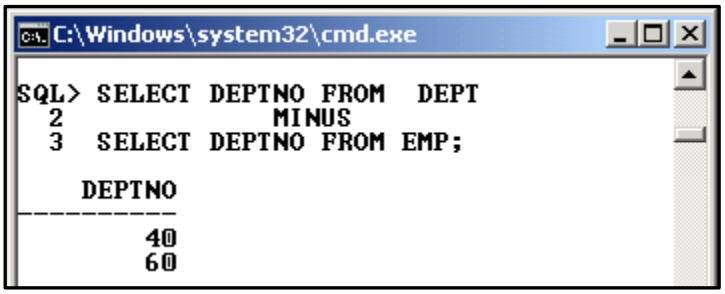


Рис. 2.51. **Результат різниці для поля DEPTNO**

Приклад 2.47. Визначити табельні номери співробітників, що не були у відпустці (рис. 2.52).

```
SELECT EMPNO FROM EMP
     MINUS
SELECT EMPNO FROM VACATION;
```

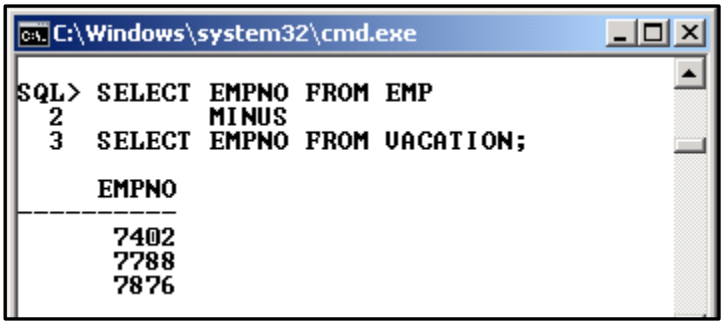


Рис. 2.52. **Результат різниці для поля EMPNO**

2.10.4. Операція перетину

Визначення

Перетин двох відношень R1 і R2 – це нове відношення, що містить усі кортежі, які належать як до відношення R1, так і до відношення R2.

Якщо виконується перетин двох таблиць у базі даних, таблиця, яка є результатом, містить однакові рядки, що є в обох таблицях.

Для ілюстрації операції перетину слід розв'язати завдання протилежне прикладу 2.45.

Приклад 2.48. Визначити посади співробітників, які є як у 10-му відділі, так і у 30-му (рис. 2.53).

```
SELECT JOB FROM EMP WHERE DEPTNO = 10
      INTERSECT
SELECT JOB FROM EMP WHERE DEPTNO = 30;
```

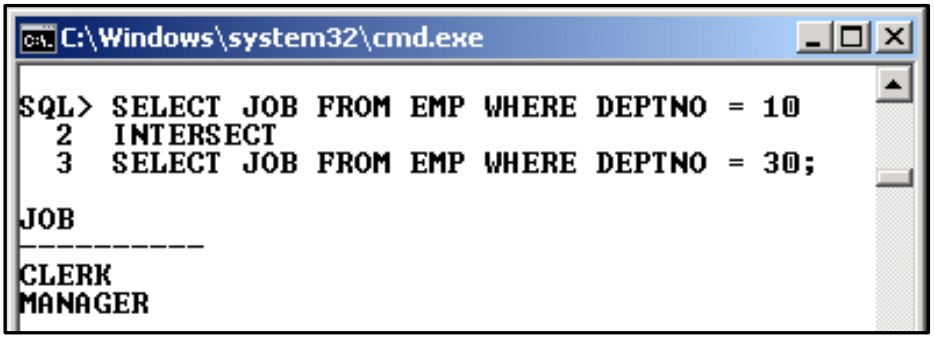


Рис. 2.53. **Результат перетину для поля JOB**

Приклад 2.49. Визначити співробітників, що мають однакове прізвище і працюють у відділах 20-му та 30-му (рис. 2.54).

```
SELECT ENAME FROM EMP WHERE DEPTNO = 20
INTERSECT
SELECT ENAME FROM EMP WHERE DEPTNO = 30;
```

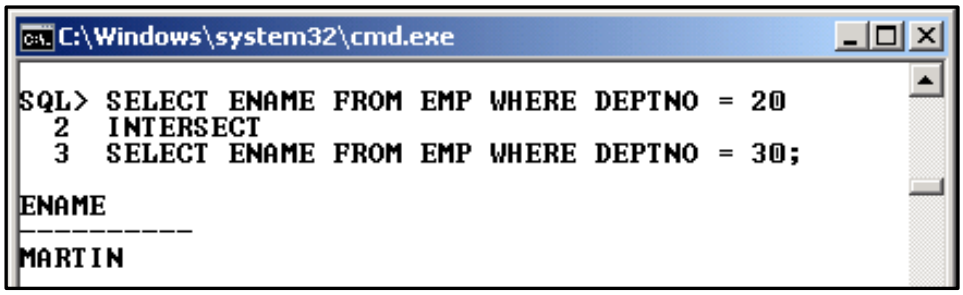


Рис. 2. 54. **Результат перетину для поля ENAME**

2.10.5. Операція ділення

Визначення

Операція ділення відношення R1 на відношення R2 – це нове відношення R3, яке містить тільки ті атрибути відношення R1, які відсутні у відношенні R2, і тільки ті кортежі, декартовий добуток яких з відношенням R2 існує у відношенні R1:

R3(A) = R1(A,B) DEVIDE BY R2(B).

Типові запити, що реалізуються за допомогою операції ділення, зазвичай у своєму формулюванні містять слово "усі". Наприклад: "Які виробники випускають усі види продукції?", "Які співробітники приймали товари від усіх виробників?" тощо.

Операція ділення є додатковою операцією реляційної алгебри й як окрема операція не визначена у мові SQL, але вона може бути реалізована через інші операції з використанням вкладених запитів. З конкретними прикладами реалізації операції ділення можна ознайомитись у [2; 22; 24].

Запитання і завдання

1. Які засоби мови SQL використовують для реалізації операції декартового добутку?
2. Які операції вживають для об'єднання таблиць?
3. У чому полягає дія операції MINUS?
4. Яким чином у мові SQL можна реалізувати операцію перетину?
5. У чому полягає ділення? Як його реалізувати засобами мови SQL?

2.11. Агрегатні функції

2.11.1 Що таке агрегатні функції

Під час роботи з базою даних часто виникають ситуації, які не потребують повної чи детальної вибірки даних. Користувача можуть цікавити певні узагальнені характеристики даних, що зберігаються у базі. Наприклад: загальна сума реалізації продукції за місяць; середня заробітна плата по відділах; кількість співробітників у кожному відділі; кількість видів товарів, що реалізуються через торгову точку тощо.

Мовою SQL запити такого типу подаються за допомогою ***агрегатних*** функцій. Агрегатні функції генерують одне сумарне значення для групи значень у вказаному стовпці, а якщо група не вказана, то для всієї таблиці. За допомогою агрегатних функцій у рамках SQL-запиту можна отримати ряд узагальнених статистичних відомостей про множину відібраних значень вихідного набору.

Агрегатні функції часто застосовують для підбиття підсумків за кожною групою рядків, тому їх ще називають ***підсумковими***.

Користувачеві доступні такі основні агрегатні функції:

COUNT – визначає кількість записів у вихідному наборі SQL-запиту;

MIN – визначає найменше значення з множини значень у певному стовпці запиту;

MAX – визначає найбільше значення з множини значень у певному стовпці запиту;

SUM – обчислює суму множини значень, що містяться в певному стовпці відібраних запитом рядків;

AVG – визначає середнє арифметичне значення множини значень, що зберігаються в певному стовпці відібраних запитом рядків, тобто суму значень поділену на їх кількість.

Окрім зазначених агрегатних функцій, які є стандартними для мови SQL і підтримуються усіма СКБД, Oracle додатково дозволяє використовувати функції STDDEV і VARIANCE для обчислення середньоквадратичного відхилення та дисперсії.

2.11.2. Використання агрегатних функцій за повним вихідним набором SQL-запиту

При обчисленні значень агрегатних функцій за повним вихідним набором SQL-запиту, тобто коли уся таблиця розглядається як одна велика група, результатом є тільки одне число – результат роботи функції. Сама агрегатна функція записується в реченні **SELECT**.

Приклад 2.50. Обчислити загальну кількість співробітників.

```
SELECT  COUNT(EMPNO) FROM EMP;
```

Приклад 2.51. Обчислити загальну середню заробітну плату співробітників.

```
SELECT  AVG (SAL) FROM EMP;
```

Приклад 2.52. Обчислити мінімальну та максимальну заробітну плату співробітників.

```
SELECT  MIN (SAL) AS MINS, MAX(SAL) AS MAXS FROM EMP;
```

Результати виконання прикладів наведені на рис. 2.55.

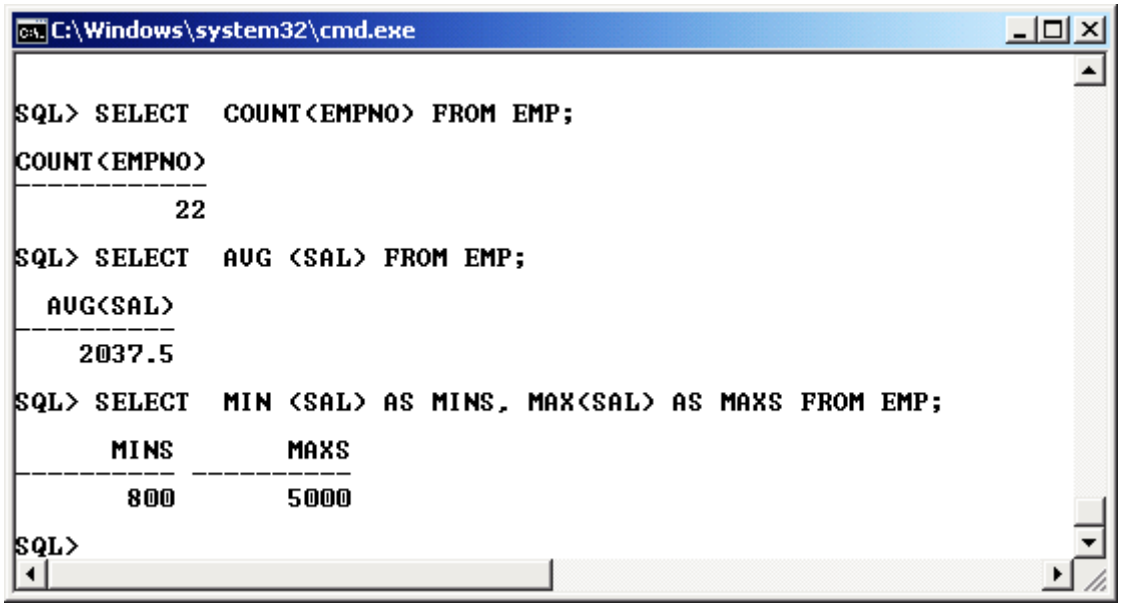


Рис. 2.55. **Реалізація прикладів 2.50 – 2.52**

Приклад 2.53. Підрахувати кількість співробітників, що займають певні посади, та кількість різноманітних посад, що є в організації (рис. 2.56).

```
SELECT  COUNT(JOB) C_JOB,
        COUNT (DISTINCT JOB) CD_JOB
FROM EMP;
```

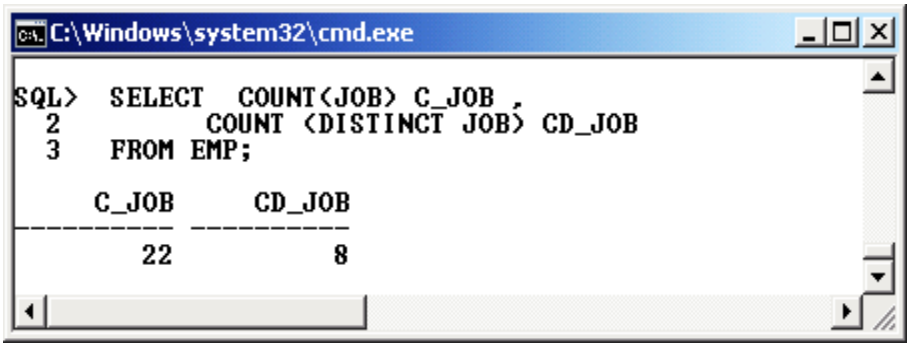


Рис. 2.56. **Використання DISTINCT у агрегатній функції**

Корисно Як видно із отриманого результату, значення функції відрізняються. Використання ключового слова DISTINCT вимагає від функції підраховувати тільки кількість різноманітних (тобто без повторювань) назв посад співробітників.

Важливо Якщо агрегатні функції використовують у якості аргументу стовпців, які можуть мати значення **NULL,** то за стандартом SQL такі значення ігноруються.

Приклад 2.54. Отримати значення однієї і тієї ж функції COUNT для різних стовпців, включаючи ті, що мають значення NULL (рис. 2.57).

```
SELECT  COUNT (EMPNO) EMP_CNT,
        COUNT (DEPTNO)  D_CNT,
        COUNT (DISTINCT DEPTNO)  DEPT_CNT,
        COUNT (COMM) COMM_CNT,
        COUNT (*) ROWS_CNT
FROM EMP;
```

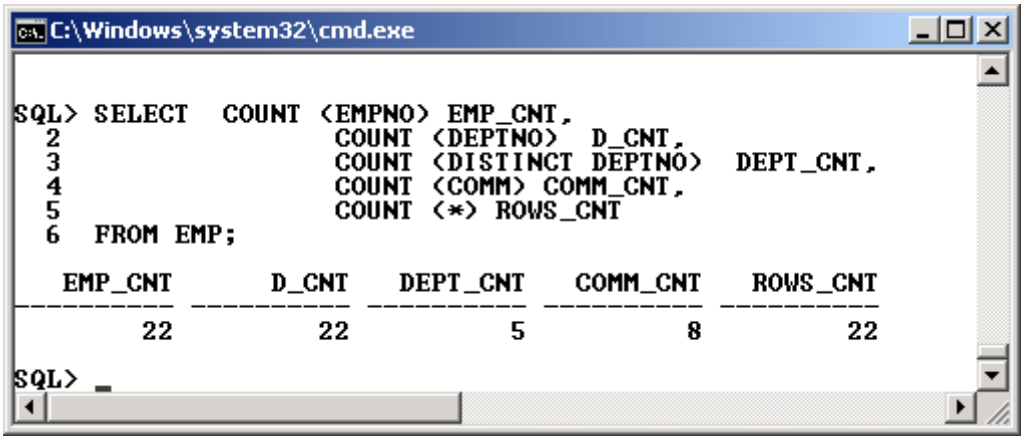


Рис. 2.57. **Результат обчислення різних функції COUNT**

Оскільки стовпець COMM містить значення NULL, то функція COUNT підрахувала тільки кількість тих рядків, що мають значення, відмінні від NULL.

Важливо У стандарті ANSI/ISO визначені точні правила обробки значень **NULL** в агрегатних функціях [8]:

- якщо будь-які значення, що містяться в стовпці, дорівнюють **NULL**, то під час обчислення результату функції вони виключаються;

• якщо усі значення в стовпці дорівнюють **NULL**, то функції **SUM** (), **AVG** (), **MIN** () та **MAX** () повертають значення **NULL**; функція **COUNT** () повертає нуль;

• якщо в стовпці немає значень (тобто стовпець порожній), то функції **SUM** (), **AVG** (), **MIN** () і **MAX** () повертають значення **NULL**; функція **COUNT** () повертає нуль;

• функція **COUNT**(*) підраховує кількість рядків і не залежить від наявності або відсутності в стовпці значень **NULL**; якщо рядків у таблиці немає, ця функція повертає нуль.

2.11.3. Групування даних. Речення GROUP BY

Усі приклади, які наведені у розділі, визначали одне значення для кожної агрегатної функції, і результат містив тільки один рядок. Іншими словами, результати агрегатних функцій попередніх прикладів обчислювали певне підсумкове значення за усією сукупністю рядків таблиці, що відповідали умові пошуку.

Зазвичай потрібні дані, що відповідають окремим групам показників чи об'єктів. Наприклад, необхідно обчислили кількість співробітників, що займають певну посаду чи володіють конкретною іноземною мовою, або підрахувати кількість різноманітних товарів, що випускає кожне підприємство-виробник, визначити сумарну реалізацію за кожною групою товарів тощо. У цьому разі стає в нагоді спеціальне речення **GROUP BY**.

У реченні **GROUP BY** вказується ім'я стовпця (або стовпців), за однаковими значеннями у яких проводиться групування (агрегація) результівних рядків. Без використання агрегатних функцій результат вживання конструкції **GROUP BY** нагадує використання ключового слова **DISTINCT.** Речення **GROUP BY** дає можливість обчислити значення агрегатних функцій для кожної групи окремо.

Приклад 2.55. Обчислити середню заробітну плату по кожному відділу. Результат впорядкувати за зростанням середньої заробітної плати (рис. 2.58).

```
SELECT DEPTNO, AVG (SAL) FROM EMP
GROUP BY DEPTNO
ORDER BY 2;
```

Приклад 2.56. Підрахувати кількість співробітників, що працюють у кожному відділі (рис. 2.59).

```
SELECT DEPTNO, COUNT(EMPNO) FROM EMP
GROUP BY DEPTNO
ORDER BY 2 DESC;
```

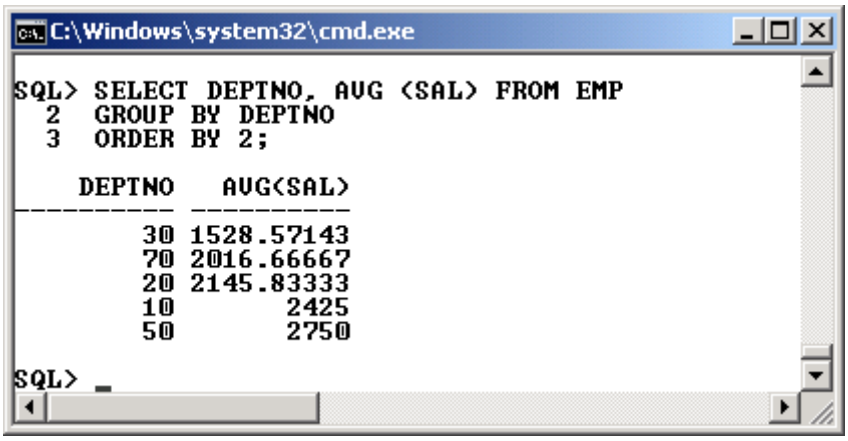


Рис. 2.58. **Результат середньої заробітної плати за відділами**

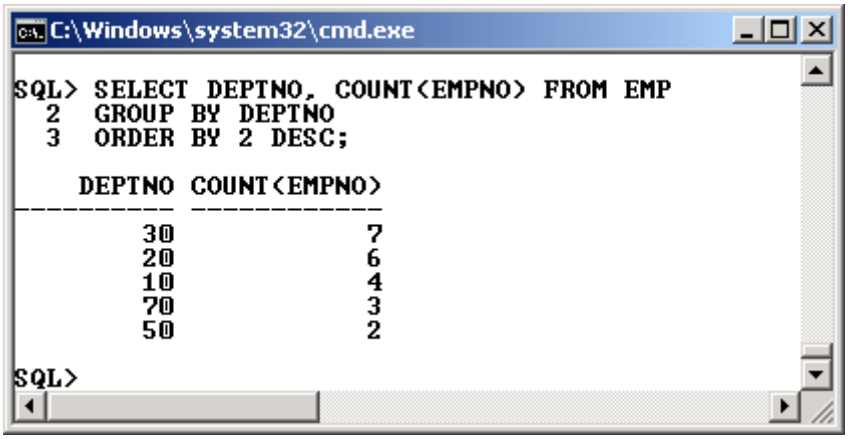


Рис. 2.59. **Кількість співробітників у кожному відділі**

Проаналізувавши отримані результати, можна побачити, що відділів у навчальній БД більше, тому для отримання повного результату запити для прикладів 2.55 та 2.56 слід переписати таким чином (рис. 2.60):

```
SELECT DEPT.DEPTNO, DNAME, AVG (SAL)
FROM DEPT LEFT JOIN EMP ON DEPT.DEPTNO=EMP.DEPTNO
GROUP BY DEPT.DEPTNO, DNAME
ORDER BY 3;
```

```
SELECT DEPT.DEPTNO, DNAME,  COUNT (EMPNO)
FROM DEPT LEFT JOIN EMP ON DEPT.DEPTNO=EMP.DEPTNO
GROUP BY DEPT.DEPTNO, DNAME
ORDER BY 3;
```

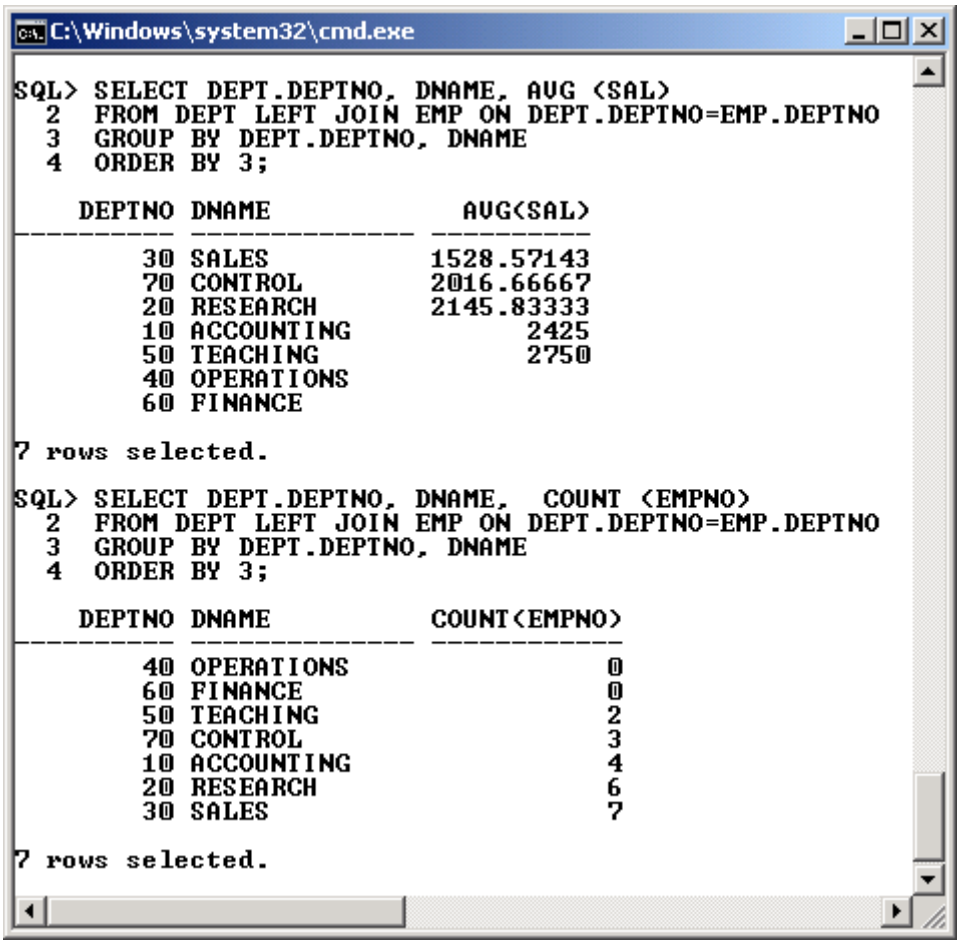


Рис. 2.60. **Результат модифікації для прикладів 2.55 та 2.56**

Пояснення. У запитах використано ліве зовнішнє з'єднання з провідною таблицею DEPT (тобто довідником відділів), з якої до результівної таблиці додадуться всі відділи, і для кожного з них будуть розраховані, відповідно, середня заробітна плата або кількість співробітників

Приклад 2.57. Для кожного співробітника визначити кількість відпусток, загальну кількість днів у відпустках та середню тривалість відпуски. Результат впорядкувати за значення середньої тривалості відпустки (рис. 2.61).

```
SELECT EMP.EMPNO, ENAME,  COUNT(VACATION.EMPNO) CNT_VAC,
       SUM (HDATE_END – HDATE_BEGIN +1) SDAY_VAC,
```

```
        SUM (HDATE_END – HDATE_BEGIN +1) /
COUNT(VACATION.EMPNO) AS AVG_DAY
FROM EMP LEFT JOIN VACATION
      ON EMP.EMPNO = VACATION.EMPNO
GROUP BY EMP.EMPNO, ENAME
ORDER BY 5;
```

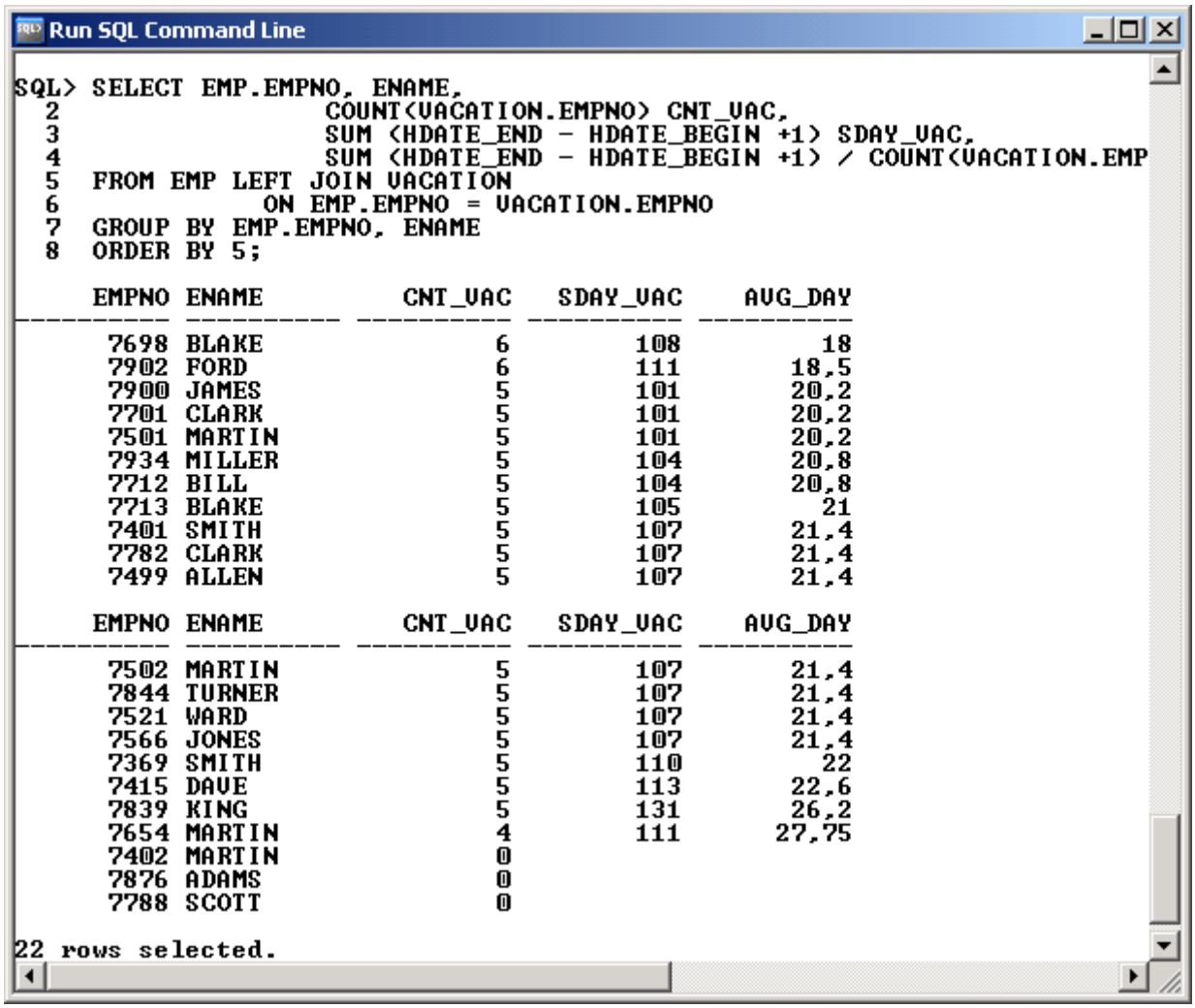


Рис. 2.61. **Результат запиту про відпустки співробітників**

Отриманий результат ще раз підтверджує особливості агрегатних функцій, коли вони працюють з невизначеними значеннями (NULL). Три співробітники, що є останніми у результаті, жодного разу не були у відпустці, тому функція COUNT (другий стовпець) повернула для них значення 0 (нуль), а функція SUM (третій стовпець) – значення NULL.

Результат четвертого стовпця визначається діленням третього стовпця на другий і теж повертає NULL, хоча на нуль ділити не можна. А ось якби функція SUM повернула значення 0 (нуль), то виникла б помилка при діленні і запит би не виконався.

При використанні **GROUP BY** слід пам'ятати наступне: якщо у реченні **SELECT** поряд з агрегатними функціями використовуються також окремі імена стовпців або функції над стовпцями, то їх необхідно вказати і в конструкції **GROUP BY**. У останньому наведеному прикладі такими стовпцями є EMP.EMPNO й ENAME. Зворотне не обов'язкове. Тобто якщо деякі стовпці перераховані у конструкції **GROUP BY**, то у **SELECT** їх можна не вказувати.

2.11.4. Групування та значення NULL

Деякі ускладнення можуть виникнути, якщо стовпці групування містять значення **NULL**. Виникає доречне питання: "До якої групи слід віднести такі значення?" Адже при порівнянні двох значень **NULL** результат має значення **NULL** (а не TRUE), тобто два значення **NULL** не вважаються однаковими. Якщо таку угоду застосувати у реченні GROUP BY, це призведе до того, що кожний рядок із значенням **NULL** в стовпці групування буде створювати окрему групу, яка складається з одного цього рядка.

У стандарті ANSI/ISO [11; 42] визначено, що два значення **NULL** у реченні **GROUP BY** рівні. Тобто, якщо два рядки мають значення **NULL** в однакових стовпцях групування й ідентичні значення в інших стовпцях групування, вони створюють одну групу.

Приклад 2.58. Визначити кількість співробітників, що отримали певного розміру премію (рис. 2.62).

```
SELECT COMM, COUNT(*), COUNT(COMM)
FROM EMP
GROUP BY COMM
ORDER BY 1 DESC;
```

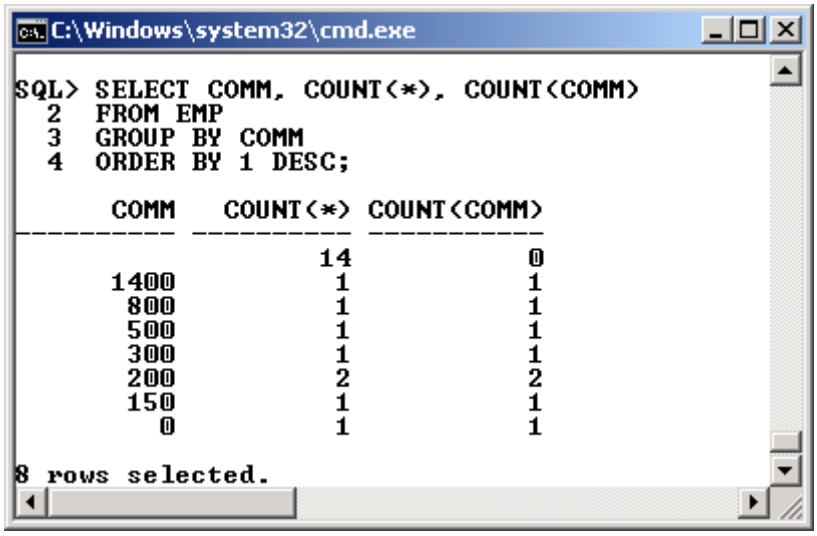


Рис. 2.62. **Кількість отриманих премій на певну суму**

Як видно з результату, всі рядки для співробітників, де значення премії дорівнює NULL, згруповані разом; віддінність є тільки у значеннях функцій COUNT(*) і COUNT(COMM), особливість роботи яких наводилась у прикладі 2.54.

2.11.5. Речення HAVING – умова відбору груп.

Аналогічно реченню **WHERE**, яке використовується для відбору рядків таблиць, що беруть участь у запиті, речення **HAVING** використовується для того, щоб відібрати ті згруповані рядки, для яких значення агрегатної функції відповідає певній умові. У наступному ряді прикладів показано, як вживається **HAVING** у запитах з групуванням.

Приклад 2.59. Знайти відділи, де середня заробітна плата менша за 2 200 (рис. 2.63).

За своєю суттю, це завдання схоже на те, де розраховувалась середня заробітна плата для усіх відділів без будь-яких умов. Тому за основу обрано запит з прикладу 2.55 і додано до нього додаткову умову на значення агрегатної функції.

```
SELECT DEPTNO, AVG (SAL) FROM EMP
GROUP BY DEPTNO
HAVING AVG (SAL) < 2200
ORDER BY 2;
```

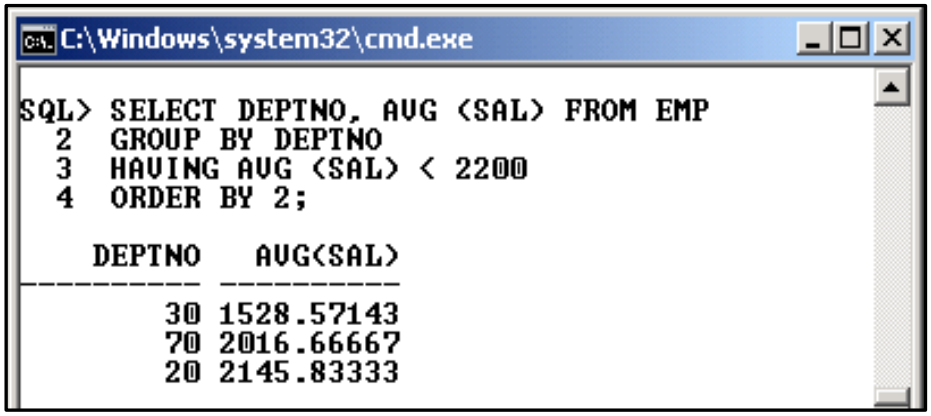


Рис. 2.63. **Умова на середню заробітну плату відділа**

Приклад 2.60. Визначити відділи, у яких працює менше чотирьох співробітників (рис. 2.64).

```
SELECT DEPT.DEPTNO, DNAME,  COUNT (EMPNO)
FROM DEPT LEFT JOIN EMP ON DEPT.DEPTNO=EMP.DEPTNO
GROUP BY DEPT.DEPTNO, DNAME
HAVING COUNT (EMPNO) < 4
ORDER BY 3;
```

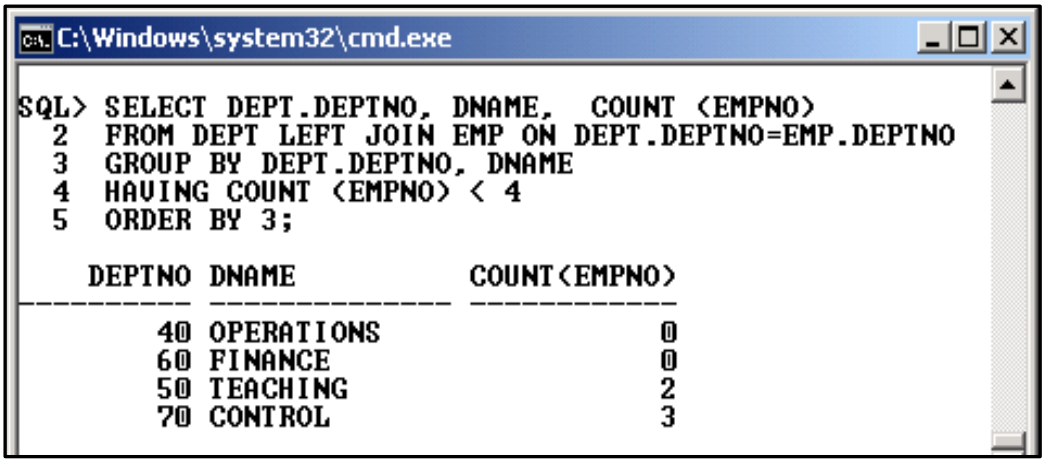


Рис. 2.64. **Визначення малочисельних відділів**

Умова на значення агрегатних функцій може накладуватися навіть тоді, коли вони не зустрічаються у реченні SELECT.

Приклад 2.61. Розрахувати середню заробітну плату співробітників за їх посадами, якщо кількість співробітників, що займають однакову посаду, більше двох (рис. 2.65).

```
SELECT JOB, AVG(SAL)  FROM EMP
GROUP BY JOB
HAVING COUNT(JOB) >2  ORDER BY 2;
```

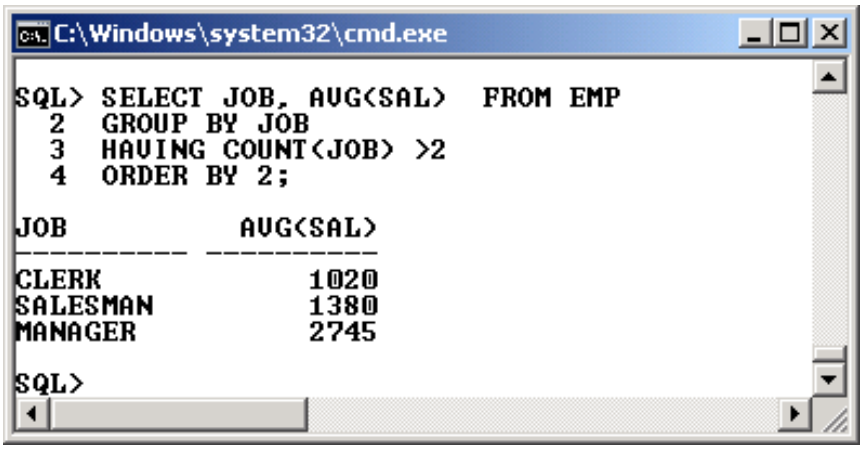


Рис. 2.65. **Умова на агрегатну функцію, яка не зустрічається у SELECT**

При складанні запитів умови на рядки таблиць та умови на агрегатні функції можуть зустрічатися одночасно.

Приклад 2.62. Визначити загальну та середню тривалість відпусток для співробітників кожного відділу за 2014 рік за умови, що у відділі працює більше двох співробітників (рис. 2.66)

```
SELECT EMP.DEPTNO, DNAME,
          SUM(HDATE_END – HDATE_BEGIN + 1) TOTAL_DAY,
          SUM(HDATE_END – HDATE_BEGIN + 1) /
          COUNT (EMP.EMPNO)    AVG_DAY
FROM DEPT, EMP, VACATION
WHERE DEPT.DEPTNO = EMP.DEPTNO AND
         EMP.EMPNO = VACATION.EMPNO AND
         HDATE_BEGIN LIKE '_____2014'
GROUP BY EMP.DEPTNO, DNAME
HAVING COUNT (EMP.EMPNO) > 2
ORDER BY AVG_DAY;
```

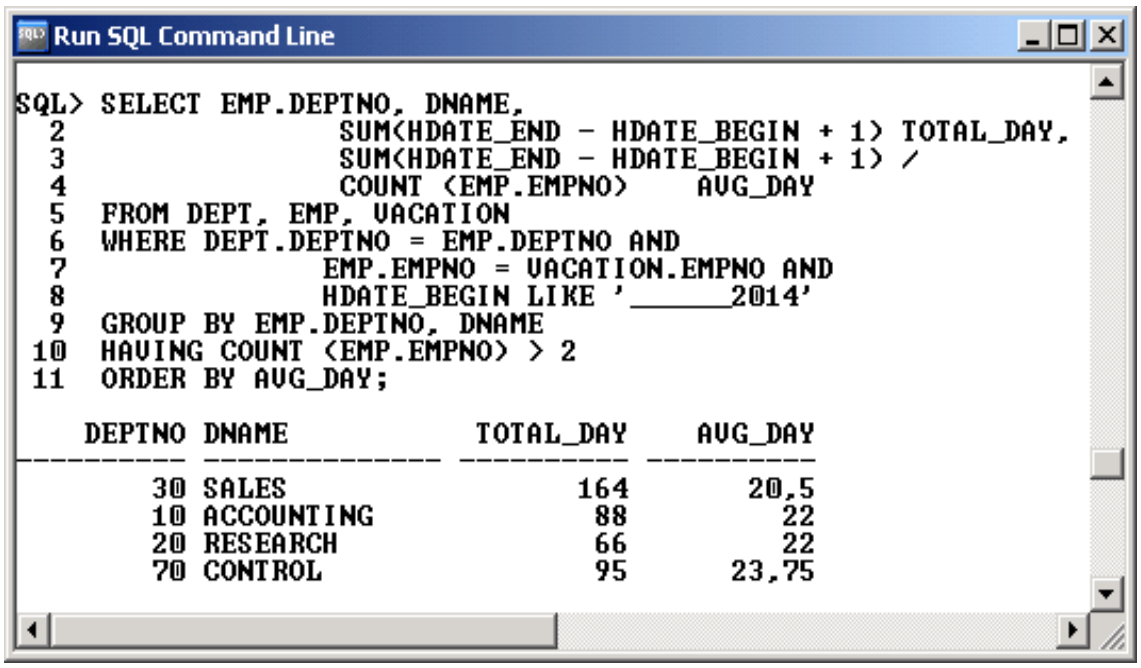


Рис. 2.66. **Одночасне використання WHERE та HAVING**

Запитання і завдання

1. Які агрегатні функції застосовують для підбиття підсумків за кожною групою рядків?
2. У чому полягає відмінність між використанням агрегатних функцій за повним вихідним набором SQL-запиту та групою рядків?
3. У яких випадках використовують речення HAVING?
4. За якими правилами здійснюють групування, якщо відповідні стовпці містять значення NULL?

2.12. Розширення Oracle SQL для агрегації у сховищах даних

Одним із ключових понять у системах підтримки прийняття рішень є багатовимірний аналіз [33; 46; 52]. Термін "вимір" означає будь-яку категорію, що використовується для здійснення аналізу, наприклад, час, місце розташування, товар, відділ, співробітник тощо. У реальних умовах потенційні розміри настільки ж нескінченні, як і різновиди підприємницької діяльності. Для проведення багатовимірного аналізу зазвичай використовують сховища даних – предметно-орієнтований, інтегрований, незмінний набір даних, що підтримує хронологію та здатний бути комплекс-

ним джерелом достовірної інформації для оперативного аналізу та прийняття рішень.

Агрегація є фундаментальною частиною сховищ даних. Для підвищення продуктивності операцій агрегації Oracle Database забезпечує додаткову функціональність, а саме [19; 58; 74]:

- розширення CUBE та ROLLUP для конструкції GROUP BY;
- три функції GROUPING;
- вираз GROUPING SETS;
- операція повороту (Pivoting).

Розширення CUBE, ROLLUP та GROUPING SETS для SQL-запитів дозволяють зробити звіти простіше і швидше. CUBE, ROLLUP та GROUPING SETS генерують результівний набір одним запитом, еквівалентним конструкції UNION ALL (див. п. 2.10.2) по-різному згрупованим рядкам.

ROLLUP обчислює агрегати, такі як, SUM, COUNT, MAX, MIN і AVG на більш високому рівні агрегації – від детальних до загального підсумку.

CUBE схожий на ROLLUP, але дозволяє виконати розрахунок усіх можливих комбінацій агрегатів. У розширеннях CUBE, ROLLUP та GROUPING SETS вказуються тільки ті групи, які необхідні у конструкції GROUP BY.

Три GROUPING-функції дозволяють визначити приналежність рядка результату до кожної групи та додати сортування проміжних підсумків рядків і результати фільтрації.

Слід звернути особливу увагу на інтерпретацію значення NULL у прикладах, що будуть розглянуті. Значення NULL, що повертаються розширеннями до конструкції GROUP BY, не завжди є традиційним значенням NULL. NULL також може означати, що його рядок є підсумковим. Щоб не вводити ще одне якесь NULL-значення у систему баз даних, ці підсумкові значення не отримали спеціальної позначки. Однак використовуючи GROUPING-функції, можна отримати докладну інформацію про те, як значення NULL, що є проміжними підсумками, відрізняються від NULL, що зберігаються у даних.

Подальші приклади ілюструють використання мовних розширень SQL для здійснення агрегацій у сховищах даних.

2.12.1. Розширення ROLLUP для конструкції GROUP BY

ROLLUP дозволяє розрахувати кілька рівнів проміжних підсумків за певною групою вимірів та розраховує загальний підсумок. ROLLUP є простим розширенням у конструкції GROUP BY, і його синтаксис дуже простий у використанні. Розширення ROLLUP є досить ефективним і створює мінімальне навантаження на запит.

Дія ROLLUP проста: він створює проміжні підсумки, які починаються з більш детального рівня і закінчуються загальним підсумком відповідно до списку групування, що подано у фразі ROLLUP. Спочатку він обчислює стандартні агрегації, передбачені конструкцією GROUP BY. Далі створюються проміжні підсумки більш високого рівня, рухаючись справа наліво по списку групування стовпців. Нарешті, створюється загальний підсумок.

ROLLUP створює проміжні підсумки для n + 1 рівнів, де n – число стовпців у списку групування. Наприклад, якщо запит вказує ROLLUP на групування стовпців за часом, регіонами та відділами (n = 3), то набір результатів включатиме в себе рядки на чотирьох рівнях агрегації.

Використовувати розширення ROLLUP рекомендується у задачах, пов'язаних з підведенням проміжних підсумків.

Для адміністраторів сховищ даних з використанням зведених таблиць ROLLUP може спростити та прискорити обслуговування зведених таблиць.

Приклад 2. 63. Підрахувати кількість співробітників у кожному відділі (рис. 2.67) з одночасним підведенням загального підсумку.

```
SELECT DEPTNO, COUNT(EMPNO) FROM EMP
GROUP BY ROLLUP(DEPTNO);
```

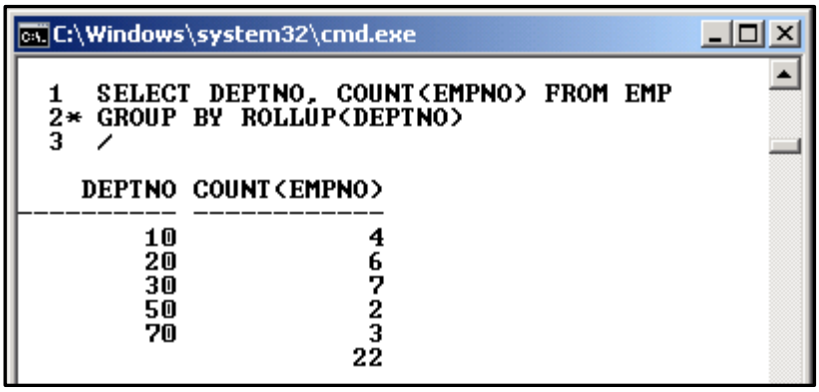


Рис. 2.67. **Загальна кількість співробітників у відділах**

Для підбиття підсумків у кожному відділі ще й за посадами треба ввести додатковий рівень (рис. 2.68).

```
SELECT DEPTNO, JOB, COUNT(EMPNO) FROM EMP
GROUP BY ROLLUP(DEPTNO, JOB);
```

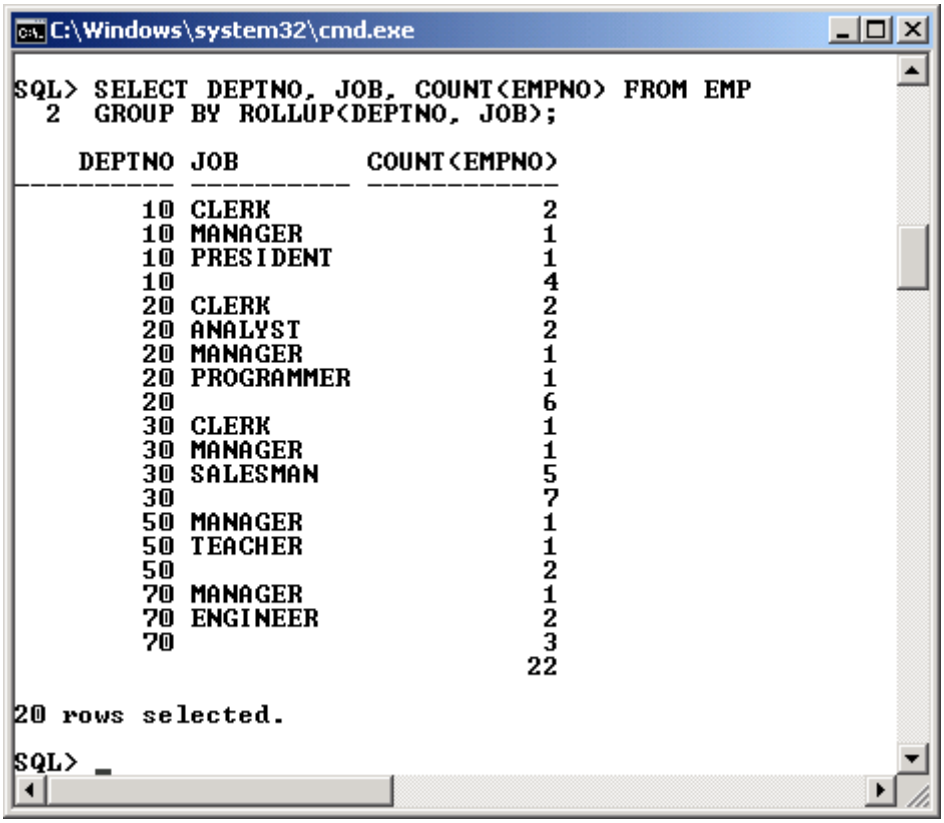


Рис. 2.68. **Підбиття підсумків за посадами, відділами та загальні**

З отриманого результату видно, що спочатку відображається кількість співробітників на певній посаді у конкретному відділі, потім кількість співробітників у відділі – для кожного відділу окремо. Останній рядок видає загальну кількість робітників.

Включаючи функцію ROLLUP у фразу GROUP BY, Oracle отримує наказ підсумувати дані за рівнями зазначених стовпців і підвести загальний підсумок. Необхідно звернути увагу на те, що коли Oracle підбиває загальний підсумок, то залишається незаповненим рядок у стовпці, по якому будувалася фраза GROUP BY. Якщо стовпець GROUP BY також містить порожні значення, то може бути важко відрізнити його значення від підсумку по рядку. Однак у цьому випадку рекомендується використовувати спеціальну функцію GROUPING, яка повідомляє про статус (summarization) поточного рівня. Функція повертає два значення: "0" вка-

зує, що поточний рядок є групою, специфікованою рівнем GROUP BY, а "1" вказує, що рядок згрупований на більш високому рівні.

Приклад 2.64. Ілюстрація використання функції GROUPING для стовпців групування (рис. 2.69).

```
SELECT DEPTNO, JOB, COUNT(EMPNO),
GROUPING(DEPTNO), GROUPING(JOB)
FROM EMP
GROUP BY ROLLUP(DEPTNO, JOB);
```

Як видно з отриманого результату, за значенням функції GROUPING можна визначити, до якого рівня агрегації відносяться значення відповідної функції (у прикладі, це функція COUNT(EMPNO).

Таким чином, за допомогою наведеного запиту можна підрахувати співробітників за трьома рівнями агрегації:

- за відділами та посадами;
- за відділами;
- загальним підсумком.

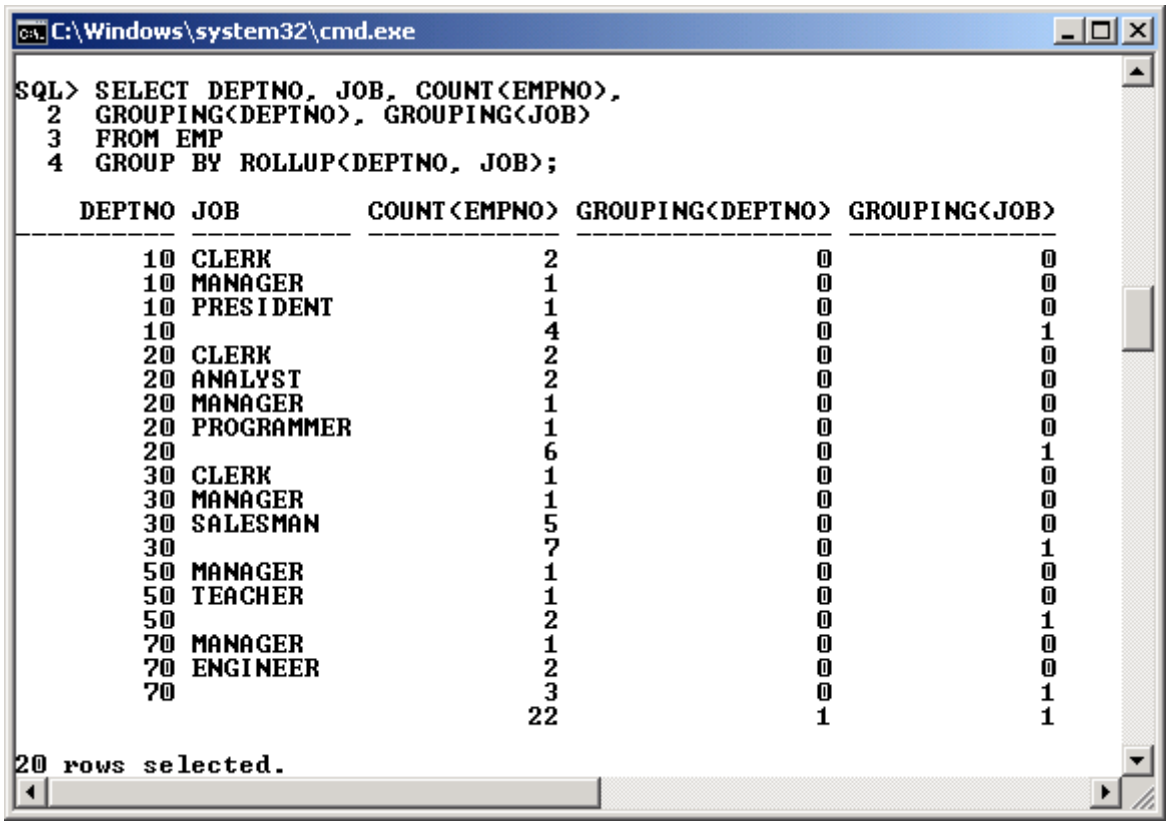


Рис. 2.69. **Приклад використання функції GROUPING**

2.12.2. Розширення CUBE для конструкції GROUP BY

Розширення CUBE на основі певного набору стовпців, що групуються, створює проміжні підсумки для всіх своїх можливих комбінацій. В аспекті багатовимірного аналізу CUBE генерує всі проміжні підсумки, які можуть бути розраховані для куба даних із заданими розмірами. Наприклад, якщо у запиті вказано CUBE (час, регіон, відділ), набір результатів включатиме всі значення, які будуть включені до еквівалентної конструкції ROLLUP, плюс додаткові комбінації, наприклад: (регіон, відділ), (час, відділ) тощо.

Розширення CUBE рекомендується застосовувати, коли потрібні звіти у вигляді перехресних таблиць. Як ROLLUP, CUBE може бути корисним у створенні зведених таблиць.

CUBE, як правило, найбільш доречні у запитах, що використовують стовпці у декількох вимірах, а не стовпці, що презентують різні рівні в одному вимірі. Наприклад усі комбінації місяців, держав та продуктів створюють три незалежні виміри, і аналіз усіх можливих проміжних підсумків комбінацій є звичайною справою. На відміну від цього, перехресні таблиці, що показують усі можливі комбінації років, місяців та днів, матимуть обмежений інтерес, тому що є природна ієрархія у часовому вимірі.

Приклад 2.65. Проілюструвати використання розширення CUBE для запита, аналогічного попередньому (2.64), але замість ROLLUP вказати CUBE (рис. 2.70).

```
SELECT DEPTNO, JOB, COUNT(EMPNO),
GROUPING(DEPTNO), GROUPING(JOB)
FROM EMP
GROUP BY CUBE(DEPTNO, JOB);
```

У отриманому результаті поряд з рядками, що були присутні при виконанні запиту з ROLLUP, є й додаткові, а саме ті, що підбивають підсумки тільки за посадами.

Підвищення читабельності результату запиту.

Комбінуючи функцію GROUPING та, наприклад, функцію DECODE (див. п. 2.17.5) можна значно підвищити читабельність отриманого результату, виводячи замість порожніх значень певні коментарі або пояснювальний текст.

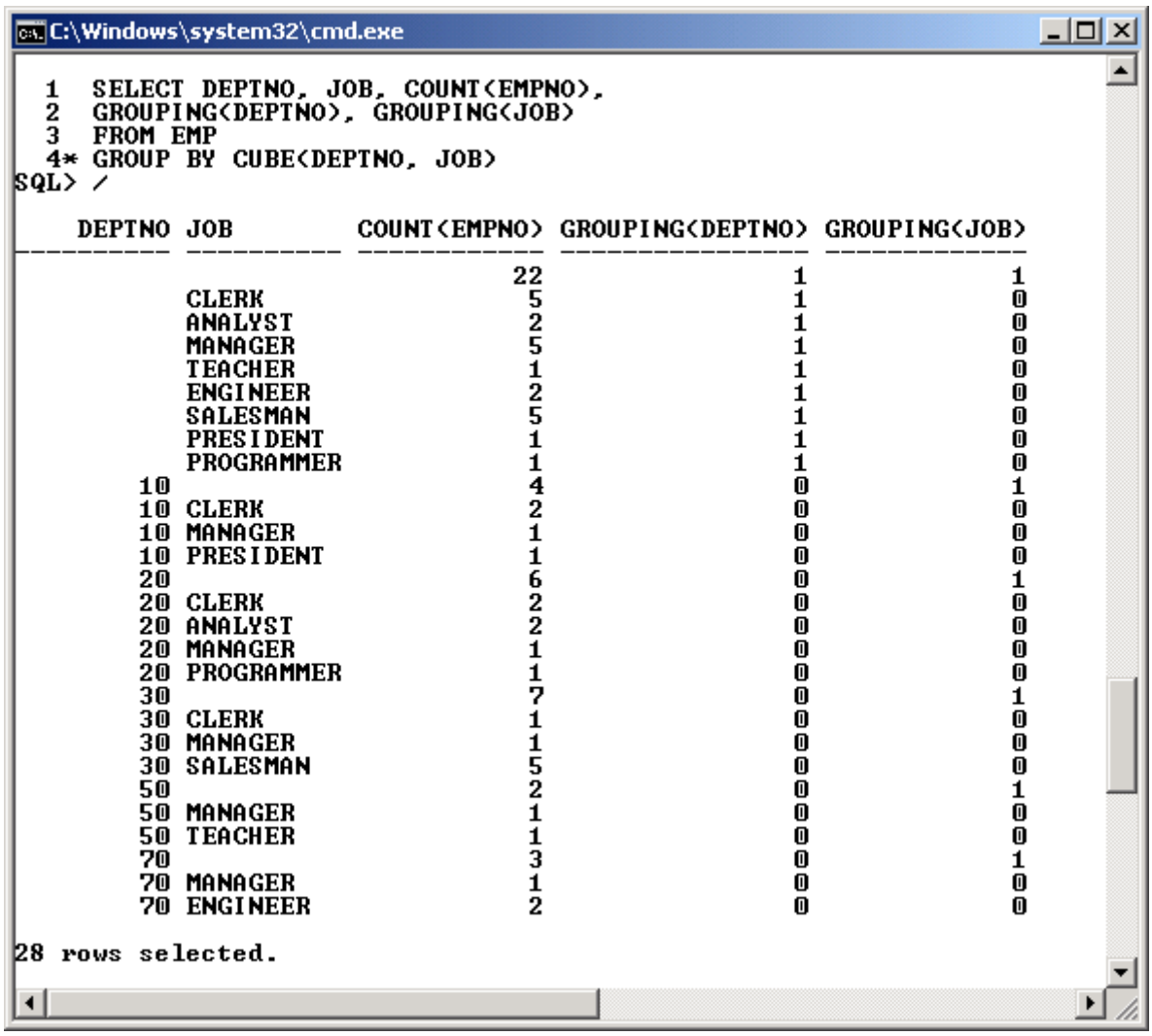


Рис. 2.70. **Приклад використання CUBE у GROUP BY**

Приклад 2.66. Отримати результат попереднього запиту з використанням розширення CUBE, але замість порожніх значень вивести текст, що пояснює групу, для якої обчислювалися підсумки. Четвертий та п'ятий стовпці запиту використовуються тільки для ілюстрації, у даному випадку їх можна не вказувати (рис. 2.71).

```
COLUMN NDEPT FORMAT A12
COLUMN NJOB  FORMAT A12
SELECT
DECODE(GROUPING(DEPTNO), 1, '  Total Dept', TO_CHAR(DEPTNO)) AS
NDEPT,
DECODE(GROUPING(JOB), 1, '  Total Job', JOB) AS NJOB,
COUNT(EMPNO),
GROUPING(DEPTNO), GROUPING(JOB)
```

FROM EMP
GROUP BY CUBE(DEPTNO, JOB);

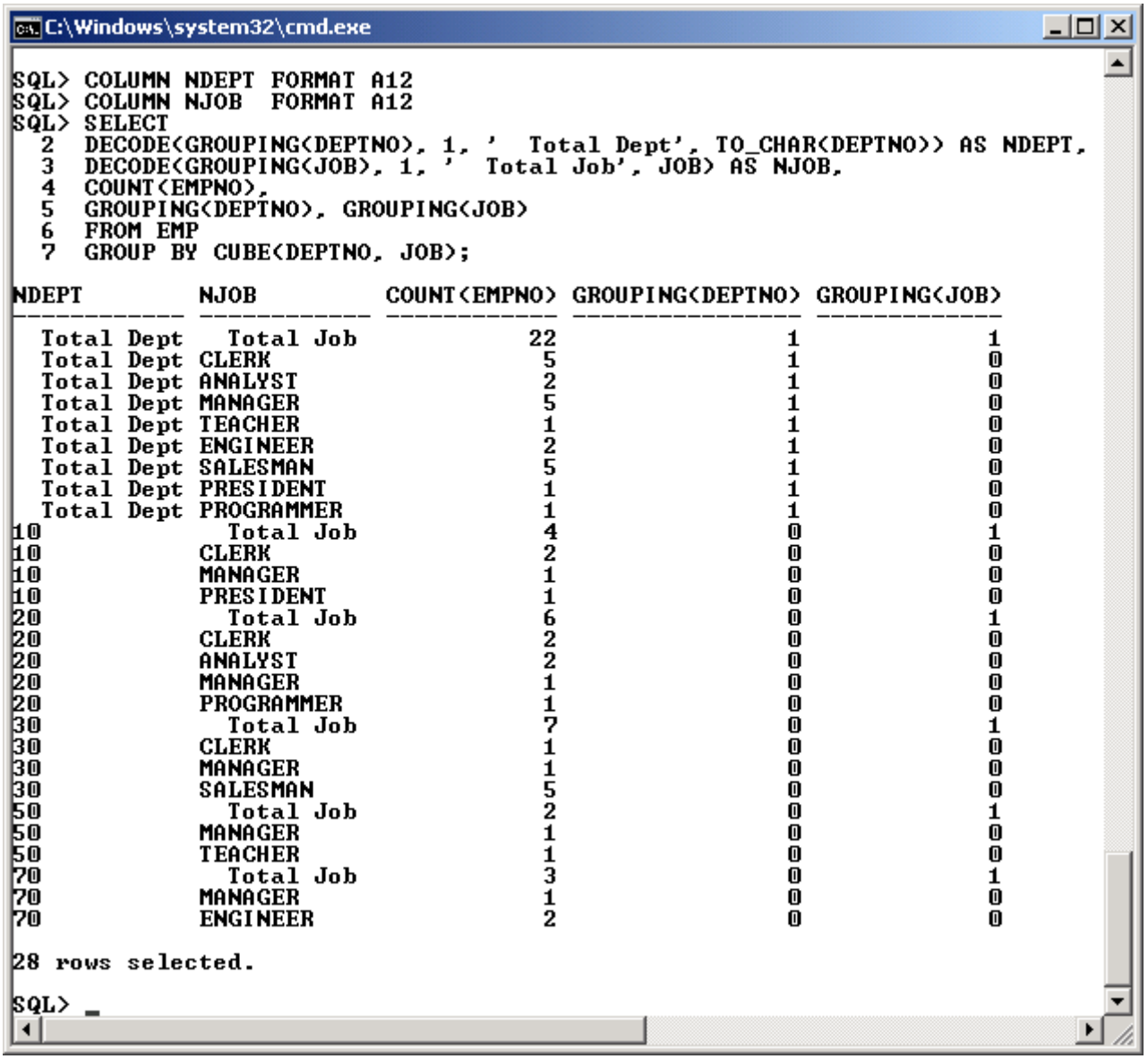


Рис. 2.71. **Виведення пояснювального тексту функцією DECODE**

2.12.3. Використання функції GROUPING_ID

Функція GROUPING_ID корисна, коли виникає необхідність працювати з агрегатами (підсумковими записами). Тобто функцію GROUPING_ID використовують, щоб визначити, яке агрегування (групування) містить кожен рядок. Функція GROUPING_ID повертає 0 або 1, якщо викликається для одного стовпця (у цьому випадку вона працює як функція GROUPING). Функція GROUPING_ID відрізняється від функції GROUPING тим, що можна вказувати не один стовпець, а список стовпців. Тоді функція GROUPING_ID буде розглядати його як бітовий вектор,

повертаючи десяткове число. Це означає, що коли викликана GROUPING_ID (a, b, c), то може бути повернуте число у діапазоні від 0 до 7 (000 … 111), оскільки повертаються різні комбінації 0 і 1. Це дозволяє використовувати в запиті оператор CASE, щоб визначити, чим є рядок.

Приклад 2.67. Визначити значення функції GROUPING_ID для кожного рядка з попереднього прикладу (2.66) (рис. 2.72).

```
SELECT DEPTNO, JOB, COUNT(EMPNO) AS CNT,
GROUPING(DEPTNO) AS G_DEPT,
GROUPING(JOB) AS G_JOB,
GROUPING_ID (DEPTNO, JOB) AS G_ID
FROM EMP
GROUP BY CUBE(DEPTNO, JOB);
```

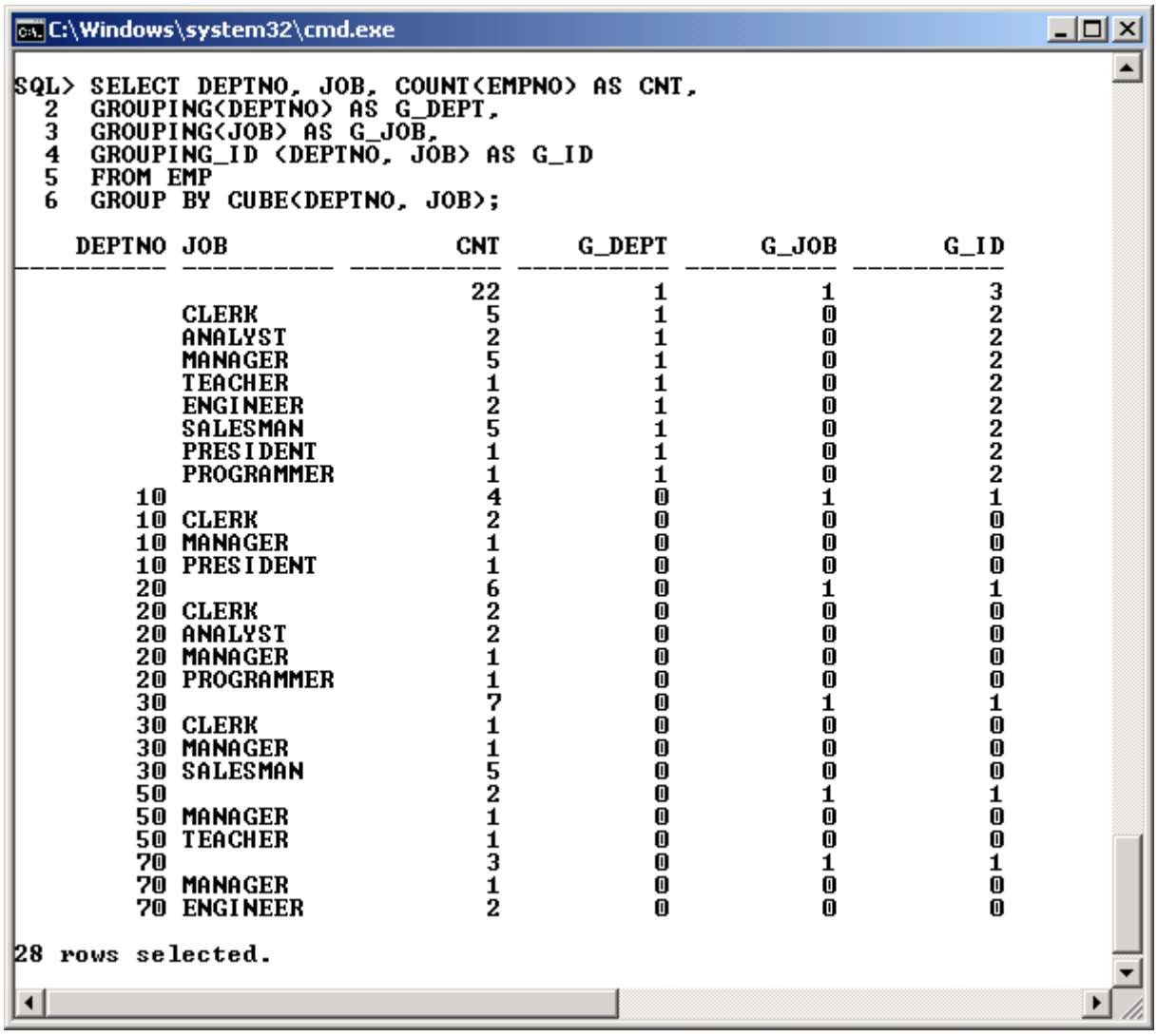


Рис. 2.72. **Використання функції GROUPING_ID**

Пояснення. Для першого рядка значення функції GROUPING_ID дорівнює 3, тобто це бінарне значення 11. Таким чином, значення цього рядка відповідають одночасній агрегації за номером відділа та посадою. Рядки, де значення функції GROUPING_ID дорівнює 2, відповідають агрегації за номером відділа. Для значення 1 – агрегації за посадами. Нарешті значення 0 означає, що рядок містить детальні дані.

2.12.4. Вираз GROUPING SETS у конструкції GROUP BY

За потребою можна вибірково вказати набір груп, які необхідно створити за допомогою виразу GROUPING SETS у межах конструкції GROUP BY. Це дозволяє задати точну специфікацію на декілька вимірів без обчислення усього CUBE.

Приклад 2.68. Використовуючи GROUPING SETS у конструкції GROUP BY, обчислити лише потрібні значення агрегатних функцій (рис. 2.73).

```
SELECT DEPTNO, JOB, COUNT(EMPNO) AS CNT,
GROUPING(DEPTNO) AS G_DEPT,
GROUPING(JOB) AS G_JOB,
GROUPING_ID (DEPTNO, JOB) AS G_ID
FROM EMP
GROUP BY GROUPING SETS(DEPTNO, JOB), (JOB);
```

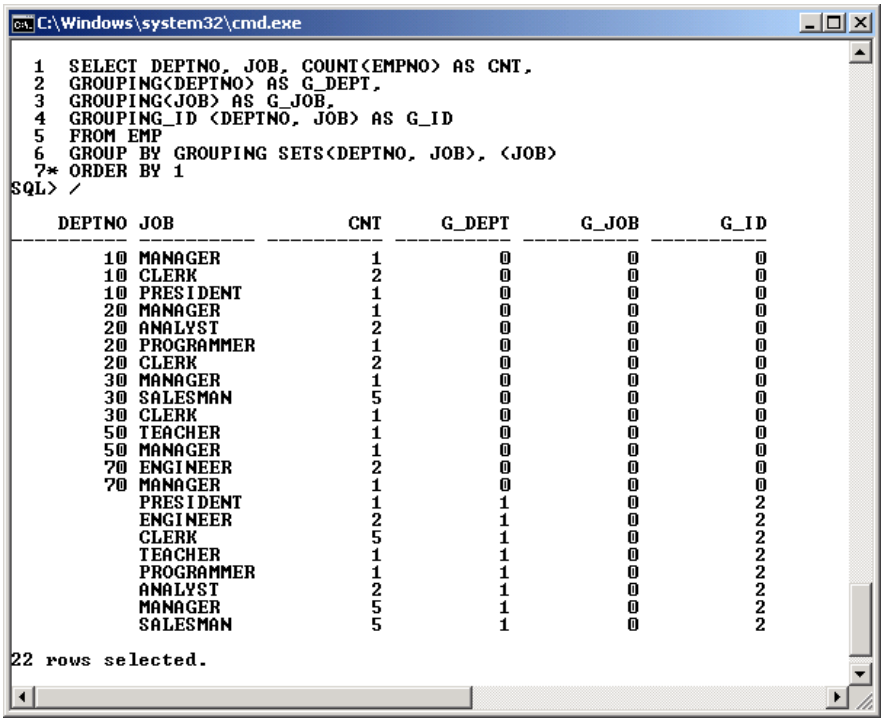


Рис. 2.73. **Використання GROUPING SETS у конструкції GROUP BY**

Пояснення. При виконанні запиту розраховується значення агрегатної функції COUNT(EMPNO) окремо для групи (DEPTNO, JOB) – верхні рядки та групи (JOB) – нижні рядки.

Один із наборів повинен включати усі стовпці, що є у SELECT без агрегатних функцій.

2.12.5. Функція GROUP_ID

Функція GROUP_ID дозволяє відрізнити повторювані групи, які є результатом використання конструкції GROUP BY. Це корисно у фільтрації дублікатів групування за результатами виконання запиту. Для кожного рядка з повторюваної групи, функція повертає її порядковий номер, починаючи з 0. Ця функція може бути застосована тільки у команді SELECT, яка містить конструкцію GROUP BY.

Приклад 2.69. Проілюструвати використання функції GROUP_ID для того, щоб відрізнити повторювані групи у результаті (рис. 2.74).

```
SELECT DEPTNO, JOB, NORDER, COUNT(*), GROUP_ID()
FROM EMP, VACATION
WHERE EMP.EMPNO=VACATION.EMPNO AND DEPTNO = 50
GROUP BY DEPTNO, JOB, NORDER,
ROLLUP (JOB, NORDER)   ORDER BY 1,2;
```

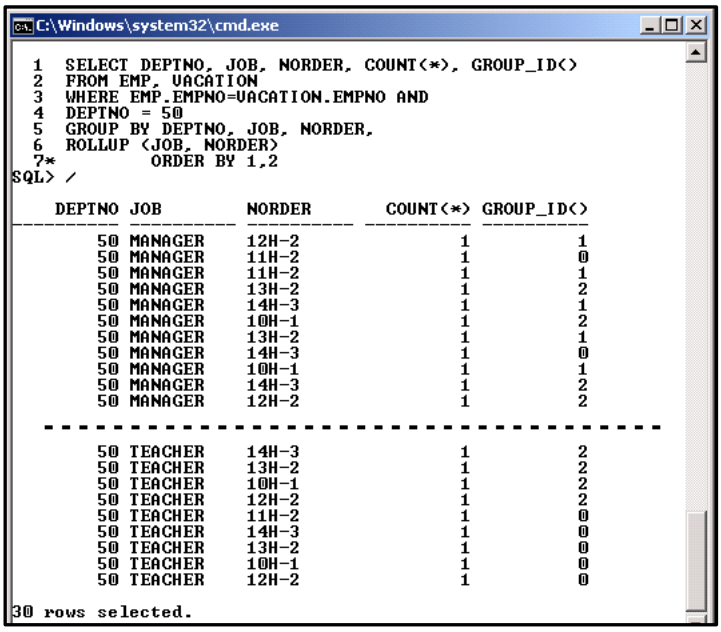


Рис. 2.74. **Приклад використання функції GROUP_ID**

Пояснення. Без фрази ROLLUP (JOB, NORDER) запит повернув би 10 рядків, і усі вони були б відмінні між собою. Однак, оскільки ROLLUP розраховує кілька рівнів проміжних підсумків по вказаним у запиті групам, певні значення у перших трьох рядках можуть повторюватися. Саме використання функції GROUP_ID (четвертий стовпець) дозволяє вказати відповідний порядковий номер повторюваного рядка.

Запитання і завдання

1. Які види функціональностей забезпечує агрегація Oracle у сховищах даних?
2. Яке призначення має розширення ROLLUP?
3. Чим відрізняється розширення CUBE від розширення ROLLUP?
4. З якою метою використовують функцію GROUPING_ID?
5. Яке призначення має вираз GROUPING SETS у конструкції GROUP BY?

2.13. Підзапити

2.13.1 Що таке підзапити

Підзапит (вкладений запит) – потужний засіб мови SQL, який дозволяє будувати складні ієрархії запитів, що багаторазово виконуються у процесі побудови результату або виконання одного з операторів модифікації даних (**DELETE**, **INSERT**, **UPDATE**).

Підзапит (subquery) зазвичай формується за допомогою команди **SELECT**, що може бути вкладена:

- у речення **WHERE**, **HAVING** або **SELECT** іншого оператора **SELECT**;
- в оператор **INSERT**, **UPDATE, DELETE** або **CREATE TABLE**;
- в інший підзапит.

Кожна зовнішня команда має більш високий рівень, ніж вкладена в нього. Зазвичай підзапити повертають свої результати у зовнішню команду, тобто результат виконання зовнішньої команди залежить (визначається) від результату виконання внутрішнього запиту – це так звані ***некорельовані підзапити***. Однак є запити, що називаються ***корельова-***

ними, у яких результат вкладеного підзапиту визначається умовами, що задані у зовнішньому запиті.

Як корельовані, так і некорельовані підзапити можна поділити на три типи:

- підзапити, що не повертають жодного або повертають декілька елементів (значень або рядків). Вони починаються з оператора IN або оператора порівняння та можуть містити ключові слова **ALL**, **ANY** (SOME);
- скалярні підзапити, що повертають одне значення. Вони починаються з оператора порівняння;
- підзапити, що перевіряють існування певних даних за умовами. Вони починаються з ключового слова **EXISTS**.

До підзапитів застосовуються такі правила й обмеження:

- фраза **ORDER BY** не використовується, хоча може зустрічатися в зовнішньому підзапиті;
- список у реченні **SELECT** складається з імен окремих стовпців або складених з них виразів, за винятком випадку, коли в підзапиті є ключове слово **EXISTS**;
- за замовчуванням, імена стовпців у підзапиті відносяться до таблиці, ім'я якої вказане в реченні **FROM**. Проте допускається посилання і на стовпці таблиці, що вказана у фразі **FROM** зовнішнього запиту. Для цього застосовуються повні імена стовпців (тобто зі вказівкою таблиці або аліасу);
- якщо підзапит є одним з двох операндів, що беруть участь в операції порівняння, підзапит повинен вказуватися у правій частині цієї операції.

2.13.2. Підзапити, що повертають одне значення

Перед тим як розглядати використання вкладених запитів, слід написати запит, що дає відповідь на таке завдання: отримати відомості про співробітників, чия заробітна плата більша за 2 500.

```
SELECT * FROM EMP WHERE SAL > 2500;
```

Яким чином зміниться запит, якщо потрібно отримати відомості про співробітників, чия заробітна плата більша за середню по усій організації? Таке завдання можна вирішити у два етапи: спочатку отримати значення середньої заробітної плати, а потім, підставивши її у попередній запит замість значення 2 500, отримати відповідь на основне питання. Але саме підзапити дозволяють замінити подібний багатокроковий алгоритм вирішення завдання одним запитом, вказавши замість значення 2 500 вкладений запит, що отримує значення середньої заробітної плати.

Приклад 2.70. Отримати відомості про співробітників, чия заробітна плата більша за середню по усій організації (рис. 2.75).

```
SELECT * FROM EMP
        WHERE SAL >  (SELECT AVG(SAL) FROM EMP);
```

Що є чудовим в цьому випадку, так це те, що навіть не потрібно знати значення середньої заробітної плати, оскільки воно може динамічно змінюватися з часом, а сам запит остається незмінним.

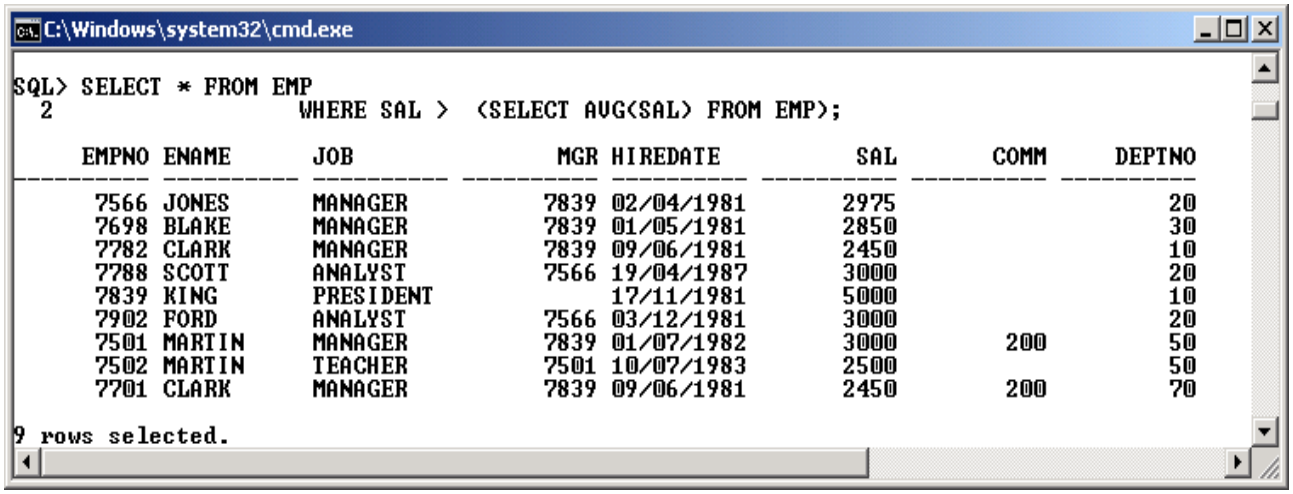


Рис. 2.75. **Підзапит "заробітна плата більша за середню по організації"**

Приклад 2.71. Отримати відомості про співробітників, чия заробітна плата відрізняється від мінімальної не більше як на 20 %. Використати вкладений запит (рис. 2.76).

```
SELECT ENAME, EMPNO, SAL, DEPTNO
FROM EMP
WHERE SAL<= 1.2 * (SELECT MIN(SAL) FROM EMP);
```

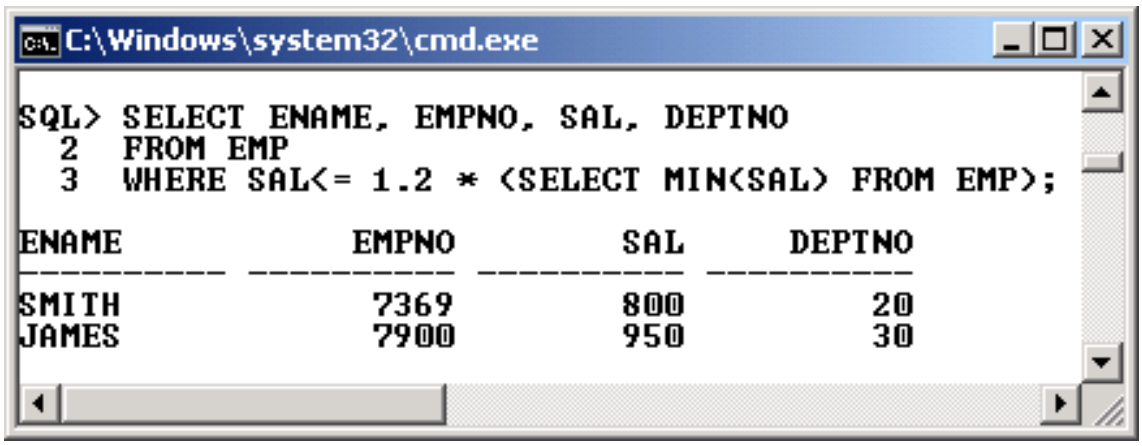


Рис. 2.76. **Арифметичний вираз з підзапитом в умові пошуку**

Приклад 2.72. Отримати відомості про співробітників, які працюють у відділі, чия назва є останньою у алфавітному розташуванні назв відділів (рис. 2.77).

```
SELECT ENAME, EMPNO, SAL, DEPTNO
FROM EMP WHERE DEPTNO =
    (SELECT DEPTNO FROM DEPT WHERE DNAME =
            (SELECT  MAX(DNAME) FROM DEPT));
```

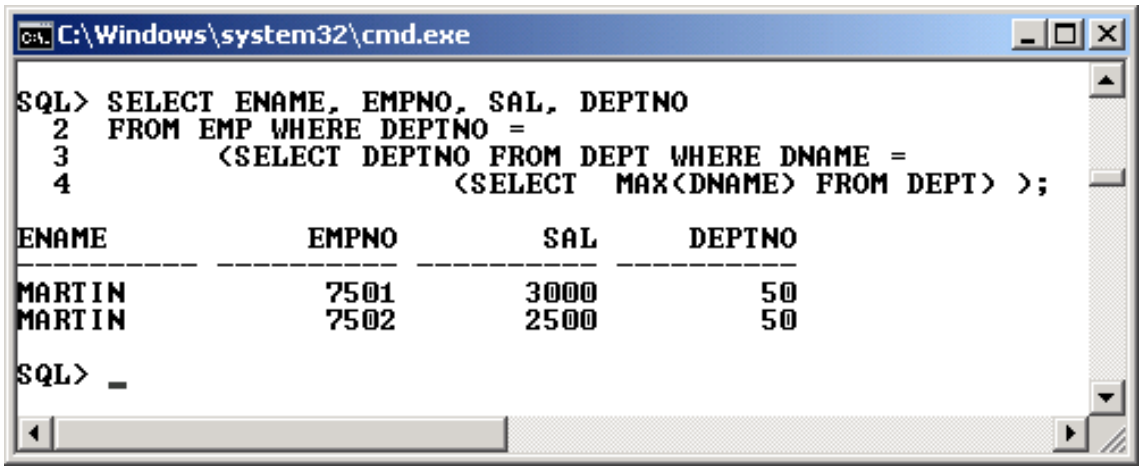


Рис. 2.77. **Декілька рівнів підзапитів**

Останній приклад ілюструє, по-перше, використання вкладених запитів на декількох рівнях вкладеності та по-друге, застосування функції MAX до нечислового атрибуту. Робота запиту потребує пояснення.

На найнижчому рівні вкладеності виконується запит

SELECT MAX(DNAME) FROM DEPT,

який знаходить назву відділу, що є останньою у алфавітному порядку назв. Це можливе тому, що коди символів є впорядкованими для комп'ютера, тому слова на літеру 'A' є меншими за слова на літеру 'C', а останні, у свою чергу, менші за слова на літеру 'Z'.

Далі для отриманої назви відділу шукається його номер у запиті на другому рівні вкладеності. І, нарешті, на найвищому рівні відбираються співробітники, що працюють у відділі зі знайденим номером.

Приклад 2.73. Отримати відомості про співробітників, чий оклад вищий за середній по відділу в якому вони працюють. Результат впорядкувати за зростанням номерів відділів, а у відділі – за зменшенням окладу (рис. 2.78).

```
SELECT ENAME, EMPNO, SAL, DEPTNO
FROM EMP E1  WHERE SAL >
      (SELECT AVG(SAL) FROM EMP E2
       WHERE E1.DEPTNO = E2.DEPTNO)
ORDER BY DEPTNO, SAL DESC;
```

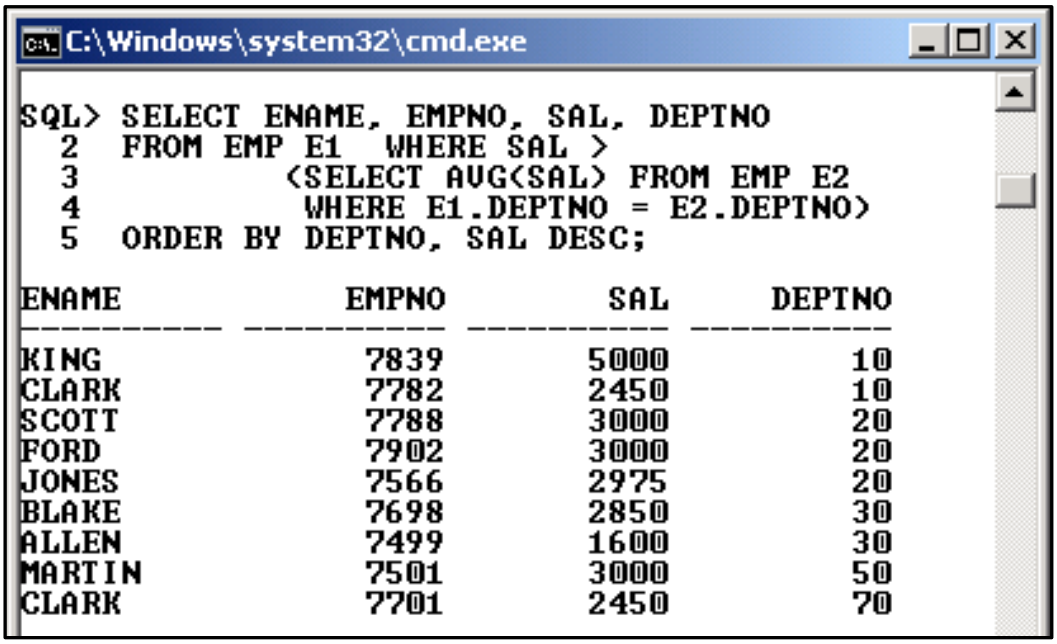


Рис. 2.78. **Корельований підзапит**

Пояснення. Наведений запит відноситься до типу корельованих, бо результат вкладеного підзапиту базується на умові, яка задається зовнішнім запитом. Алгоритм виконання виглядає таким чином:

- відбирається черговий (поточний) рядок із таблиці EMP (зовнішній запит), для нього визначається номер відділу;
- значення номеру відділу підставляється в умову вкладеного запиту;
- вкладений запит обчислює значення середньої заробітної плати для обраного відділу;
- перевіряється умова для зовнішнього запиту і якщо вона виконується, то рядок з даними про поточного співробітника додається до результату.

Попередні приклади ілюстрували використання вкладених запитів, що повертали одне значення та таким чином формували умову у конструкції **WHERE** для окремого стовпця результівної таблиці. Якщо така умова повинна накладатися на значення агрегатної функції, то, відповідно, застосовується речення **HAVING**. Тобто, якщо умова на значення агрегатної функції заздалегідь не визначена, а обчислюється у підзапиті, то такий підзапит включається в речення **HAVING**.

Приклад 2.74. Знайти відомості про відділи, де середня заробітна плата менша за середню по усій організації (рис. 2.79).

```
SELECT DNAME, DEPT.DEPTNO, AVG(SAL)
FROM DEPT INNER JOIN EMP ON DEPT.DEPTNO=EMP.DEPTNO
HAVING  AVG(SAL) < (SELECT AVG (SAL) FROM EMP)
GROUP BY DNAME, DEPT.DEPTNO;
```

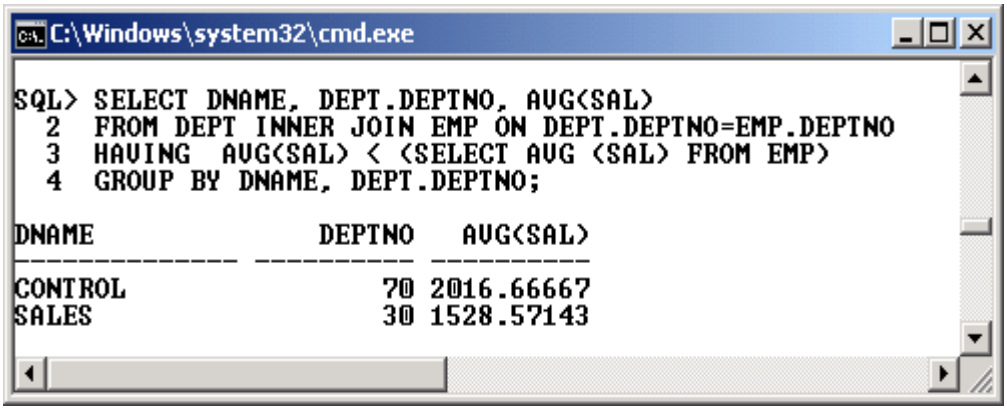


Рис. 2.79. **Умова HAVING на підзапит**

Приклад 2.75. Визначити відділи, у яких працює більше співробітників, ніж у відділі управління (CONTROL) (рис. 2.80).

```
SELECT DEPT.DEPTNO, DNAME, COUNT(*)
FROM DEPT, EMP
WHERE DEPT.DEPTNO = EMP.DEPTNO
HAVING COUNT(*) >  (SELECT COUNT (*) FROM EMP
          WHERE DEPTNO = (SELECT DEPTNO FROM DEPT
                                WHERE DNAME='CONTROL'))
GROUP BY DEPT.DEPTNO, DNAME;
```

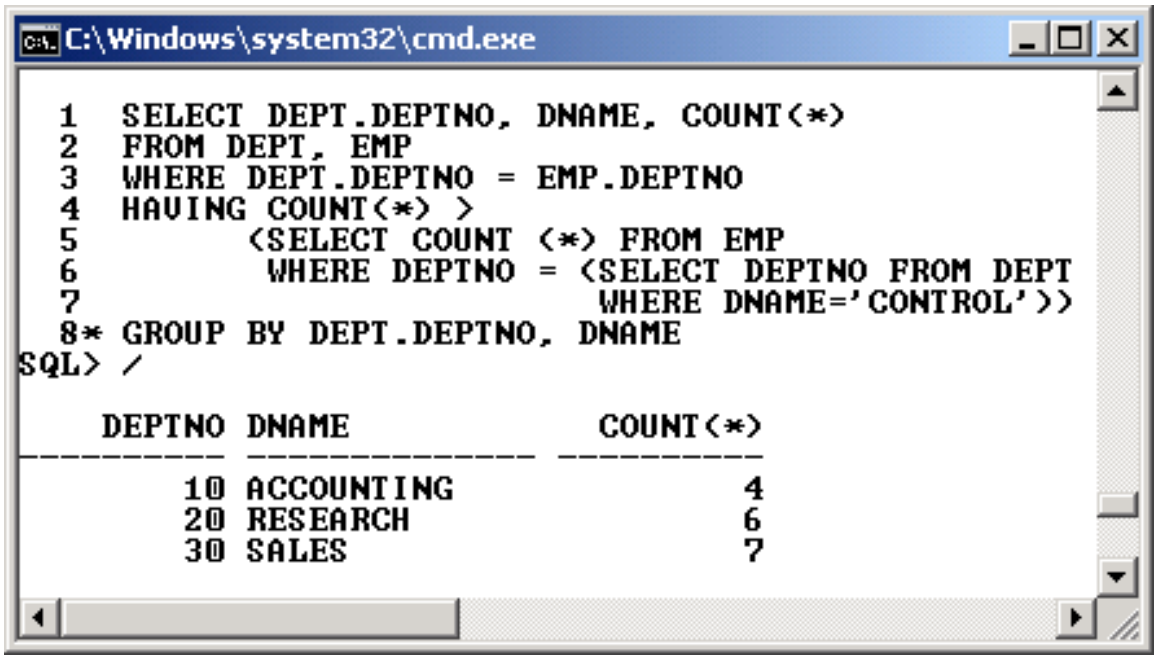


Рис. 2.80. **Умова HAVING на вкладені підзапити**

2.13.3. Підзапити, що повертають декілька значень

Приклади, що були наведені до цього часу для вкладних запитів, мали одну спільну особливість – підзапити у них повертали одне єдине значення, яке потім використовувалося в операторі порівняння у реченні **WHERE**. Але часто виникають ситуації, коли підзапит може повертати не одне, а декілька значень, в окремому випадку – 0. У цьому разі до підзапиту слід застосувати операції, які зазвичай застосовуються до множини, а саме:

• **{ WHERE | HAVING }** вираз **[NOT]** ***IN*** **(***підзапит***);**

• **{ WHERE | HAVING }** вираз оператор_порівняння **{ ALL | SOME | ANY }(***підзапит***);**

• **{WHERE | HAVING } [NOT]** ***EXISTS*** **(***підзапит***);**

Підзапити, що використовують IN або NOT IN

Результат внутрішнього підзапиту може відбирати від нуля до декількох значень. Після виконання внутрішнього запиту буде виконуватися зовнішній. Якщо в умові зовнішнього запиту використовується оператор належності до множини IN, то будуть відбиратися усі рядки у зовнішньому запиті, які відповідають умові рівності з будь-яким значенням внутрішнього запиту.

Приклад 2.76. Отримати відомості про співробітників, які працюють у Далласі чи Бостоні. Використати вкладений запит (рис. 2.81).

```
SELECT ENAME, EMPNO, SAL, DEPTNO
FROM EMP WHERE DEPTNO IN
        (SELECT DEPTNO FROM DEPT
         WHERE LOC IN ('DALLAS','BOSTON'));
```

Приклад 2.77. Визначити відділи та міста, в яких працюють співробітники на прізвище MARTIN (рис. 2.82).

```
SELECT DEPTNO, DNAME, LOC
FROM DEPT
WHERE DEPTNO IN
        (SELECT DEPTNO FROM EMP
         WHERE ENAME='MARTIN');
```

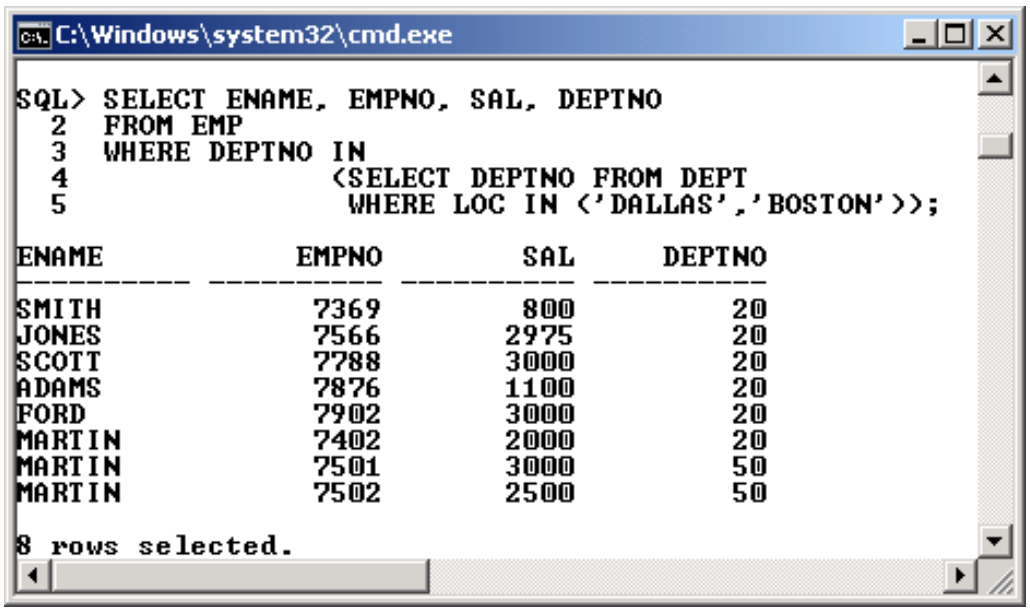


Рис. 2.81. **Використання умови IN для підзапита**

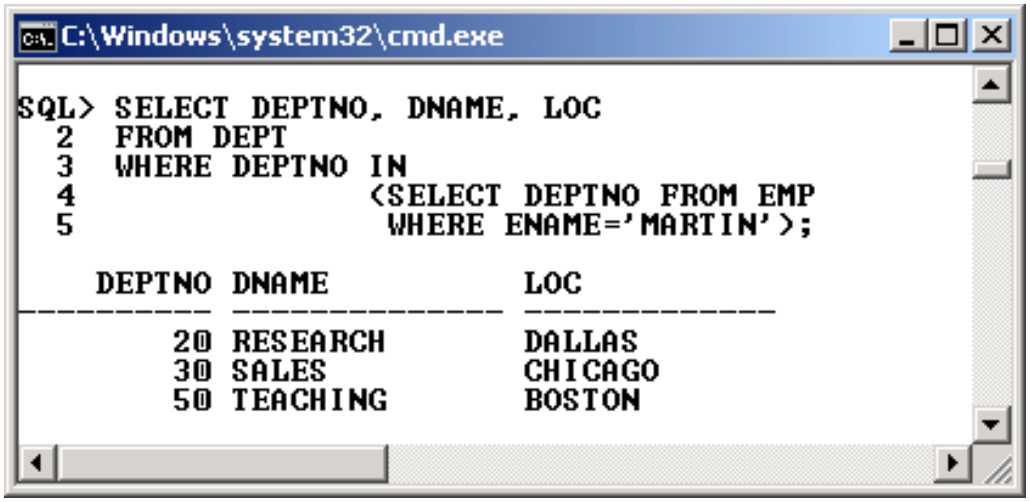


Рис. 2.82. **Відділи та міста, де працює MARTIN**

Приклад 2.78. Визначити співробітників, що були у відпустці влітку (рис. 2.83).

```
SELECT EMPNO, ENAME, JOB
FROM EMP WHERE EMPNO IN
    (SELECT EMPNO FROM VACATION
     WHERE EXTRACT(MONTH FROM HDATE_BEGIN) IN (6,7,8)
                OR
            EXTRACT(MONTH FROM HDATE_END) IN (6,7,8));
```

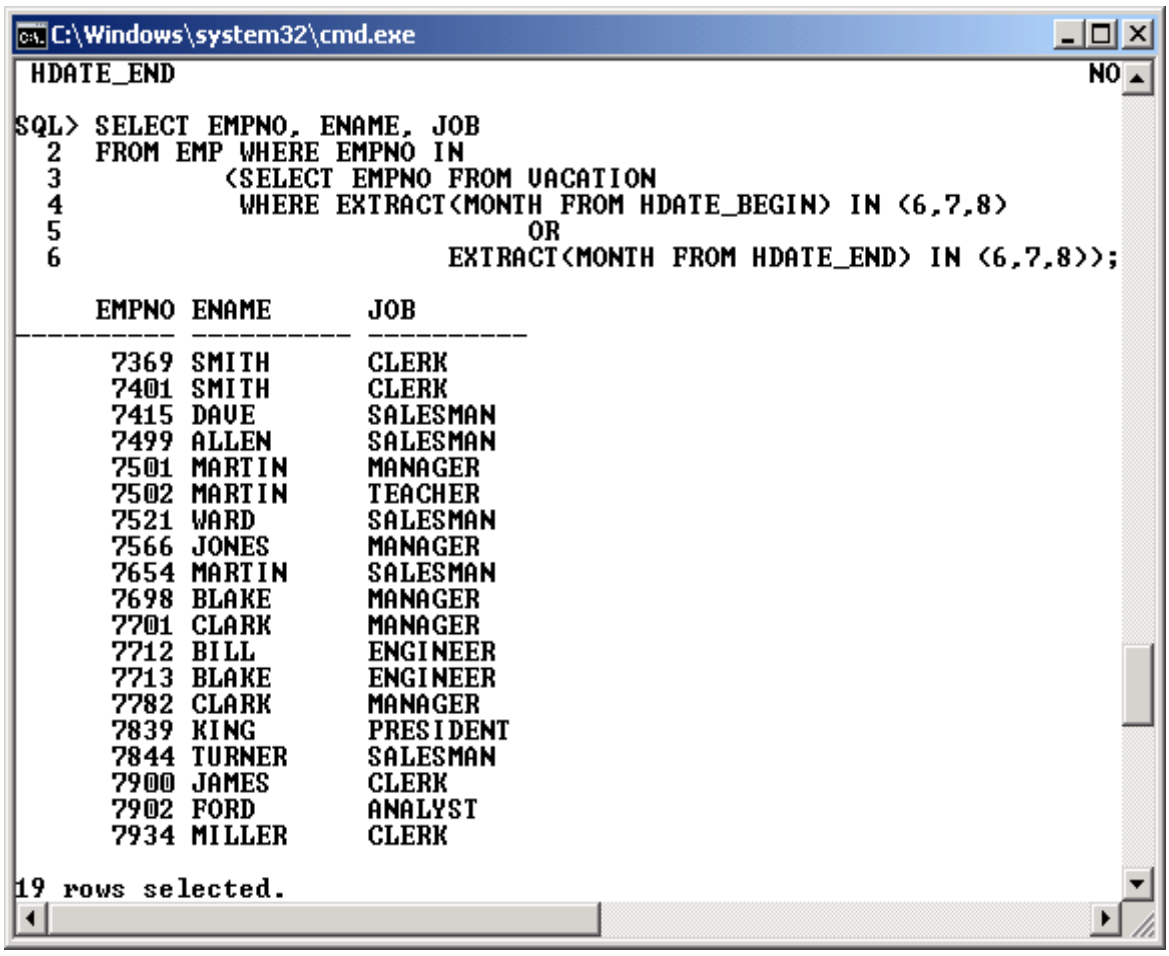


Рис. 2.83. **Співробітники, що відпочивали влітку**

У наведеному прикладі використовується функція EXTRACT, яка дозволяє (у даному прикладі) виділити з дати значення місяця.

Підзапити, що використовують ALL, ANY або SOME

У другому варіанті підзапитів, які повертають декілька рядків, використовуються ключові слова **ALL** або **ANY**. Ключове слово **SOME** еквівалентне **ANY**. Призначення цих ключових слів таке. Коли операція порівняння використовується зі словом **ALL** (усі), ця операція повинна виконуватися для усіх значень, що повертаються вкладеним запитом. Якщо ж використовується **ANY** (будь-який), то необхідно, щоб умова виконалась хоча б для одного значення, що повертає підзапит.

Приклад 2.79. Визначити співробітників, які отримують більшу заробітну плату ніж усі співробітники 30-го відділу. У даному випадку слово "усі" означає не загальну суму заробітних плат, а те, що заробітна плата співробітників, яких шукають, повинна бути більша за заробітну плату кожного (*будь-кого*) із працюючих у 30-му відділі (рис. 2.84).

```
SELECT EMPNO, ENAME, SAL, JOB
FROM EMP
WHERE SAL > ALL
        (SELECT SAL FROM EMP
         WHERE DEPTNO=30);
```

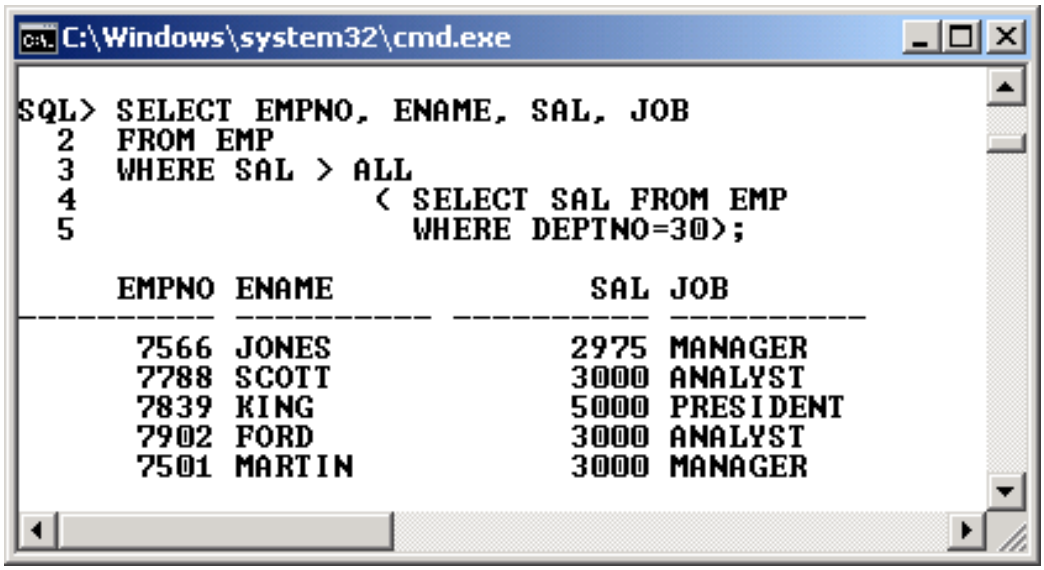


Рис. 2.84. **Використання ключового слова ALL**

Приклад 2.80. Визначити співробітників, заробітна плата яких менша за заробітну плату будь-якого співробітника 30-го відділу (рис. 2.85).

```
SELECT EMPNO, ENAME, SAL, JOB
FROM EMP
WHERE SAL < ANY
        (SELECT SAL FROM EMP
         WHERE DEPTNO=30);
```

Приклад 2.81. Визначити співробітників, що працюють у відділах, назва яких закінчується на ING (рис. 2.86).

```
SELECT EMPNO, ENAME, SAL, JOB, DEPTNO
FROM EMP
WHERE DEPTNO = ANY
        (SELECT DEPTNO FROM DEPT
         WHERE DNAME LIKE '%ING');
```

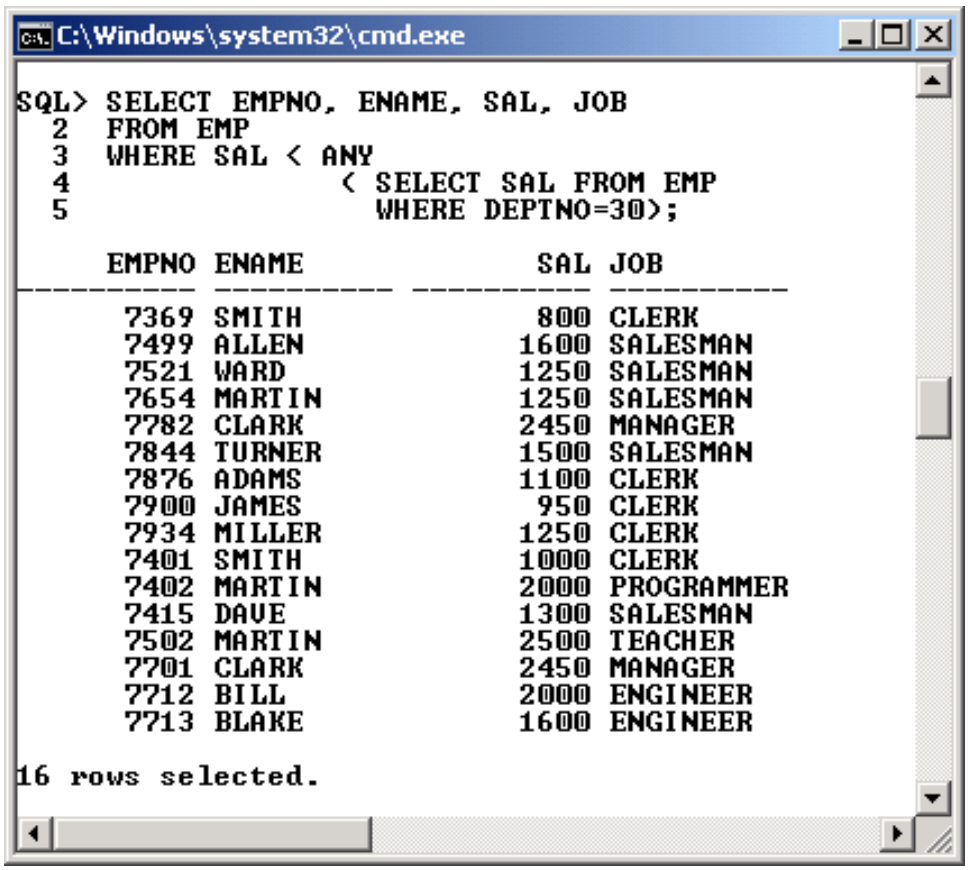


Рис. 2.85. **Використання ключового слова ANY** (операція "менше")

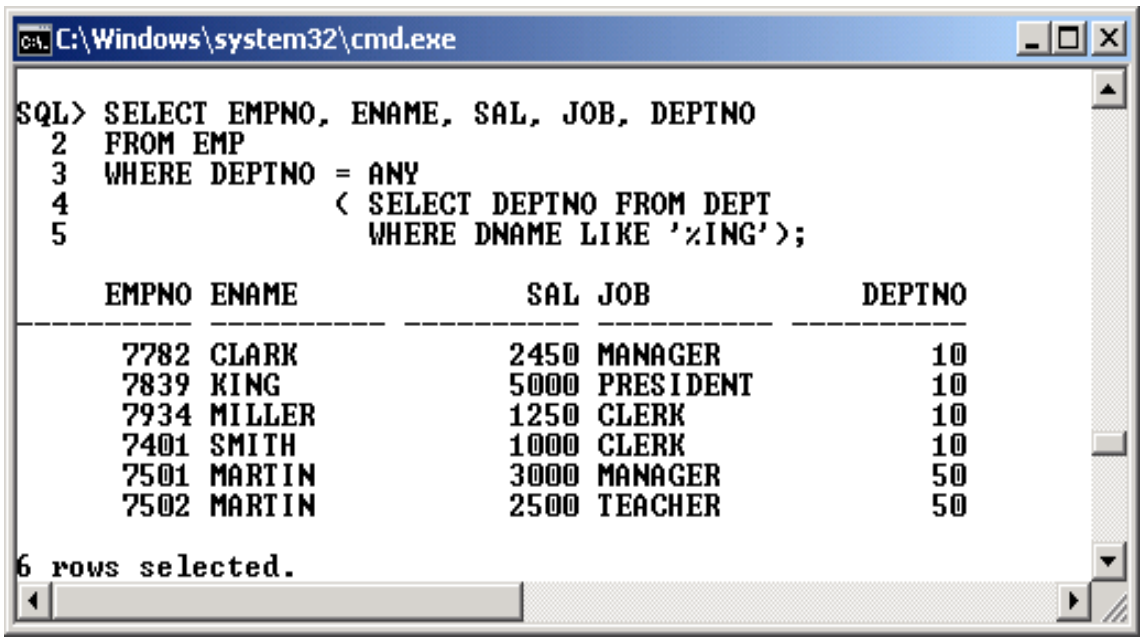


Рис. 2.86. **Використання ключового слова ANY** (операція "дорівнює")

Пояснення. У прикладі в умові відбору зовнішнього запиту була застосована конструкція = **ANY**, що еквівалентна IN.

Підзапити, що виконують перевірку на існування

У тому разі, коли підзапит починається з ключового слова **EXISTS**, він перевіряє, чи існують взагалі якісь дані, що відповідають вкладеному запиту. Іншими словами – поверне вкладений запит хоча б один рядок, чи ні? Якщо вкладений запит повертає хоча б один рядок, то результат операції **EXISTS** буде "істина", а **NOT EXISTS** – "неправда" і навпаки, якщо підзапит не повертає жодного рядка.

Правило використання **EXISTS** у підзапитах має свої особливості:

1. Перед словом **EXISTS** не записується жодне ім'я стовпця або вираз.

2. У фразі **SELECT** підзапиту, що починається з **EXISTS**, майже завжди використовується зірочка, оскільки важливим є тільки сам факт існування чи відсутності результату підзапиту.

Приклад 2.82. Визначити співробітників, які починали свою відпустку у липні. У запиті використати операцію перевірки існування **EXISTS** (рис. 2.87).

```
SELECT EMPNO, ENAME, SAL, JOB, DEPTNO
FROM EMP E1  WHERE EXISTS
        (SELECT * FROM VACATION V1
         WHERE V1.EMPNO=E1.EMPNO AND
                EXTRACT(MONTH FROM HDATE_BEGIN)=7);
```

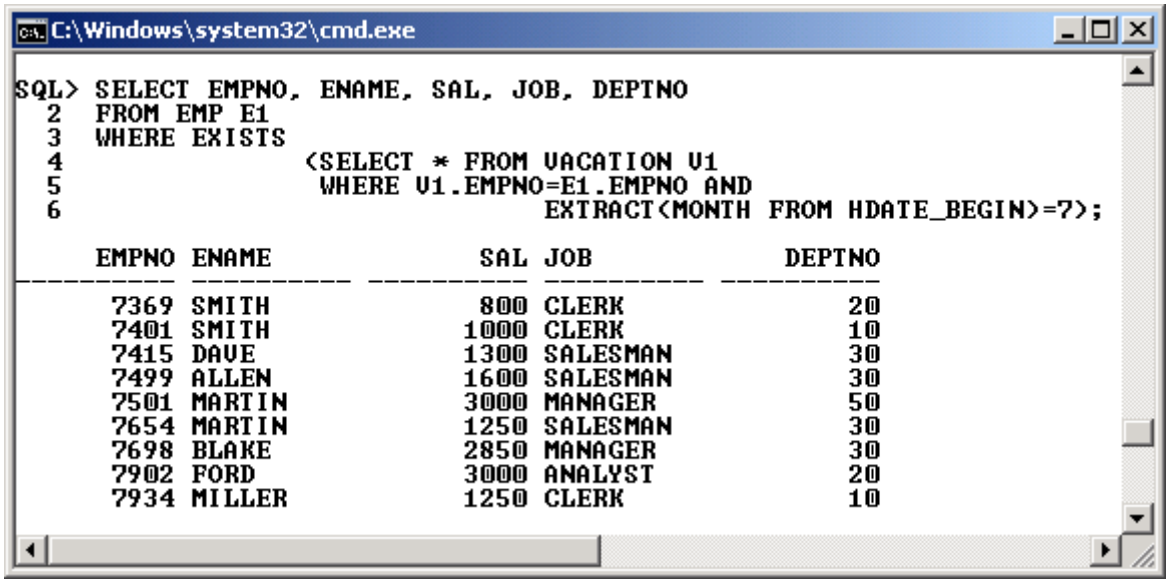


Рис. 2.87. **Використання ключового слова EXISTS**

Пояснення. Підзапити з **EXISTS** завжди є корельованими, тобто умова для відбору чи існування рядків підзапиту, базується на зовнішньому запиті. У наведеному прикладі у зовнішньому запиті перебираються рядки таблиці EMP. Для кожного такого рядка, а фактично – для кожного співробітника у внутрішньому запиті у таблиці VACATION шукається рядок з відповідним номером співробітника та місяцем початку відпустки, що дорівнює 7. Якщо буде відібраний хоч один рядок у підзапиті, то це еквівалентно результату "правда" операції **EXISTS** і, як наслідок, такий співробітник з верхнього запиту попаде у результат команди SELECT. Для скорочення запису використовуються псевдоніми (E1 та V1).

Приклад 2.83. Знайти відділи, у яких ще відсутні працівники (рис. 2.88).

```
SELECT * FROM DEPT
WHERE NOT EXISTS
      (SELECT * FROM EMP
       WHERE EMP.DEPTNO=DEPT.DEPTNO);
```

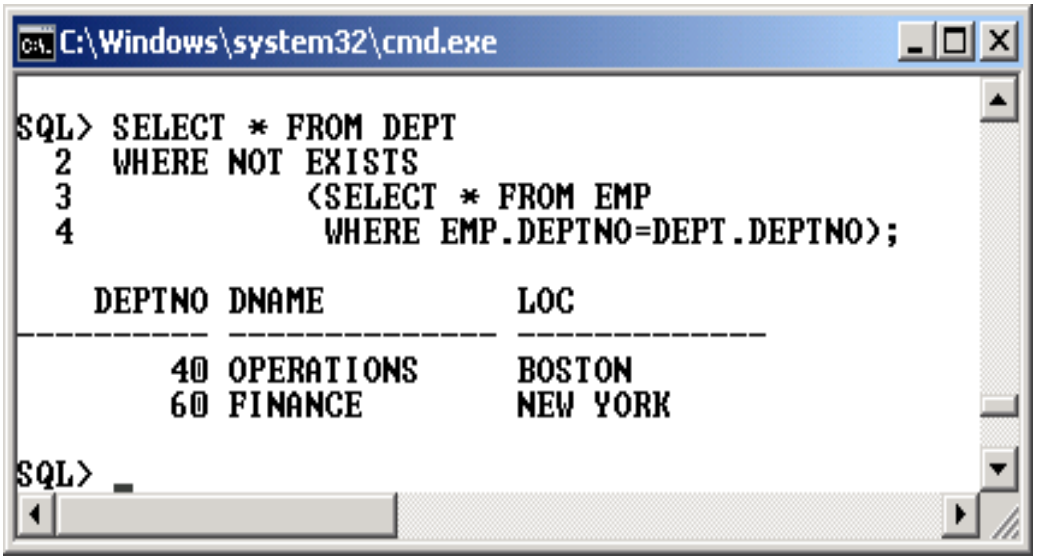


Рис. 2.88. **Використання NOT EXISTS**

Пояснення. У зовнішньому запиті для кожного відділа, що міститься у таблиці-довіднику DEPT, робиться спроба знайти у підзапиті співробітників, які у ньому працюють. У разі, коли жодного рядка у підзапиті не буде знайдено (тобто співробітників немає), назва такого відділу попаде до результівної таблиці.

Порівняння за декількома значеннями

Приклади, які були наведені для ілюстрації вкладених запитів зазвичай повертали один стовпчик, окрім операції перевірки існування **EXISTS**. Однак часто зустрічаються ситуації, коли умова накладається на декілька стовпців, що повертаються вкладеним запитом. SQL Oracle дозволяє записати таку умову достатньо витонченим шляхом.

Приклад 2.84. Визначити співробітників, що отримують найменшу заробітну плату у своєму відділі. Результат впорядкувати за номером відділу (рис. 2.89).

```
SELECT EMPNO, ENAME, SAL, JOB, DEPTNO
FROM EMP
WHERE (DEPTNO, SAL) IN
        (SELECT DEPTNO, MIN(SAL)
         FROM EMP
         GROUP BY DEPTNO)
ORDER BY DEPTNO;
```

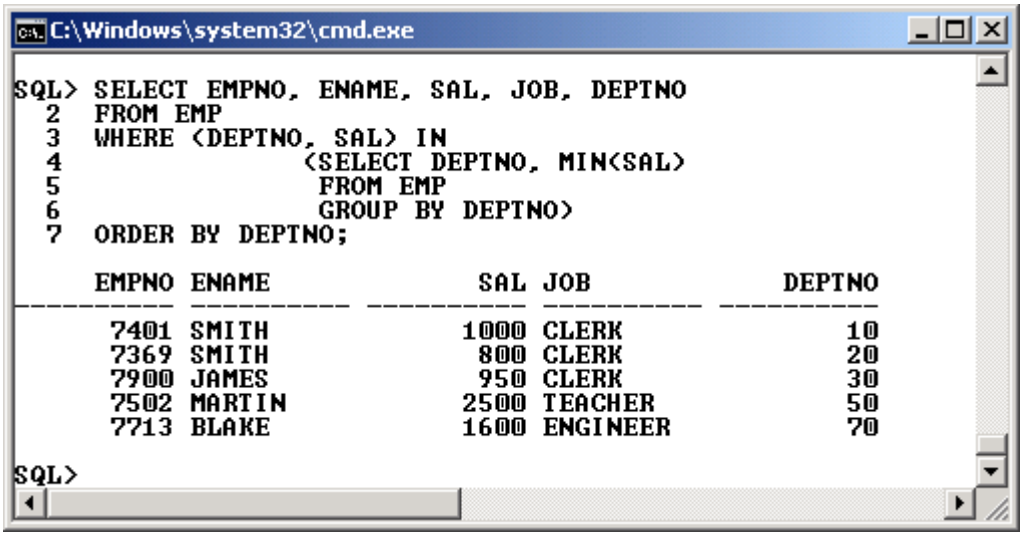


Рис. 2.89. **Умова на декілька стовпців підзапиту**

Пояснення. Підзапит повертає таблицю з двох стовпців, яка визначає мінімальну заробітну плату для кожного відділу. Зовнішній запит відбирає тільки тих співробітників з кожного відділу, заробітна плата яких дорівнює мінімальній по відділу. Фактично умова накладається не на один стовпець, а на декілька. А це потребує, щоб типи даних стовпців у реченні **WHERE** та у списку вибору підзапиту, співпадали.

Приклад 2.85. Отримати відомості про найдовшу за тривалістю відпустку для кожного співробітника. Результат впорядкувати за спаданням тривалості (рис. 2.90).

```
SELECT DISTINCT V1.EMPNO, ENAME,
        (HDATE_END – HDATE_BEGIN + 1) DUR
FROM VACATION V1 INNER JOIN EMP ON V1.EMPNO=EMP.EMPNO
WHERE (V1.EMPNO, (HDATE_END – HDATE_BEGIN +1)) IN
        (SELECT EMPNO, MAX (HDATE_END – HDATE_BEGIN +1)
         FROM VACATION
         GROUP BY EMPNO)
ORDER BY 3 DESC;
```

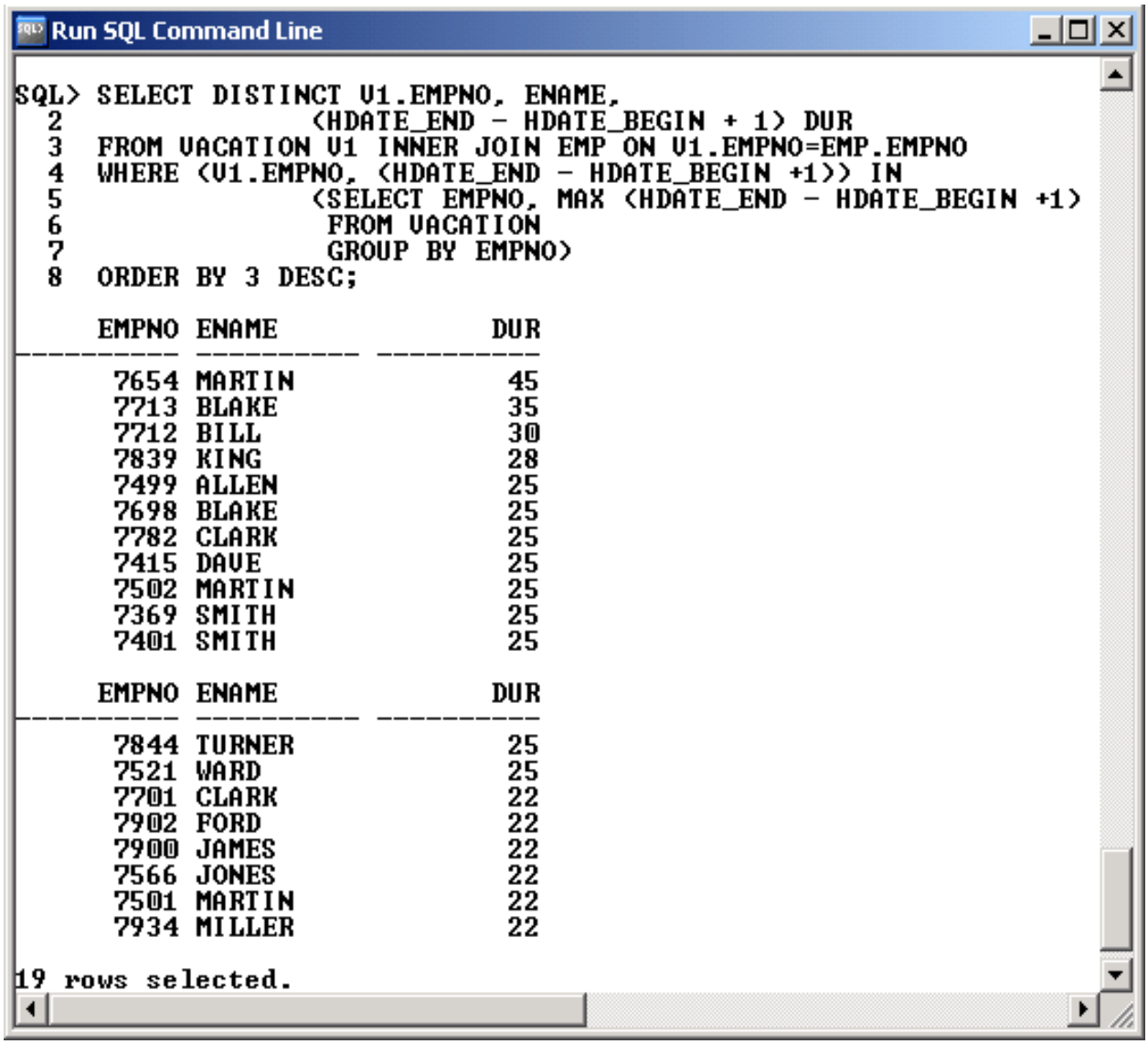


Рис. 2.90. **Умова на стовпець та вираз на основі підзапиту**

Підзапити у командах CREATE TABLE та INSERT.

Вкладені підзапити можна використовувати також у командах **CREATE TABLE** та **INSERT**, тобто у запитах на створення та заповнення таблиць.

Створення таблиці на основі вкладеного запиту

Такий підзапит формує нову таблицю у базі даних, яка за своєю структурою, тобто переліком стовпців та їх типами даних, відповідає результату вкладеного запиту SELECT, тобто запиту до наявних у БД об'єктів. Синтаксис команди такий:

CREATE TABLE table [(column1, [, column2...])]
AS SELECT...

де table – ім'я таблиці що створюється.

column – назви стовпців таблиці.

При цьому потрібно враховувати такі моменти:

Таблиця буде створена зі стовпцями, заданими в реченні SELECT, і рядками, вибраними за заданим у команді SELECT критерієм пошуку.

Тільки обмеження NULL / NOT NULL передається у створювану таблицю від таблиці-джерела і тільки тоді, коли вибирається значення стовпця, а не функція від нього.

Якщо всі стовпці в реченні SELECT мають імена, допустимі синтаксисом для імен стовпців таблиці, то (за відсутності явного задання імені стовпця створюваної таблиці) ці імена і будуть іменами стовпців створюваної таблиці.

Якщо в цьому випадку ім'я таблиці не відповідає синтаксису, то системою буде видано повідомлення про помилку.

Якщо імена стовпців таблиці задані, то їх кількість повинна відповідати кількості стовпців у реченні SELECT.

Приклад 2.86. Створити окрему таблицю, що містить дані про кількість співробітників у кожному відділі:

```
CREATE TABLE DEPT_CNT
AS SELECT DEPT.DEPTNO, DNAME,
                COUNT(EMPNO) AS EMP_CNT
   FROM DEPT, EMP
   WHERE DEPT.DEPTNO=EMP.DEPTNO
   GROUP BY DEPT.DEPTNO, DNAME;
```

Перевірте структуру та вміст таблиці DEPT_CNT за допомогою команд DESCRIBE та SELECT.

Приклад 2.87. Створити у базі даних нову таблицю на ім'я EMP_DMIN, яка повинна містити відомості про співробітників, що отримують мінімальну заробітну плату у своєму відділі (рис. 2.91)

```
CREATE TABLE EMP_DMIN AS
SELECT EMPNO, ENAME, JOB, SAL, DEPTNO
FROM EMP E1
WHERE SAL =
        (SELECT MIN(SAL) FROM EMP E2
         WHERE E1.DEPTNO = E2.DEPTNO);
```

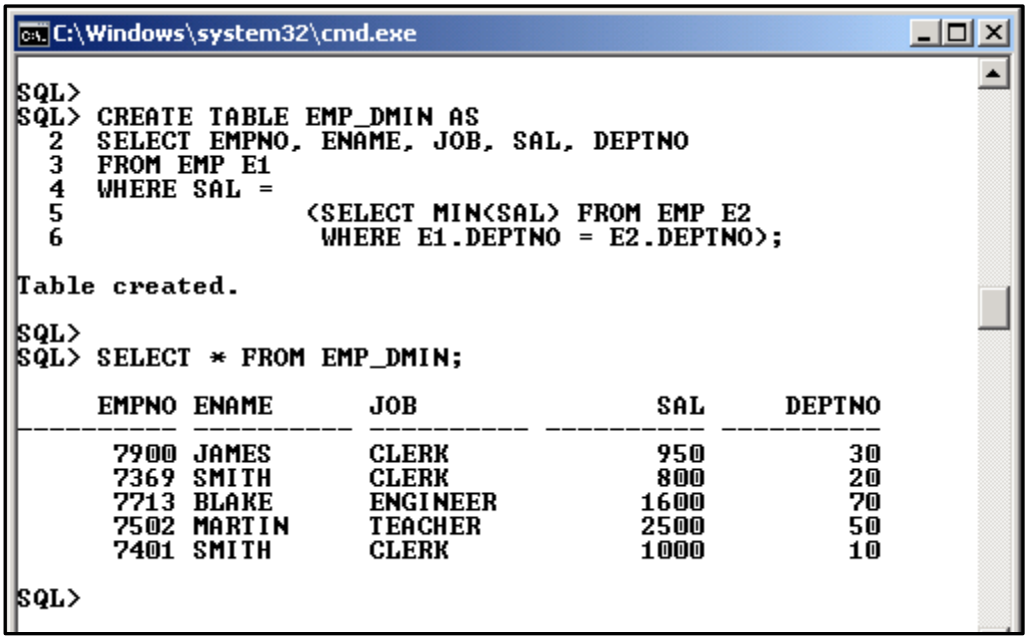


Рис. 2.91. **Таблиця співробітників з мінімальною зарплатою**

Заповнення таблиці на основі вкладеного запиту

Вкладені запити можуть також використовуватись для введення нових рядків у існуючі таблиці бази даних. У цьому разі у команді **INSERT** замість переліку значень, які записувалися у реченні **VALUES,** вказується запит **SELECT**, що вибирає рядки з існуючих таблиць бази даних та додає їх до іншої таблиці. Ця команда вимагає, щоб імена та порядок стовпців у підзапиті співпадали з іменами та порядком стовпців у команді **INSERT INTO**.

Приклад 2.88. Занести до таблиці BONUS відомості про співробітників, що отримали премію (рис. 2.92).

```
INSERT INTO BONUS
SELECT ENAME, JOB, SAL, COMM
FROM EMP
WHERE COMM IS NOT NULL;

SELECT * FROM BONUS;
```

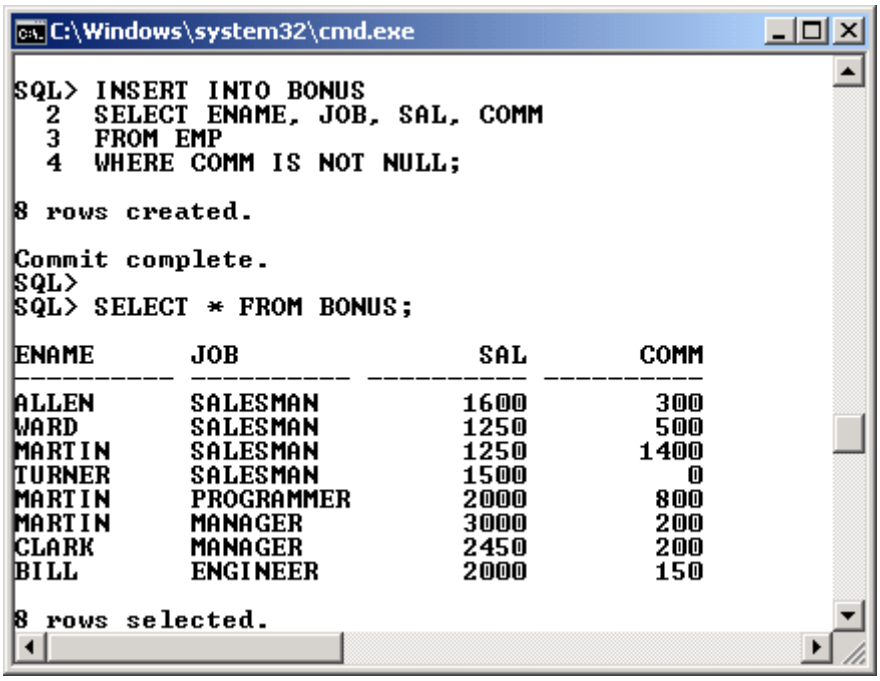


Рис. 2.92. **Заповнення таблиці BONUS**

Приклади використання вкладених підзапитів у командах UPDATE або DELETE розглянуті у підрозділі 2.16.

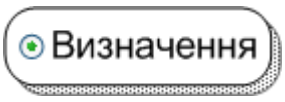
Слід зауважити, що не завжди результат команди, що містить вкладені запити працює належним чином. Чим більше рівень вкладеності, тим більш неочікуваним може вийти результат. У разі, якщо підзапити не є корельованими, можна рекомендувати поетапне їх тестування від запиту на найнижчому рівні до зовнішнього. Для корельованих підзапитів на етапі тестування умови, що посилаються на зовнішні таблиці, можна замінити на конкретні значення, а досягнувши правильного результату, відновити посилання.

Запитання і завдання

1. У яких випадках використовують підзапити?
2. Які види підзапитів вам відомі?
3. У яких випадках до підзапиту застосовують множинні операції? Наведіть приклади використання таких операцій.
4. Запишіть приклади використання підзапитів у командах CREATE TABLE та INSERT.

2.14. Що таке подання

Подання, або перегляд ("view" – англійською), є тимчасовою похідною (віртуальною) таблицею.



Подання – це запит на вибірку даних, що зберігається як окремий об'єкт бази даних Однак інформація в ньому не зберігається постійно, як у базових таблицях, а формується динамічно після звернення до нього.

Подання не може існувати самостійно, а визначається тільки в термінах однієї чи декількох таблиць або інших подань.

Подання використовують, щоб подавати дані таблиць спеціальним чином. Наприклад, для покупців роблять доступним подання ***Прайс***, що побудоване на основі таблиці ***Товари***, але в ньому відсутній стовпець ***Ціна_закупівлі***, значення якого є комерційною таємницею продавця.

Застосування подання дозволяє розробникові бази даних забезпечити кожному користувачу або групі користувачів найбільш зручні способи роботи з даними. Подання вирішує проблему простоти використання та безпеки даних. Подання – це фактично той самий запит, який виконується кожного разу, якщо бере участь у будь-якій команді.

CREATE| **ALTER**} VIEW ім'я_подання
[(ім'я_стовпця [,...n])]
[WITH ENCRYPTION] **AS SELECT**_команда [WITH CHECK OPTION]

Параметр WITH ENCRYPTION пропонує серверу шифрувати SQL-код запиту. Це унеможливлює його несанкціонований перегляд і викори-

стання. Якщо під час визначення подання необхідно приховати імена початкових таблиць і стовпців, а також алгоритм з'єднання даних, необхідно застосувати цей аргумент.

Параметр **WITH CHECK OPTION** пропонує серверу виконувати перевірку змін, що здійснюється через подання, на відповідність критеріям, які вказані в операторі **SELECT** цього ж подання.

Запит до подання на вибірку повністю аналогічний запиту **SELECT** на вибірку зі звичайної таблиці. Але на модифікацію вмісту бази даних через подання накладається декілька умов [14; 24; 26].

Не усі подання у SQL можуть бути такими, що дозволяють виконувати модифікацію базових таблиць у базі даних. Для того щоб через подання існувала можливість виконувати модифікацію бази даних, воно повинно бути таким, що:

- ґрунтується тільки на одній базовій таблиці;
- містить первинний ключ цієї таблиці;
- не містить ключового слова **DISTINCT** у своєму визначенні;
- не використовує речень **GROUP BY** або **HAVING** у своєму визначенні;
- по можливості не застосовує у своєму визначенні підзапити;
- не використовує константи або вирази значень серед вибраних стовпців;
- до подання обов'язково включають кожен стовпець таблиці, що має обмеження **NOT NULL**;
- оператор **SELECT** подання не використовує агрегатні (підсумкові) функції, з'єднання таблиць, збережені процедури і функції, які визначені користувачем;
- ґрунтується на поодинокому запиті, тому заборонено використовувати об'єднання **UNION**.

Приклад 2.89. Створити подання, що містить інформацію про співробітників, які отримали премію (рис. 2.93).

```
CREATE VIEW EMP_COMM AS
SELECT * FROM EMP WHERE COMM> 0 AND COMM IS NOT NULL;
```

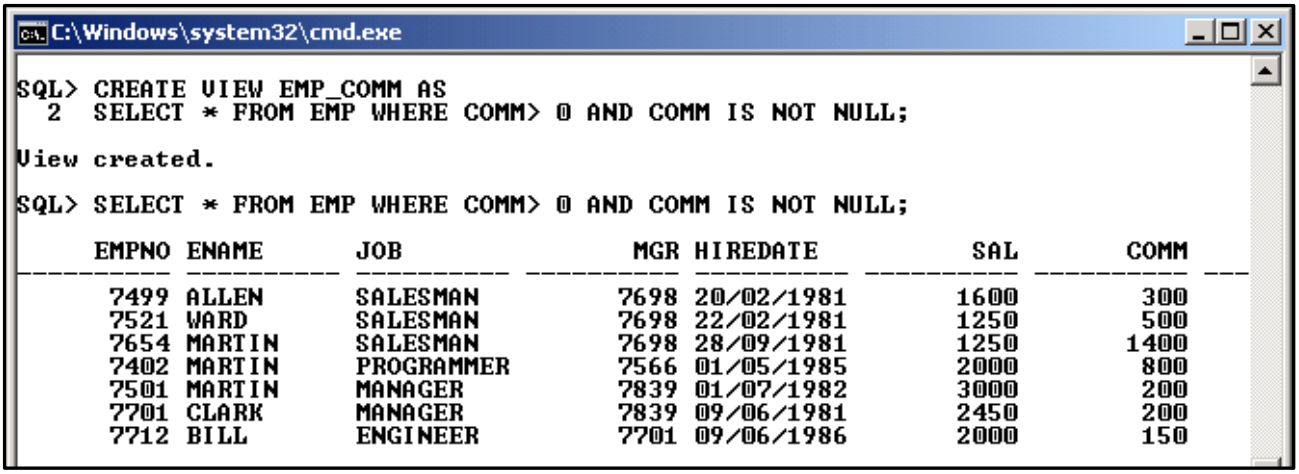


Рис. 2.93. **Подання про співробітників з премією**

. *Пояснення.* Таку складну умову у команді SELECT довелось записувати тому, що для одного зі співробітників (TURNER – 7844) у БД для премії записано значення 0, а не порожнє значення NULL.

Приклад 2.90. Створити подання, що містить відомості про відпустки співробітників, що працюють у БОСТОНІ (рис. 2.94).

```
CREATE VIEW BOSTON_VAC AS
SELECT VACATION.* FROM VACATION, EMP, DEPT
WHERE VACATION.EMPNO = EMP.EMPNO AND
        EMP.DEPTNO = DEPT.DEPTNO  AND LOC='BOSTON';
```

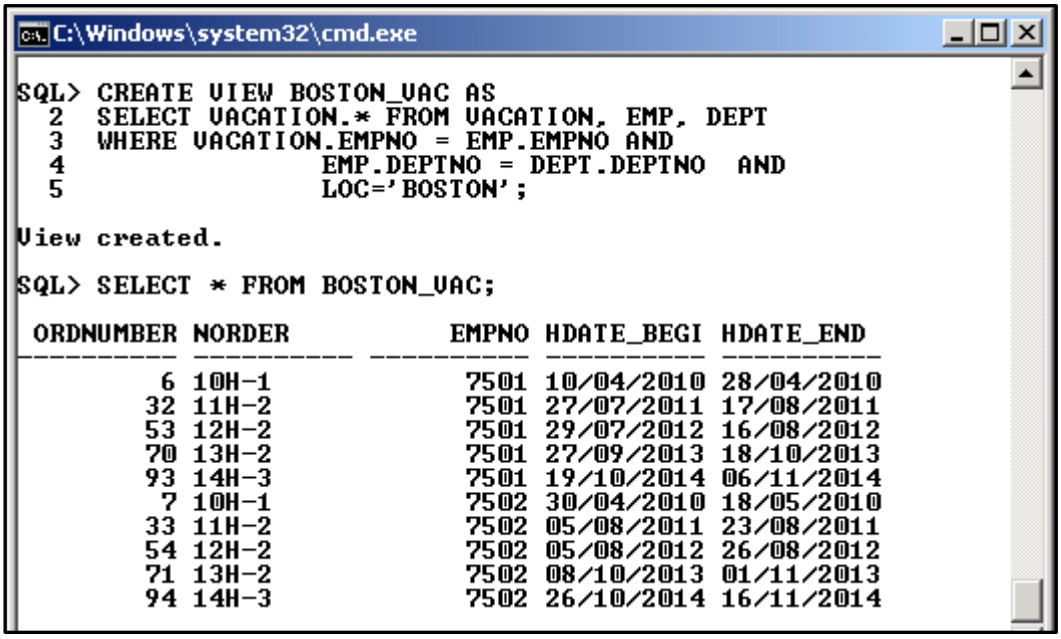


Рис. 2.94. **Подання про співробітників у Бостоні**

З наведених прикладів видно, що, перше подання дозволяє робити модифікацію даних, а друге ні. Слід відзначити, що для наведеного другого подання можна реалізувати модифікацію за допомогою тригерів (спеціальних процедур, які автоматично викликаються сервером при певних операціях модифікації – UPDATE, INSERT або DELETE), однак їх пояснення виходить за рамки цього посібника.

Запитання і завдання

1. Що таке подання?
2. У яких випадках використовують подання? Наведіть приклади.
3. Яким умовам має задовольняти подання, щоб за його допомогою можна було виконувати модифікацію вмісту бази даних?

2.15. Обробка ієрархічних структур у Oracle

Будь-яка база даних, що побудована на основі реляційної моделі, не зберігає дані у ієрархічній структурі, подібній до дерева у теорії графів. Однак у таблиці, що зберігаються у реляційній БД, можна закласти ієрархію відносин між певними об'єктами. Наприклад, це може бути класична задача розвузлування, що виникає при конструюванні тих чи інших агрегатів, машин, деталей тощо, або спроба побудувати ієрархію підпорядкування співробітників у тій чи іншій організації. Для того щоб вирішити подібну задачу, така ієрархія повинна бути закладена у відповідні таблиці реляційної БД. У базі даних, на якій демонструються приклади у посібнику, вона присутня та полягає у тому, що для кожного співробітника у таблиці EMP є значення номера його безпосереднього керівника у полі MGR.

Для обробки подібних ієрархічних структур (їх також називають деревоподібними) у СКБД Oracle часто використовують псевдостовпець LEVEL для визначення того, на якому рівні ієрархії перебуває вершина дерева (рядок) стосовно кореневої вершини. Загалом команда SELECT, що обробляє ієрархічні структури, має конструкції, наведені у табл. 2.2.

Таблиця 2.2

Елементи команди SELECT обробки деревоподібних структур

Ключове слово команди	Призначення
SELECT	Містить перелік стовпців, що вибираються з БД, плюс додатковий псевдостовпець LEVEL, котрий повертає рівень, на якому у дереві перебуває поточна вершина
FROM	Вказується тільки ім'я однієї таблиці
WHERE	Умова відбору рядків, що використовуються при побудові деревоподібної структури
CONNECT BY	Задання імен стовпців, у яких закладена ієрархічна залежність. Це речення необхідне для наведення деревоподібної структури
PRIOR	Визначає напрям, у якому "росте" дерево. Наприклад, якщо PRIOR з'являється перед стовпцем MGR, то значення стовпця MGR вибираються першими і потім шукається еквівалентне йому значення стовпця EMPNO. Дерево росте від підлеглого до керівника, знизу нагору. Якщо PRIOR стоїть перед EMPNO, то дерево буде рости зверху вниз, від керівника до підлеглих. Якщо при побудові дерева відбувається зациклення, то система видає повідомлення про помилку
START WITH	Визначає, з якої вершини почати побудову дерева. Не можна задавати в критерії стовпець LEVEL. Це речення не обов'язкове, навіть коли потрібне відтворення деревоподібної структури. При відсутності цього речення від кожного рядка-вершини буде побудоване дерево
ORDER BY	Задається наприкінці і визначає впорядкування рядків за значеннями певних стовпців.

Приклад 2.91. Отримати відомості щодо підпорядкування співробітників у організації, починаючи з керівника вищого рангу (рис. 2.95).

```
BREAK ON DEPTNO SKIP 1
SELECT LEVEL, DEPTNO, EMPNO, ENAME, JOB, SAL
FROM EMP
CONNECT BY PRIOR EMPNO = MGR
START WITH MGR IS NULL;
```

При обробці ієрархічних структур часто буває необхідно виключити з результату певні рядки (вершини). Для цього можуть бути використані конструкції WHERE та CONNECT BY.

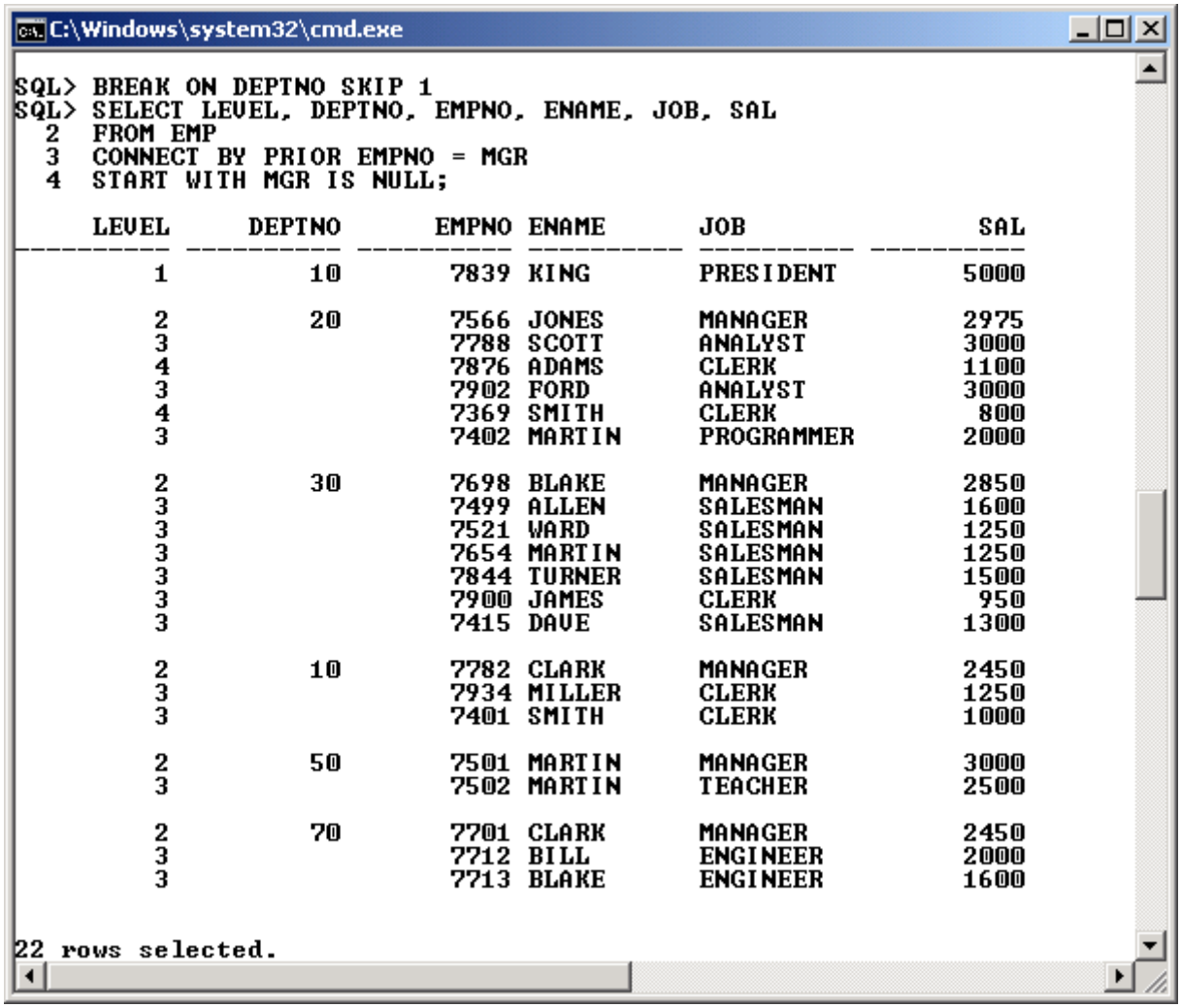


Рис. 2.95. **Підпорядкування співробітників у організації**

Якщо умова записана у WHERE, то тільки ті рядки (вершина), що відповідають умові, будуть виключені. Вершини, що перебувають по ієрархії вище й нижче, залишаться у таблиці результатів. Якщо ж обмеження на вершину задано у реченні CONNECT BY, то разом з вершиною буде виключена вся підлегла їй гілка дерева.

Приклад 2.92. Отримати відомості щодо підпорядкування співробітників у організації, починаючи з керівника вищого рангу, за виключенням співробітників на прізвище CLARK (рис. 2.96).

```
BREAK ON DEPTNO SKIP 1
SELECT LEVEL, DEPTNO, EMPNO, ENAME, JOB, SAL
FROM EMP
WHERE ENAME <> 'CLARK'
CONNECT BY PRIOR EMPNO = MGR
START WITH MGR IS NULL;
```

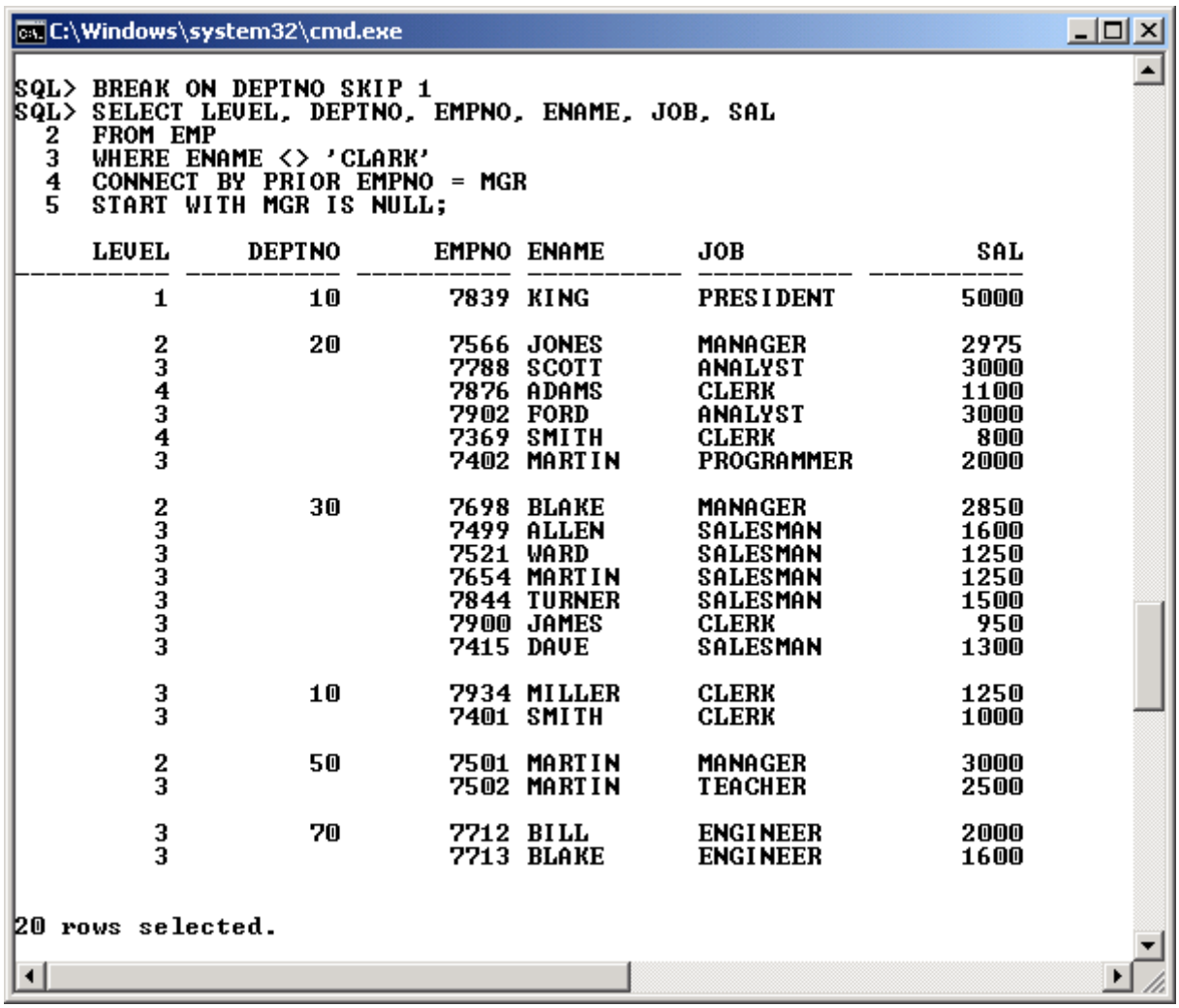


Рис. 2.96. **Підпорядкування співробітників за виключенням CLARK**

Приклад 2.93. Отримати відомості щодо підпорядкування співробітників у організації, починаючи з керівника вищого рангу, за виключенням співробітників на прізвище CLARK та їх підлеглих (рис. 2.97).

```
BREAK ON DEPTNO SKIP 1
SELECT LEVEL, DEPTNO, EMPNO, ENAME, JOB, SAL
FROM EMP
CONNECT BY PRIOR EMPNO = MGR AND ENAME <> 'CLARK'
START WITH MGR IS NULL;
```

У багатьох прикладних програмах ієрархічна (деревоподібна) структура може бути подана виведенням кожного організаційного рівня з певним відступом. Для цього, наприклад, можна використовувати пробіли на початку рядків.

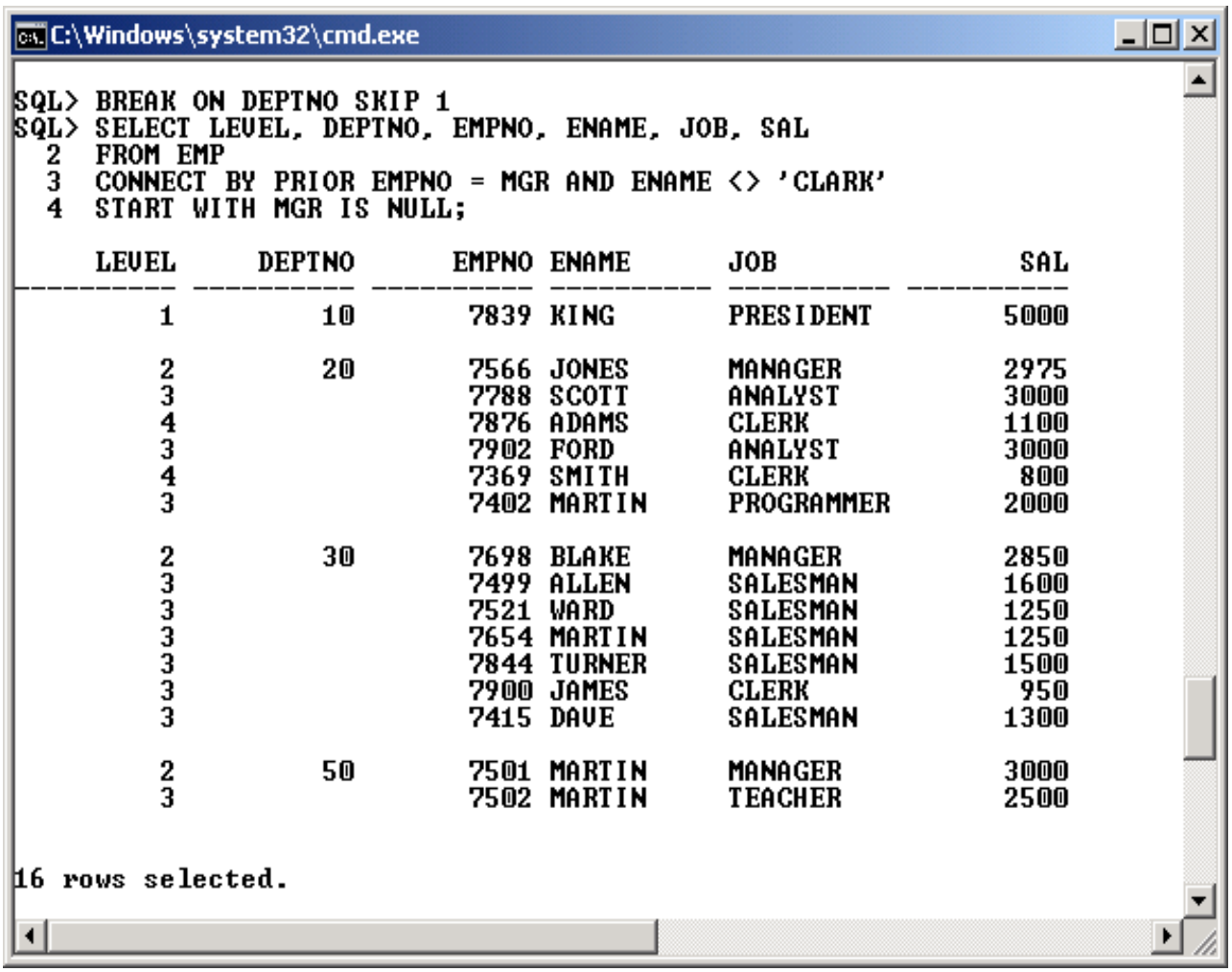


Рис. 2.97. **Підпорядкування за виключенням CLARK'а з підлеглими**

Приклад 2.94. Вивести на екран співробітників організації у вигляді ієрархічної структури (рис. 2.98).

```
SELECT RPAD('  ', 4*(LEVEL-1)) || ENAME AS HIERARCHY
FROM EMP
CONNECT BY PRIOR EMPNO = MGR
START WITH MGR IS NULL;
```

Пояснення. Функція RPAD (див п. 2.17.2) повертає рядок, доповнений справа пробілами, кількість яких визначається другим параметром. Отже, чим більший за номером рівень вершини, тим далі від краю вона відображається у результаті.

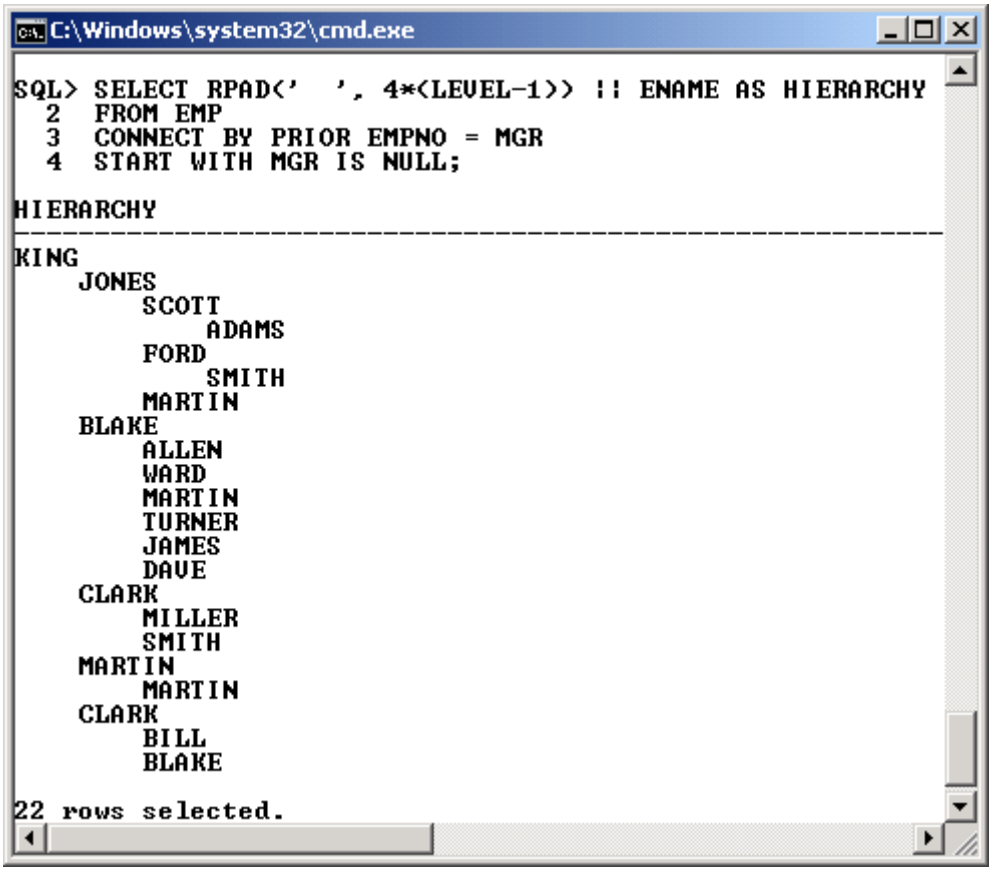


Рис. 2.98. **Ієрархічна структура організації**

Запитання і завдання

1. З якою метою у СКБД Oracle використовують псевдостовпець LEVEL?
2. Назвіть елементи команди SELECT, які використовують для обробки деревоподібних структур.
3. Наведіть приклад використання речення CONNECT BY.
4. Які засоби дозволяють вивести кожний організаційний рівень ієрархічної (деревоподібної) структури з певним відступом?

2.16. Команди модифікації даних

У процесі роботи будь-яка база даних повинна адекватно відображати поточний стан тієї предметної області, дані про яку вона містить. Реальна ситуація є такою, що дані у системі постійно змінюються. Під зміною (модифікацією) зазвичай розуміють три основні операції при роботі з таблицями: додавання (вставка), власне модифікація (оновлення) та видалення.

Операція додавання характеризує появу нових відомостей про стан предметної області, фактів, подій, що відбулися у дійсності, тощо.

Для виконання операції додавання у мові SQL існує команда **INSERT** (вставити, добавити). Вона подана у підрозділі 2.7, коли розглядались питання початкового заповнення таблиць навчальної бази даних.

Власне, *операція модифікації* (або оновлення) даних пов'язана з необхідністю вносити певні корективи в уже існуючі дані. Коригування даних диктується зміною тих чи інших обставин, фактів, подій, що відбуваються у реальній предметній області. Наприклад: зміна ціни на товар, зміна особистих даних працівника (освіти, посади, сімейного стану, окладу), зміна юридичної адреси підприємства тощо. Реалізація модифікації вже існуючих даних у базі мовою SQL здійснюється за допомогою команди **UPDATE** (обновити, скоригувати, виправити).

Третя *операція – видалення* зворотна до додавання. У реальній предметній області можуть виникати події, що призводять до необхідності виконання такої операції, наприклад: звільнення працівника з підприємства, зняття того чи іншого товару з виробництва, помилкове внесення тих чи інших даних до таблиці БД тощо. Реалізація видалення здійснюється командою **DELETE** мови SQL.

Як вже вказувалося, базова конструкція команди вставки – **INSERT** була розглянута в підрозділі 2.7, а її розширення, пов'язане з використанням вкладених запитів, було проілюстровано у п. 2.13.5.

Таким чином, більш детально у цьому підрозділі розглядаються команди **UPDATE** та **DELETE,** які відносяться до однієї таблиці або подання, яке дозволяє проводити модифікацію.

Крім команди DELETE видалити дані з таблиці у СКБД Oracle можна також за допомогою команди **TRUNCATE**.

2.16.1. Оновлення даних – команда UPDATE

Для того щоб за допомогою команди **UPDATE** виконати модифікацію даних у базі, необхідно дати відповідь на такі питання [2; 14; 24]:

- де виконують модифікацію (у якій таблиці чи поданні бази даних);
- що треба модифікувати (які рядки означеної таблиці та значення у яких стовпцях);
- на що треба змінювати значення у стовпцях (тобто нові значення).

Команда **UPDATE** якраз і дозволяє вказати відповіді на усі ці питання.

UPDATE ім'я_таблиці
SET ім'я_стовпця=значення [,ім'я_стовпця=значення]
[**WHERE** умова_пошуку]

У реченні **UPDATE** вказується ім'я таблиці, для якої необхідно виконати операцію модифікації.

Речення **SET** містить одну або декілька мовних конструкції виду ***ім'я_стовпця=значення***. У ньому вказується ім'я стовпця і його нове значення.

Речення **WHERE** виконує ту ж саму функцію, що й для команди **SELECT**, а саме – відбирає рядки таблиці, в яких повинні змінитися значення. У разі, коли речення **WHERE** відсутнє, оновлюються усі рядки означеної таблиці.

Слід також зазначити, що в умові пошуку можуть використовуватися вкладені запити, як це було для запитів на вибірку даних.

Приклад 2.95. У зв'язку з наказом керівництва змінити назву 50-го відділу на PROGRAMING (рис. 2.99) та підвищити заробітну плату співробітникам, що працюють на посаді CLERK, на 10 % (рис. 2.100).

```
UPDATE DEPT
SET DNAME='PROGRAMING'
WHERE DEPTNO = 50;


UPDATE EMP
SET SAL = SAL * 1.1   WHERE JOB = 'CLERK';


SELECT * FROM DEPT;
```

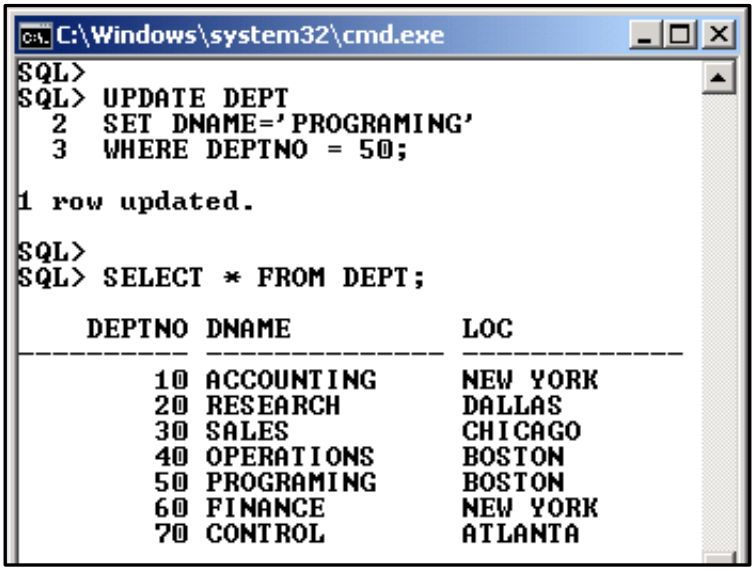


Рис. 2.99. **Зміна назви відділу**

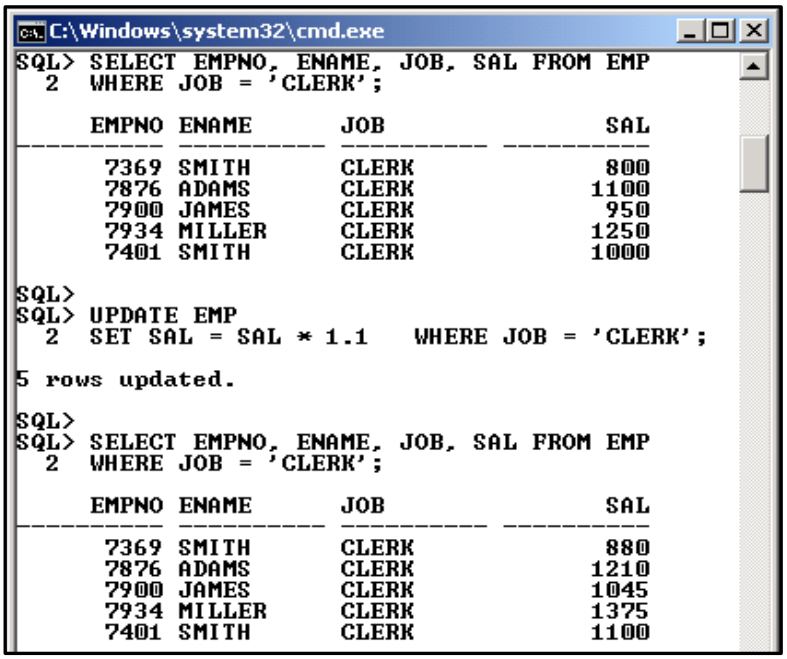


Рис. 2.100. **Модифікація заробітної плати 'CLERK'ів**

Приклад 2.96. Нагородити премією у розмірі 30 % окладу співробітників, що працюють у Далласі та Атланті (рис. 2.101).

```
UPDATE  EMP
SET COMM = SAL * 0.3
WHERE DEPTNO IN
    (SELECT DEPTNO FROM DEPT
     WHERE LOC IN ('DALLAS', 'ATLANTA'));
```

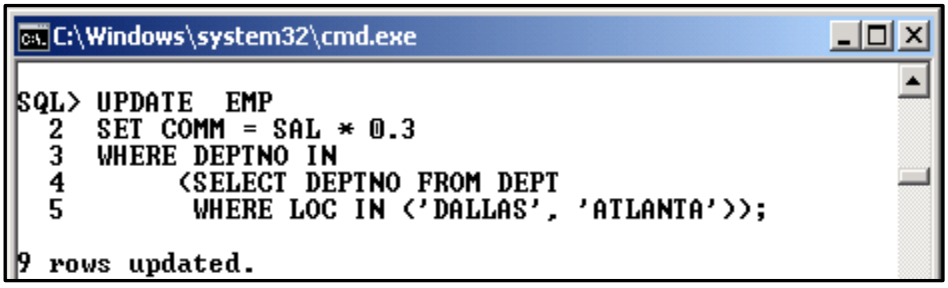


Рис. 2.101. **Вкладений підзапит у UPDATE**

Пояснення. В останньому прикладі у якості умови використовується вкладений підзапит. Використання таких підзапитів в умовах як для команди UPDATE, так і для команди DELETE нічим не відрізняється від їх використання у команді SELECT. Тому будь-які особливості у цьому випадку не існують.

Однак синтаксис команди UPDATE у СКБД Oracle має певні особливості, що відрізняють її від інших реалізацій, а саме:

- допускається використовувати аліасні імена таблиць для посилань на оновлювану таблицю у підзапитах;
- допускаються підзапити у правій частині фрази SET;
- допускається список оновлюваних колонок у лівій частині фрази SET;
- підзапити в реченні SET або WHERE можуть посилатися на оновлювану таблицю;
- команда UPDATE дозволяє виконувати оновлення підзапитів.

Проілюструвати ці особливості можна на таких прикладах:

```
UPDATE EMP E1
    SET (SAL,COMM)=(SELECT 1.05*AVG(SAL),1.25*AVG(COMM)
    FROM EMP E2 WHERE E1.DEPTNO=E2.DEPTNO);
```

У запиті змінюються значення полів SAL, COMM для усіх співробітників на нові значення, які визначаються з заданими коефіцієнтами середнім окладом та середньою премією для конкретного відділу, де працює співробітник.

```
UPDATE (SELECT * FROM EMP)
SET COMM=1000 WHERE JOB='MANAGER';
```

У цьому запиті встановлюється нове значення для премії менеджерів, що вибираються у підзапиті. Результат виконання повністю еквівалентний такому запиту на оновлення:

UPDATE EMP SET COMM=1000 WHERE JOB='MANAGER'

Спробуйте самостійно виконати наведені приклади та пояснити отримані результати, виконавши команду SELECT.

Зауваження. Рекомендується зробити копію таблиці EMP, наприклад EMP1, та усі експерименти проводити з нею.

2.16.2. Видалення даних – команда DELETE

Для того щоб за допомогою команди **DELETE** виконати видалення рядків у будь-якій таблиці БД, необхідно дати відповідь на такі питання:

- де виконують видалення (тобто для якої таблиці бази даних);
- що треба видаляти (які рядки означеної таблиці)?

DELETE FROM ім'я_таблиці [**WHERE** умова_пошуку]

Речення **DELETE** нічого не містить, воно тільки визначає саму команду видалення.

Речення **FROM**, аналогічно команді **SELECT**, містить ім'я таблиці, з якої необхідно видалити рядки. Але, на відміну від команди **SELECT** у реченні **FROM** команди **DELETE** можна вказати ім'я тільки однієї таблиці.

Речення **WHERE** виконує ту ж саму функцію, що й для команди **SELECT** або **UPDATE**, а саме – відбирає рядки таблиці, які потрібно видалити. У разі, коли речення **WHERE** відсутнє, видаляються усі рядки означеної таблиці. Тоді вона стає порожньою, тобто не містить жодного рядка.

Приклад 2.97. Видалити з таблиці EMP усі відомості про співробітників 50-го відділу.

DELETE FROM EMP WHERE DEPTNO = 50;

Приклад 2.98. Видалити з бази даних усі відомості про відпустки співробітників.

DELETE FROM VACATION;

У СКБД Oracle дозволяється не вказувати ключове слово FROM у команді DELETE. Тобто команду для приклада 2.97 можна записати так: DELETE EMP WHERE DEPTNO = 50;

2.16.3. Видалення даних – команда TRUNCATE

Команда TRUNCATE TABLE – не входить у стандарт ANSI; необоротно видаляє усі рядки з таблиці без занесення до журналу видалення окремих рядків. Ця інструкція швидко видаляє всі записи з таблиці, не впливаючи на структуру таблиці. Результати видалення командою TRUNCATE TABLE не можна скасувати після виконання.

TRUNCATE TABLE ім'я_таблиці

Команда TRUNCATE TABLE виконує дію, аналогічну команді DELETE без умови пошуку WHERE. Обидві інструкції видаляють всі рядки в цій таблиці. Однак є важливі відмінності.

Інструкція TRUNCATE TABLE працює швидше та не записується у журнал, тому не дозволяє зробити відкат у разі помилки. Крім того, інструкція TRUNCATE TABLE, на відміну від DELETE, не активує тригери. Цю команду не рекомендується використовувати у працюючих системах, що містять невідновні дані.

Приклад 2.99. Видалити з бази даних усі відомості про відпустки співробітників, використовуючи команду TRUNCATE.

```
TRUNCATE TABLE VACATION;
```

Запитання і завдання

1. Які команди модифікації даних вам відомі?
2. Наведіть приклад використання вкладеного підзапиту у команді UPDATE.
3. В чому полягає різниця між командами DELETE та TRUNCATE?

2.17. Використання вбудованих функцій СКБД Oracle

У запитах на мові SQL у СКБД Oracle, як і в інших СКБД, можна застосовувати не тільки агрегатні функції, які вже були розглянуті у підрозділі 2.11, а й звичайні скалярні вбудовані функції, яких достатньо у Oracle

[15; 20; 25; 28; 35]. Деякі функції такого роду вже використовувалися раніше у наведених запитах, наприклад NVL або EXTRACT. Необхідно також відзначити, що крім вбудованих функцій, користувач має можливість створювати мовою PL/SQL і свої функції, однак цей матеріал виходить за рамки даного посібника.

Функції можна застосовувати для виконання обчислень або інших операцій на основі значень аргументів функцій, якими найчастіше є значення, обрані з бази даних. Крім того, функції дозволяють перетворювати дані з одного типу на інший і змінювати формат відображення даних. У літературі з СКБД Oracle скалярні функції часто називають функціями одного рядка або однорядковими функціями. Ці функції можна розділити на ряд груп, а саме:

- числові функції;
- символьні функції;
- функції дати;
- функції перетворення;
- інші функції одного рядка.

Як і в мовах програмування, такі функції приймають у якості аргументів кілька значень різних типів і повертають одне значення певного типу даних. Такі функції можна використовувати у виразах замість значень, що мають аналогічний тип даних.

Необхідно більш докладно розглянути основні функції, що відносяться до різних груп.

2.17.1. Числові функції

Основні числові функції СКБД Oracle наведені у табл. 2.3.

Таблиця 2.3

Основні числові функції СКБД Oracle

Функція	Значення, що повертається
ABS (x)	Абсолютне значення величини **x**
ACOS (x)	Повертає арккосинус x. Область визначення **x**: від -1 до +1
ASIN (x)	Повертає арксинус **x**. Область визначення x: від -1 до +1
ATAN (x)	Повертає арктангенс **x**
BITAND (x, y)	Повертає результат виконання побітової операції AND над аргументами **x** та **y**, кожен з яких повинен бути невід'ємним цілим значенням
CEIL (x)	Найменше ціле, більше або рівне **x**

Закінчення табл. 2.3

Функція	Значення, що повертається
COS (x)	Косинус **x** (кута, вираженого у радіанах)
COSH (x)	Гіперболічний косинус **x**
EXP (x)	**e** у ступені **x**
FLOOR (x)	Найбільше ціле, менше або рівне **x**
LN (x)	Натуральний логарифм **x**, де x> 0
LOG (x, y)	Логарифм **y** по основі **x**
MOD (x, y)	Залишок від ділення **x** на **y**
POWER (x, y)	**x** у ступені **y**
ROUND (x [, y])	**x,**округлене до **y**-позицій після десяткової точки. За замовчуванням **y** дорівнює 0
SIGN (x)	-1 (Якщо x <0); 0 (якщо x = 0); 1 (якщо x> 0)
SIN (x)	Синус **x** (кута, вираженого у радіанах)
SINH (x)	Гіперболічний синус **x**
SQRT (x)	Квадратний корінь від **x**. Якщо x<0, повертає значення NULL
TAN (x)	Тангенс **x** (кута, вираженого у радіанах)
TANH (x)	Гіперболічний тангенс **x**
TRUNC (x [, y])	**x**, усічене до **y**-позицій після десяткової точки
WIDTH_BUCKET (x, min, max, num_buckets)	Створює гістограми однакової довжини на основі вхідних параметрів

Для ілюстрації роботи функцій у СКБД Oracle зазвичай використовують спеціальну системну таблицю DUAL. Ця таблиця доступна усім користувачам і завжди містить один стовпець з ім'ям DUMMY типу VARCHAR2 (1) і один рядок.

Приклад 2.100. Отримати значення тригонометричних функцій для кута у 1 радіан (рис. 2.102).

SELECT SIN(1), COS(1), TAN(1) FROM DUAL;

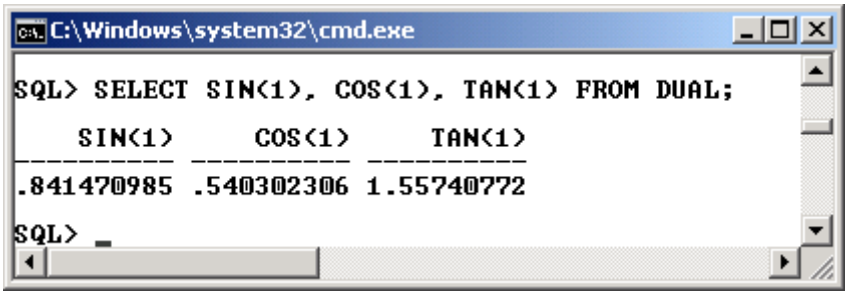


Рис. 2.102. **Використання тригонометричних функцій**

Для ілюстрації використання функцій сумісно з даними таблиць БД треба створити найпростішу таблицю з одного стовпця і занести до неї декілька рядків.

```
CREATE TABLE INT_NUM (N NUMBER(2));
INSERT INTO INT_NUM VALUES (1);
INSERT INTO INT_NUM VALUES (2);
INSERT INTO INT_NUM VALUES (3);
INSERT INTO INT_NUM VALUES (4);
INSERT INTO INT_NUM VALUES (5);
```

Приклад 2.101. Проілюструвати використання функції експоненти, натурального логарифма, квадратного кореня та функції усікання для даних, що занесені у таблицю INT_NUM (рис. 2.103).

```
SELECT N, EXP(N) EXPON, LN(N) NAT_LOG,
SQRT(N) KV_KOR, TRUNC(SQRT(N), 3) TRUNC3
FROM INT_NUM;
```

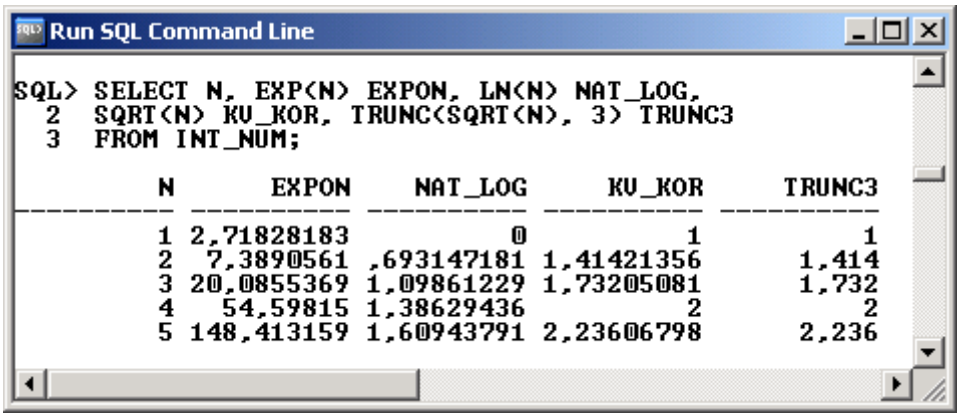


Рис. 2.103. **Використання експоненти, логарифма та інших функцій**

Особливого пояснення потребує функція WIDTH_BUCKET(x, min, max, num_buckets).

WIDTH_BUCKET дозволяє створювати гістограми однакової довжини на основі вхідних параметрів. Діапазон min... max ділиться num_buckets відрізків однакової довжини. Повертається відрізок, в який потрапляє x. Якщо x менше min, повертається 0. Якщо x більше або дорівнює max, повертається num_backets + 1. Ні min, ні max не можуть бути NULL, а num_buckets повинен бути позитивним цілим числом. Якщо x містить NULL, то повертається NULL.

Приклад 2.102. Розбити діапазон отримуваних заробітних плат (від 500 до 5 500) на десять рівних відрізків і визначити, до якого діапазону відноситься заробітна плата кожного співробітника (рис. 2.104).

```
SELECT EMPNO, ENAME, SAL,
       WIDTH_BUCKET(SAL, 500, 5500, 10) AS NOM
FROM EMP;
```

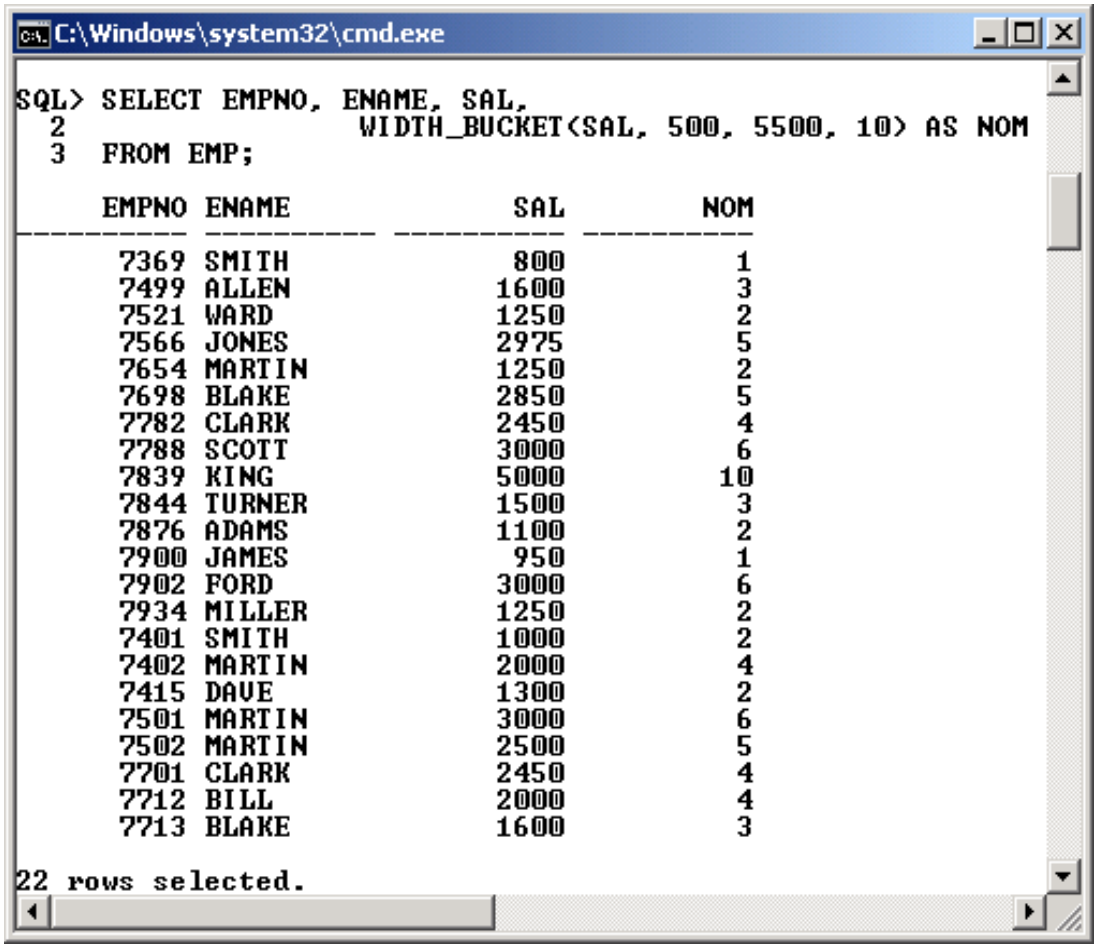


Рис. 2.104. **Визначення приналежності окладу до певної групи**

Приклад 2.103. Підрахувати кількість співробітників, чия заробітна плата відноситься до певного діапазону. Результат впорядкувати за номером діапазону (рис. 2.105).

```
SELECT   WIDTH_BUCKET(SAL, 500, 5500, 10) AS NOM, COUNT(*)
FROM EMP
GROUP BY WIDTH_BUCKET(SAL, 500, 5500, 10)
ORDER BY 1;
```

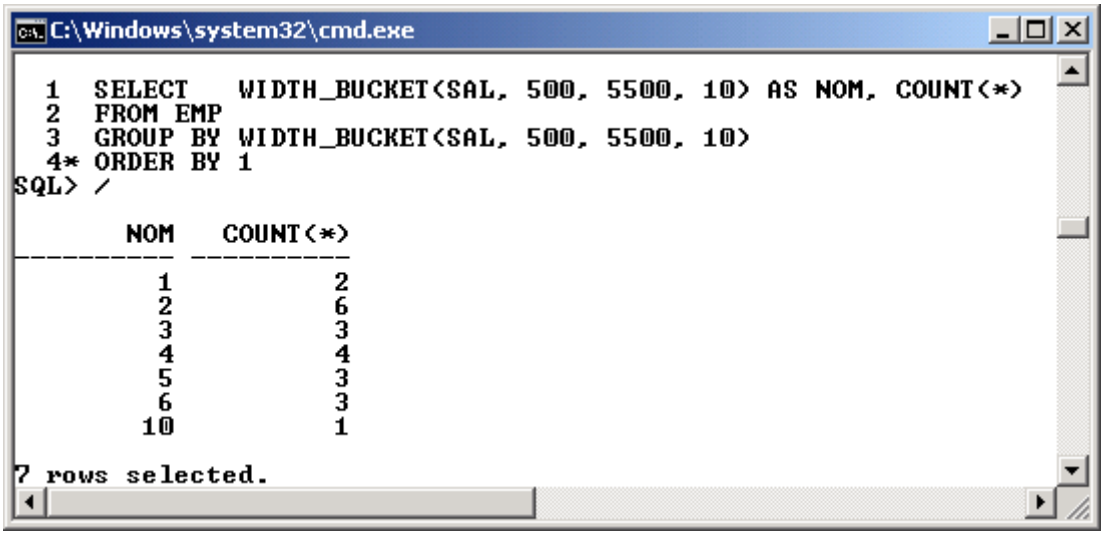


Рис. 2.105. **Числова гістограма розподілення окладів**

2.17.2. Символьні функції

Символьні функції можна поділити на дві групи: функції, які повертають числові значення (табл. 2.4) та функції, які повертають символьні значення (табл. 2.5). Серед аргументів цих функції зазвичай є такий, що містить символьне значення (рядок).

Таблиця 2.4

Символьні функції, що повертають числове значення

Функція	Призначення
ASCII (sfring)	Повертає десяткове подання першого байта рядка string із набору символів бази даних. Ця функція обернена до функції CHR. CHR повертає символ для коду символу, а ASCII – код символа для символу.
INSTR (string1, string2 [, a] [, b])	Повертає позицію входження рядка string2 у рядок string1; величини **a** і **b** вимірюються в символах (починаючи з символа a шукається входження за номером **b**. За замовчуванням a=b=1)
INSTRB (string1, string2 [, a] [, b])	Повертає позицію входження рядка string2 в рядок string1; величини **a** і **b** вимірюються в байтах
INSTRC (string1, string2 [, a] [, b])	Повертає позицію входження рядка string2 в рядок string1; величини **a** і **b** вимірюються в повних символах Unicode
LENGTH (string)	Повертає довжину рядка, виміряну у символах
LENGTHB (string)	Повертає довжину рядка, виміряну у байтах
LENGTHC (string)	Повертає довжину рядка, виміряну у повних символах Unicode

Таблиця 2.5

Символьні функції, що повертають символьне значення

Функція	Призначення
CHR (x [USING NCHAR_CS])	Повертає символ, який має значення, еквівалентне **x** у наборі символів бази даних. CHR і ASCII є протилежними функціями. CHR повертає символ для заданого номера символу, а ASCII повертає номер заданого символу. Якщо вказано USING NCHAR_CS, то у якості набору символів використовується національний набір символів бази даних
CONCAT (string1, string2)	Повертає рядок string1 у з'єднанні з рядком string2. Ця функція ідентична оператору ||. Значення, що повертається, завжди має тип VARCHAR2
INITCAP (string)	Повертає рядок, у якому перший символ кожного слова є прописним, а решта символів кожного слова – малі. Слова розділяються пробілами або не алфавітно-цифровими символами. Символи, які не є буквами, не змінюються. Результат функції має той же тип даних, що і рядок string
LOWER (string)	Повертає рядок, де всі символи малі. Символи, які не є буквами, не змінюються. Результат функції має той же тип даних, що і рядок string
LDAP (string1, x x [, string2])	Повертає рядок string1, доповнений зліва до довжини **x** символами з рядка sting2. Якщо string2 містить менше **x** символів, він дублюється. Якщо string2 містить більше **x** символів, використовуються тільки перші **x** символів string2. Якщо рядок string2 не вказаний, то за замовчуванням застосовується одиночний пробіл. LDAP веде себе аналогічно RDAP, за винятком того, що доповнює рядок зліва, а не справа
LPAD (string1, x x [, string2])	Повертає рядок string,1 доповнений ліворуч до довжини **x** символами рядка string2. Якщо string2 містить менше **x** символів, він дублюється. Якщо string2 містить більш **x** символів, застосовуються тільки перші **x** символів. Якщо string2 не вказаний, то за замовчуванням використовується одиночний пробіл. LPAD веде себе аналогічно RPAD, за винятком того, що додає символи ліворуч
LTRIM (string1, string2)	Повертає рядок string1 з видаленими лівими символами, що зустрічаються у рядку string2. За замовчуванням, string2 відповідає одиночному пробілу. База даних буде переглядати string1, починаючи з самої лівої позиції. Дійшовши до першого символу, що не зустрічається у string2, вона поверне результат. LTRIM веде себе аналогічно RTRIM, за винятком того, що видаляє символи ліворуч, а не праворуч

Продовження табл. 2.5

Функція	Призначення
REPLACE (string, search_str [, replace_str])	Повертає рядок, де кожне входження символу search_str замінено символом replace_str. Якщо replace_str не вказаний, то всі примірники search_str видаляються. REPLACE реалізує підмножину функціональності, що надається функцією TRANSLATE
RPAD (string1, x x [, string2])	Повертає рядок string1 доповнений праворуч до довжини **x** символами рядка string2. Якщо string2 містить менше **x** символів, він дублюється. Якщо string2 містить більше **x** символів, застосовуються тільки перші **x** символів. Якщо string2 не вказаний, то за замовчуванням використовується одиночний пробіл. RPAD веде себе аналогічно LPAD, за винятком того, що додає символи праворуч, а не ліворуч
RTRIM (string1 [, string2])	Повертає рядок string1 з видаленими правими символами, присутніми в рядку string2. За замовчуванням, string2 є пробілом. База даних буде переглядати string1, починаючи з самої правої позиції. Дійшовши до першого символу, що не зустрічається в рядку string2, вона поверне результат. RTRIM веде себе аналогічно LTRIM, за винятком того, що видаляє символи праворуч, а не ліворуч
SOUNDEX (string)	Повертає фонетичне подання рядка string. Ця функція дозволяє порівнювати слова з різним написанням, але які звучать схоже (у англійській мові)
SUBSTR (string, a [, b])	Повертає частину рядка string починаючи з символу **a** – **b** символів
SUBSTRB (string, a [, b])	Повертає частину рядка string; величини **a** і **b** вимірюються в байтах
SUBSTRC (string, a [, b])	Повертає частину рядка string; величини **a** і **b** вимірюються в повних символах Unicode
TRANSLATE (string, from_str, to_str)	Повертає рядок string, де входження кожного символу з рядка from_str замінюється відповідним символом з рядка to_str. Якщо from_str довший to_str, то всі символи з from_str, відсутні у to_str, видаляються із string, бо вони не мають відповідних символів. Рядок to_str не може бути порожнім. Oracle інтерпретує порожній рядок як NULL, і якщо будь-який аргумент TRANSLATE буде NULL, то результат також буде NULL
TRIM ([{{LEADING | TRAILING | BOTH} [trim_char]) | trim_char} FROM] string)	Повертає рядок string з видаленими символами trim_char на початку або/та наприкінці. Trim_char повинен бути одиночним символом, і за замовчуванням використовується пробіл. Якщо не вказано LEADING, TRAILING або BOTH, екземпляри символу trim_char видаляються з обох кінців рядка string

Закінчення табл. 2.5

Функція	Призначення
UPPER (string)	Повертає рядок, де всі символи переведені у верхній регістр. Якщо рядок має тип CHAR, повернене значення також буде CHAR. Якщо рядок має тип VARCHAR2, повертається значення VARCHAR2. Символи, які не є буквами, залишаються без зміни

Приклад 2.104. Знайти довжину назви відділів, позицію підрядка 'CO' у назві відділів та позицію другої літери 'E' у назві відділів із таблиці DEPT (рис. 2.106).

```
SELECT DNAME,  LENGTH(DNAME) LD,
          INSTR(DNAME,'CO') PCO, INSTR(DNAME,'E',1,2) E2
FROM DEPT;
```

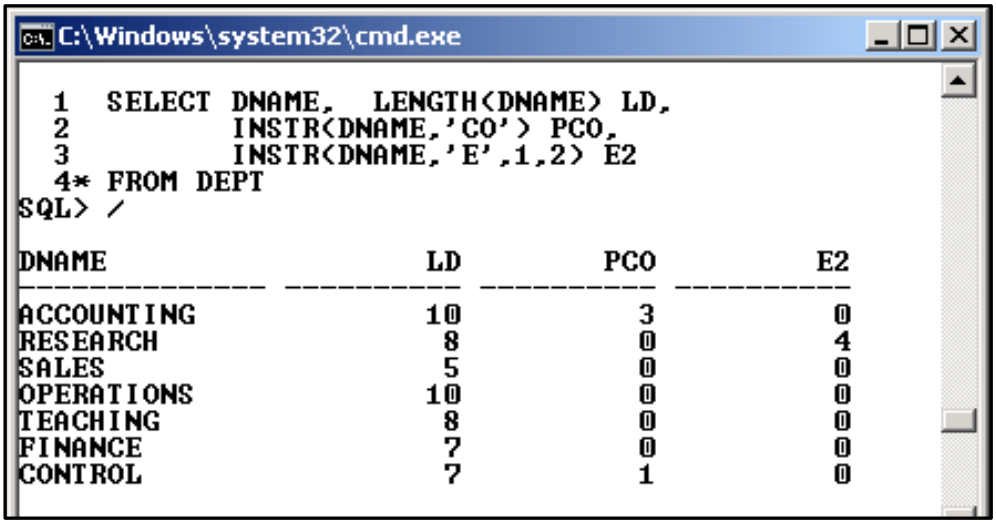


Рис. 2.106. **Символьні функції, що повертають числове значення**

Символьні функції, що повертають символьне значення

Ці функції в якості аргументів приймають символи (крім функцій CHR і NCHR) і повертають символьні значення. Більшість з них повертають значення, що мають тип VARCHAR2. При використанні в процедурних операторах ці значення можна присвоїти змінним PL/SQL, які мають тип VARCHAR2 або CHAR.

Приклад 2.105. Видати відомості про співробітників у такій формі: прізвище – з великої літери, а інші літери малі; чотири перших літери з назви відділу; значення посади вирівняти праворуч, доповнивши його пробілами зліва (рис. 2.107).

```
SELECT INITCAP(ENAME) Surname, EMP.DEPTNO,
       SUBSTR(DNAME,1,4) DN,
       LPAD(JOB,14) JOB
FROM EMP INNER JOIN DEPT ON EMP.DEPTNO = DEPT.DEPTNO;
```

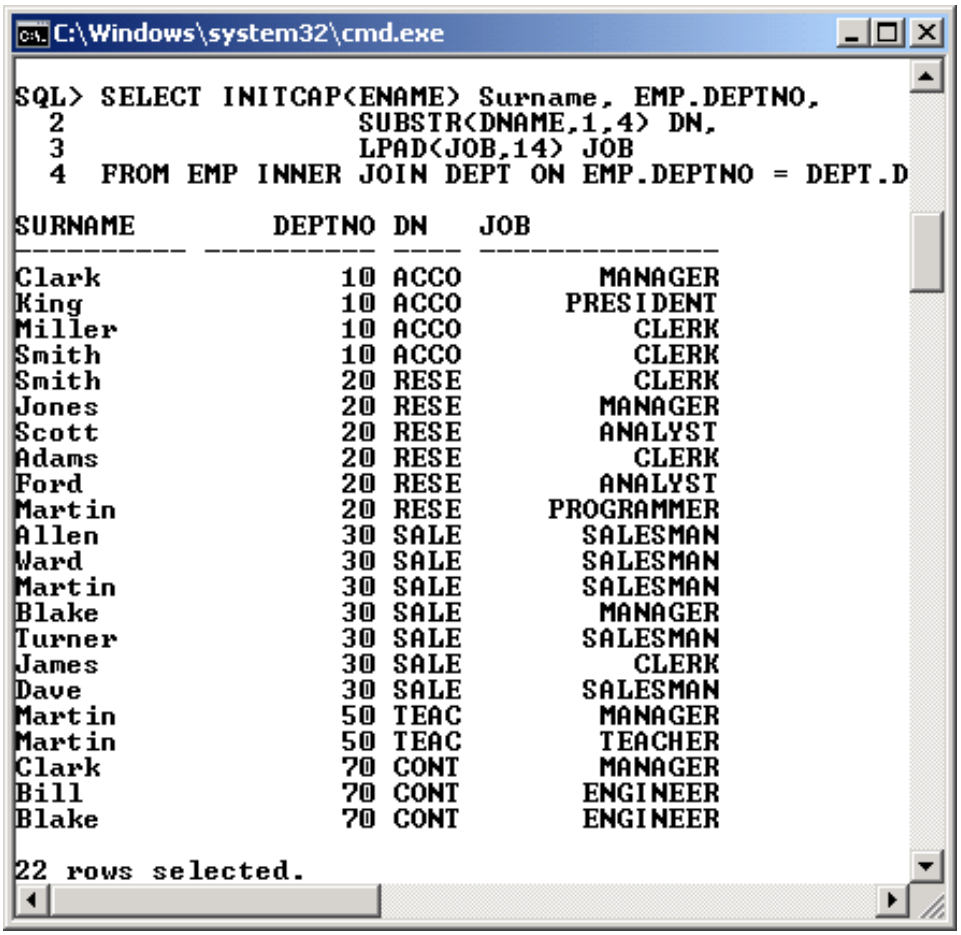


Рис. 2.107. **Використання функцій INITCAP, SUBSTR та LPAD**

Приклад 2.106. Проілюструвати використання функції SOUNDEX для декількох близьких за звучанням слів (рис. 2.108).

```
SELECT SOUNDEX('BAD') BAD,
       SOUNDEX('BAT') BAT,
       SOUNDEX('BED') BED,
       SOUNDEX('RAT') RAT,
       SOUNDEX('CAT') CAT,
       SOUNDEX('SAD') SAD,
       SOUNDEX('SEND') SEND  FROM DUAL;
```

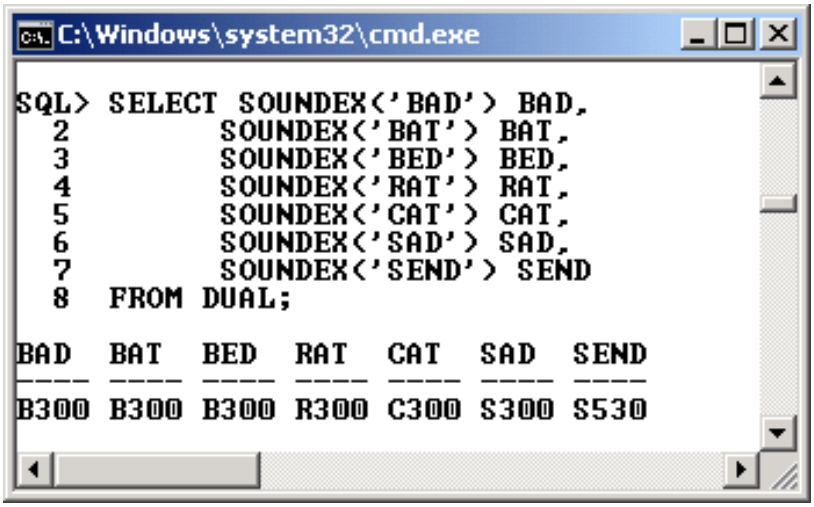


Рис. 2.108. **Результат функції SOUNDEX**

2.17.3. Функції для роботи з датами

Oracle зберігає дані у внутрішньому цифровому форматі: століття, рік, місяць, день, години, хвилини, секунди. За замовчуванням дата видається у форматі "DD-MON-YY". Основні функції цього виду подані у табл. 2.6.

Таблиця 2.6

Основні функції для роботи з датою та часом

Функція	Призначення
ADD_MONTHS (date, n)	Додавання календарних місяців до дати
CURRENT_DATE	Повертає поточну дату в тимчасовій зоні сеансу як значення типу DATE. Ця функція аналогічна SYSDATE за винятком того, що SYSDATE не залежить від тимчасової зони поточного сеансу
EXTRACT({YEAR|MONTH|DAY|HOUR|MINUTE|SECOND} from date)	Виділяє та повертає значення зазначеного поля дати з виразу типу DateTime або Interval
MONTHS_BETWEEN (date1, date2)	Визначає число місяців, що розділяють дві дати. Дробова частина результату є часткою місяця
NEXT_DAY (date, 'char')	Найближча дата, коли настане заданий день. Аргумент 'char' може задавати порядковий номер або назву дня тижня
LAST_DAY (date)	Визначення останнього дня місяця, що містить задану дату
ROUND (date [, 'fmt'])	Округлення до цілого числа діб. Якщо fmt = YEAR, визначає перший день року

Закінчення табл. 2.6

Функція	Призначення
TRUNC (date [, 'fmt'])	Повертає перший день місяця, зазначеного в аргументі date. Якщо fmt = YEAR, повертає дату першого дня року
SYSDATE ()	Повертає поточну дату та час локальної бази даних

Приклад 2.107. Отримати відомості про те, на скільки місяців пізніше були прийняті на роботу співробітники по відношенню до найдавніше прийнятого (рис. 2.109).

```
SELECT EMPNO, ENAME, JOB, DEPTNO,
    HIREDATE, MONTHS_BETWEEN(HIREDATE,
    (SELECT MIN(HIREDATE) FROM EMP)) DMONTH
FROM EMP
ORDER BY 6;
```

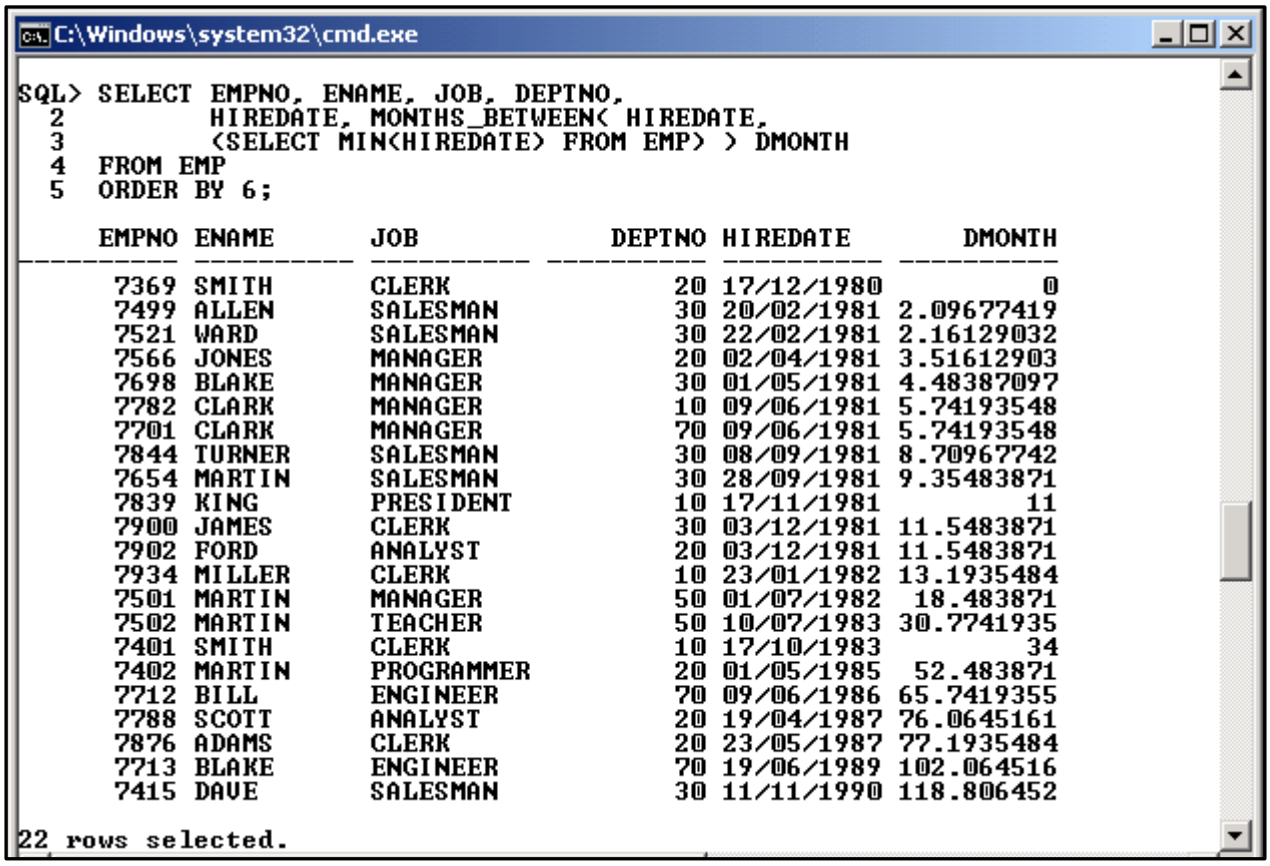


Рис. 2.109. **Визначення різниці у місяцях**

Зверніть увагу на те, що у наведеному прикладі вкладений запит використовується як аргумент функції.

При використанні функції ROUND та TRUNC слід уважно ставитися до використання другого параметра, який задає так звану форматну маску, основні значення якої наведено у табл. 2.7.

Таблиця 2.7

Основні форматні маски для функцій ROUND та TRUNC

Маска	Призначення
CC	Перший день сторіччя
YEAR, або YYYY, або YY, або Y	Перший день року
Q	Перший день кварталу
MONTH, або MON, або MM	Перший день місяця
WW	Той же день тижня, що й перший день поточного року
W	Той же день тижня, що й перший день поточного місяця
DDD або DD	День
DAY, або DY, або D	Перший день тижня
HH, або HH12, або HH24	Година
MI	Хвилина

Приклад 2.108. Проілюструвати використання функцій ROUND і TRUNC для різноманітних форматних масок (рис. 2.110).

```
SELECT TO_CHAR(SYSDATE, 'DD.MM.YY HH24:MI') CUR_DT,
    TRUNC(SYSDATE,'DD') TDD,
    TRUNC(SYSDATE,'MM') TMM,
    TRUNC(SYSDATE,'HH') THH
FROM DUAL;

SELECT TO_CHAR(SYSDATE, 'DD.MM.YY HH24:MI') CUR_DT,
    ROUND(SYSDATE,'DD') RDD,
    ROUND(SYSDATE,'MM') RMM,
    ROUND(SYSDATE,'HH') RHH
FROM DUAL;
```

Арифметичні операції з датою та часом

У СКБД Oracle над датами (d), датами та часом (dt), інтервалами (i) та числами (n) можна виконувати арифметичні операції, наприклад:

різниця двох дат покаже, скільки між ними днів; значення інтервалу, помножене на число n, збільшує його у n разів тощо. Сутність арифметичних операцій з датами наведена у табл. 2.8.

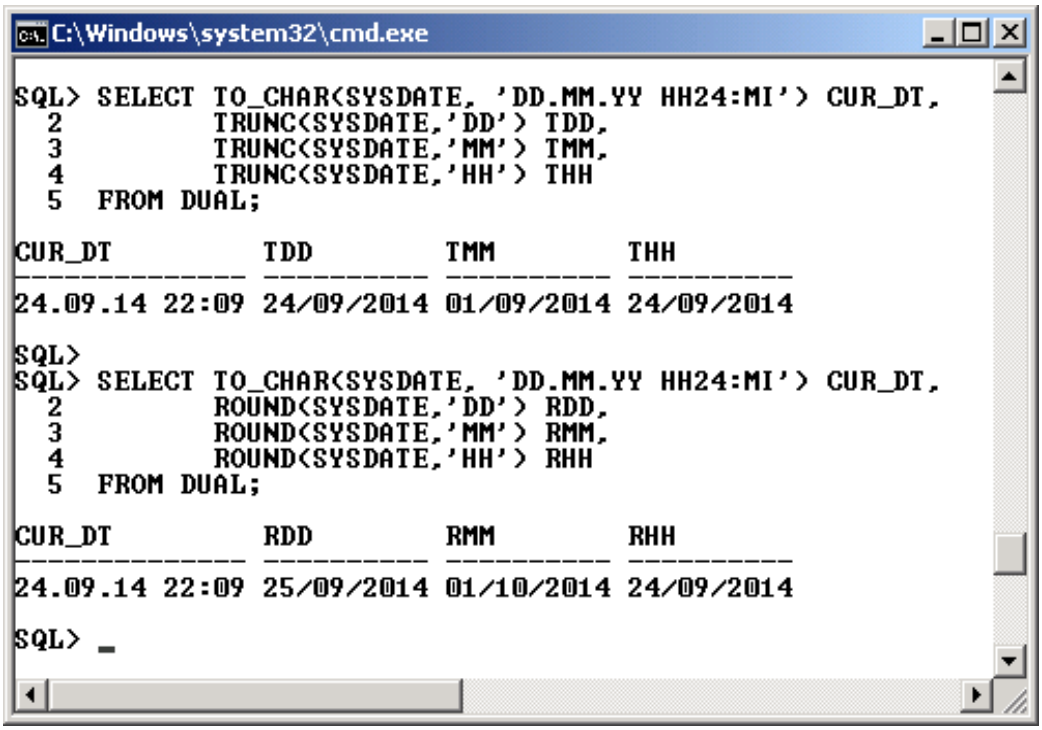


Рис. 2.110. **Використання функцій TRUNC та ROUND**

Таблиця 2.8

Арифметичні операції з датою та часом

Операція	Тип значення, що повертається	Результат
1	2	3
d1-d2	NUMBER	Повертається різниця у днях між d1 і d2. Це значення є дійсним числом, де дробова частина означає неповний день
dt1 – dt2	INTERVAL	Повертає інтервал між dt1 та dt2
i1-i2	INTERVAL	Повертає різницю між i1 і i2
d1 + d2	Не визначено	Заборонена операція. Допустимо тільки віднімання однієї дати від іншої
dt1 + dt2	Не визначено	Заборонена операція. Допустимо тільки віднімання однієї дати_часу від іншої
i1 + i2	INTERVAL	Повертає суму i1 та i2
d1 + n	DATE	До d1 додається n днів, і повертається результат, що має тип DATE. Операнд **n** може бути дійсним числом, що містить неповний день

Закінчення табл. 2.8

1	2	3
d1-n	DATE	3 d1 віднімається **n** днів, і повертається результат, що має тип DATE. Операнд **n** може бути дійсним числом, що містить неповний день
dt1 + i1	DATETIME	Повертає суму dt1 і i1
dt1-i1	DATETIME	Повертає різницю dt1 і i1
i1 * n	INTERVAL	Повертає значення i1, помножене на **n**
i1 / n	INTERVAL	Повертає значення i1, поділене на **n**

Приклад 2.109. Визначити кількість днів, що кожний співробітник вже працює в організації. Результат відсортувати за спаданням кількості днів роботи (рис. 2.111).

```
SELECT EMPNO, ENAME, JOB, DEPTNO,
    FLOOR(SYSDATE-HIREDATE) AS WORK_DAYS
FROM EMP
ORDER BY 5 DESC;
```

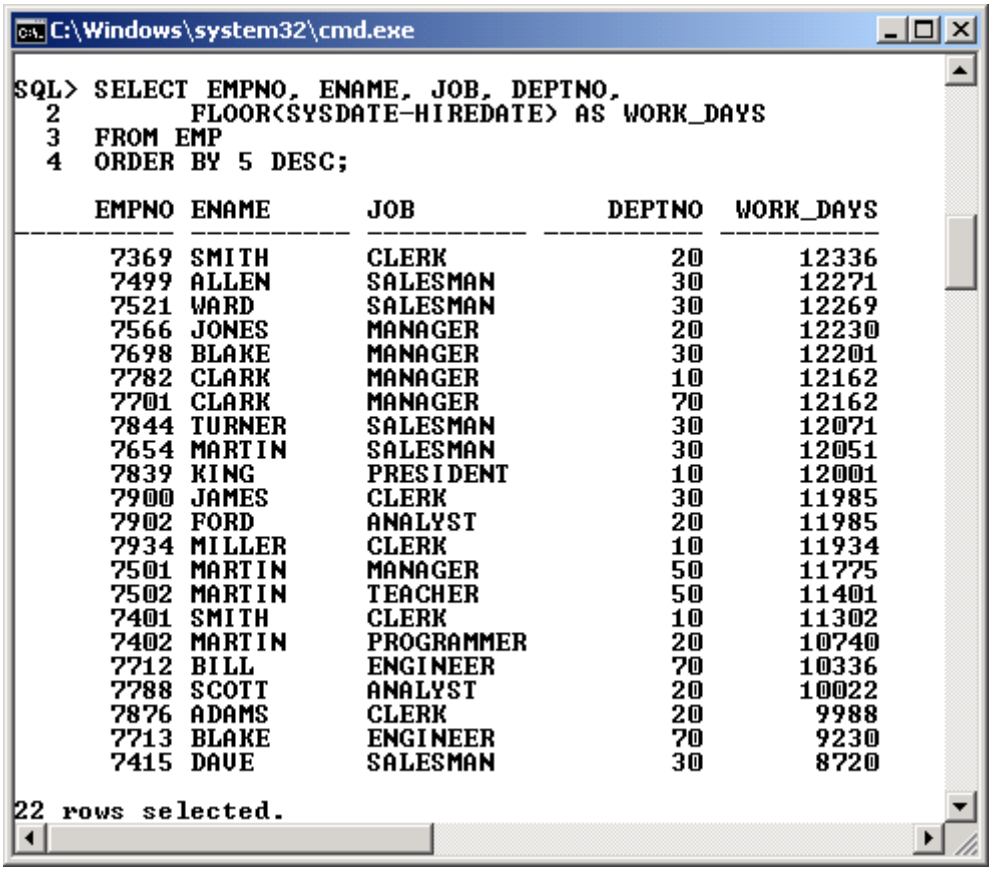


Рис. 2.111. **Використання арифметичних операцій з датою**

2.17.4. Функції перетворення

Функції перетворення використовуються для перетворення типів даних. Більшу частину перетворень Oracle виконує автоматично, неявно викликаючи певну функцію. Однак користувач не може керувати форматом даних під час неявного виклику функції, що іноді робить результати запиту або текст програми незрозумілим. Тому вважається доречним явно вказувати функції перетворення, а не покладатися на неявне перетворення типів даних, яке виконує сама система. Найчастіше використовуються функції перетворення, що наведені у табл. 2.9. З розширеним їх переліком можна ознайомитись у [25].

Таблиця 2.9

Функції перетворення типу

Функція	Значення, що повертається
TO_CHAR (date [, 'fmt'])	Перетворення дати в рядок символів відповідно до форматної моделі fmt
TO_CHAR (number [, 'fmt'])	Перетворення числа в рядок символів відповідно до форматної моделі fmt
TO_NUMBER (char)	Перетворення рядка символів у числовий формат
TO_DATE (char [, 'fmt'])	Перетворення рядка символів у формат дати відповідно до форматної моделі fmt

Як видно з таблиці, результат перетворення значень з одного типу на інший, в основному визначається форматною моделлю (форматними масками). Форматні маски повинні бути укладені в апострофи, і в них різняться символи верхнього та нижнього регістрів. Основні форматні маски та їх опис для значень дати – часу та чисел наведені відповідно, в табл. 2.10 та 2.11. Більш докладно з ними можна ознайомитися у [25].

Таблиця 2.10

Моделі формату для дати

Маска	Призначення
Пунктуація "текст"	Усі символи пунктуації відтворюються у результівному рядку. Текст, що міститься в подвійних лапках, відтворюється без змін
AM або PM	Виведення ознаки "до півдня" – AM та "після півдня" – PM. Вказівка AM і PM у форматній масці рівнозначні. Дана маска часто вживається спільно з HH / HH12. TO_CHAR (SYSDATE, 'HH AM')

Закінчення табл. 2.10

Маска	Призначення
CC	Сторіччя
D	День тижня
DAY	День тижня прописом, при необхідності доповнюється до дев'яти символів пробілами
DD	День місяця цифрами
DDD	День року
DY	День тижня прописом, скорочений до трьох символів
HH12 або HH	Година дня за 12-годинною шкалою (1-12)
HH24	Години цифрами у 24-х годинному форматі
J	Дата юліанського календаря. Є числом днів від 1.01.4712 до н. е.
MI	Хвилини цифрами (0-59)
MM	Двозначне цифрове позначення місяця
MON	Назва місяця прописом у скороченому до трьох символів вигляді
MONTH	Назва місяця прописом
Q	Номер кварталу
RM	Номер місяця, записаний з імітацією римських цифр за допомогою символів X та I
SCC	Сторіччя, причому перед датами до н. е. ставиться знак "мінус"
SP	Форматний суфікс. Його додавання до елементу форматної маски, повертає число, призводить до форматування цього числа прописом. TO_CHAR (SYSDATE, 'DDSP') d2
SS	Секунди цифрами. (0-59)
SSSSS	Кількість секунд після півночі. Дана форматна маска корисна для виміру часових інтервалів у секундах
SYEAR і YEAR	Рік, записаний прописом з урахуванням поточної національної мови
SYYYY	Аналогічно YYYY, але перед датами до н. е. ставиться знак "мінус"
W	Тиждень місяця
WW	Тиждень року
YEAR	Рік прописом
YYYY	Повний рік цифрами
YYY, YY, Y	Аналогічні YYYY, але повертаються, відповідно, останні 3, 2 або 1 цифра року

Таблиця 2.11

Числові моделі формату

Формат	Призначення
$	Плаваючий знак долара. Повертає значення з провідним знаком долара незалежно від символу валюти ($9999)
, (Кома)	Повертає кому в зазначеній позиції незалежно від групового роздільника
. (крапка)	Виведення десяткової точки (9999.99)
0	Виведення цифри, якщо ведучий нуль, виведення нуля
9	Виведення цифри з відкиданням провідних нулів
D	Повертає десятковий роздільник у зазначеній позиції. Кількість дев'яток на кожному боці визначає максимальну кількість цифр (99D9)
EEEE	Повертає значення в експоненційній формі (9.99EEEE)
G	Повертає груповий роздільник у зазначеній позиції. G може бути присутнім більше одного разу на рядку формату
L	Плаваючий символ місцевої валюти у заданій позиції 999L
S	Повертає провідний знак '+' для позитивних чисел і '–' – для негативних (S99999)
V	Повертає значення, помножене на 10^n, де n дорівнює числу дев'яток після V. У разі необхідності це значення округляється (99V9)

Приклад 2.110. Вивести дату зарахування на роботу співробітників, використовуючи різноманітні форматні маски (рис. 2.112).

```
SELECT EMPNO, ENAME, HIREDATE,
       TO_CHAR(HIREDATE,'Day') DAY,
       TO_CHAR(HIREDATE,'DD') DD,
       TO_CHAR(HIREDATE,'RM') M_ROME,
       TO_CHAR(HIREDATE,'Month') MONTH,
       TO_CHAR(HIREDATE,'YYYY') YEAR
FROM EMP;
```

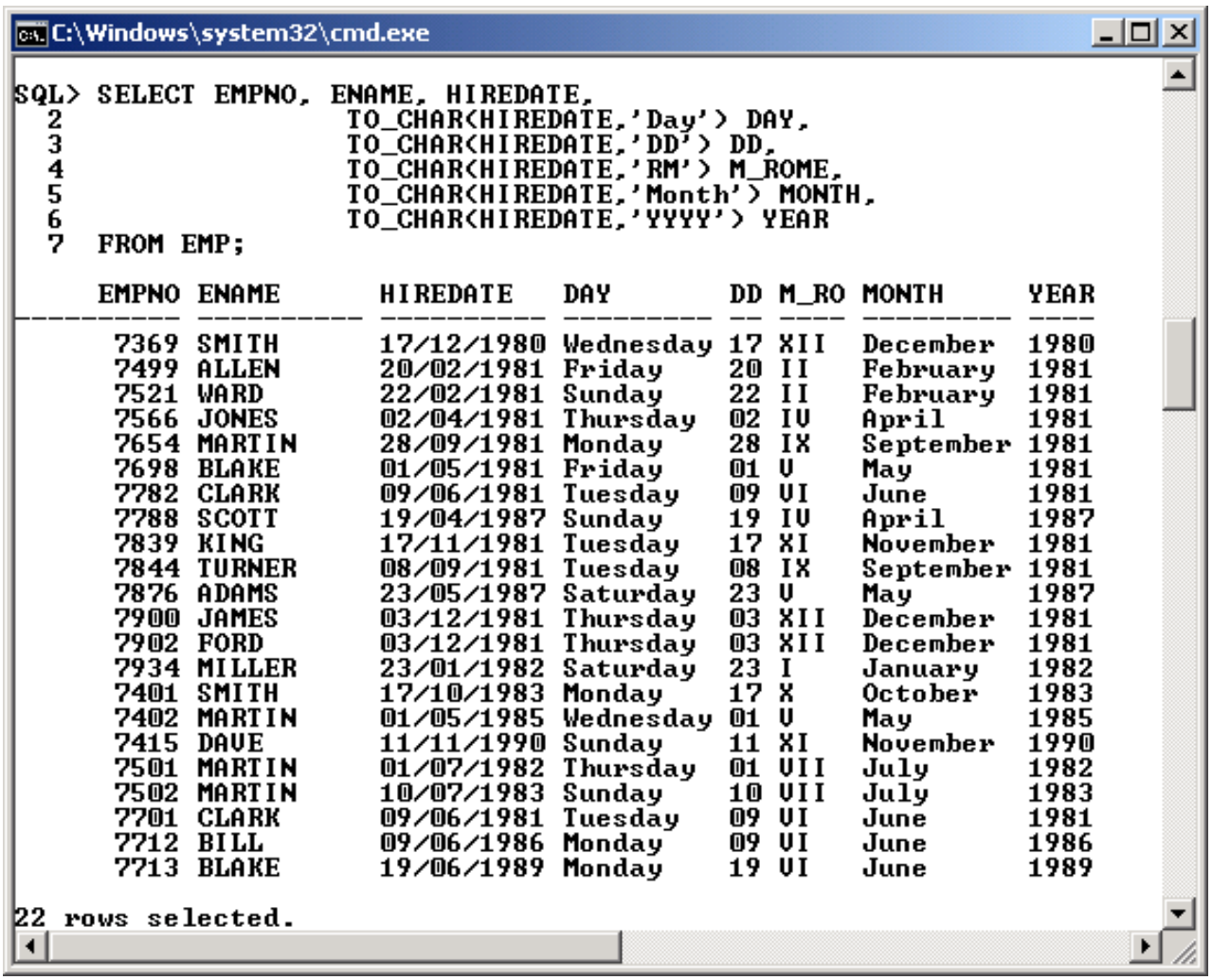


Рис. 2.112. **Використання форматних масок для дати**

Приклад 2.111. Вивести відомості про заробітну плату співробітників, використовуючи різноманітні форматні маски (рис. 2.113).

```
SELECT EMPNO, ENAME,
       SAL,
       TO_CHAR(SAL,'00000') S0,
       TO_CHAR(SAL,'$99999') S$,
       TO_CHAR(SAL,'99999.99L') SLP,
       TO_CHAR(SAL,'S9.99EEEE') SE,
       TO_CHAR(SAL,'99,999') SC
FROM EMP;
```

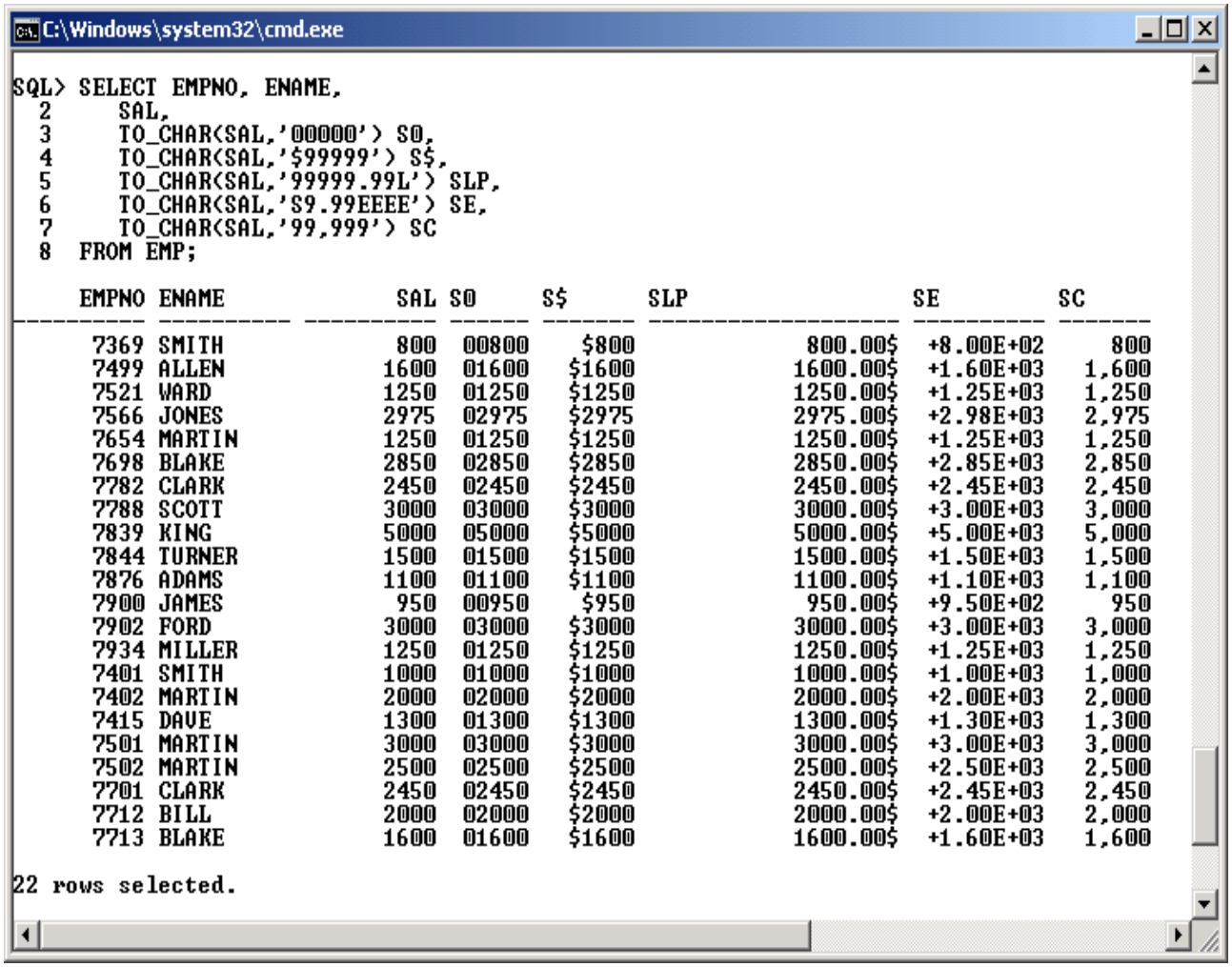


Рис. 2.113. **Використання форматних масок для чисел**

2.17.5. Додаткові функції

Крім наведених категорій функцій, СКБД Oracle має ще додаткові, що не входять до жодної групи. До них відносяться як функції, що можуть часто використовуватися користувачами системи (табл. 2.12), так і функції, що необхідні переважно адміністраторам системи [25].

Таблиця 2.12

Додаткові функції СКБД Oracle

Функція	Застосування
COALESCE (expr1,... [exprn])	Повертає перший вираз, який не є NULL, зі списку аргументів. Якщо усі вирази будуть NULL, COALESCE поверне NULL. Це узагальнення функції NVL

Закінчення табл. 2.12

Функція	Застосування
DECODE (base_expr, compare1, value1, compare2, value2,....default)	Функція DECODE аналогічна послідовності вкладених операторів IF-THEN-ELSE. Базовий вираз base_expr послідовно порівнюється з виразами compare1, compare2 і т.д. Якщо базовий вираз відповідає і-му пункту порівняння, повертається і-е значення (valuei). Якщо базове вираз не відповідає жодному пункту, повертається значення за замовчуванням (default). Вирази порівняння розглядаються по черзі. Якщо знайдена відповідність, то пункти порівняння, що залишилися, не розглядаються. Якщо базовий вираз є NULL-значенням, воно вважається еквівалентним висловом порівняння типу NULL. Кожне значення виразу порівняння перетвориться, якщо це необхідно, у тип даних значення першого виразу порівняння. Цей тип даних є також типом значення, що повертається
GREATEST (expr1 [, expr2]...)	Повертає найбільше значення серед своїх аргументів. Перед порівнянням кожен вираз неявно перетвориться до типу виразу expr1. Якщо expr1 має символьний тип, то порівняння виконується без доповнення пробілами, причому результат має тип VARCHAR2
LEAST (expr1 [, expr2] ".)	Повертає найменше значення зі списку виразів. Функція LEAST схожа на функцію GREATEST тим, що всі вирази неявно перетворюються до типу даних першого виразу. Операції порівняння символів виконуються методом порівняння без доповнення пробілами
NULLIF (a, b)	Повертає NULL, якщо **a** дорівнює **b**, та **a**, якщо аргументи не рівні
NVL {expr1, expr2)	Повертає expr2, якщо expr1 має NULL-значення; в іншому випадку повертає expr1. Якщо expr1 не є рядком символів, то значення, що повертається, має той же тип даних, що й expr1. Інакше значення, що повертається, має тип даних VARCHAR2
NVL2 (expr1, expr2, expr3)	Якщо expr1 містить NULL, повертається expr2, інакше повертається expr3. Значення, що повертається, має тип expr2, якщо expr2 не є символьним; в іншому випадку повертається VARCHAR2
USER	Повертає значення типу VARCHAR2, що містить ім'я поточного користувача Oracle. Функція USER не має аргументів
VSIZE (x)	Повертає число байтів у внутрішньому поданні **x**

Приклад 2.112. Визначити найбільше та найменше значення базових тригонометричних функцій для кутів від 1 до 5 радіан (рис. 2.114).

```
SELECT N, SIN(N) SIN, COS(N) COS, TAN(N) TG,
       GREATEST (SIN(N), COS(N), TAN(N)) GREATEST,
       LEAST (SIN(N), COS(N), TAN(N)) LEAST
FROM INT_NUM;
```

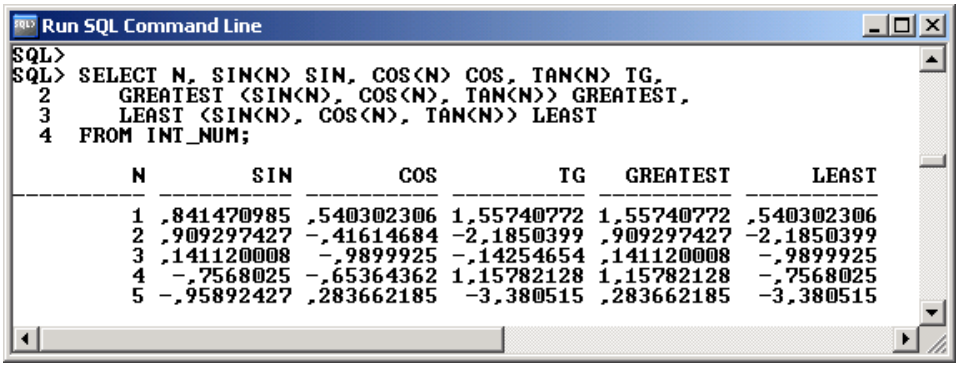


Рис. 2.114. **Використання функцій GREATEST та LEAST**

Приклад 2.113. Визначити річний дохід співробітників з використанням функції COALESCE (рис. 2.115).

```
SELECT EMPNO, ENAME, SAL, COMM,
       SAL*12+COALESCE(COMM,0) YSAL
FROM EMP;
```

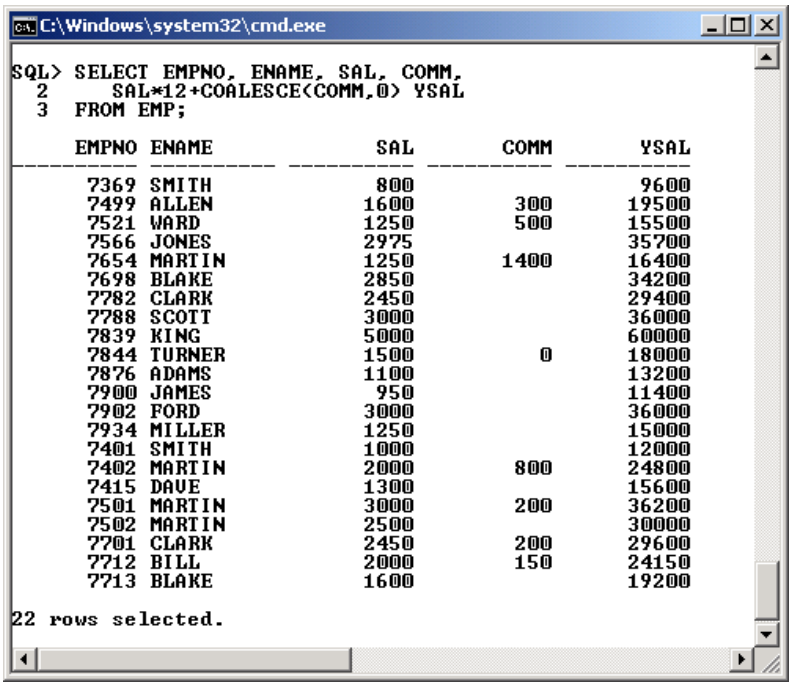


Рис. 2.115. **Використання функції COALESCE**

Приклад 2.114. Використовуючи функцію DECODE, вивести номери відділів словами (рис. 2.116).

```
SELECT DNAME, DEPTNO,
       DECODE(DEPTNO, 10, 'Ten', 20, 'Twenty',
                      30, 'Thirty', 40, 'Forty',
                      50, 'Fifty', 60, 'Sixty',
                      70, 'Seventy') NDEPT
FROM DEPT;
```

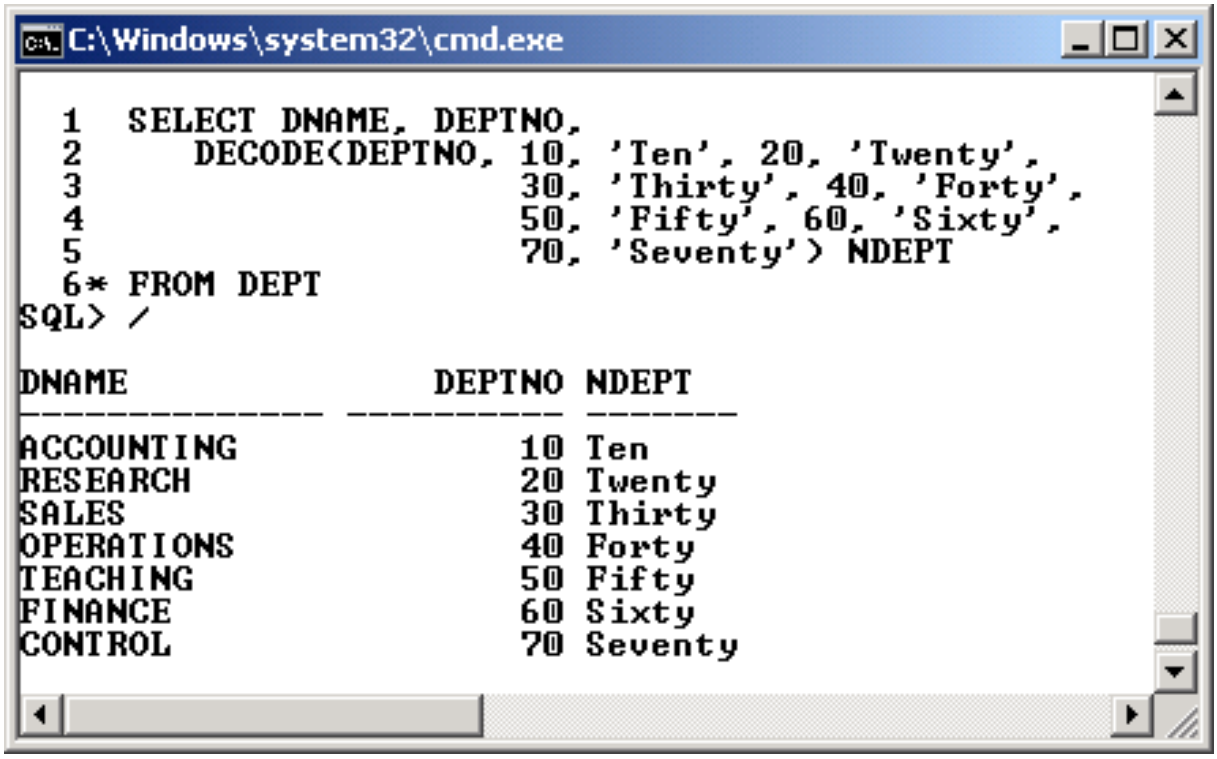


Рис. 2.116. **Використання функції DECODE**

2.17.6. Використання у запитах SELECT виразу CASE

У запитах на вибірку даних мовою SQL СКБД Oracle підтримує так звані CASE-вирази, які можна поділити на прості та пошукові.

Простий вираз CASE дозволяє вибрати для обчислення одне з декількох виразів на основі заданого скалярного значення. Пошуковий вираз CASE послідовно обчислює вирази з заданого списку, поки одне з них не виявиться рівним TRUE, а потім повертає результат пов'язаного з ним виразу. Їх синтаксис такий:

Простий_вираз_Case: =
CASE вираз
WHEN результат_1 THEN результівний_вираз_1
WHEN результат_2 THEN результівний_вираз_2
ELSE результівний_вираз_else
END;

Пошуковий_вираз_Case: =
CASE
WHEN вираз_1 THEN р результівний_вираз_1
WHEN вираз_2 THEN результівний_вираз_2
ELSE
результівний_вираз_else
END;

Приклад 2.115. Використовуючи простий вираз (рис. 2.117) та пошуковий вираз CASE (рис. 2.118), вивести номери відділів словами.

Для простого виразу запит буде таким:

```
SELECT DNAME, DEPTNO,
       CASE DEPTNO
       WHEN 10 THEN 'Ten'
       WHEN 20 THEN 'Twenty'
       WHEN 30 THEN 'Thirty'
       WHEN 40 THEN 'Forty'
       WHEN 50 THEN 'Fifty'
       WHEN 60 THEN 'Sixty'
       WHEN 70 THEN 'Seventy'
       ELSE 'Unknown'
END  AS NDEPT
FROM DEPT;
```

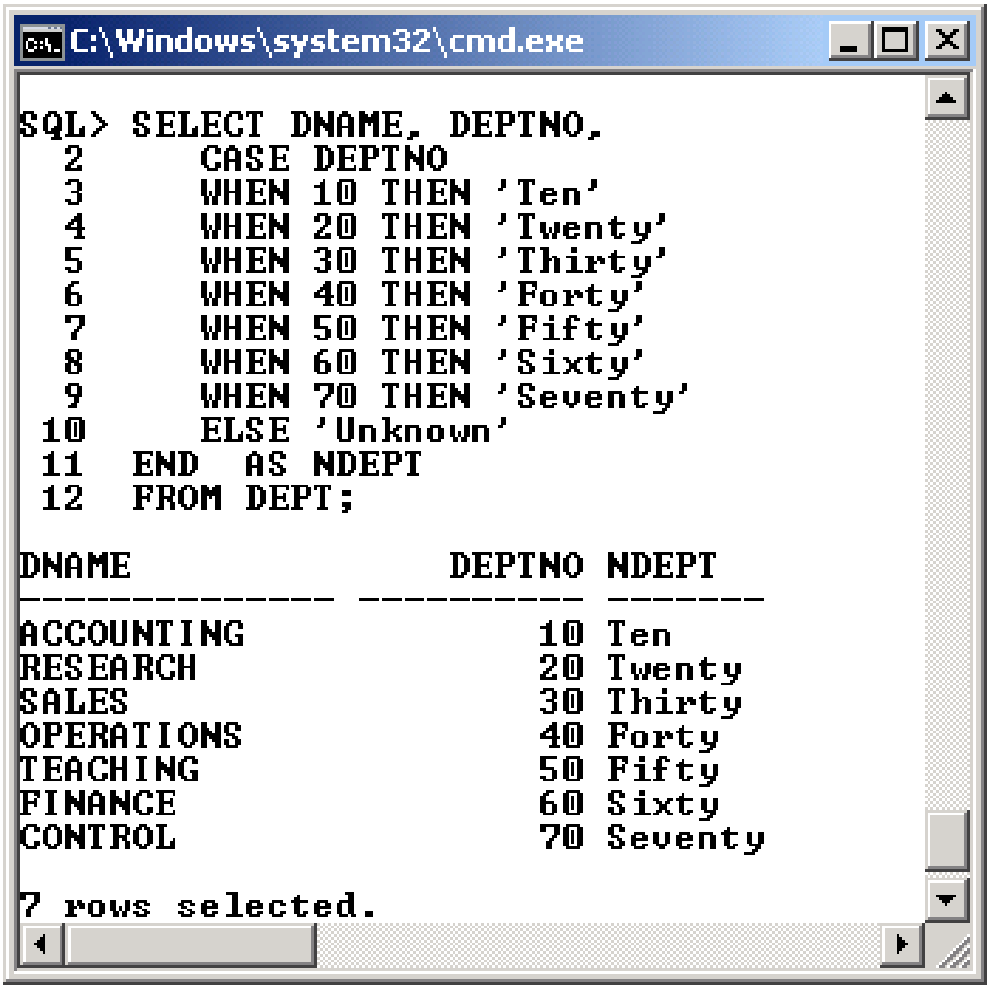


Рис. 2.117. **Використання простого виразу CASE**

У пошуковому виразі умова записується у реченні WHEN

```
SELECT DNAME, DEPTNO,
      CASE
      WHEN DEPTNO=10 THEN 'Ten'
      WHEN DEPTNO=20 THEN 'Twenty'
      WHEN DEPTNO=30 THEN 'Thirty'
      WHEN DEPTNO=40 THEN 'Forty'
      WHEN DEPTNO=50 THEN 'Fifty'
      WHEN DEPTNO=60 THEN 'Sixty'
      WHEN DEPTNO=70 THEN 'Seventy'
      ELSE 'Unknown'
END AS NDEPT
FROM DEPT;
```

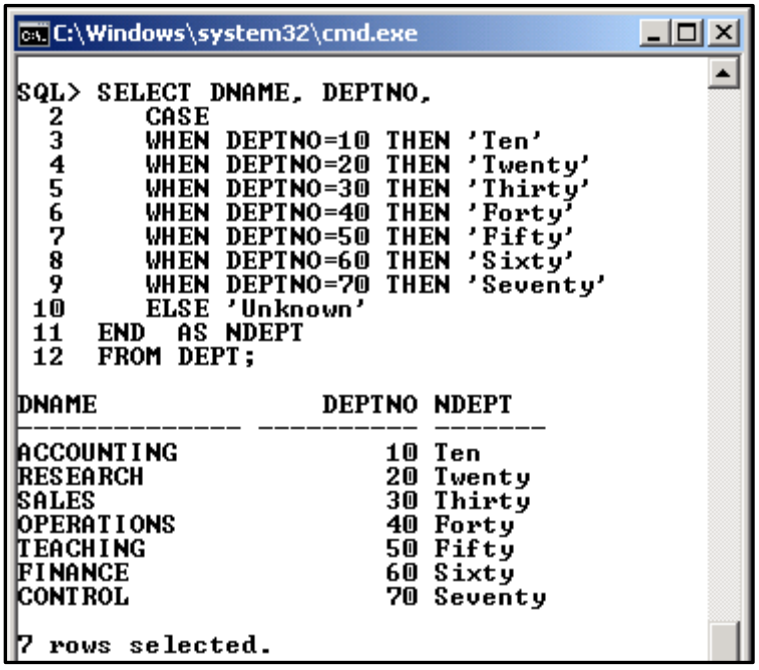


Рис. 2.118. **Використання пошукового виразу CASE**

Запитання і завдання

1. Які групи скалярних функцій СКБД Oracle вам відомі?
2. Запишіть приклад, у якому використовуються числові функції СКБД Oracle?
3. Які функції для роботи з датами є в СКБД Oracle?
4. Які додаткові функції СКБД Oracle вам відомі? Наведіть приклади їх використання.

Лабораторна робота № 2.
Використання мови SQL у СКБД Oracle

Цілі лабораторної роботи:

1. Набуття практичних навичок створення та модифікації об'єктів БД мовою DDL SQL Oracle.
2. Вироблення вмінь та навичок додавання інформації до БД та виконання простих пошукових запитів..
3. Вироблення вмінь та навичок виконання складних запитів до БД з використанням агрегатних функцій, вкладених корельованих та некорельованих запитів.
4. Набуття практичних навичок здійснення операцій модифікації даних, збережених у базі.

Перед виконанням лабораторної роботи студент повинен знати:

1. Способи підключення до БД Oracle та виконання запитів за допомогою програм SQL*PLUS та Oracle SQL Developer.
2. Правила узгодження синтаксису команд SQL*PLUS та SQL.
3. Мовні конструкції SQL для створення та модифікації об'єктів БД.
4. Структуру та призначення основних складових запиту на пошук у БД.
5. Правила формування вкладених запитів мовою SQL.
6. Основні команди, призначені для здійснення операцій модифікації у БД.

Після виконання лабораторної роботи студент повинен уміти:

1. Самостійно підключатися та відключатися від бази даних Oracle засобами SQL*PLUS та Oracle SQL Developer та використовувати команди SQL*PLUS для налаштування режимів роботи утиліти.
2. Виконувати запити на мові SQL на створення об'єктів у БД та модифікацію їх структури.
3. Заносити дані до таблиць за допомогою команд SQL.
4. Виконувати прості та складні пошукові запити до БД.
5. Здійснювати модифікацію даних засобами мови SQL Oracle.
6. Самостійно здійснювати постановку та реалізацію задач у БД.

2.1. Підготовка до виконання лабораторної роботи

Для виконання лабораторної роботи спочатку треба виконати певні підготовчі кроки:

1. Завантажити на свій носій (флешку) zip-архів з лабораторною роботою з сайту дистанційного навчання ХНЕУ та розпакувати архів.
2. Ознайомитися зі вмістом архіву. Архів містить опис поточної лабораторної роботи та скрипт SCOTT_XE2.SQL для створення у БД тестових таблиць.
3. Підключитися до сервера Oracle кафедри інформаційних систем за допомогою SQL*PLUS чи Oracle SQL Developer аналогічно тому, як це було зроблено у лабораторній роботі №1.
4. Для ознайомлення зі вмістом скрипта SCOTT_XE2.SQL відкрити його у Блокноті, та визначити, які таблиці будуть створені у тестовій БД та які дані до них будуть занесені.

5. Для створення тестової БД ввести команду:

START D:\path\scott_xe2.sql,

де path – це шлях до скрипта scott_xe2.sql на вашому флеш-носії.

6. Ввести та виконати команду **SET AUTOCOMMIT ON** для встановлення режиму автоматичного збереження змін у базі даних.

7. Ввести команду **SPOOL D:\filename**, (де filename – ім'я файлу з розширенням LST) для зберігання у файлі протоколу роботи з базою даних (при роботі з Oracle SQL Developer цю команду можна не вводити, а зберегти протокол роботи засобами програми.

8. Ввести команду **SET LINESIZE 150,** що збільшує ширину рядка для виведення результатів запитів.

9. За допомогою команди **DESC** визначити структуру всіх таблиць, імена яких були визначені у п.4. Наприклад, для таблиці DEPT:

```
SQL> DESC DEPT
Name                      Null?    Type
------------------------- -------- ----------------
DEPTNO                    NOT NULL NUMBER(2)
DNAME                              VARCHAR2(14)
LOC                       NOT NULL VARCHAR2(13)
```

Оскільки при роботі зі встановленою версією Oracle Express Edition виникають певні проблеми з символами кирилиці, то рекомендується при виконанні всіх наступних лабораторних робіт вводити значення до бази даних тільки символами латиниці.

Виконання лабораторної роботи складається з трьох рівнів складності. При бездоганному виконанні завдань першого рівня студент може отримати максимум 6 балів за 12-бальною системою оцінювання (73 – за 100-бальною). При бездоганному виконанні завдань другого рівня студент може отримати максимум 9 балів за 12-бальною системою оцінювання (89 – за 100-бальною). Третій рівень (при успішних перших двох) виконується за індивідуальними варіантами і студент повинен проявити вміння як у постановці складних задач, так і в їх реалізації. За успішне виконання завдань цього рівня студент може отримати максимальний бал..

Кожний запит починати коментарем із прізвищем студента!

2.2. Мова визначення даних (DDL) – рівень 1

Завдання 1. Створіть таблицю на ім'я PROJECTS_N зі стовпцями, як показано нижче. Визначте стовпець PROJID як первинний ключ і унеможливте ситуацію, щоб дата в полі P_END_DATE виявилась більш ранньою, ніж дата в полі P_START_DATE.

PROJID	**NOT NULL NUMBER(4)**
P_DESC	**VARCHAR2(20)**
P_START_DATE	**DATE**
P_END_DATE	**DATE**
BUDJET_AMOUNT	**NUMBER(7,2)**
MAX_NO_STAFF	**NUMBER(2)**

Завдання 2. Створіть другу таблицю на ім'я ASSIGMENTS_N, як показано нижче. Визначте в ній стовпець PROJID як зовнішній ключ, що посилається на стовпець PROJID таблиці PROJECTS_N. Визначте також стовпець EMPNO як зовнішній ключ, що посилається на стовпець EMPNO таблиці EMP. Стовпці PROJID і EMPNO не повинні мати порожніх значень полів.

PROJID	**NOT NULL NUMBER(4)**
EMPNO	**NOT NULL NUMBER(4)**
A_START_DATE	**DATE**
A_END_DATE	**DATE**
BULL_RATE	**NUMBER(4,2)**
ASSIGN_TYPE	**VARCHAR2(2)**

Завдання 3. Додайте до таблиці PROJECTS_N стовпець типу LONG на ім'я COMMENTS. Додайте також до таблиці ASSIGMENTS_N числовий стовпець на ім'я HOURS.

Завдання 4. Задайте обмеження на таблицю ASSIGMENTS_N, що забезпечує унікальність комбінацій полів PROJID і EMPNO

Завдання 5. Занесіть до таблиці PROJECTS_N наступні дані:

PROJID	**1**	**2**
P_DESC	**WRITE C030**	**PROOF READ NOTES**
P_START_DATE	**02-JAN-88**	**01-JAN-89**
P_END_DATE	**07-JAN-88**	**10-JAN-89**
BUDJET_AMOUNT	**600**	**500**
MAX_NO_STAFF	**2**	**1**
COMMENTS	**BE CREATIVE**	**YOUR CHOICE**

Завдання 6. Занесіть до таблиці ASSIGMENTS_N наступні дані:

PROJID	**1**	**1**	**2**
EMPNO	**7369**	**7902**	**7844**
A_START_DATE	**02-JAN-88**	**02-JAN-88**	**02-JAN-88**
A_END_DATE	**02-JAN-88**	**02-JAN-88**	**02-JAN-88**
BULL_RATE	**50.00**	**55.00**	**45.50**
ASSIGN_TYPE	**WR**	**WR**	**PF**
HOURS	**15**	**20**	**30**

2.3. Мова маніпулювання даними (DML) – рівень 1

Завдання 7. Скласти команду SQL для вибірки всієї інформації з таблиці DEPT.

Завдання 8. Скласти команду SQL для вибірки даних про співробітників 20-го та 30-го відділів за алфавітом за їх іменами.

Завдання 9. Скласти команду SQL для вибірки імен співробітників, що працюють на посаді менеджерів у 10-му та 20-му відділах, та їх окладів.

Завдання 10. Скласти команду SQL для знаходження всіх співробітників із зарплатою у діапазоні від 1 000 до 2 000.

Завдання 11. Скласти команду SQL для виведення номерів відділів у таблиці DEPT та їх назв за алфавітом.

Завдання 12. Скласти команду SQL для відображення усіх категорій посад з таблиці EMP у впорядкованому вигляді.

Завдання 13. Скласти команду SQL для видачі імен співробітників, їх посад та окладів, що мають менеджера.

Завдання 14. Скласти команду SQL для знаходження всіх імен співробітників, що містять комбінації символів "TH" або "AR".

Завдання 15. Знайдіть імена всіх співробітників та їх відділів. Відсортуйте результати за найменуванням відділів.

Завдання 16. Виберіть імена всіх співробітників, номери й найменування їх відділів.

Завдання 17. Видайте інформацію про співробітників третьої категорії оплати.

Завдання 18. Видайте ім'я співробітника, розташування йнайменування його відділу для всіх співробітників, у яких зарплата перевищує 1 500.

Завдання 19. Сформуйте таблицю, що відбиває градацію співробітників за рівнем їх зарплати.

Завдання 20. Виберіть ім'я, посаду, оклад, категорію окладу, найменування відділу для всіх співробітників компанії, крім реалізаторів. Упорядкуйте рядки стосовно зарплати в порядку убування.

Завдання 21. Видайте інформацію про співробітників, які отримали премію у розмірі від 1 000 до 2 000, а також за всіма реалізаторами.

Завдання 22. Знайдіть усіх співробітників, що працюють у Далласі.

Завдання 23. Знайдіть відділи, у яких немає співробітників.

Завдання 24. Знайдіть кількість співробітників у кожному відділі.

Завдання 25. Виберіть номери й імена всіх співробітників, а також номери й імена їх менеджерів. Використайте просте з'єднання.

2.4. Мова визначення даних (DDL) – рівень 2

Завдання 26. Створіть подання, що накладає такі обмеження на таблицю ASSIGMENTS_N:

- Значення поля PROJID повинне бути менше 2 000.
- Дата завершення проекту (A_END_DATE) повинна бути пізніше його початку (A_START_DATE).
- Припустимі типи призначення (ASSIGN_TYPE) – це PF, WT, ED.
- Значення поля BULL_RATE повинне бути менше 50.00 при типі призначення PF, менше 60.00 при типі призначення WT і менше 70.00 при типі призначення ED.
- Службовий номер (EMPNO) повинен відповідати таблиці EMP.

Не забудьте речення WITH CHECK OPTION.

Завдання 27. Занесіть кілька рядків до таблиці ASSIGMENTS_N через створене подання та перевірте результат.

Завдання 28. Створіть подання, що містить наступні відомості: номер відділу; середню зарплату; максимальну зарплату; мінімальну зарплату; сумарну зарплату у відділі; кількість людей, що отримують зарплату; кількість людей, що отримали премію. Виконайте запит до подання для перегляду його змісту. Ім'я подання View_N.

Завдання 29. Скористайтеся створеним поданням для отримання такого рядка: EMPNO, ENAME, JOB, SAL, HIREDATE; мінімальна, максимальна, середня зарплата у відділі співробітника, номер якого повинен вводитися з клавіатури.

Завдання 30. Самостійне виконання

1. Створити таблицю SOTRUD_N, (CREATE TABLE).
2. Переглянути структуру таблиці (DESC SOTRUD_N).
3. Додати до таблиці новий стовпець (ALTER TABLE).
4. Модифікувати стовпець.
5. Занести кілька нових рядків у таблицю (INSERT).
6. Виправити рядок (UPDATE).
7. Видалити рядок (DELETE).
8. Створити таблицю OTDEL_N, що містить табельний номер, ім'я, посаду й оклад співробітників N-го відділу (CREATE TABLE...AS SELECT).
9. Перейменувати таблицю OTDEL_N в OTD_N (RENAME).
10. Видалити цю таблицю (DROP TABLE).
11. Що, на ваш погляд, потрібно зробити, щоб скопіювати рядки з таблиці SOTRUD_N іншого користувача БД?

Завдання 31. Створити у базі даних таблицю EMP1, ідентичну таблиці EMP. Створити послідовність за допомогою команди SQL CREATE SEQUENCE для генерації номерів співробітників для стовпця EMPNO таблиці EMP1.

Дати пояснення вибору основних параметрів команди (INCREMENT BY, START WITH, NOMAXVALUE, NOCYCLE, CACHE);

Зробіти зміни у послідовності командою SQL: ALTER SEQUENCE з параметрами:

```
ALTER SEQUENCE emp_sequence
        INCREMENT BY 10
        MAXVALUE 10000    CYCLE    CACHE 20;
```

Як здійснюється звертання до послідовності в реченнях SQL? Як видалити послідовність?

За допомогою послідовності занести три рядка у таблицю EMP1. Перевірити результат.

Завдання 32. Створити подання, що з'єднує дані з таблиць EMP і DEPT з умовою вибірки WHERE EMP.DEPTNO IN (10, 30).

Завдання 33. Перевірити або створити обмеження цілісності, гарантуючи, щоб кожний співробітник у таблиці EMP значився в одному з відділів, зазначених у таблиці DEPT.

Завдання 34. Підключитися до бази з логіном **SCOTT** та паролем **TIGER**. Ввести команду **grant SELECT on emp to public.** Повернутися до своєї бази (логін studGGNN).

Створити синонім за допомогою команди SQL: CREATE SYNONYM з ім'ям PUBLIC_EMP таблиці EMP, що знаходиться у схемі користувача SCOTT.

Вибрати дані з PUBLIC_EMP.

Видалити непотрібний синонім, використовуючи команду SQL: DROP SYNONYM.

Завдання 35. Визначити й дати пояснення обмеженням цілісності у командах CREATE TABLE:

```
CREATE TABLE dept (
   deptno  NUMBER(3) PRIMARY KEY,
   dname   VARCHAR2(15),
   loc     VARCHAR2(15)
           CONSTRAINT dname_ukey UNIQUE (dname, loc),
           CONSTRAINT loc_check1
            CHECK (loc IN ('NEW YORK', 'BOSTON', 'CHICAGO')));
CREATE TABLE emp (
   empno    NUMBER(5) PRIMARY KEY,
   ename    VARCHAR2(15) NOT NULL,
   job      VARCHAR2(10),
   mgr      NUMBER(5) CONSTRAINT mgr_fkey
            REFERENCES emp,
   hiredate DATE,
   sal      NUMBER(7,2),
   comm     NUMBER(5,2),
   deptno   NUMBER(3) NOT NULL
           CONSTRAINT dept_fkey
            REFERENCES dept ON DELETE CASCADE);
```

Зробити аналіз визначення обмеження цілісності за допомогою фрази CONSTRAINT команди ALTER TABLE:

```
ALTER TABLE dept
   ADD PRIMARY KEY (deptno);
```

```
ALTER TABLE emp
  ADD CONSTRAINT dept_fkey FOREIGN KEY (deptno)
REFERENCES dept
  MODIFY (ename VARCHAR2(15) NOT NULL);
```

Випробувати включення й вимикання обмежень цілісності за допомогою опції ENABLE або опції DISABLE;

2.5. Мова маніпулювання даними (DML) – рівень 2

Завдання 36. Скласти команду SQL для вибірки імені, річної зарплати та премій для всіх реалізаторів, у яких місячна зарплата перевершує премію. Відсортувати рядки за значенням окладу в порядку спадання. Якщо декілька співробітників отримують однакову зарплату, то в межах рядків з однаковою зарплатою впорядкуєте їх за іменами співробітників.

Завдання 37. Скласти команду SQL для отримання імен і річного доходу всіх співробітників з урахуванням премії.

Завдання 38. Скласти команду SQL для отримання даних про всіх співробітників, зарахованих на роботу влітку 1981 року.

Завдання 39. Скласти команду SQL для побудови таблиці результатів, як показано нижче:

```
EMPLOYEE JOB              DEPT       FROM
-------------------------------------------------
SMITH    works as CLERK  at 20 dept from 17.12.1998
```

Завдання 40. Скласти команду SQL для занесення інформації про двох нових співробітників відділу №40 з номерами "NN1", "NN2" і посадами "програміст", де "NN" – номер варіанта (тобто номер студента у журналі групи). Зафіксувати транзакцію й переглянути результати.

Завдання 41. Після виконання завдання 38 скласти команду SQL для відновлення рядків у таблиці EMP, замінивши посади "програміст" на "адміністратор" для співробітників з номерами "NN", де "NN" – номер варіанта. Зафіксувати транзакцію й переглянути результати.

Завдання 42. Після виконання завдання 39 скласти команду SQL, яка створює таблицю EMP_COPY, на основі усіх даних таблиці EMP та

таблицю JOBS з двох полів EMPNO та JOB на основі даних таблиці EMP. Зафіксувати транзакції та переглянути результати.

Завдання 43. Після виконання завдання 40 скласти команду SQL для видалення з таблиці "EMP_COPY" усіх рядків, в яких у полях з номером відділа та номером співробітника є ідентифікатор виду "NN", де NN – номер варіанта. Зафіксувати транзакцію й переглянути результати.

Завдання 44. Знайдіть посади, на які призначали співробітників взимку та восени 1981 року.

Завдання 45. Модифікуйте запит 23 так, щоб у таблиці результатів з'явився KING, що не має менеджера.

Завдання 46. Визначте всіх співробітників, зарахованих у компанію раніше своїх менеджерів.

Завдання 47. Знайдіть усі відділи, що не мають співробітників, використовуючи підзапит.

Завдання 48. Знайти відділ, в якому сумарний річний дохід співробітників максимальний.

Завдання 49. Знайдіть співробітників, що отримують більше від всіх на своїй посаді. Упорядкуйте дані за убуванням значення зарплати.

Завдання 50. Знайдіть співробітників, що отримують менше за всіх на своїй посаді. Упорядкуєте дані за зростанням значення зарплати.

Завдання 51. Визначте, хто з співробітників у кожному з відділів був зарахований на роботу останнім за часом. Результати впорядкуйте за датою зарахування.

Завдання 52. Видайте інформацію про співробітників, що отримують більше за середню зарплату у відділі. Впорядкуйте дані за номерами відділів.

Завдання 53. Хто входить до трійки самих високооплачуваних у компанії? Знайдіть їх імена та оклад.

Завдання 54. Підрахуйте кількість співробітників, які були зараховані у компанію за кожним роком.

Завдання 55. Напишіть запит, що друкує зірочку навпроти співробітника, зарахованого на роботу останнім за часом. Вкажіть його ім'я, дату зарахування на роботу, а також стовпець HIRED_LAST, у який буде поміщена ця зірочка.

2.5. Завдання для самостійного виконання – рівень 3

Завдання І3-1. Написати запити мовою DDL SQL для створення таблиць за індивідуальними варіантами.

Завдання І3-2. Проаналізувавши розвиток постановки задачі – внести певні зміни до структури таблиць, а за необхідністю – створити додаткові таблиці.

Завдання І3-3. Написати запити для додавання рядків до таблиці.

Завдання І3-4. Сформулювати п'ять простих запитів до бази даних, та реалізувати їх на SQL.

Завдання І3-5. Сформулювати не менше п'яти складних запитів для бази даних та реалізувати їх на SQL (вкладені запити; використання EXISTS; накладення умов на групові функції; з'єднання таблиць; реалізація операцій реляційної алгебри; формування таблиць, для аналізу даних тощо).

Завдання І3-6. Сформулювати задачі аналізу, які можна провести за даними індивідуальної бази, та реалізувати їх за допомогою функцій формування сховищ даних та аналітичних функцій.

Висновки

1. Стандартом мови для усіх реляційних систем керування базами даних є мова SQL (Structured Query Language – мова структурованих запитів). Вона є універсальною комп'ютерною мовою для створення, модифікації та керування даними у реляційних базах даних.

2. Мова SQL в СКБД Oracle має такі особливості, як "рідні" типи даних, стандартні функції системи, використання підзапитів для оновлення у команді UPDATE чи скорочена форма команди DELETE, особливості використання зовнішніх з'єднань, деревоподібні запити тощо.

3. В Oracle таблиці, за замовчуванням, присвоюються тому користувачеві, який їх створив. Кожна з таблиць повинна мати ім'я, відмінне від імен інших таблиць користувача, тобто у користувача не може бути двох таблиць з однаковими іменами. Однак різні користувачі можуть створювати таблиці, що мають одне і теж ім'я. Усі стовпці в межах таблиці повинні мати унікальні імена.

4. Найбільш часто використовують такі типи даних: NUMBER, CHAR, VARCHAR2 (або VARCHAR) та DATE. У стовпцях DATE розмір не задається, у стовпцях VARCHAR2 задання розміру необхідно.

5. Виходячи з правила посилальної цілісності, база даних не повинна містити неузгоджених значень зовнішнього ключа, тобто значень, яких немає для потенційного ключа у посилальному відношенні.

6. Для підтримки посилальної цілісності у СКБД Oracle передбачені такі дії: RESTRICT або NO ACTION (обмежити) – стратегія у Oracle передбачена за замовчуванням і не вказується у визначенні посилальної цілісності; CASCADE (каскадувати) та SET NULL (встановити в NULL) – стратегії дозволяються лише для операції видалення (DELETE).

7. Індекс є спеціальним об'єктом бази даних, який створюється для підвищення ефективності виконання пошукових операцій. Індекс може створюватися для одного або декількох стовпців таблиці та забезпечує більш швидкий доступ до БД за рахунок прямих посилань на місце зберігання рядків, які містять необхідні дані.

8. Послідовності (Sequences) у Oracle найчастіше використовуються для автоматичної генерації послідовних чисел для значень полів таблиці (сурогатних ключів), оскільки у системі не передбачена така властивість поля, як IDENTITY у Microsoft SQL Server або тип даних Counter у Microsoft Access.

9. Зв'язок бази даних (database link) призначений для опису шляху до віддаленої бази даних, що дозволяє звертатися до її таблиць та ін-ших об'єктів у SQL-командах, що виконуються локально. Зв'язок бази даних – це основний спосіб опису шляху до інших баз даних у розподіленому середовищі, що дозволяє виконувати віддалені транзакції.

10. Коли необхідно внести корективи до структури вже створеної таблиці, використовують команду ALTER TABLE.

11. Список значень команди INSERT повинен відповідати списку стовпців згідно з такими правилами:

кількість елементів в обох списках має бути однаковою;

повинна існувати пряма відповідність між позицією одного і того ж елемента в обох списках, тому перший елемент списку значень має відноситися до першого стовпця в списку стовпців, другий – до другого стовпця і так далі;

типи елементів у списку значень мають бути сумісні з типами відповідних стовпців таблиці;

порожній рядок у Oracle сприймається як значення NULL.

12. Операція вибірки даних займає найважливіше місце серед усіх команд SQL. Вона є, з одного боку, найбільш поширеною, а з іншого – найбільш складною. У реальних умовах операції вибірки даних, тобто пошуку необхідної інформації у базі даних і подання її в зручному для опрацювання вигляді, займають понад 80 % усіх операції, що виконуються з базою даних.

13. Для з'єднання таблиць обов'язковою є вимога, щоб вони мали один або декілька "однакових" атрибутів.

14. Для запису операції з'єднання можна використовувати два варіанти команди SELECT. У першому разі з'єднання за спільним атрибутом або атрибутами описується в реченні WHERE, де найчастіше вказується операція порівняння між "однаковими" стовпцями таблиць, а в другому – у реченні FROM з використанням спеціального ключового слова – операції JOIN (з'єднати).

15. Класична операція з'єднання у мові SQL позначається як INNER JOIN і може мати дві модифікації: за операцією "дорівнює" – еквіз'єднання та операціями, відмінними від "дорівнює", наприклад "більше", "не дорівнює" тощо. Це так зване тета-з'єднання. Часто зустрічаються модифікації класичного з'єднання – зовнішні з'єднання або ліве та праве з'єднання (LEFT JOIN та RIGHT JOIN), повне зовнішнє з'єднання FULL OUTER JOIN До цього ж типу операцій можна віднести операцію декартового добутку.

16. Реалізація еквіз'єднання у Oracle може бути здійснена також додатковими мовними конструкціями з використанням фраз NATURAL JOIN та USING.

17. Для зовнішніх з'єднань СКБД Oracle має особливу мовну конструкцію (оператор з'єднання), яка поміщається в умову з'єднання та позначається символом плюс, що береться в круглі дужки (+). Він знаходиться на тому боці з'єднувальної умови, що відповідає таблиці з відсутніми даними. В оператора є ефект створення одного або більшої кількості порожніх рядків, до яких будуть приєднуватися безпарні рядки другої таблиці. Тобто, якщо з'єднання ліве, знак плюс ставиться справа; якщо з'єднання праве, то знак плюс ставиться зліва.

18. Агрегатні функції часто застосовують для підбиття підсумків за кожною групою рядків, тому їх ще називають підсумковими.

19. У реченні **GROUP BY** вказується ім'я стовпця (або стовпців), за однаковими значеннями в якому (яких) проводиться групування (агрегація) результівних рядків. Речення **GROUP BY** дає можливість обчислити значення агрегатних функцій для кожної групи окремо.

20. Для підвищення продуктивності операцій агрегації Oracle Database забезпечує додаткову функціональність, а саме:

розширення CUBE і ROLLUP для конструкції GROUP BY;

три функції GROUPING;

вираз GROUPING SETS;

операція повороту (Pivoting).

21. Підзапит (вкладений запит) дозволяє будувати складні ієрархії запитів, що багаторазово виконуються у процесі побудови результівного набору або виконання одного з операторів зміни даних.

22. Подання – це запит на вибірку даних, що зберігається як окремий об'єкт бази даних, але інформація в якому не зберігається постійно як у базових таблицях, а формується динамічно після звернення до нього.

23. Для обробки ієрархічних структур (їх також називають деревоподібними) у СКБД Oracle часто використовують псевдостовпець LEVEL для визначення того, на якому рівні ієрархії перебуває вершина дерева (рядок) стосовно кореневої вершини.

24. Під зміною (модифікацією) даних зазвичай розуміють три основні операції при роботі з таблицями: додавання (вставка), власне модифікація (оновлення) та видалення.

25. Скалярні вбудовані функції у Oracle можна розділити на такі групи: числові функції; символьні функції; функції дати; функції перетворення; інші функції одного рядка.

Як і в мовах програмування, такі функції у якості аргументів приймають кілька значень різних типів і повертають одне значення якогось певного типу даних. Такі функції можна використовувати у виразах замість значень, що мають аналогічний тип даних.

Тема 3. Керування доступом у СКБД ORACLE. Словник даних ORACLE

Мета розділу – ознайомитися з засобами забезпечення безпеки даних у СКБД Oracle та використання словника даних Oracle для отримання необхідної інформації про об'єкти бази даних.

Основні питання

Основні підходи до забезпечення безпеки даних – описується різниця між вибірковим та обов'язковим підходом до безпеки даних.

Види привілеїв – роз'яснюється сенс системних та об'єктних привілеїв та відмінність між ними.

Створення користувачів і ролей у базі даних – на прикладах пояснюється використання мовних засобів SQL Oracle для створення ролей та користувачів у базі даних.

Команди призначення і скасування привілеїв – на прикладах пояснюється використання мовних засобів SQL Oracle для надання та скасування як системних, так і об'єктних привілеїв.

Привілеї, необхідні для виконання основних операцій у БД, – наводиться перелік привілеїв, які повинні мати користувачі, щоб створювати, використовувати, змінювати та вилучати основні об'єкти бази даних.

Словник даних Oracle – наводиться опис системних подань Oracle та приклади отримання інформації зі словника за допомогою запитів мовою SQL.

3.1. Керування доступом у СКБД ORACLE

У сучасних СКБД з метою убезпечення даних підтримуються два основних підходи: вибірковий та обов'язковий [6; 47]. В обох підходах одиницею даних ("об'єктом даних"), для яких створюється система безпеки, може бути вся база даних цілком, певний набір відносин, певне значення даних для заданого атрибута всередині кортежу в певному відношенні тощо. Ці підходи відрізняються наступними властивостями:

- у разі вибіркового управління деякий користувач володіє різноманітними правами (привілеями чи повноваженнями) при роботі з різними об'єктами. Більш того, різні користувачі зазвичай володіють і різними

правами доступу до одного і того ж об'єкту. Тому вибіркові схеми характеризуються значною гнучкістю;

- у разі обов'язкового управління, навпаки, кожному об'єкту даних присвоюється певний класифікаційний рівень, а кожен користувач має певний рівень допуску. Отже, при такому підході доступом до певного об'єкту даних мають тільки користувачі з відповідним рівнем допуску. Тому обов'язкові схеми досить жорсткі та статичні.

Вибіркове управління доступом підтримується багатьма СКБД, і воно підтримується у мові SQL. У загальному випадку система безпеки таких СКБД базується на трьох компонентах: користувачі, об'єкти БД та привілеї [31; 32].

Користувачі. СКБД виконує будь-які дії з БД від імені деякого користувача. Кожному користувачеві присвоюється ідентифікатор – коротке ім'я, яке однозначно визначає користувача в СКБД. Для підтвердження того, що користувач може працювати з введеним ідентифікатором використовується пароль. Таким чином, за допомогою ідентифікатора та пароля проводиться ідентифікація й аутентифікація користувача. Більшість комерційних СКБД дозволяє об'єднувати користувачів з однаковими привілеями в групи – це дозволяє спростити процес адміністрування.

Об'єкти БД. За стандартом SQL2, захищеними об'єктами в БД є таблиці, подання, домени та визначені користувачем набори символів. Більшість комерційних СКБД розширює список об'єктів, додаючи до нього збережені процедури та інші об'єкти.

Привілеї. Привілеї показують набір дій, які можна проводити над тим чи іншим об'єктом. Наприклад, користувач має привілей для перегляду таблиці.

Захист в середовищі Oracle забезпечується за допомогою засобів створення, зміни та знищення облікових записів користувача бази даних Oracle та ролей, а також за допомогою привілеїв, які регулюють можливість створювати та змінювати об'єкти бази даних. Тобто система захисту Oracle належить саме до вибіркового підходу. Роль – це сукупність привілеїв, що можуть призначатися користувачам або іншим ролям [31; 32]. Призначивши ролі, наприклад Student, у подальшому певні привілеї можна надати користувачам-студентам, призначивши тільки роль Student, і тим самим вони отримають усі привілеї цієї ролі. Іншими словами,

використання ролей дозволяє спростити роботу адміністратора, бо відпадає необхідність у призначенні цілої низки привілеїв кожному користувачу системи.

Таким чином, загальна послідовність кроків щодо надання певних привілеїв користувачам системи є такою:

1. Створення користувача або ролі за допомогою команди CREATE USER (CREATE ROLE).

2. Надання створеному користувачу або ролі певних привілеїв на роботу з системою та об'єктами бази даних за допомогою команди GRANT.

3. За необхідності скасування тих чи інших привілеїв у користувача чи ролі, використовують команду REVOKE.

3.1.1. Види привілеїв

Для роботи з Oracle в будь-якій якості, будь то розробник бази даних або адміністратор, потрібні ім'я користувача та пароль. Дані аутентифікації, задані при вході, визначають той набір прав і привілеїв, який буде доступний користувачеві бази даних Oracle.

Зазвичай виділяють два типи привілеїв: системні та об'єктні.

Системні привілеї, або привілеї доступу, регламентують можливі дії користувача на рівні всієї бази даних в цілому.

Іншим видом є привілеї на об'єкти бази даних (об'єктні привілеї). З кожним об'єктом бази даних пов'язаний набір привілеїв доступу до нього. Власник об'єкта має повний набір привілеїв, доступний для даного об'єкта при створенні. Призначувані об'єктні привілеї дозволяють виконувати певні операції над зазначеним об'єктом.

3.1.2. Системні привілеї

Системні привілеї дозволяють користувачам виконувати конкретну дію на рівні системи або з конкретним типом об'єктів. Більшість системних привілеїв доступні тільки адміністраторам і розробникам застосувань, оскільки такі привілеї досить потужні.

Зазвичай виділяють п'ять основних груп таких привілеїв [8]:

Користувальницькі привілеї (CONNECT). Відносяться до користувачів, які звертаються до системи, але не створюють в ній об'єктів. Наприклад, CREATE SESSION, ALTER SESSION.

Привілеї розробника (RESOURCE). Даний набір привілеїв дозволяє створювати об'єкти бази даних. Наприклад: CREATE TABLE, CREATE VIEW.

Привілеї адміністратора (DBA). Це виключно потужний набір привілеїв. Зазвичай їх слід надавати тільки DBA та відповідальним розробникам. Дані привілеї застосовуються не тільки до об'єктів самого власника бази даних, але і до об'єктів, якими володіють інші користувачі (включаючи системних). Наприклад: CREATE TABLESPACE, ALTER DATABASE.

Привілеї супроводу бази даних. Забезпечують супровід та підтримки бази даних. Дані привілеї повинні мати тільки DBA і інший обслуговуючий комп'ютерний персонал.

Привілеї контролю. Відносяться до контролю над роботою бази даних Oracle. Зазвичай призначаються DBA та деяким системним адміністраторам

Перелік основних системних привілеїв СКБД Oracle наведено у табл. Д.1 додатка Д.

Щоб призначати системний привілей, необхідно, щоб користувач мав цей привілей, причому він повинен бути призначений йому з опцією WITH ADMIN OPTION або мати системний привілей GRANT ANY PRIVILEGE.

Щоб призначати роль, користувачу необхідно або мати цю роль, причому вона повинна бути призначена йому з опцією WITH ADMIN OPTION, або мати системний привілей GRANT ANY ROLE.

3.1.3. Об'єктні привілеї

Об'єктні привілеї дозволяють користувачам виконувати конкретні дії на конкретному об'єкті, наприклад, видаляти рядки чи здійснювати модифікацію у зазначеній таблиці.

До привілеїв об'єктного рівня зазвичай відносять такі:

SELECT – дозволяє іншому користувачеві Oracle зробити запит до даних таблиці;

INSERT – дозволяє іншому користувачеві Oracle вставляти рядки в таблицю за допомогою команди INSERT;

UPDATE – дозволяє користувачеві Oracle оновлювати рядки в таблиці незалежно від того, були ці рядки створені цим користувачем чи ні;

DELETE – дозволяє видаляти з таблиці будь-які існуючі рядки. Використовуючи подання можна обмежити множину рядків, які можуть бути видалені;

EXECUTE – дає можливість користувачу Oracle, який володіє процедурним кодом бази даних (процедурами, функціями або пакетами), дозволити іншому користувачеві Oracle викликати його процедурні об'єкти;

ALTER – дає можливість користувачу Oracle змінити визначення таблиці або послідовності;

INDEX – дозволяє користувачеві Oracle створювати індекси на таблиці власника;

REFERENCES – дозволяє створювати обмеження зовнішнього ключа, з посиланням на батьківську таблицю. Цей привілей не можна призначати ролі;

Щоб призначити привілей на об'єкт, необхідно мати цей об'єкт у своїй схемі або мати цей привілей, причому він повинен бути призначений з опцією GRANT OPTION. Власник об'єкта має на нього повні права і може надавати права доступу до об'єкта іншим користувачам за своїм вибором.

3.1.4. Створення користувачів та ролей у базі даних

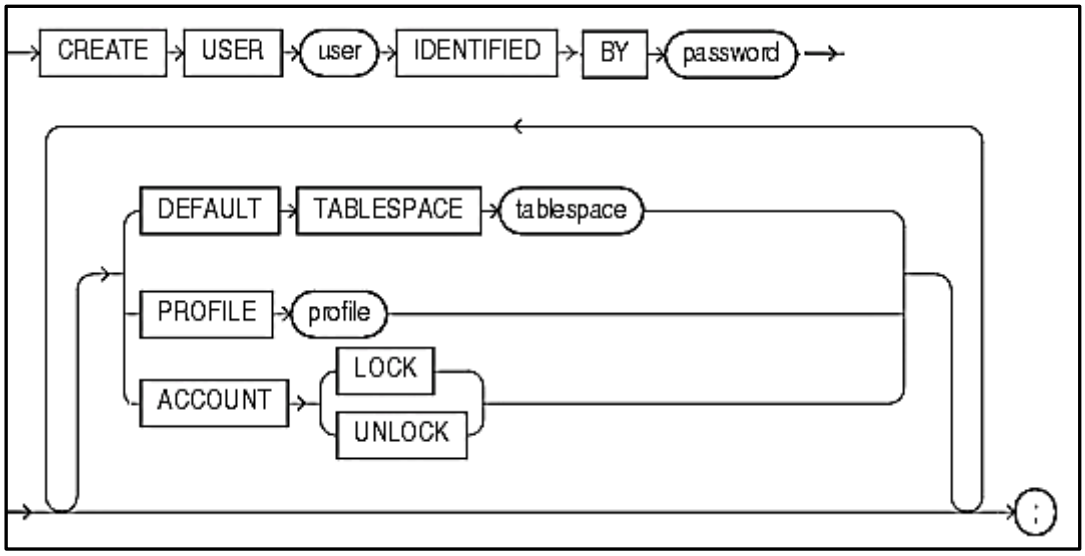
Для створення облікового запису користувача у базі даних Oracle використовується команда CREATE USER [32]. Після створення облікового запису користувача Oracle він не може використовуватися, доки користувач не отримає щонайменше один системний привілей. Системний привілей CREATE SESSION дозволяє користувачеві створювати сеанс по відношенню до бази даних Oracle. Без цього доступ до системи неможливий.

Під час першого створення користувача Oracle можна визначити заданий за замовчуванням табличний простір, в якому будуть створюватися об'єкти користувача. Табличний простір у Oracle – це логічні структури збереження даних, у яких створюються та зберігаються об'єкти бази даних. Усі інші дії виконуються самим сервером Oracle [15; 16].

Якщо заданий за замовчуванням табличний простір не визначений, користувачеві буде призначено табличний простір SYSTEM в якості заданого за замовчуванням. У складі оператора CREATE USER може використовуватися фраза DEFAULT TABLESPACE для визначення того, що об'єкти користувача повинні бути розміщені у табличному просторі,

відмінному від SYSTEM. Користувачеві Oracle також повинна бути призначена квота, яка визначає, скільки пам'яті він може використовувати у табличному просторі.

Формат Спрощений синтаксис команди створення користувача виглядає так:



USER – вказується ім'я користувача, обліковий запис якого повинен бути створений. Це ім'я може містити тільки символи з допустимого набору символів поточної бази даних і зазвичай не повинне перевищувати тридцяти символів.

IDENTIFIED BY password – вказує для створюваного користувача, пароль для входу до бази даних. Паролі чутливі до регістру символів і можуть містити будь-які однобайтові, багатобайтові, спеціальні символи або будь-яку комбінацію з них з допустимого набору символів бази даних.

DEFAULT TABLESPACE – вказує табличний простір за замовчуванням для об'єктів, які створюються у схемі користувача. Якщо опустити цей пункт, то об'єкти користувача зберігаються у табличному просторі за замовчуванням бази даних. Якщо за замовчуванням значення табличного простору не було зазначено для бази даних, то об'єкти користувача зберігаються у системному табличному просторі.

Більш детально табличні простори розглянуті у [45].

PROFILE – вказує профіль, який призначається користувачеві. Профіль у Oracle дозволяє обмежити доступ користувача до ресурсів або вказати умову обробки паролів користувача [9; 10].. Якщо опустити цей пункт, то Oracle Database привласнює профіль для користувача за замовчуванням.

ACCOUNT – фраза ACCOUNT LOCK блокує обліковий запис користувача та забороняє йому доступ. Фраза ACCOUNT UNLOCK розблокує обліковий запис користувача та відкриває йому доступу до системи.

Найбільш повний опис команди CREATE USER наведено у [82].

Можна використовувати команду ALTER USER для зміни таких параметрів користувача, як пароль, задані за замовчуванням тимчасові табличні простори та квоту пам'яті.

Для видалення користувача з бази даних використовується команда DROP USER, яка видаляє запис користувача із словника даних Oracle. Якщо користувач Oracle володіє будь-якими об'єктами бази даних, можна видалити кожен з об'єктів перед використанням команди DROP USER, або використовувати у DROP USER опцію CASCADE для автоматичного знищення всіх об'єктів при видаленні облікового запису користувача.

Аналогічно до команди CREATE USER, командою CREATE ROLE можна створювати ролі у системі з призначенням їй певних привілеїв, а у подальшому призначити цю роль тому чи іншому користувачу [49; 79].

Деякі ролі створюються автоматично самою СКБД Oracle, і система автоматично призначає їм певні системні привілеї. Перелік основних ролей, визначених у Oracle, наведений у табл. 3.1.

Таблиця 3.1

Основні ролі, визначені у ORACLE

Роль	Призначені системні привілеї та ролі
CONNECT	ALTER SESSION CREATE CLUSTER CREATE DATABASE LINK CREATE SEQUENCE CREATE SESSION CREATE SYNONYM CREATE TABLE CREATE VIEW
RESOURCE	CREATE CLUSTER CREATE PROCEDURE CREATE SEQUENCE CREATE TABLE CREATE TRIGGER
DBA	Всі системні привілеї WITH ADMIN OPTION роль EXP_FULL_DATABASE роль IMP_FULL_DATABASE

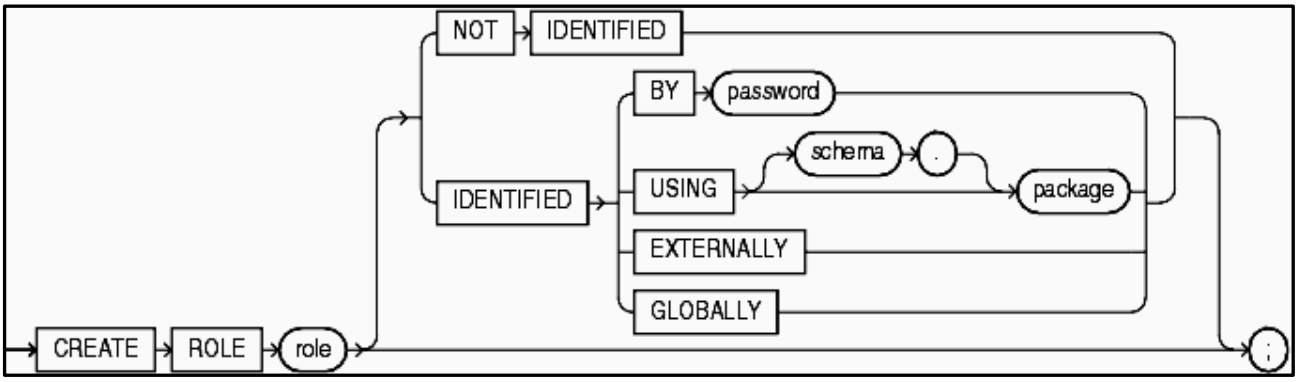
При призначенні користувачу ролі RESOURCE або DBA система неявно призначає йому системний привілей UNLIMITED TABLESPACE.

Ролі CONNECT, RESOURCE та DBA надані для суміщення з попередніми версіями Oracle. Корпорація Oracle рекомендує, щоб проектувальники бази даних використовували свої власні ролі для захисту бази даних, а не покладалися на вже визначені, оскільки вони можуть втратити підтримку в майбутніх версіях Oracle.

Ролі EXP_FULL_DATABASE й IMP_FULL_DATABASE надаються для зручності роботи з утилітами імпорту та експорту бази даних, тобто вони потрібні для здійснення повного експорту або імпорту бази даних.

Для створення ролі необхідно мати системний привілей CREATE ROLE.

Синтаксис команди створення ролі такий:



де **role** – вказує ім'я ролі, яка створюється. Максимальна довжина імені ролі є 30 байт;

NOT IDENTIFIED – вказує на те, що для використання ролі пароль не потрібен;

IDENTIFIED BY password – визначає необхідність вказувати пароль у команді SET ROLE перед тим, як включити привілеї ролі;

USING package – створюється роль додатка (application role), яку можна застосовувати тільки за допомогою програми, що використовує пакет PL/SQL з ім'ям package. Якщо схема не вказана, платформа передбачає, що пакет знаходиться у схемі поточного користувача;

EXTERNALLY – створюється зовнішня роль, аутентифікацію якої виконує операційна система або сторонній постачальник відповідних

послуг. Залежно від операційної системи користувачу, можливо, доведеться вказати пароль для операційної системи перед тим, як роль буде включена;

GLOBALLY – створюється глобальна роль, аутентифікацію якої виробляє корпоративна служба каталогів (enterprise directory service).

Якщо опустити фрази NOT IDENTIFIED та IDENTIFIED, то роль за замовчуванням є NOT IDENTIFIED.

Приклад 3.1. Створити роль **USER_ROLE** з привілеями створення основних об'єктів у базі даних та доступу до словника (рис. 3.1).

```
-- CREATE THE ROLE
CREATE ROLE USER_ROLE;
-- GRANT/REVOKE SYSTEM PRIVILEGES
GRANT RESOURCE TO USER_ROLE;
GRANT CREATE TABLE TO USER_ROLE;
GRANT CREATE VIEW TO USER_ROLE;
GRANT CREATE SEQUENCE TO USER_ROLE;
GRANT CREATE SYNONYM TO USER_ROLE;
GRANT CREATE PROCEDURE TO USER_ROLE;
GRANT CREATE TYPE TO USER_ROLE;
GRANT CREATE SESSION TO USER_ROLE;
GRANT CREATE TRIGGER TO USER_ROLE;
GRANT SELECT ANY DICTIONARY TO USER_ROLE;
```

Приклад 3.2. Створити користувача системи STUD0001 з паролем доступу STUD0001, табличним простором для нього на ім'я USERS та профілем за замовчуванням (рис. 3.2).

```
CREATE USER STUD0001
  IDENTIFIED BY STUD0001
  DEFAULT TABLESPACE USERS
  TEMPORARY TABLESPACE TEMP
  PROFILE DEFAULT;
```

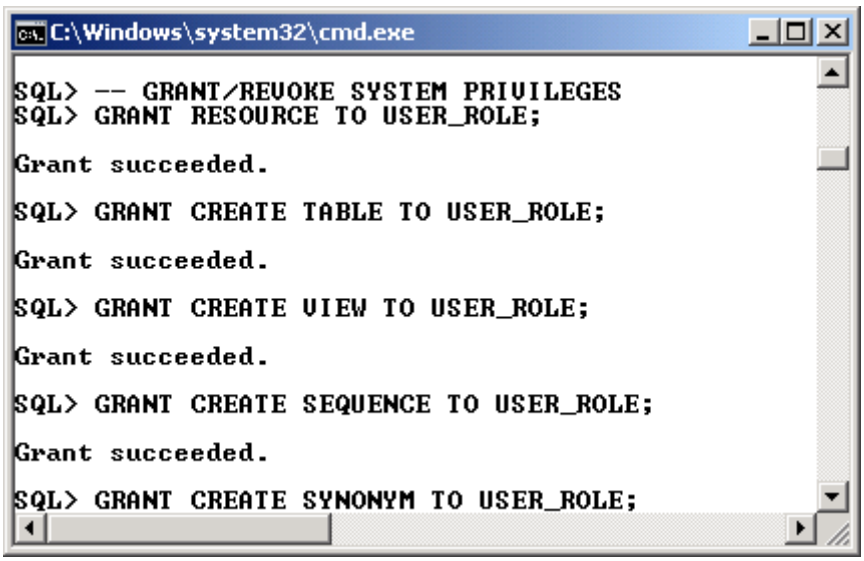


Рис. 3.1. **Створення ролі USER_ROLE**

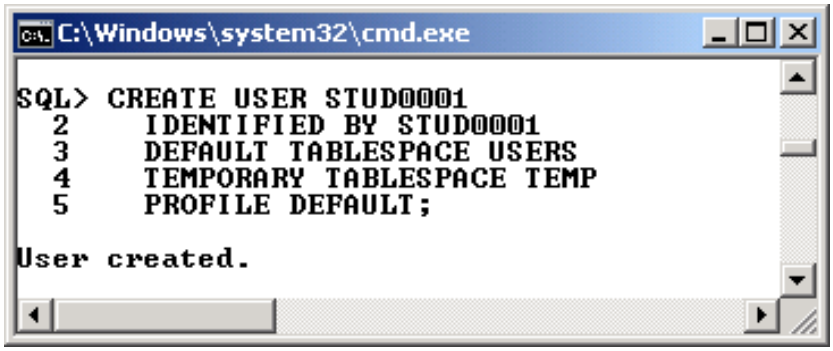
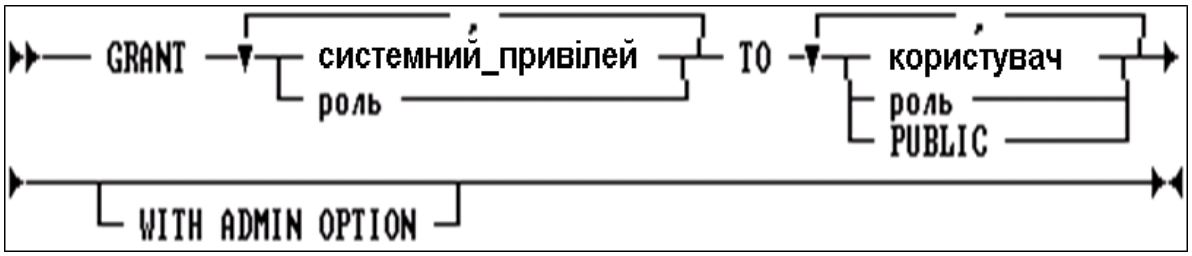


Рис. 3.2. **Створення користувача STUD0001**

3.1.5. Призначення системних привілеїв

Для призначення системних привілеїв та ролей користувачам і ролям використовується команда GRANT.

Формат Синтаксис команди GRANT такий:



де **системний_привілей** – призначається ролі або користувачу;

роль – призначається іншій ролі або користувачу системи;

TO – ідентифікує користувачів або ролі, яким призначаються системні привілеї та ролі;

PUBLIC – призначає системні привілеї або ролі усім користувачам;

WITH ADMIN OPTION – дозволяє користувачеві, що отримав привілей або роль, призначати її іншим користувачам або ролям.

Повторне призначення привілею без опції ADMIN OPTION не перекриває попередню аналогічну команду з опцією ADMIN OPTION. Тобто, якщо необхідно відкликати від користувача опцію ADMIN OPTION за деяким системним привілеєм або роллю, то треба спочатку повністю відкликати цей системний привілей або роль, а потім знову призначити їх, але вже без опції ADMIN OPTION.

Призначення системного привілею CREATE SESSION користувачеві STUD0002 з дозволом підключення до Oracle:

```
GRANT CREATE SESSION TO STUD0002;
```

Призначення системного привілею CREATE TABLE ролі SCHOOL_MASTER:

```
GRANT CREATE TABLE  TO SCHOOL_MASTER;
```

Приклад 3.3. Надати користувачеві STUD0001 право підключатися до системи й усі права, визначені для ролі **USER_ROLE.** Зняти обмеження на розмір табличного простору для користувача STUD0001 (рис. 3.3).

```
GRANT CONNECT TO STUD0001;
GRANT USER_ROLE TO STUD0001;
GRANT UNLIMITED TABLESPACE TO STUD0001;
```

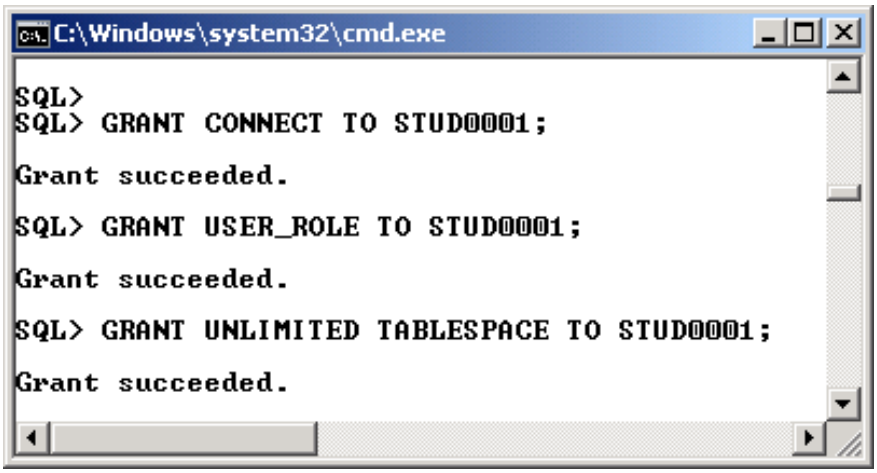
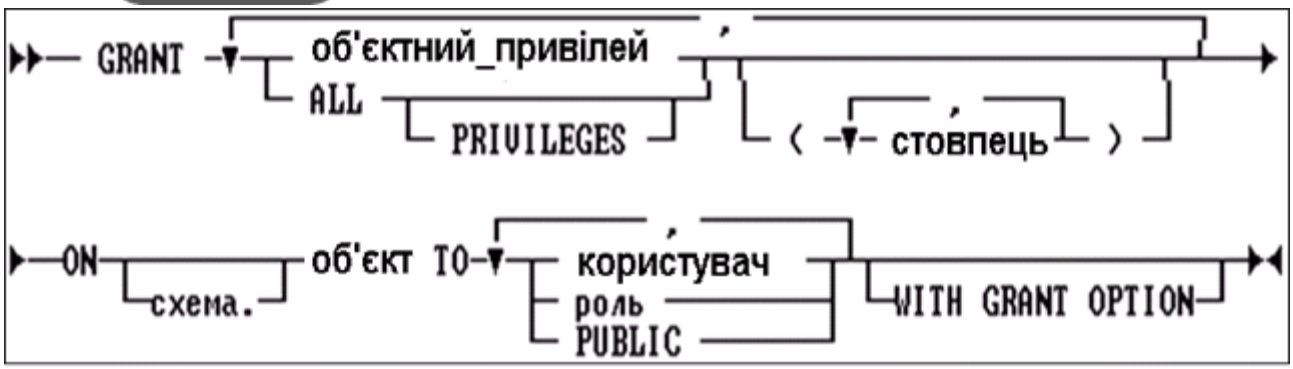


Рис. 3.3. **Надання привілеїв ролі користувачу**

3.1.6. Призначення об'єктних привілеїв

Призначення об'єктних привілеїв (аналогічно системним) здійснюється командою GRANT.

Формат Синтаксис команди такий:



де **об'єктний_привілей** – конкретний об'єктний привілей (ALTER, DELETE, EXECUTE, INDEX, INSERT, REFERENCES, SELECT, UPDATE), що призначується на той чи інший об'єкт;

ALL PRIVILEGES призначає на об'єкт усі привілеї з опцією GRANT OPTION, які були призначені користувачу, що видає команду. Користувач, якому належить схема, що містить об'єкт, автоматично має всі привілеї на об'єкт;

Стовпець – специфікує стовпець таблиці або подання, по якому призначаються привілеї. Стовпці вказуються тільки при призначенні привілеїв INSERT, REFERENCES або UPDATE. Якщо стовпці не вказуються, то привілей призначається за всіма стовпцями таблиці або подання;

ON – ідентифікує об'єкт, за яким призначаються привілеї. Якщо **об'єкт** не уточнюється **схемою**, то Oracle вважає, що об'єкт перебуває у схемі поточного користувача. Допускаються наступні типи об'єктів: таблиця; подання; послідовність; процедура; функція; пакет; знімок; синонім для таблиці, подання, знімка, функції, процедури, пакета або послідовності;

TO – ідентифікує користувачів або ролі, яким призначаються привілеї на об'єкт;

PUBLIC – призначає привілеї на об'єкт усім користувачам;

WITH GRANT OPTION– дозволяє користувачеві, що отримав об'єктний привілей, призначати ці привілеї іншим користувачам або ролям.

Приклад 3.4. Призначити усі привілеї за таблицею VACATION користувачеві STUD05 з опцією GRANT OPTION.

```
GRANT ALL  ON VACATION TO STUD05  WITH GRANT OPTION
```

STUD05 може тепер виконувати будь-які операції з таблицею VACATION, яку створив користувач SCOTT, і одночасно надавати права на роботу з цією таблицею іншим користувачам (рис. 3.4).

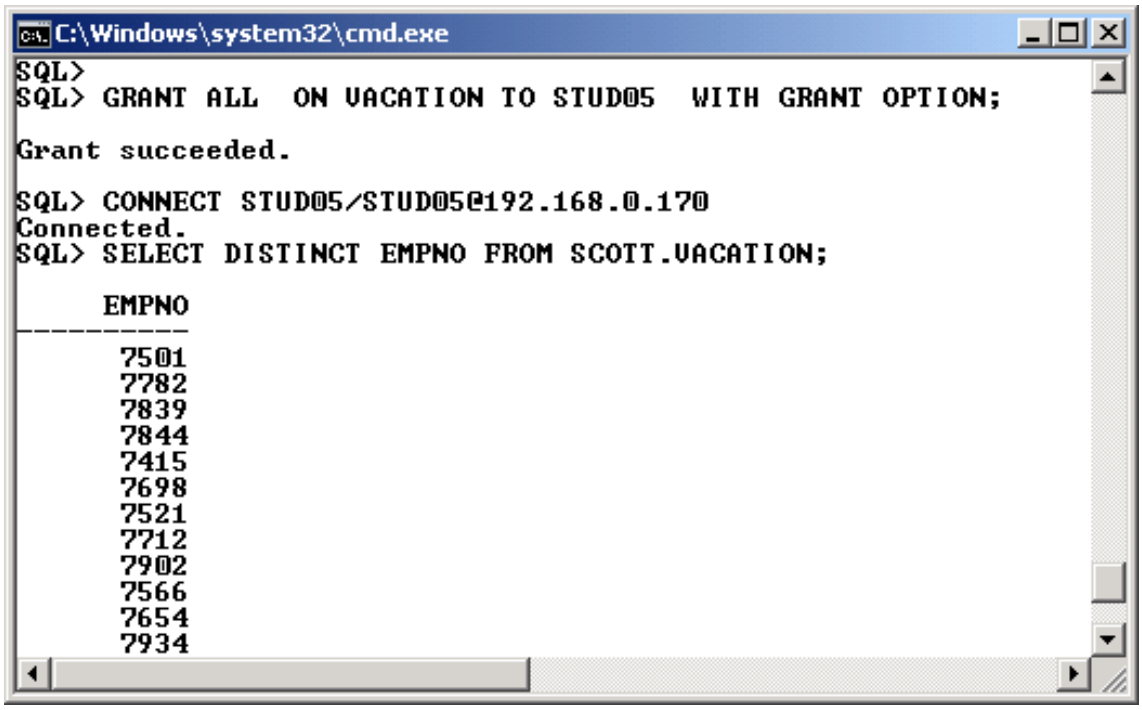


Рис. 3.4. **Надання привілеїв на таблицю**

Приклад 3.5. Призначити привілеї SELECT і UPDATE до подання BOSTON_VAC (приклад 2.90) усім користувачам (рис. 3.5).

```
GRANT SELECT, UPDATE  ON BOSTON_VAC
      TO PUBLIC;
```

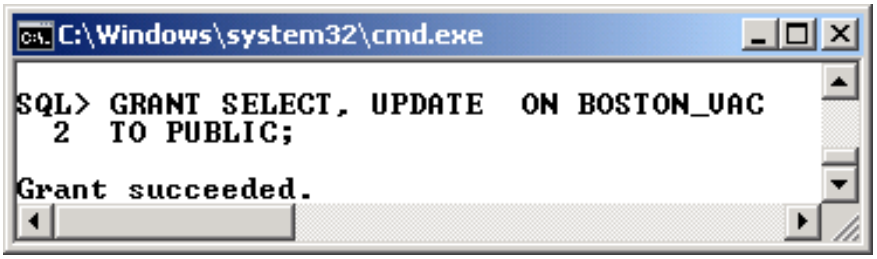


Рис. 3.5. **Надання привілеїв на подання**

Після виконання цієї команди усі користувачі мають право виконувати команди SELECT та UPDATE до подання BOSTON_VAC.

Приклад 3.6. Призначити привілей SELECT за послідовністю NUM_VACATION у схемі SCOTT користувачеві STUD05 (рис. 3.6).

```
GRANT SELECT ON SCOTT.NUM_VACATION
TO STUD05;
```

STUD05 тепер має право генерувати чергове значення цієї послідовності за допомогою наступної команди:

```
SELECT SCOTT.NUM_VACATION.NEXTVAL
FROM DUAL;
```

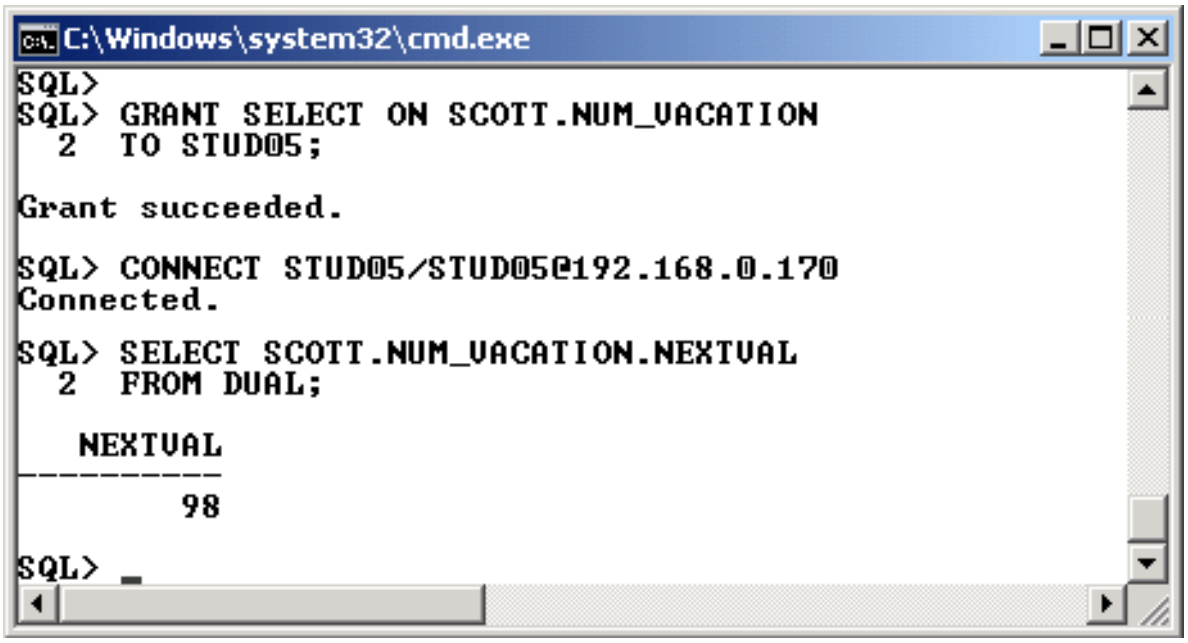


Рис. 3.6. **Надання привілеїв на послідовність**

Приклад 3.7. Призначити користувачеві STUD05 привілей REFERENCES за стовпцем EMPNO та привілей UPDATE за стовпцями EMPNO та SAL таблиці EMP у схемі SCOTT (рис. 3.7).

```
GRANT REFERENCES (EMPNO), UPDATE (EMPNO, SAL)
  ON SCOTT.EMP
  TO STUD05;
```

Користувач STUD05 може тепер оновляти значення стовпців EMPNO та SAL, а також визначати обмеження посилальної цілісності, які посилаються на стовпець EMPNO. Однак, оскільки у команді GRANT перераховані конкретні стовпці, STUD05 не може виконувати будь-які операції над іншими стовпцями таблиці EMP.

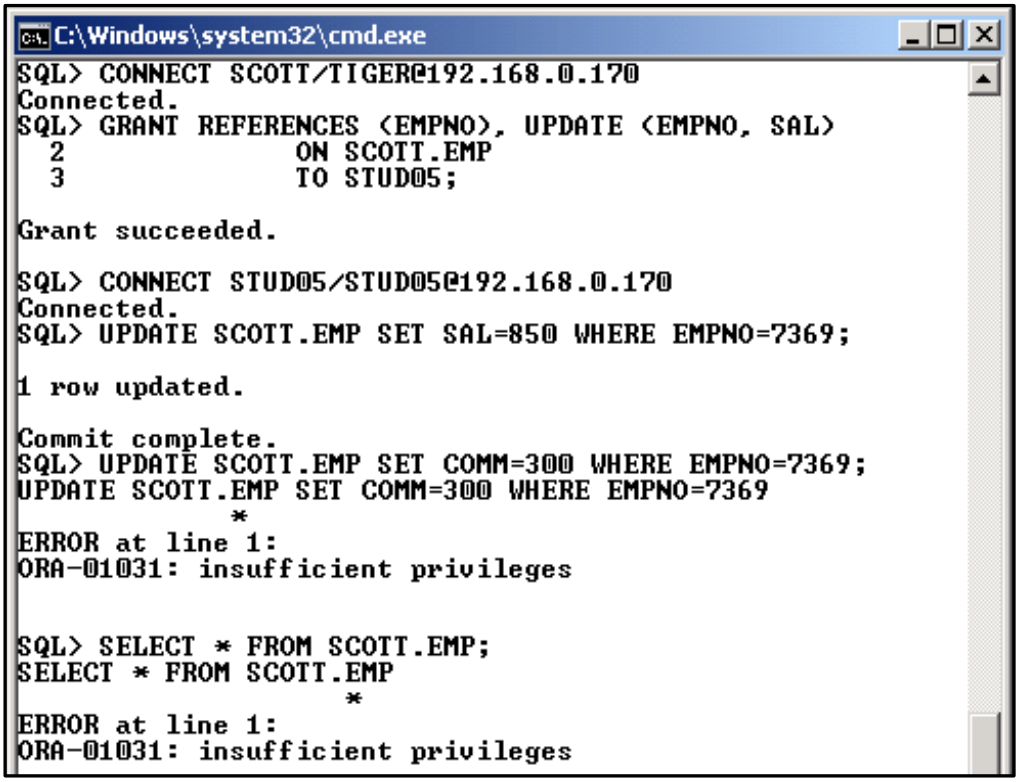


Рис. 3.7. **Надання привілеїв REFERENCES та UPDATE**

Однак, маючи привілей REFERENCES, користувач STUD05 може створити таблицю з відповідним обмеженням (рис. 3.8):

```
CREATE TABLE ORDERS
      (NORDER INT PRIMARY KEY,
      NAME_ORD  VARCHAR2(30),
      SUM   NUMBER(7,2),
      EMPLOYEE  NUMBER(4)
      CONSTRAINT c_emp  REFERENCES SCOTT.EMP(EMPNO));
```

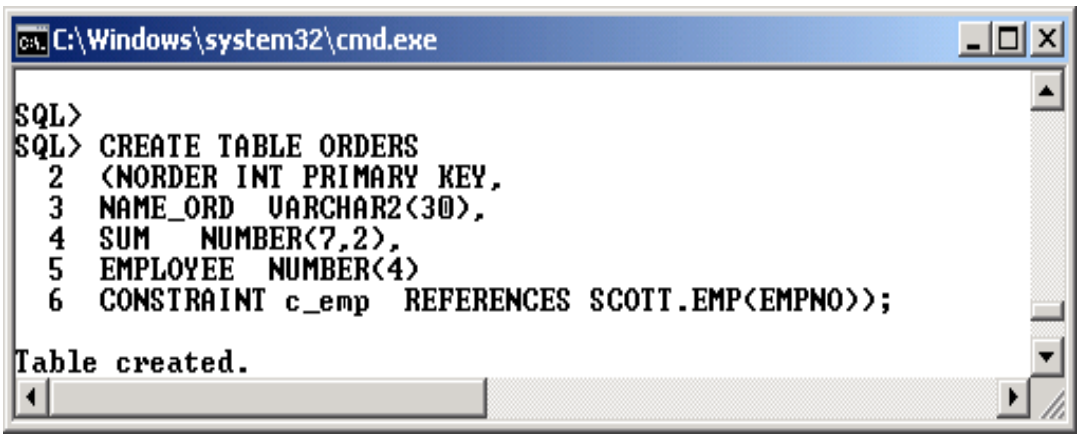


Рис. 3.8. **Створення таблиці з зовнішнім ключем**

Обмеження C_EMP гарантує, що кожне значення стовпця EMPLOYEE у таблиці ORDERS відповідає певному значенню первинного ключа EMPNO у таблиці EMP схеми SCOTT.

Для перевірки обмеження треба додати два рядка до таблиці ORDERS (рис. 3.9).

```
INSERT INTO ORDERS VALUES (1,'BOOKS',300,7900);
INSERT INTO ORDERS VALUES (2,'PAPER',120,7915);
```

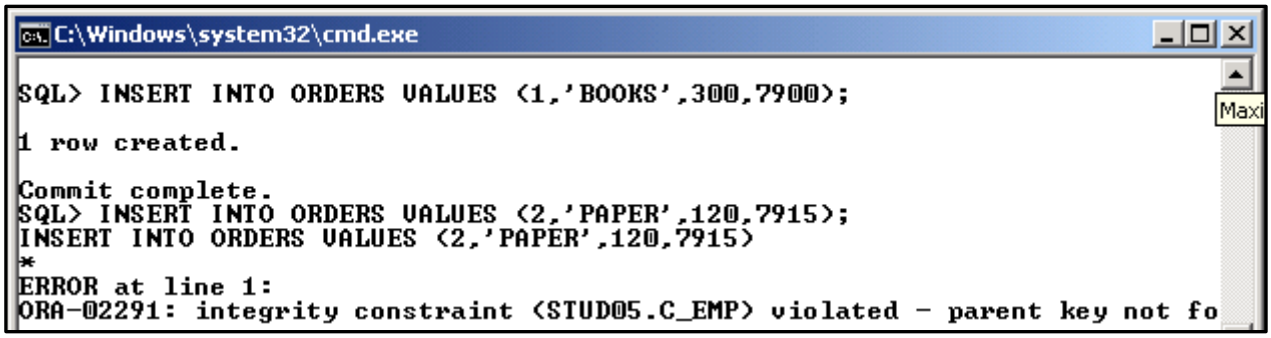


Рис. 3.9. **Контроль введення даних за зовнішнім ключем**

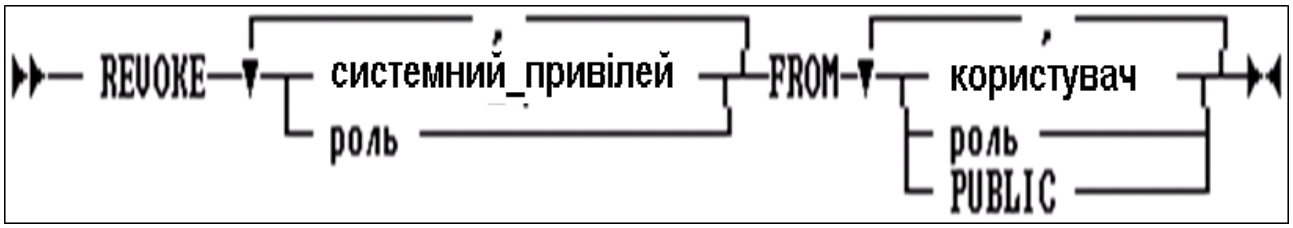
Як видно з отриманого результату, спроба додати до таблиці ORDERS другий рядок зі значенням 7915 у стовпці EMPLOYEE викликало помилку, оскільки співробітника з таким номером у таблиці SCOTT.EMP не існує.

3.1.7. Скасування системних привілеїв

Як для скасування системних привілеїв, так і для скасування об'єктних використовується команда REVOKE.

Для скасування саме системних привілеїв користувач, по-перше, повинен мати ці привілеї, а по-друге, вони повинні бути призначені користувачу з опцією WITH ADMIN OPTION. Крім того, користувач може відкликати будь-яку роль, якщо має системний привілей GRANT ANY ROLE.

Синтаксис команди REVOKE для скасування системних привілеїв такий:



де **системний_привілей** – системний привілей, що скасовується (відзивається);

роль – роль, що скасовується;

FROM – ідентифікує користувачів або ролі, у яких скасовуються системні привілеї та ролі;

PUBLIC – скасовує системні привілеї або ролі від всіх користувачів.

Слід звернути увагу на те, що при скасуванні ролі з опцією PUBLIC, вона не скасовується у тих користувачів, які отримали її безпосередньо або через інші ролі.

Приклад 3.8. Відкликати роль USER_ROLE від користувача STUD0001:

REVOKE USER_ROLE FROM STUD0001;

Користувач STUD0001 більше не зможе використовувати привілеї, що були йому надані за допомогою ролі USER_ROLE.

Приклад 3.9. Відкликати системний привілей DROP ANY TABLE від користувачів TEACHER та STUDENT:

REVOKE DROP ANY TABLE FROM TEACHER та STUDENT

Користувачі TEACHER та STUDENT надалі не зможуть видаляти таблиці не у своїх схемах.

Приклад 3.10. Відкликати роль USER_ROLE від ролі STUD_ROLE.

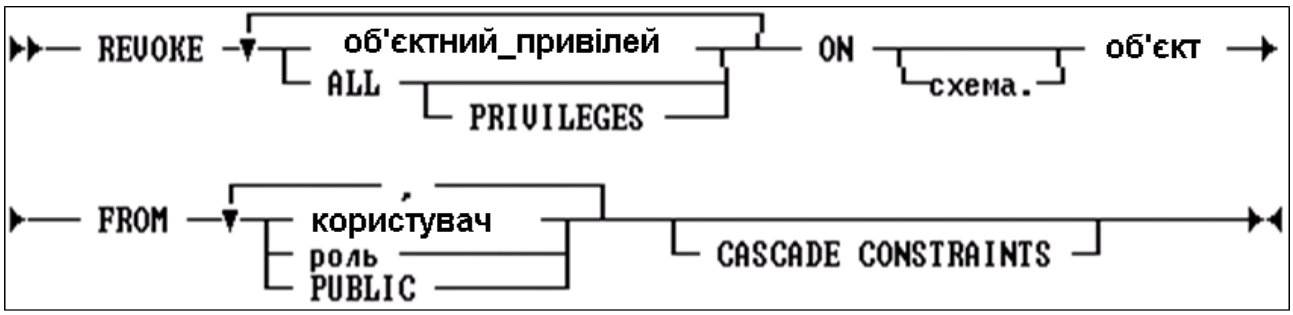
REVOKE USER_ROLE FROM STUD_ROLE.

Роль USER_ROLE більше не призначена ролі STUD_ROLE.

3.1.8. Скасування об'єктних привілеїв

Команда REVOKE з переліком певних об'єктних привілеїв на конкретний об'єкт бази даних скасовує їх у ролях та у користувачів.

Синтаксис команди REVOKE для скасування (відклику, відзиву) об'єктних привілеїв такий:



де, **об'єктний_привілей** – об'єктний привілей (ALTER, DELETE, EXECUTE, INDEX, INSERT, REFERENCES, SELECT, UPDATE), що скасовується у користувача або ролі;

ALL PRIVILEGES – скасовує усі привілеї на об'єкт, які були призначені користувачеві або ролі;

ON – визначає об'єкт, для якого скасовуються привілеї. Якщо не вказується **схема,** то Oracle вважає, що об'єкт перебуває у схемі поточного користувача. Допускаються наступні типи об'єктів: таблиця; подання; послідовність; процедура; функція або пакет; знімок; синонім для таблиці, подання, знімка, функції, процедури, пакета або послідовності;

FROM – визначає користувачів або ролі, у яких скасовуються привілеї на об'єкт;

PUBLIC – скасовує привілеї на роботу з об'єктом від усіх користувачів;

CASCADE CONSTRAINTS – видаляє будь-які обмеження посилальної цілісності, які цільовий користувач визначав, користуючись привілеєм REFERENCES. Цю опцію необхідно вказувати, якщо скасовується привілей REFERENCES або задана опція ALL PRIVILEGES. Крім того, якщо користувач скасовує об'єктний привілей, який був призначений іншим користувачам або ролям (завдяки можливості GRANT OPTION), то Oracle також скасовує цей привілей від усіх таких користувачів та ролей.

Приклад 3.11. Дано: користувач STUD04 має привілеї DELETE, INSERT та UPDATE за таблицею SALGRADE у схемі SCOTT, що були призначені командою:

GRANT ALL ON SALGRADE TO STUD04;

Відкликати привілей DELETE від STUD04.

REVOKE DELETE ON SALGRADE FROM STUD04;

Відкликати привілеї, що залишилися, за таблицею SALGRADE, які були призначені STUD04.

REVOKE ALL ON SALGRADE FROM STUD04;

Приклад 3.12. Призначити привілеї SELECT і UPDATE за поданням BOSTON_VAC усім користувачам за допомогою призначення цих привілеїв ролі PUBLIC:

GRANT SELECT, UPDATE ON BOSTON_VAC TO PUBLIC;

Скасувати привілей UPDATE за поданням BOSTON_VAC від усіх користувачів:

REVOKE UPDATE ON BOSTON_VAC FROM PUBLIC;

Користувачі більше не можуть оновлювати подання BOSTON_VAC, хоча, як і раніше, можуть опитувати його. Однак якщо привілей UPDATE був призначений за поданням BOSTON_VAC будь-яким конкретним користувачам (безпосередньо або через ролі), то такі користувачі збережуть цей привілей.

Приклад 3.13. Призначити користувачеві STUD06 привілей SELECT за послідовністю NUM_VACATION у схемі SCOTT.

GRANT SELECT ON SCOTT.NUM_VACATION TO STUD06;

Відкликати привілей SELECT від STUD06.

REVOKE SELECT ON SCOTT.NUM_VACATION FROM STUD06;

Однак якщо користувач SCOTT також призначив користувачеві STUD06 привілей SELECT за своєю послідовністю NUM_VACATION, то STUD06, як і раніше, зможе користуватися цим привілеєм.

Приклад 3.14. Користувач STUD05 отримав привілеї REFERENCES та UPDATE за таблицею EMP у схемі SCOTT (приклад 3.7).

Користувач STUD05 скористався привілеєм REFERENCES, щоб визначити обмеження за своєю власною таблицею ORDERS, що посилається на таблицю EMP у схемі SCOTT.

Скасувати привілей REFERENCES на таблицю SCOTT.EMP у користувача STUD05 з опцією CASCADE CONSTRAINTS.

```
REVOKE REFERENCES  ON SCOTT.EMP
FROM STUD05   CASCADE CONSTRAINTS
```

Відкликання привілею REFERENCES змушує систему видалити обмеження C_EMP, тому що користувачеві STUD05 був потрібний цей привілей для визначення даного обмеження.

Однак якщо користувач STUD05 отримав привілей REFERENCES за таблицею SCOTT.EMP ще від якого-небудь користувача, то Oracle не видаляє це обмеження, тому STUD05, як і раніше, зберігає необхідний привілей.

3.1.9. Привілеї, необхідні для виконання операцій у базі даних

Для створення, використання, модифікації чи видалення основних об'єктів бази даних користувач повинен мати певні привілеї. Слід розглянути основні з них, що відносяться до найбільш використовуваних об'єктів бази даних – таких, як таблиці, подання, послідовності, синоніми та індекси [32; 48; 56; 64].

Привілеї, необхідні для створення таблиць

Щоб створити нову таблицю у власній схемі, користувач повинен мати системний привілей CREATE TABLE. Для створння таблиці у схемі іншого користувача необхідно мати системний привілей CREATE ANY TABLE. Крім того, власник таблиці повинен мати квоту для табличного простору, в якому міститься таблиця, або системний привілей UNLIMITED TABLESPACE.

Привілеї, необхідні для зміни таблиць

Щоб змінити таблицю, вона повинна міститися у власній схемі користувача або він повинен мати об'єктний привілей ALTER для цієї таблиці чи системний привілей ALTER ANY TABLE.

Привілеї, необхідні для видалення таблиць

Щоб видалити таблицю, вона повинна міститися у власній схемі, або користувач повинен мати системний привілей DROP ANY TABLE.

Привілеї, необхідні для створення подання

Щоб створити подання у власній схемі, користувач повинен мати привілей CREATE VIEW. Для створення подання у схемі іншого користувача, необхідно мати системний привілей CREATE ANY VIEW. Обидва ці привілеї можуть бути отримані користувачем явно або через роль.

Крім того, власник подання повинен явно володіти привілеями для доступу до об'єктів, на які посилається визначення подання; ці привілеї не можуть бути отримані через роль.

Якщо власник подання має намір надати доступ до нього іншим користувачам, він повинен володіти об'єктними привілеями для базових об'єктів подання з опцією GRANT OPTION або системними привілеями з опцією ADMIN OPTION. В іншому випадку власник не може надати доступ до подання іншим користувачам.

Привілеї, необхідні для видалення подання

Користувач може видаляти будь-яке подання, що міститься у його власній схемі. Щоб видалити подання у схемі іншого користувача, він повинен мати системний привілей DROP ANY VIEW.

Привілеї, необхідні для заміни подання

Щоб замінити подання, користувач повинен мати усі привілеї, необхідні для видалення та створення подання.

Привілеї, необхідні для використання подання

Щоб видавати для подання запити INSERT, UPDATE або DELETE, користувач повинен мати для цього подання об'єктний привілей SELECT, INSERT, UPDATE або DELETE, відповідно, отриманий явно або через роль.

Привілеї, необхідні для створення послідовностей

Щоб створити послідовність у власній схемі, користувач повинен мати системний привілей CREATE SEQUENCE. Для створення послідовності у схемі іншого користувача, необхідно мати системний привілей CREATE ANY SEQUENCE.

Привілеї, необхідні для зміни послідовностей

Щоб змінити послідовність, користувач повинен мати її у власній схемі або мати системний привілей ALTER ANY SEQUENCE.

Привілеї, необхідні для використання послідовностей

Щоб використовувати послідовність, користувач повинен володіти нею у власній схемі або мати об'єктний привілей SELECT для послідовності, що належить іншому користувачеві.

Привілеї, необхідні для видалення послідовностей

Користувач може видаляти будь-яку послідовність, яка міститься у його власній схемі. Щоб видалити послідовність у схемі іншого користувача, необхідно мати системний привілей DROP ANY SEQUENCE.

Привілеї, необхідні для створення синонімів

Щоб створити особистий синонім у власній схемі, користувач повинен мати привілей CREATE SYNONYM. Для створення особистого синоніму в схемі іншого користувача, необхідно мати системний привілей CREATE ANY SYNONYM. Щоб створити загальний синонім, користувач повинен мати системний привілей CREATE PUBLIC SYNONYM.

Привілеї, необхідні для використання синоніма

Користувач може використовувати будь-який особистий синонім, що міститься у його схемі, а також будь-який загальний синонім за умови, що він має необхідні привілеї для доступу до об'єктів, на які фактично вказує синонім, що отримані явно, через діючу роль, або через групу PUBLIC. Користувач може також звертатися до будь-якого синоніму, що міститься у чужій схемі, якщо йому були призначені необхідні об'єктні привілеї для цього синоніма.

Привілеї, необхідні для видалення синонімів

Користувач може видалити будь-який особистий синонім, що міститься у його власній схемі. Щоб видалити особистий синонім у схемі іншого користувача, необхідно мати системний привілей DROP ANY SYNONYM. Для видалення загального синоніма, користувач повинен мати системний привілей DROP PUBLIC SYNONYM.

Привілеї, необхідні для створення індексів

Щоб створити новий індекс, користувач повинен володіти відповідною таблицею або мати для неї об'єктний привілей INDEX. Схема, в якій створюється індекс, повинна також мати квоту для табличного простору, в якому буде міститися індекс, або системний привілей UNLIMITED TABLESPACE. Щоб створити індекс у схемі іншого користувача, необхідно мати системний привілей CREATE ANY INDEX.

Привілеї, необхідні для видалення індексу

Щоб видалити індекс, користувач повинен мати його у власній схемі або володіти системним привілеєм DROP ANY INDEX.

Привілеї, необхідні для перейменування об'єкта

Для перейменуваня об'єкту, користувач повинен бути його власником.

Спробуйте самостійно визначити перелік користувачів, які будуть мати право користуватися базою даних, яка аналогічна навчальній БД, наведеній у підрозділі 2.1, та їх права на виконання тих чи інших операцій над об'єктами цієї бази.

Запитання і завдання

1. У чому полягає відмінність між вибірковим та обов'язковим підходами до безпеки даних? Опишіть переваги та недоліки кожного з них.

2. У яких випадках використовують системні привілеї, а в яких – об'єктні?

3. Які мовні засоби SQL Oracle використовують для створення ролей та користувачів у базі даних?

4. Наведіть приклади використання мовних засобів SQL Oracle для надання та скасування системних та об'єктних привілеїв.

5. Які привілеї повинні мати користувачі, щоб створювати, використовувати, змінювати та вилучати основні об'єкти бази даних?

3.2 Словник даних Oracle

Словник даних Oracle становить значну кількість таблиць і об'єктів бази даних, які зберігаються у спеціальній області бази даних і ведуться виключно ядром Oracle. Словник даних містить інформацію про об'єкти бази даних, користувачів та події. До цієї інформації можна звернутися за допомогою подань словника даних. Запити на зчитування з бази даних чи її оновлення обробляються ядром Oracle з використанням інформації зі словника даних [31; 34].

Словник даних – не тільки центральне сховище у кожній базі даних ORACLE, а також важливий інструмент для всіх користувачів – від кінцевих користувачів до розробників додатків і адміністраторів бази даних.

Словник даних є однією з найважливіших частин бази даних ORACLE та може надавати таку інформацію про базу даних:

- імена користувачів ORACLE;
- привілеї таролі, які були надані кожному користувачеві;
- імена об'єктів схем (таблиць, подань, знімків, індексів, кластерів, синонімів, послідовностей, процедур, функцій, пакетів, тригерів тощо);
- інформацію про обмеження цілісності;
- значення за замовчуванням для стовпців;
- об'єм зовнішньої пам'яті, розподілений і на даний час використовуваний об'єктами у базі даних;
- інформацію аудиту, наприклад: хто звертався до різних об'єктів і оновлював їх тощо [31].

Інформація у словнику даних призначена для підтвердження існування об'єктів, забезпечення доступу до них і опису фактичного фізичного розташування у пам'яті.

Першими таблицями, що створюються у будь-якій базі даних, є системні таблиці, або словник даних Oracle. Системні таблиці зберігають інформацію про структуру бази даних і об'єктів усередині неї. Oracle звертається до них, коли необхідна інформація про базу даних або коли виконується команда DDL чи DML [34].

Таблиці словника ніколи безпосередньо не оновлюються, проте відновлення у них відбувається у фоновому режимі щоразу, коли виконується оператор DDL [34].

Головні таблиці словника даних містять нормалізовану інформацію, яка є досить важкою для сприйняття. Тому в Oracle передбачений набір подань, які видають інформацію з головних системних таблиць у більш зрозумілому вигляді.

Подання словника даних виступають як довідники для всіх користувачів бази даних. Доступ до них здійснюється через запити мовою SQL. Деякі подання доступні усім користувачам, інші – призначені тільки для адміністраторів.

Словник даних завжди доступний при відкритій базі даних. Він розміщується у табличному просторі SYSTEM і завжди знаходиться у стані online.

Словник даних складається з декількох наборів подань. У багатьох випадках такий набір складається з трьох подань, що містять аналогічну інформацію, а відрізняються лише префіксами в іменах [43] (табл. 3.2) :

Таблиця 3.2

Характеристика префіксів імен для подань словника даних

Префікс	Призначення
USER	Погляд користувача на БД – містить об'єкти його власної схеми
ALL	Розширений погляд користувача – об'єкти, до яких він має доступ
DBA	Погляд адміністратора – усі об'єкти, до яких можуть мати доступ усі користувачі

У поданнях із префіксом USER зазвичай немає стовпця на ім'я OWNER (власник); у поданнях USER власником є користувач, що видав запит.

Подання з префіксом ALL повертають інформацію про всі об'єкти, до яких користувач має доступ. Тобто вони включать усю інформацію про об'єкти, що містяться у поданнях з префіксом USER, і додатково – про ті об'єкти, привілеї доступу до яких поточному користувачу були надані іншими користувачами.

Подання з префіксом DBA показують загальне подання про базу даних і призначені тільки для адміністраторів бази даних. Тобто опитувати подання словника з префіксом DBA може будь-який користувач, що має системний привілей SELECT ANY TABLE.

Перелік основних подань словника бази даних Oracle наведений у табл. Е.1 додатка Е. З повним переліком можна ознайомитися у [34; 85].

Отримати відомості про структуру кожного подання, що входить до словника, можна за допомогою команди SQL*PLUS – DESCRIBE.

Приклад 3.15. Отримати відомості про структуру подання USER_CONSTRAINTS, що містить дані про наявні обмеження, що були створені для об'єктів поточного користувача (рис. 3.10).

DESCRIBE USER_CONSTRAINTS

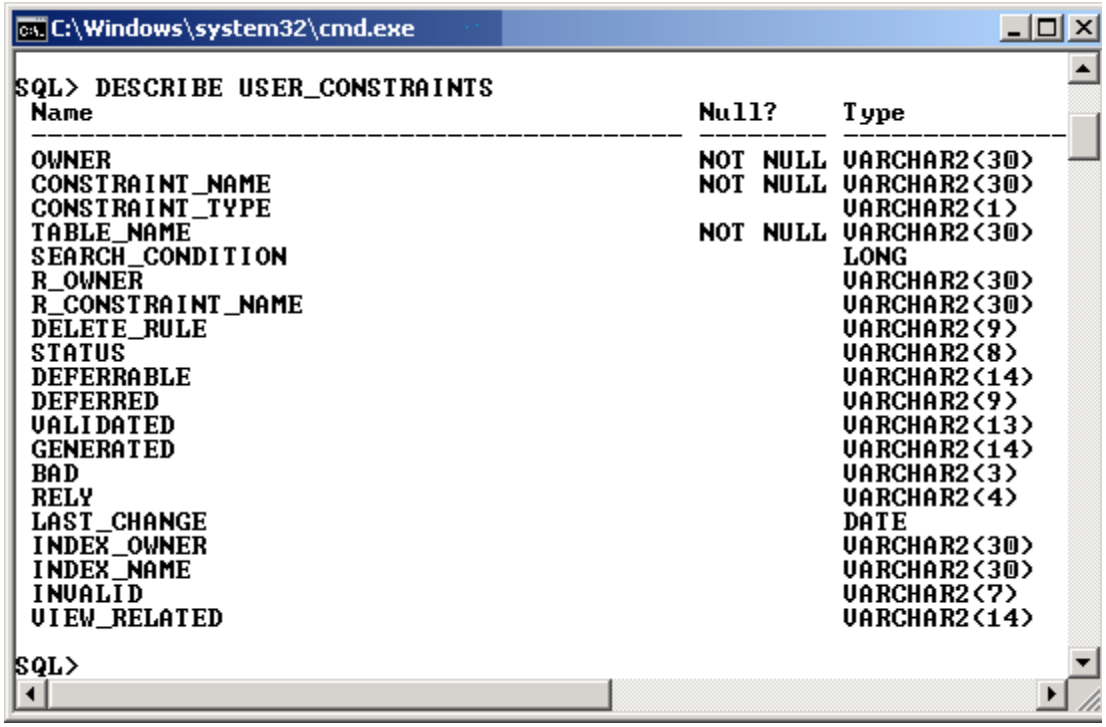


Рис. 3.10. **Структура подання USER_CONSTRAINTS**

Як видно з отриманого результату, назви стовпців подання англійською мовою характеризують сенс значень для конкретного стовпця.

Приклад 3.16. Отримати відомості, про обмеження, які діють для таблиці EMP, а саме – тип обмеження, його статус, назву та умови (рис. 3.11).

```
COLUMN SEARCH_CONDITION FORMAT A30 WRAP
SELECT CONSTRAINT_TYPE, STATUS,
       CONSTRAINT_NAME, SEARCH_CONDITION
FROM USER_CONSTRAINTS
WHERE TABLE_NAME = 'EMP';
```

В отриманому результаті CONSTRAINT_TYPE (C) характеризує тип обмеження (P – первинний ключ, C – перевірка, R – обмеження за зовнішнім ключем). Крім того, значеннями цього стовпця можуть бути: U – унікальний ключ, V – режим WITH CHECK OPTION для подання, O – тільки для читання для подання.

У назвах обмежень зустрічаються значення, що починаються зі сполучення SYS_C (SYS_C004252,…, SYS_C004276) тощо. Ці імена Oracle привласнив сам, оскільки вони не були вказані при створенні таблиці. Інші мають ті імена обмежень, що вказував безпосередньо користувач.

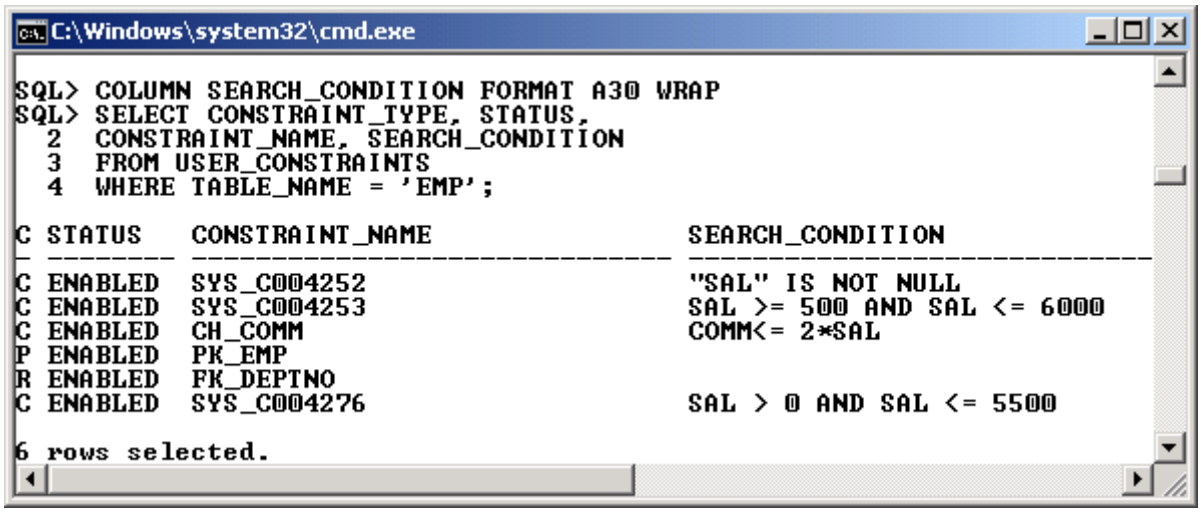


Рис. 3.11. **Відомості про обмеження таблиці EMP**

Приклад 3.17. Отримати відомості про об'єкти схеми користувача та їх типи (рис. 3.12).

```
COLUMN OBJECT_NAME FORMAT A30 WRAP
COLUMN OBJECT_TYPE FORMAT A10 WRAP
SELECT OBJECT_NAME, OBJECT_TYPE FROM USER_OBJECTS;
```

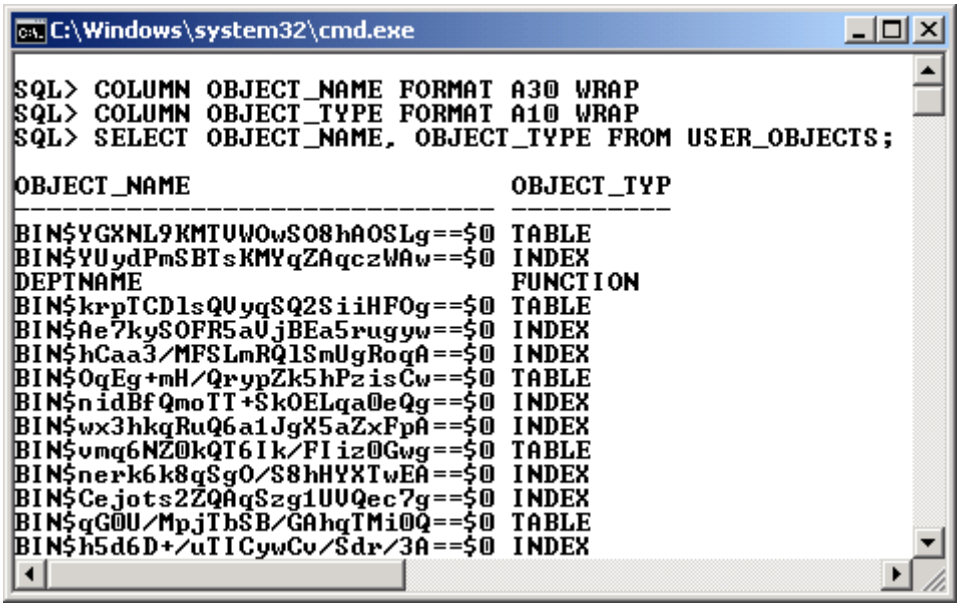


Рис. 3.12. **Відомості про об'єкти схеми користувача**

У результаті цього запиту було отримано більше вісімдесяти рядків, тому вони наведені не усі.

Приклад 3.18. Отримати відомості тільки про таблиці та послідовності у схемі поточного користувача, за винятком службових (рис.3.13).

```
SELECT OBJECT_NAME, OBJECT_TYPE
FROM USER_OBJECTS
WHERE OBJECT_TYPE IN ('TABLE', 'SEQUENCE')
      AND INSTR(OBJECT_NAME, '$')=0;
```

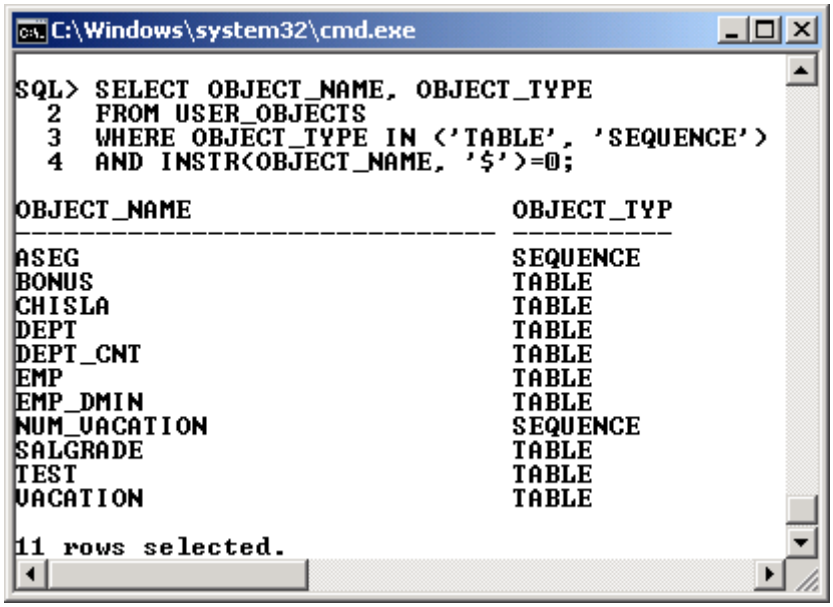


Рис. 3.13. **Відомості про таблиці та послідовності**

Якщо необхідно отримати відомості тільки про таблиці та послідовності, до яких має доступ користувач, то замість USER_OBJECTS у фразі FROM слід вказати ALL_OBJECTS.

При створенні подання або синоніма, вони опираються на певний базовий об'єкт. Подання словника даних із суфіксом _DEPENDENCIES можна використовувати для отримання інформації про залежності для подання, а подання словника даних із суфіксом _SYNONYMS – для показу базового об'єкта синоніма.

Приклад 3.19. Отримати відомості про залежності для існуючих подань у схемі користувача (рис. 3.14).

```
COLUMN REFERENCED_NAME FORMAT A20 WRAP
COLUMN REFERENCED_TYPE FORMAT A10 WRAP
COLUMN NAME FORMAT A12 WRAP
SELECT NAME, TYPE, REFERENCED_NAME,
REFERENCED_TYPE, DEPENDENCY_TYPE
FROM USER_DEPENDENCIES
WHERE TYPE='VIEW';
```

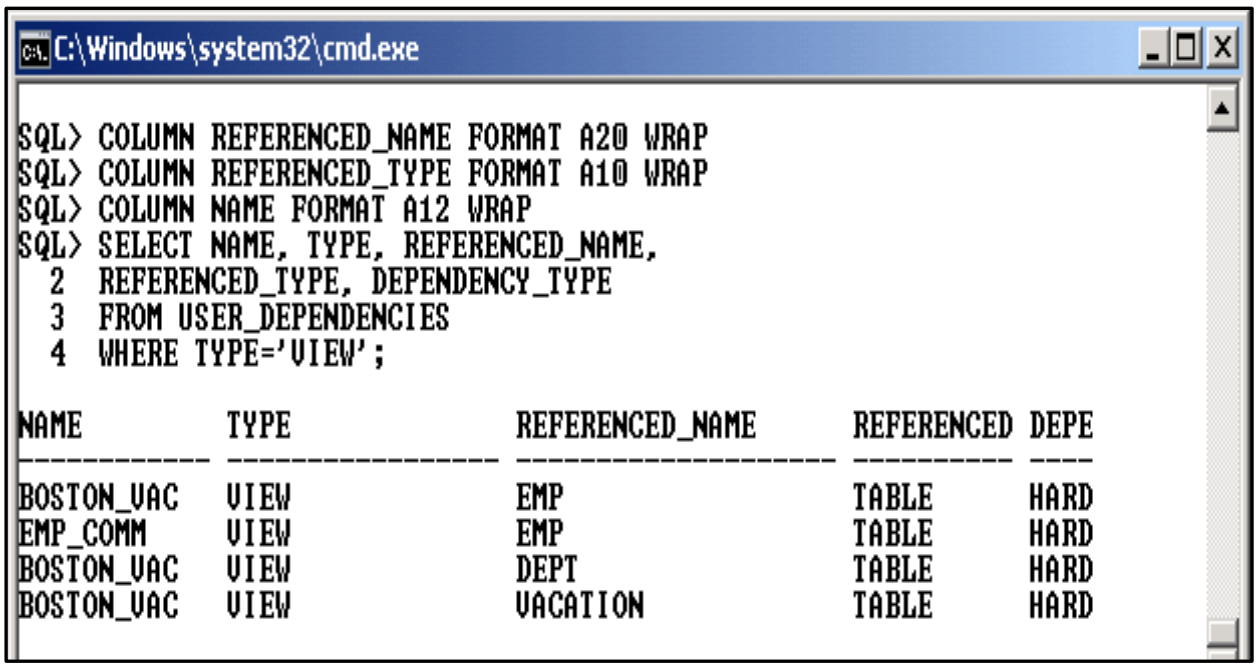


Рис. 3.14. **Відомості про залежності для існуючих подань.**

Приклад 3.20. Скористатися поданням DICTIONARY для отримання переліку усіх подань та системних таблиць (рис. 3.15).

```
COLUMN TABLE_NAME FORMAT A15
COLUMN COMMENTS FORMAT A30 WRAP
SELECT * FROM DICTIONARY;
```

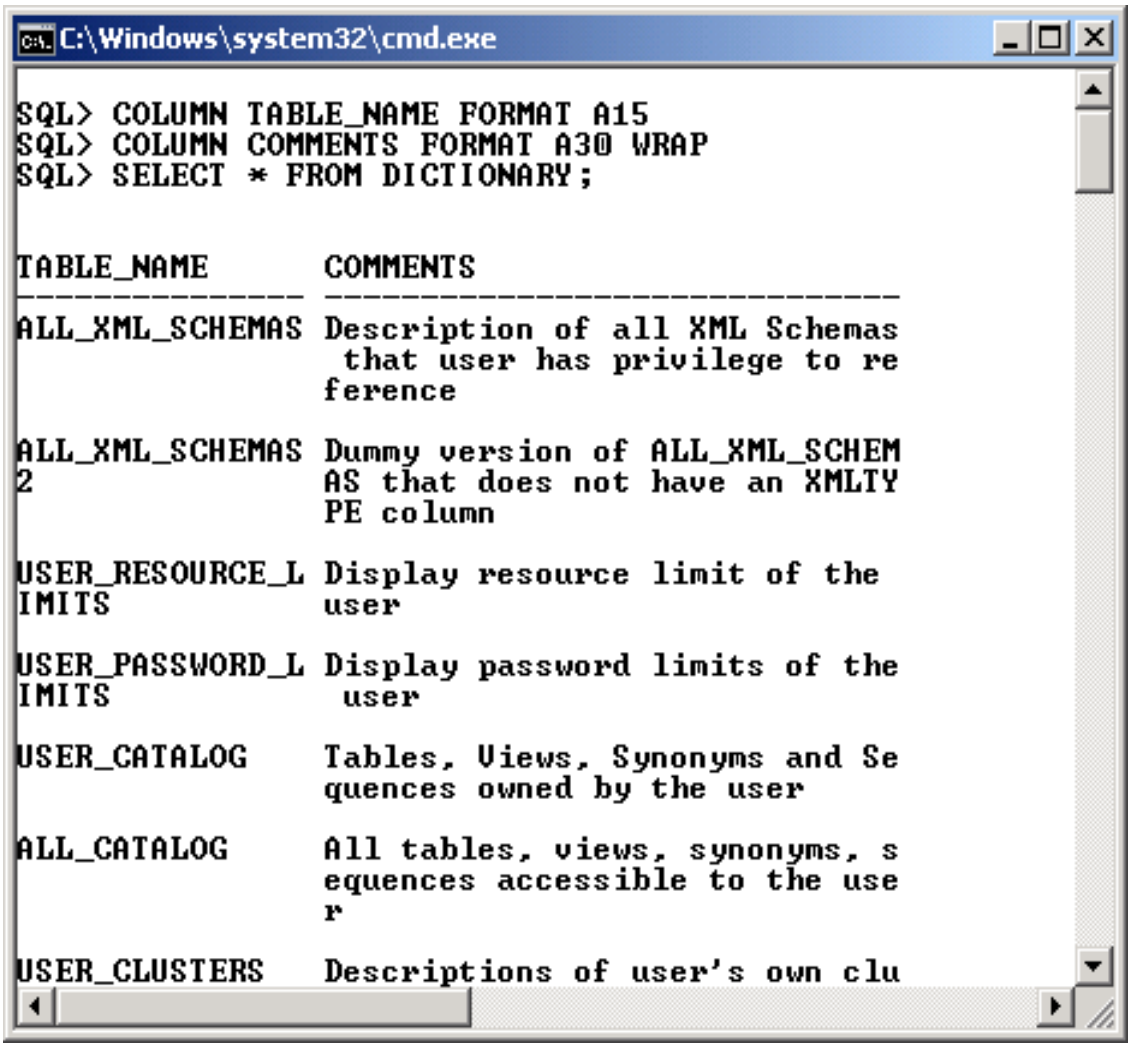


Рис. 3.15. **Перелік усіх подань та системних таблиць**

Приклад 3.21. Отримати відомості про перелік стовпців, типи даних та значення за замовчуванням для таблиць EMP та DEPT (рис. 3.16).

```
COLUMN TABLE_NAME FORMAT A5 WRAP
COLUMN COLUMN_NAME FORMAT A10 WRAP
COLUMN DATA_TYPE FORMAT A10 WRAP
COLUMN DD FORMAT A10 WRAP
```

```
SELECT TABLE_NAME, COLUMN_NAME, DATA_TYPE,
DATA_LENGTH, DATA_PRECISION,
NULLABLE, DATA_DEFAULT AS DD
FROM USER_TAB_COLUMNS
WHERE TABLE_NAME IN ('EMP', 'DEPT');
```

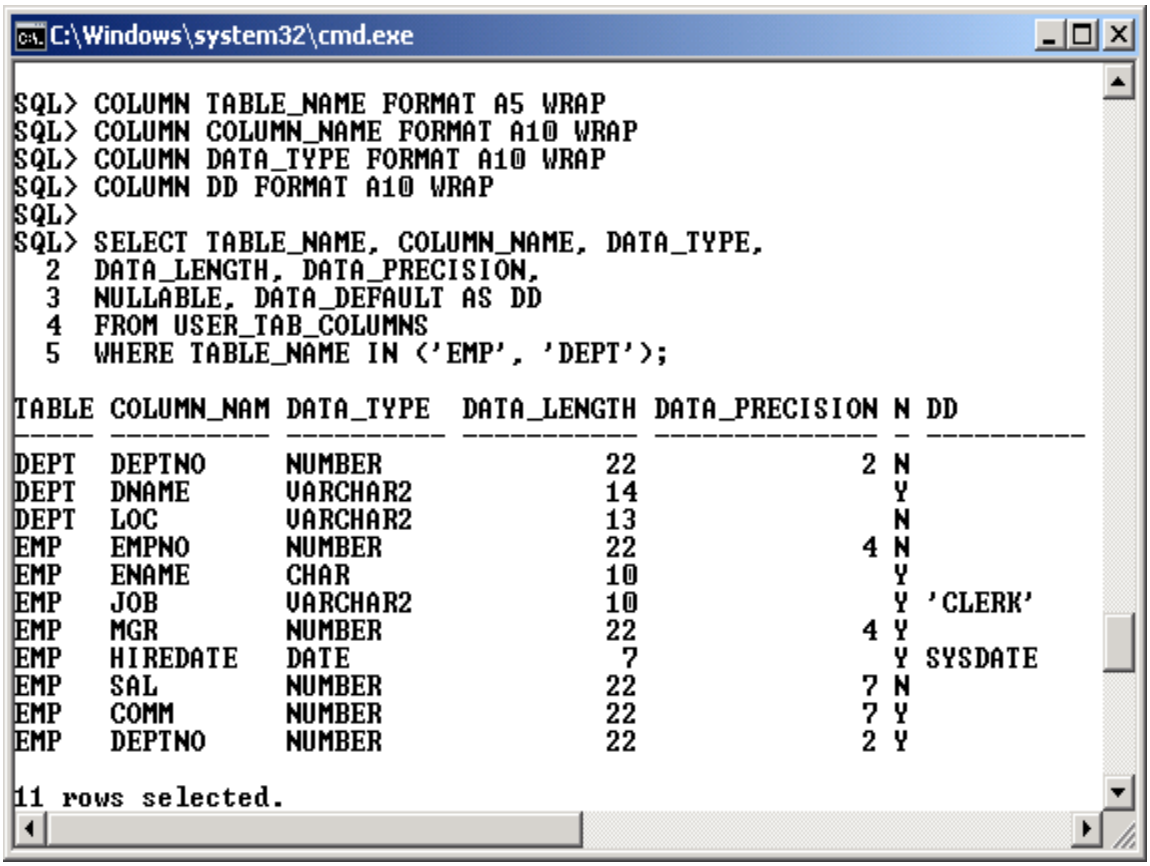


Рис. 3.16. **Відомості про складові таблиць EMP та DEPT**

Запитання і завдання

1. Яку інформацію містить словник даних Oracle?
2. Які основні подання словника бази даних Oracle вам відомі? Опишіть їх призначення.
3. Наведіть приклади отримання інформації зі словника бази даних Oracle за допомогою запитів мовою SQL.

Лабораторна робота № 3
Керування доступом у СКБД ORACLE.
Робота зі словником ORACLE

Цілі лабораторної роботи:

1. Набуття практичних навичок створення ролей та користувачів СКБД Oracle.
2. Набуття практичних навичок надання та вилучення системних привілеїв створеним користувачам.
3. Набуття практичних навичок надання та вилучення об'єктних привілеїв на об'єкти власної схеми БД іншим користувачам системи.
4. Ознайомлення з призначенням та основними складовими словника даних Oracle.
5. Набуття практичних навичок у складанні запитів мовою SQL до словника даних з метою отримання інформації щодо переліку, структури, обмежень тощо від основних об'єктів бази даних.

Перед виконанням лабораторної роботи студент повинен знати:

1. Основні підходи до забезпечення безпеки даних у сучасних СКБД.
2. Види привілеїв та їх призначення.
3. Базові команди SQL, що використовуються для забезпечення захисту даних у СКБД.
4. Поняття "словник даних" та його роль для сучасних СКБД.

Після виконання лабораторної роботи студент повинен уміти:

1. Самостійно від імені адміністратора системи створювати ролі та користувачів.
2. Надавати певні системні привілеї ролям та користувачам на виконання операції у БД та скасовувати їх у разі необхідності.
3. Надавати об'єктні привілеї на створені об'єкти БД іншим користувачам та скасовувати їх.
4. Виконувати запити до словника даних з метою отримання необхідної інформації щодо об'єктів бази даних.

Робота виконується самостійно на локальній машині від імені адміністратора та створених користувачів системи.

3.1. Керування доступом у СКБД ORACLE

Завдання 1. Підключитися до бази даних від імені адміністратора та, базуючись на прикладі, наведеному у скрипті ***CreateRole.SQL***, створити ролі, які потребують (manager) чи не потребують (user_role) паролю для своєї активізації.

```
Drop role manager;
-- Create the manager identified by passw
create role manager identified by passw;
-- Grant/Revoke system privileges
grant resource to manager;
grant create table to manager;
grant create view to manager;
grant create sequence to manager;
grant create synonym to manager;
grant create procedure to manager;
grant create type to manager;
grant create session to manager;
grant create trigger to manager;
grant select any dictionary to manager;

Drop role user_role;
create role user_role;
-- Grant/Revoke system privileges
grant resource to user_role;
grant create table to user_role;
grant create view to user_role;
grant create sequence to user_role;
grant create synonym to user_role;
grant create session to user_role;
grant select any dictionary to user_role;
```

Завдання 2. Базуючись на скрипті ***CreateUser.SQL***, створити кілька користувачів та надати їм привілеї на виконання певних операцій у базі даних, опираючись на вже створені ролі.

```
-- CREATE USER STUDP1
drop user studp1 cascade;
create user studp1
  identified by studp1
  default tablespace USERS
  temporary tablespace TEMP
  profile DEFAULT;

  grant connect to studp1;
grant manager to studp1;
grant unlimited tablespace to studp1;

-- CREATE USER STUDP2
drop user studp2 cascade;
create user studp2
  identified by studp2
  default tablespace USERS
  temporary tablespace TEMP
  profile DEFAULT;

  grant connect to studp2;
grant user_role to studp2;
grant unlimited tablespace to studp2;

-- CREATE USER STUDP3
drop user studp3 cascade;
create user studp3
  identified by studp3
  default tablespace USERS
  temporary tablespace TEMP
  profile DEFAULT;

  grant connect to studp3;
grant user_role to studp3;

grant manager to studp3;
grant unlimited tablespace to studp3;
```

Завдання 3. Підключитися до БД першим створеним користувачем та здійснити зміну його паролю на інший.

```
ALTER USER  studp1 IDENTIFIED BY studparol;
```

Завдання 4. Відключити для користувача надані йому ролі та спробувати створити нову таблицю у БД.

```
SQL> SET ROLE NONE;
Role set.

SQL> CREATE TABLE TestTable (A INT);
CREATE TABLE TestTable (A INT)
*
ERROR at line 1:
ORA-01031: insufficient privileges
```

Завдання 5. Активізувати для користувача роль manager, відновивши привілеї, та спробувати повторно створити нову таблицю у БД.

```
SQL> SET ROLE manager IDENTIFIED BY passw;
Role set.

SQL> CREATE TABLE TestTable (A INT);
Table created.
```

Завдання 6. Підключитися до БД як другий користувач (studp2), стварити та заповнити таблицю SUBJECT (Предмет).

```
CREATE TABLE SUBJECT (ID_SUB NUMBER(3) PRIMARY KEY,
                      SUBJECT VARCHAR2(15));
INSERT INTO SUBJECT VALUES (1,'Phisics');
INSERT INTO SUBJECT VALUES (2,'English');
```

Завдання 7. Підключитися до БД як третій користувач та спробувати виконати запит: ***SELECT * FROM STUDP2.SUBJECT;***

Який результат був отриманий та чому?

Завдання 8. Підключитися до БД як адміністратор та надати ролі manager додатковий привілей *SELECT ANY TABLE.*

```
GRANT SELECT ANY TABLE TO manager;
```

Після цього повторити завдання 7, тобто підключитися до БД як третій користувач та виконати запит SELECT * FROM STUDP2.SUBJECT;

Поясніть отриманий результат.

Завдання 9. Створити таблицю Teacher (Викладач), у схемі користувача **studp3** і заповнити її даними,

```
CREATE TABLE TEACHER (ID_TEACHER NUMBER(3) PRIMARY KEY,
                      TEACHER VARCHAR2(15),
                      DEPARTMENT VARCHAR2(20));
INSERT INTO TEACHER VALUES(1, 'IVANOV', 'IS');
INSERT INTO TEACHER VALUES(2, 'PETRENKO', 'IS');
INSERT INTO TEACHER VALUES(3, 'IVANENKO', 'KSIT');
```

Потім підключитися до БД як користувач studp2 та спробувати отримати дані зі створеної таблиці про викладачів.

Поясніть отриманий результат.

Що потрібно зробити, щоб команда виконалась успішно?

Завдання 10. Від імені адміністратора системи відібрати привілей *SELECT ANY TABLE* від ролі **manager**.

```
REVOKE SELECT ANY TABLE FROM manager;
```

До чого це може призвести? Перевірте Ваші здогадки виконанням конкретних операцій у базі даних.

Завдання 11. Від імені користувача **studp3** надати привілеї SELECT та UPDATE на поле DEPARTMENT для таблиці TEACHER користувачу studp1, а також SELECT, INSERT та DELETE користувачу studp2 на ту ж саму таблицю.

```
GRANT SELECT, UPDATE(DEPARTMENT) ON TEACHER TO STUDP1;
GRANT SELECT, INSERT, DELETE ON TEACHER TO STUDP2;
```

Підключитися як користувач studp1 та виконати команди до таблиці TEACHER, на які є привілеї та на які вони відсутні. Пояснити отриманий результат.

Аналогічні дії виконати від імені користувача studp2.

Завдання 12. Від імені користувача studp3 відібрати усі привілеї на таблицю TEACHER від користувача studp1.

```
REVOKE ALL ON TEACHER FROM STUDP1;
```

Надати користувачу studp2 право виконувати операції SELECT, INSERT, DELETE та UPDATE на поле DEPARTMENT з правом надавати їх іншим користувачам.

```
GRANT SELECT, INSERT, DELETE, UPDATE(DEPARTMENT)
ON TEACHER TO STUDP2
WITH GRANT OPTION;
```

Самостійно від імені користувача studp2 надати певні права на таблицю TEACHER у схемі studp3 користувачу studp1. Перевірити виконані дії.

Завдання 13. Самостійно від імені користувача studp3 відібрати привілей SELECT на таблицю TEACHER від користувача studp2. Перевірити виконання команди.

Спробувати виконати запит SELECT на таблицю TEACHER від імені користувача studp1. Пояснити отриманий результат.

Завдання 14 (самостійне виконання)

- Створити декілька додаткових ролей з різними привілеями.
- Створити декілька додаткових користувачів та надати їм права від різних створених ролей.
- Перевірити можливість використання користувачами наданих їм та відсутніх у них привілеїв.
- Самостійно сформулювати декілька завдань на надання та скасування системних привілеїв. Перевірити результат виконання команд.
- Самостійно сформулювати декілька завдань на надання та скасування об'єктних привілеїв. Перевірити результат виконання команд.
- Самостійно сформулювати виконати та перевірити виконання команд на надання та скасування привілеїв, використовуючи ключове слово PUBLIC.

3.2. Робота зі словником ORACLE

Завдання 15. Отримати відомості про структуру системного подання USER_OBJECTS.

Завдання 16. Базуючись на отриманому результаті попереднього завдання, визначити усі об'єкти бази даних, що створені поточним користувачем.

Завдання 17. Задати обмеження на будь-яку таблицю поточного користувача, що забезпечує унікальність комбінацій полів. Виконати повторно запит до подання USER_OBJECTS. Пояснити отриманий результат.

Завдання 18. Знайти у словнику даних відомості про обмеження, накладені на створені таблиці. Для цього слід скористатися поданням USER_ CONSTRAINTS.

Завдання 19. Скористатися системним поданням ALL_TABLES для визначення усіх доступних користувачу таблиць та їх власників.

Завдання 20. Скористатися системним поданням ALL_TABLES для визначення усіх доступних користувачу таблиць, у яких в імені міститься рядок "NEXT".

Завдання 21. Створити подання, яке базується на двох довільних таблицях з індивідуального завдання. Задати запит до словника даних (USER_VIEWS), щоб переконатися, що створене подання відбите у ньому.

Завдання 22. Використати подання SESSION-PRIVS, щоб дізнатися, які привілеї користувач має у даний момент.

Завдання 23. За допомогою подання USER_ROLE_PRIVS отримати інформацію про ролі, які користувач отримав у системі.

Завдання 24. Для отриманих ролей з попереднього завдання дізнатися, які привілеї надані цим ролям.

Завдання 25. Скористатися системним поданням ALL_TAB_PRIVS для визначення привілеїв на об'єкти, які користувач отримав безпосередньо, через роль або як PUBLIC.

Примітка. Повний запит видає надто багато рядків таблиці.

Висновки

1. У сучасних СКБД для убезпечення даних підтримуються широко поширені підходи: вибірковий та обов'язковий.

2. У разі вибіркового управління користувач має різноманітні права (привілеї чи повноваження) при роботі з різними об'єктами. Різні користувачі зазвичай мають різні права доступу до того самого об'єкту, тому вибіркові схеми характеризуються значною гнучкістю.

3. У разі обов'язкового управління кожному об'єкту даних надається певний класифікаційний рівень, а кожен користувач має певний рівень допуску. Отже, при такому підході доступ до певного об'єкту даних мають тільки користувачі з відповідним рівнем допуску. Тому обов'язкові схеми досить жорсткі та статичні.

4. За допомогою ідентифікатора та пароля проводиться ідентифікація та аутентифікація користувача. Більшість комерційних СКБД дозволяє об'єднувати користувачів з однаковими привілеями в групи. Це спрощує процес адміністрування.

5. Привілеї показують набір дій, які можна проводити над тим чи іншим об'єктом. Наприклад, користувач має привілей для перегляду таблиці.

6. Роль – це сукупність привілеїв, що можуть призначатися користувачам або іншим ролям.

7. Захист в середовищі Oracle забезпечується за допомогою засобів створення, зміни та знищення облікових записів користувача бази даних Oracle та ролей, а також за допомогою привілеїв, які регулюють можливість створювати та змінювати об'єкти бази даних. Тобто система захисту Oracle належить саме до вибіркового підходу.

8. Зазвичай виділяють два типи привілеїв: системні та об'єктні.

9. Системні привілеї дозволяють користувачам виконувати конкретну дію на рівні системи або з конкретним типом об'єктів. Більшість системних привілеїв доступні тільки адміністраторам і розробникам застосувань, оскільки такі привілеї достатньо потужні.

10. Об'єктні привілеї дозволяють користувачам виконувати конкретні дії на конкретному об'єкті, наприклад: видаляти рядки чи здійснювати модифікацію у зазначеній таблиці.

11. Для створення облікового запису користувача у базі даних Oracle використовується команда CREATE USER.

12. Командою CREATE ROLE можна створювати ролі у системі з призначенням їй певних привілеїв, а у подальшому призначити цю роль тому чи іншому користувачу. Деякі ролі створюються автоматично самою СКБД Oracle, і система автоматично призначає їм певні системні привілеї.

13. Для призначення системних та об'єктних привілеїв та ролей користувачам і ролям використовується команда GRANT, а для їх скасування – команда REVOKE.

14. Словник даних Oracle становить значну кількість таблиць і об'єктів бази даних, які зберігаються у спеціальній області бази даних і ведуться виключно ядром Oracle.

15. Словник даних містить таку інформацію про базу даних: імена користувачів ORACLE; привілеї та ролі, які були надані кожному користувачеві; імена об'єктів схем (таблиць, подань, знімків, індексів, кластерів, синонімів, послідовностей, процедур, функцій, пакетів, тригерів тощо); інформацію про обмеження цілісності; значення за замовчуванням для стовпців; об'єм зовнішньої пам'яті, який було розподілено і наразі використовується об'єктами у базі даних; інформацію аудиту, наприклад: хто звертався до різних об'єктів і оновлював їх, тощо.

16. Головні таблиці словника даних містять нормалізовану інформацію, яка є досить важкою для сприйняття. Тому в Oracle передбачений набір подань, які видають інформацію з головних системних таблиць у більш зрозумілому вигляді.

17. Подання словника даних виступають як довідники для всіх користувачів бази даних. Доступ до них здійснюється через запити мовою SQL. Деякі подання доступні усім користувачам, інші – призначені тільки для адміністраторів.

18. Словник даних завжди доступний при відкритій базі даних. Він розміщується у табличному просторі SYSTEM і завжди знаходиться у стані online.

19. Словник даних складається з трьох наборів подань, що містять аналогічну інформацію, а відрізняються лише префіксами в іменах. Подання мають такі префікси:

USER – з боку користувача на БД (об'єкти його власної схеми);

ALL – розширений погляд користувача (об'єкти, до яких він має доступ);

DBA – пз боку адміністратора (усі об'єкти, до яких можуть мати доступ усі користувачі).

Глосарій

Агрегатні функції (aggregate functions) – функції, які зазвичай використовуються у реченнях **GROUP BY** і **HAVING**. Агрегатні функції генерують одне сумарне значення для групи значень у вказаному стовпці. До їх числа входять функції **AVG**, **COUNT**, **COUNT** (*), **MIN**, **MAX**, **SUM**.

Аргумент (argument) – значення (також називається параметром), яке передається у функцію.

Атрибут (attribute) – значення, що описує одну характеристику об'єкта. Також часто називається полем або стовпцем.

Багато-до-багатьох (many-to-many) – залежність між таблицями (наприклад, між таблицями "Студенти" та "Предмети"), при якій студент може вивчати декілька предметів, а предмет може вивчатися декількома студентами.

Багаторядковий підзапит (multirow subquery) – підзапит, який може повертати більше одного рядка.

База даних (database) – набір зв'язаних таблиць, що містять як самі дані, так і визначення об'єктів бази даних.

Булевий вираз (boolean expressions) – вираз, що повертає значення "істина" або "неправда".

Булеві операції (boolean operators) – логічні операції **AND**, **OR** та **NOT**.

Вибір (selection) – визначення умов на вибірку рядків з таблиці.

Вибірка даних (data retrieval) – пошук і відображення даних з бази даних за допомогою запитів (команда **SELECT**).

Визначення даних (data definition) – процес створення (видалення) бази даних і її об'єктів.

Вираз (expression) – константа, ім'я стовпця, функція або будь-яка їх комбінація з арифметичними операціями.

Відношення (relation) – синонім таблиці.

Віртуальна таблиця (virtual table) – те саме, що подання.

Включаючий діапазон (inclusive range) – діапазон, визначений з ключовим словом **BETWEEN**, у якому при пошуку враховуються як проміжні, так і граничні значення.

Власник (owner) – творець об'єкта бази даних, що є його власником і зазвичай має на нього повний набір повноважень.

Головна таблиця (parent table) – таблиця, що бере участь у зв'язку "головний – підлеглий", на один рядок (або запис) якої можуть посилатися багато рядків (записів) підпорядкованої таблиці.

Група (group.) – сукупність рядків, що зазвичай є результатом SQL-запиту.

Групові значення (group values) – значення (набори значень), загальні для всіх рядків групи. У конструкції HAVING використовуються для фільтрації груп цілком.

Групові функції (group functions) – те саме, що агрегатні функції.

Декартовий добуток (Cartesian product) – з'єднання двох таблиць, у результаті якого кожен рядок однієї таблиці комбінується з кожним рядком іншої таблиці.

Десятковий тип даних (decimal datatype) – тип даних, що використовується для запису десяткових даних.

Діаграма "сутність-зв'язок" (Entity Relationship Diagram, ERD) – діаграма, на якій відображена структура реляційної бази даних. Таблиці подаються певними стандартними символами ("сутностями"), які з'єднуються за допомогою стандартних символів і ліній ("зв'язків").

Запис (record) – набір пов'язаних полів, що описує певний об'єкт. Також називається кортежем або рядком.

Запит (query) – вимога на вибірку інформації з бази даних.

Зв'язок "головний-підлеглий" (parent-child relationship) – зв'язок між двома таблицями, коли на один рядок першої таблиці (головної) можуть посилатися багато рядків другої таблиці (підпорядкованої).

Зв'язок "один-до-багатьох" (one-to-many relationship) – зв'язок між таблицями, при якому на один рядок в одній таблиці можуть посилатися багато рядків іншої таблиці.

Зв'язок (relationship) – асоціація між двома і більше таблицями, встановлена на рівні стовпців.

Значення (value) – елемент даних, що знаходиться, наприклад, на перетині рядка та стовпця.

Значення за замовчуванням (default) – значення, яке автоматично вводиться системою, якщо користувач не вказує конкретне значення стовпця.

Зовнішній ключ (foreign key) – стовпець у таблиці, що відповідає первинному або унікальному стовпцю в іншій таблиці.

Зчеплений індекс (concatenated index) – те саме, що складений індекс.

Індекс (index) – спеціальний механізм для швидкого пошуку даних.

Каскадування (cascade) – поширення операцій оновлення або видалення на зв'язані таблиці у базі даних.

Ключове слово (keyword) – слово, що використовується у синтаксисі SQL. Також називається "зарезервованим словом".

Ключові значення (key VALUES) – первинні ключі, що однозначно ідентифікують рядки, тоді як зовнішні ключі забезпечують доступ до них з інших таблиць.

Команда (command) – будь-яка команда (оператор) SQL, наприклад: **INSERT** або **CREATE TABLE**.

Команда (statement) – команда визначення, маніпулювання або адміністрування даних.

Конкатенація (concatenation) – операція, у результаті якої дві частини тексту з'єднуються в один текст.

Константа (constant) – вираз з фіксованим значенням.

Контрольні обмеження (check constraint) – обмеження, що дозволяє вказати набір умов, яким повинні задовольняти вхідні дані, щоб бути вставленими у таблицю бази даних.

Корельований підзапит (correlated subquery) – підзапит, який не може виконуватися незалежно від зовнішнього запиту. Також називається повторюваним підзапитом, оскільки виконується для кожного рядка, що вибирається у зовнішньому запиті.

Кортеж (tuple) – набір пов'язаних атрибутів, що описують певний об'єкт. Також називається рядком або записом.

Літерал (literal) – фіксований текст у команді або програмі, який не підлягає зміні і інтерпретується буквально, а не розглядається як ім'я змінної.

Логічні операції (logical operators) – те саме, що булеві операції.

Маніпулювання даними (data manipulation) – вибірка та модифікація даних за допомогою команд **SELECT**, **INSERT**, **DELETE** та **UPDATE**.

Мова структурованих запитів (SQL) – мова для визначення, зміни й управління даними в реляційних базах даних. Скорочення від Structured Query Language.

Модифікація даних (data modification) – зміна даних за допомогою команд SQL **INSERT**, **DELETE** та **UPDATE**.

Набори (sets) – групи рядків, до яких застосовуються агрегатні функції.

Невизначене значення (NULL) – пропущене, невідоме або непридатне значення в стовпцях таблиць бази даних.

Незв'язаний підзапит (nonrelated subquery) – запит, який може виконуватися незалежно від зовнішньої команди SQL.

Непроцедурна мова (nonprocedural language) – мова, що дозволяє описати бажані результати без вказівки способу їх отримання.

З'єднання (join) – вибірка з декількох таблиць на основі порівняння значень у певних стовпцях.

Об'єктні привілеї (object privileges) – привілеї на визначений об'єкт бази даних – таблицю, послідовність тощо.

Обмеження (constraints) – речення у команді **CREATE TABLE** (**CHECK**, **PRIMARY KEY**, **UNIQUE**, **REFERENCES**, **FOREIGN KEY**), котрі забезпечують посилальну цілісність і підтримують бізнес-правила.

Обчислення з датами (date math) – арифметичні операції над датами, наприклад, визначення кількості днів між двома датами.

Один-до-багатьох (one-to-many) – відношення, в якому рядок першої таблиці може бути пов'язаний з декількома рядками другої таблиці, але будь-який рядок другої таблиці може бути пов'язаний тільки з єдиним рядком першої таблиці.

Однорядковий підзапит (single-row subquery) – підзапит, що повертає не більше одного рядка.

Операції порівняння (comparison operators) – операції, що використовуються для порівняння значень виразів у реченнях **WHERE** або **HAVING**. До їх числа входять операції "порівняння" (=), "більше ніж" (>), "менше ніж" (<), "більше або дорівнює" (>=), "менше або дорівнює2 (<=) і операція "не дорівнює" (!= чи <>).

Первинний ключ (primary key) – стовпець або стовпці таблиці, значення котрих однозначно ідентифікують рядки таблиці.

Перетворення даних (data conversion) – операція перетворення даних від одного типу до іншого. Найчастіше виконуються перетворення між числами, текстом та датами.

Перетин (intersection) – операція теорії множин та реляційної алгебри, за допомогою якої знаходять загальні рядки двох і більше таблиць.

Підзапит (subquery) – команда **SELECT**, вкладена в речення **WHERE** або в іншу команду **SELECT**.

Підпорядкована таблиця (child table) – таблиця у зв'язку "головний-підлеглий", яка посилається на рядки головної таблиці.

Підрядок (substring) – частина рядка.

Подання (view) – альтернативний спосіб перегляду даних з однієї або декількох таблиць. Фактично це запит, що зберігається у базі даних під визначеним ім'ям.

Поле (field) – атрибут об'єкта, стовпець таблиці.

Посилальна цілісність (referential integrity) – правила, що керують цілісністю даних. Відповідно до них зовнішній ключ повинен повністю співпадати з відповідним первинним ключем або мати значення **NULL**.

Послідовність (sequence) – лічильник бази даних, який автоматично збільшується або зменшується при виборі з нього чергового значення. Послідовність може бути налаштована так, щоб видавати значення тільки з обмеженого діапазону; може також допускати циклічне повторювання по досягненні певного значення.

Похідні таблиці (derived tables) – те саме, що "подання", також відомі як "віртуальні таблиці".

Правила перевірки (validation rules) – правила, що визначають умови на значення, які можуть бути введені у певний стовпець.

Привілей (privilege) – надана деяким користувачам можливість виконувати певні дії над базою даних, які можуть бути недоступні іншим користувачам. Наприклад, одні користувачі можуть створювати нові таблиці у базі даних, а інші – ні.

Проекція (projection) – вибір стовпців, значення яких мають бути включені у результати виконання запиту.

Псевдонім таблиці (table alias) – скорочене ім'я, яке можна надати таблиці у SQL-команді і потім використовувати у межах цієї команди.

Реляційна база даних (relational database) – база даних, у якій інформація організована у вигляді зв'язаних таблиць.

Різниця (difference) – операція теорії множин та реляційної алгебри, результатом якої є рядки першої таблиці, відсутні в другій таблиці.

Роль (role) – набір привілеїв. Ролі спрощують надання однакових наборів привілеїв багатьом користувачам.

Рядок (row) – один ряд елементів таблиці.

Рядок (string) – набір з однієї або більше літер, чисел або спеціальних символів (таких, як знаки питання, зірочки тощо), інакше – текст.

Самоз'єднання (self-join) – вибірка з таблиці на основі порівняння значень з одного або декількох стовпців цієї ж таблиці.

Символьний тип даних (character datatype) – тип даних для зберігання символьної інформації – такої, як букви, числа та спеціальні символи.

Символьні функції (character functions) – функції Oracle, призначені для роботи з текстовими рядками.

Синонім (synonym) – альтернативне ім'я, що присвоюється існуючому об'єкту бази даних.

Синтаксис (syntax) – правила запису команди або мовної конструкції. Тільки команди з правильним і повним синтаксисом можуть бути успішно виконані.

Системний каталог (system catalog) – системні таблиці, що містять описи об'єктів бази даних та їх структур.

Системний привілей (system privilege) – привілей на виконання певних дій у базі даних, наприклад, підключення до бази чи створення нової таблиці.

Скалярна агрегатна функція (scalar aggregate) – агрегатна функція, що повертає з команди **SELECT** одне значення.

Складені індекси (composite indexes) – індекси, що ґрунтуються більше, ніж на одному стовпці таблиці.

Словник даних (data dictionary) – набір таблиць та подань, які автоматично створюються СКБД для зберігання поточної інформації про всі об'єкти бази даних.

Список вибору (select list) – зірочка (для завдання усіх стовпців) або список стовпців і виразів, які будуть включені у результати запиту.

Список таблиць (table list) – список таблиць, подань або обох разом після ключового слова **FROM** у команді **SELECT**.

Стовпець (column) – атрибут або характеристика об'єкта в таблиці. Також називається полем.

Стовпці з'єднання (connecting column) – стовпці, за якими виконується з'єднання таблиць. Стовпці з'єднання, що належать одній або декільком таблицям, повинні містити сумісні значення.

Схема (schema) в SQL/92 – набір об'єктів бази даних, що належать одному користувачеві. Також використовується для опису загальної структури бази даних.

Таблиці користувача (user tables) – таблиці з інформацією, що складають основу будь-якої бази даних.

Таблиця (table) – представлення даних у вигляді рядків і стовпців.

Тип даних (datatype) – тип даних, що зберігаються у базі, наприклад, числовий (NUMBER), символьний (CHAR) тощо.

Транзакція (transaction) – механізм, що забезпечує виконання цілого ряду дій у якості одного неподільного завдання.

Умова пошуку/відбору (qualifications) – умови на відбір рядків таблиць, що містяться у реченнях **WHERE** або **HAVING**.

Умовна обробка (conditional processing) – виконання різних груп операторів залежно від результату перевірки певної умови.

Унікальний індекс (unique index) – індекс, що створюється для реалізації обмеження унікальності.

Фактичні параметри (actual parameters) – реальні змінні або константи, які зазвичай копіюються у формальні параметри функції або процедури при її виклику.

Фіксована довжина (fixed length) – деякі типи даних можуть мати фіксовану або змінну довжину. Правильний вибір може впливати на розміри пам'яті, що витрачається, і на продуктивність роботи системи.

Формальні параметри (formal parameters) – список імен та типів змінних, які функція або процедура приймає при виклику. Формальні параметри можна розглядати як локальні змінні функції або процедури.

Функція (function) – частина коду програми, що вбудована у систему або написана користувачем. Функція зазвичай приймає певний набір вхідних параметрів (можуть бути відсутні) і повертає інший (вихідний) набір значень, обчислених на основі вхідних значень.

Цілісність (integrity) – несуперечність і правильність даних.

Цілісність сутності (entity integrity) – правило, що вимагає, щоб кожен рядок таблиці відрізнявся від іншого рядка, тобто мав первинний ключ, для якого недопустимі значення **NULL**.

Шаблони (wildcards) – символи, які використовуються у ключовому слові LIKE, що представляють один символ (підкреслення _) або будь-яку кількість символів (знак відсотка %).

Шлях (path) – ім'я дискового накопичувача та каталогу, де зберігається файл.

Юліанський календар (Julian dates) – система літочислення, яка базується на підрахунку кількості днів, що минули з певного початкового дня. У Oracle початковим днем вважається 1 січня 4 712 до н.е. Час доби виражається як дрібна частина дати. Наприклад, 55 123.5 означає опівдні 55123-го дня з 1 січня 4712 до н. е.

Використана література

1. Алапати С. Oracle Database 11g: руководство администратора баз данных / С. Алапати. – М. : ООО "И. Д. Вильямс", 2010. – 1440 с."
2. Андон Ф. Язык запросов SQL : учебный курс / Ф. Андон, В. Резниченко. – СПб. : Питер ; К. : BHV, 2006. – 416 с.
3. Астахова И. Ф. SQL в примерах и задачах / И. Ф. Астахова, А. П. Толстобров, И. М. Мельников. – Мн. : Новое знание, 2002. – 176 с.
4. Бази даних у питаннях і відповідях : навч. посіб. / укл. В. В. Чубук, Р. М. Чен, Л. А. Павленко та ін. – Х. : Вид. ХНЕУ, 2004. – 288 с.
5. Боуман Д. Практическое руководство по SQL / Д. Боуман, С. Эмерсон, М. Дарновски. – М. : Вильямс, 2002. – 352 с.
6. Бураков П. В. Введение в системы баз данных / П. В. Бураков, В. Ю. Петров. – СПб. : Изд. СПбГУ ИТМО, 2010. – 128 с.
7. Галузевий стандарт вищої освіти України з напряму підготовки 6.050101 "Комп'ютерні науки" // Збірник нормативних документів вищої освіти. – К. : Видавнича група BHV, 2011. – 85 с.
8. Грин Д. Oracle 8/8i Server. Энциклопедия пользователя / Д. Грин – К. : "ДиаСофт", 2000. – 576 с.
9. Гринвальд Р. Oracle 11g. Основы / Р. Гринвальд, Р. Стаковьяк, Дж. Стерн. – СПб. : СимволПлюс, 2009. – 464 с.
10. Гринвальд Р. Oracle. Справочник / Р. Гринвальд, Д. Крейнс. – СПб. : Символ-Плюс, 2005. – 976 с.
11. Грофф Д. SQL: Полное руководство / Д. Грофф, П. Вайнберг. – К. : BHV, 2001. – 816 с.
12. Гудов А. М. Введение в язык структурированных запросов SQL: учеб. пособ. / А. М. Гудов, Л. Е. Шмакова. – Кемерово : Кемеровский гос. ун-т, 2001. – 118 с.
13. Дейт Дж. Введение в системы баз данных / Дж. Дейт .– 8-е изд. – М. : Вильямс, 2005. – 1328 с.
14. Дунаев В. В. Базы данных. Язык SQL / В. В. Дунаев. – СПб. : БХВ-Петербург, 2006. – 288 с.
15. Кайт Т. Oracle для профессионалов / Т. Кайт. – СПб. : ДиаСофтЮП, 2003. – 672 с.
16. Кайт Т. Oracle для профессионалов: архитектура, методики програмирования и особенности версий 9i, 10g и 11g/ Т. Кайт. – М. : ООО "И.Д.Вильямс", 2011. – 848 с.

17. Когаловский М. Р. Энциклопедия технологий баз данных (Эволюция технологий. Технологии и стандарты. Инфраструктура. Терминология) / М. Р. Когаловский. – М. : Финансы и статистика, 2002. – 836 с.

18. Конноли Т. Базы данных: проектирование, реализация и сопровождение / Т. Конноли. – М. : Вильямс, 2000. – 1 120 с.

19. Кузнецов С. Д. Математическая модель OLAP-кубов / С. Д. Кузнецов, Ю. А. Кудрявцев // Программирование, 2009. – № 5. – с. 26–36.

20. МакДональд К. Oracle PL/SQL для профессионалов: практические решения / К. МакДональд, Х. Кац, Б. Кристофер. – СПб. : ООО "ДиаСофтЮП", 2005. – 560 с.

21. Миллсап К. Oracle. Оптимизация производительности / К. Миллсап, Д. Хольт. – СПб. : Символ-Плюс, 2006. – 464 с

22. Пушников А. Ю. Введение в системы управления базами данных. Реляционная модель данных : учеб. пособ. / А. Ю. Пушников.– Уфа : Изд. Башкирского ун-та, 1999. – 108 с.

23. Тарасов О. В. Проектування баз даних : навч. посіб. / О. В. Тарасов, В. В Федько, М. Ю.Лосєв. – Х. : Вид. ХНЕУ, 2011. – 200 с.

24. Тарасов О. В. Використання мови SQL для роботи з сучасними системами керування базами даних / О. В. Тарасов, М. Ю. Лосєв, В. В. Федько. – Х. : Вид. ХНЕУ, 2013. – 348 с.

25. Урман С. Oracle 9i. Программирование на языке PL/SQL / С. Урман. – М.: Лори, 2004. – 560 с.

26. Форта Б. Освой самостоятельно SQL / Б. Форта. – М. : Вильямс, 2005. – 288 с.

27. Хернандес М. Д. SQL-запросы для простых смертных. Практическое руководство по манипулированию данными в SQL / М. Д. Хернандес, Д. Л. Вьескас. – М. : Лори, 2003. – 460 с.

28. Earp R. Advanced SQL functions in Oracle 10g / R.W. Earp, S.S.Bagui. – Texas : – Wordware Publishing, Inc. 2006. – 398 p.

29. Ostrowski C. Oracle Application Server Portal Handbook / C. Ostrowski. – N.Y. : – The McGraw-Hill Companies, Inc. 2007. – 600 p.

30. Powell G. Oracle Data Warehouse Tuning for 10g / G. Powell. – Elsevier Digital Press, 2005. – 468 p.

31. Администрирование баз данных Oracle. Представления словаря данных [Электронный ресурс]. – Режим доступа : http://interway.ucoz.ru/publ/Oracle/administrirovanie_baz_dannykh_Oracle_predstavlenija_slovarja_dannykh/4-1-0-155.

32. Администрирование баз данных Oracle. Управление пользователями базы данных [Электронный ресурс]. – Режим доступа :

http://interway.ucoz.ru/publ/Oracle/administrirovanie_baz_dannykh_Oracle_upravlenie_polzovateljami_bazy_dannykh/4-1-0-157.

33. Альперович М. Введение в OLAP и многомерные базы данных [Электронный ресурс] / М. Альперович. – Режим доступа : http://www.olap.ru/basic/alpero2i.asp.

34. Власов А.И. Краткое практическое руководство разработчика информационных систем на базе СКБД Oracle. [Электронный ресурс] / А. И. Власов. – Режим доступа : http://citforum.ru/database/Oraclepr /index.shtml.

35. Встроенные функции Oracle [Электронный ресурс]. – Режим доступа : http://www.quizful.net/post/Oracle-inline-functions.

36. Генник Д. Взаимодействие с удаленными базами данных. [Электронный ресурс] / Д. Генник. – Режим доступа : http://baks.gaz.ru/oradoc /ora/ora056.htm.

37. Грубер М. Понимание SQL (Understanding SQL) [Электронный ресурс] / М. Грубер. – Режим доступа : http://www.sql.ru/docs/sql/u_sql.

38. Гудов А. М. Введение в язык структурированных запросов SQL [Электронный ресурс] / А. М. Гудов. – Режим доступа : http://www.math. kemsu.ru/library/sql.

39. Деревянко А. С. Язык SQL в диалектах Oracle и IBM DB2 [Электронный ресурс] / А. С. Деревянко. – Режим доступа : http://khpi-iip.mipk.kharkiv.edu/ library/extent/dbms/sql/index.html.

40. Косембаев Р. СКБД Oracle [Электронный ресурс] / Р. Косембаев. – Режим доступа : http://bourabai.kz/dbt/servers/Oracle.htm.

41. Кравчук В. С. Утилита SQL*Plus. Создание и выполнение сценариев [Электронный ресурс] / В. Кравчук. – Режим доступа : http://citforum.ru/database/Oracle/sqlplus/.

42. Кузнецов С. Д. Стандарты языка реляционных баз данных SQL: краткий обзор [Электронный ресурс] / С. Д. Кузнецов. – Режим доступа: http://citforum.ru/ database/articles/art_2.shtml.

43. Лекции по дисциплине "Администрирование баз данных и приложений" [Электронный ресурс]. – Режим доступа : http://www.opennet.ru /docs/RUS/db_admin.

44. Некоторые примеры нестандартных возможностей синтаксиса SQL. Часть вторая: форматы дат [Электронный ресурс]. – Режим доступа : http://habrahabr.ru/post/128682.

45. Обзор табличных пространств [Электронный ресурс]. – Режим доступа: http://Oracle-dba.ru/database/basics/Oracle_database_tablespaces /how_to_create_new_tablespace/.

46. Оперативная аналитическая обработка информации (OLAP) [Электронный ресурс]. – Режим доступа : http://bourabai.kz/tpoi/olap.htm.

47. Паламаренко А. С. Информационная безопасность в системах управления базами данных [Электронный ресурс] / А. С. Паламаренко, А. А. Мартыненко. – Режим доступа : http://ir.nmu.org.ua/bitstream/handle /123456789/1838/IS%20in%20BD.pdf?sequence=1.

48. Пользователи в Oracle: Управление профилями [Электронный ресурс]. – Режим доступа : http://www.all-Oracle.ru/content/view/ ?part=1&id=90.

49. Пржиялковский В. Как организовать двойную парольную защиту данных в Oracle [Электронный ресурс] / В. Пржиялковский. – Режим доступа : http://www.realcoding.net/article/view/2734.

50. Робоча програма навчальної дисципліни "Організація баз даних та знань" для студентів напряму підготовки 6.050101 "Комп'ютерні науки" всіх форм навчання / укл. О. В. Тарасов, В. В. Федько, М. Ю. Лосєв; – Х. : ХНЕУ ім. С. Кузнеця, 2014. – 105 с.; [Електронний ресурс]. – Режим доступу : http://www.repository.hneu.edu.ua/jspui/handle/123456789/6615.

51. Сайт персональних навчальних систем ХНЕУ ім. С. Кузнеця [Електронний ресурс]. – Режим доступу : http://www.ikt.hneu.edu.ua.

52. Создание структуры интеллектуального анализа данных OLAP [Электронный ресурс]. – Режим доступа : http://msdn.microsoft.com/ru-ru/library/hh230818.aspx.

53. Суррогатный_ключ [Электронный ресурс]. – Режим доступа : http://access.avorut.ru/publ/bazy_dannykh_osnovnye_ponjatija/s/surrogatnyj_kljuch/46-1-0-35.

54. Суррогатный_ключ [Электронный ресурс]. – Режим доступа : http://ru.wikipedia.org/wiki/Суррогатный_ключ.

55. Типы данных Oracle [Электронный ресурс]. – Режим доступа : http://sd-company.su/article/sql/data_type_Oracle.

56. Управление пользователями и безопасностью [Электронный ресурс]. – Режим доступа : http://www.oranet.ru/OraDoc10gXE/admin.102 /b25107/users_secure.htm.

57. Федоров А. Введение в базы данных. Часть 6. Введение в язык SQL [Электронный ресурс] / А. Федоров, Н. Елманова – Режим доступа : http://www.compress.ru/article.aspx?id=11944&iid=463#06/.

58. Фернстайн С. Функции ROLLUP и CUBE в предложении SELECT [Электронный ресурс] / С. Фернстайн. – Режим доступа : http://www.olap.ru/desc/Oracle/rollup_and_cube.asp.

59. Эквивалентные типы данных ANSI SQL [Электронный ресурс]. – Режим доступа : http://office.microsoft.com/ru-ru/access-help/HP001032 229.aspx.

60. Язык запросов SQL. Типы данных SQL [Электронный ресурс]. – Режим доступа : http://sql-language.ru/sqldatetype.html.

61. Oracle 7. Server Application Developer's Guide. Руководство разработчика приложений [Электронный ресурс]. – Режим доступа : http://www.interface.ru/fset.asp?Url=/Oracle/ora7/ora7a00.htm&anchor=1.

62. SQL для Oracle. SQL-операторы по сравнению с командами SQL*Plus [Электронный ресурс]. – Режим доступа : http://sql-Oracle.ru/sql-operatory-po-sravneniyu-s-komandami-sql-plus.html.

63. SQL для Oracle. Взаимодействие SQL и SQL*Plus [Электронный ресурс]. – Режим доступа : http://sql-Oracle.ru/vzaimodejstvie-sql-i-sql-plus.html.

64. SQL для Oracle. Управление пользовательским доступом [Электронный ресурс]. – Режим доступа : http://sql-Oracle.ru/upravlenie-polzovatelskim-dostupom.

65. Difference between inner join and equi join and natural join [Electronic resource]. – Access mode : http://www.dotnet-tricks.com/Tutorial /sqlserver/1VEW230213-Difference-between-inner-join-and-equi-join-and-natural-join.html.

66. Natural join operation [Electronic resource]. – Access mode : http://docs.Oracle.com/javadb/10.6.2.1/ref/rrefsqljnaturaljoin.html.

67. Oracle Database Express Edition 11g Release 2 [Electronic resource]. – Access mode : http://www.Oracle.com/technetwork/database /database-technologies/express-edition/downloads/index.html.

68. Oracle Database. Overview [Electronic resource]. – Access mode : http://www.Oracle.com/ru/products/database/overview/index.html.

69. Oracle Help Center. Database Error Messages [Electronic resource]. – Access mode : http://docs.Oracle.com/cd/B28359_01/server.111/b28278/toc.htm.

70. Oracle Help Center. Database PL/SQL User's Guide and Reference [Electronic resource]. – Access mode : http://docs.Oracle.com/cd/B19306_01 /appdev.102/b14261/toc.htm.

71. Oracle Help Center. Database SQL Language Reference [Electronic resource]. – Access mode : https://docs.Oracle.com/cd/B28359_01/server.111 /b28286/toc.htm.

72. Oracle Help Center. Database SQL Reference [Electronic resource]. – Access mode : http://docs.Oracle.com/cd/B19306_01/server.102/b14200 /toc.htm.

73. Oracle Help Center. SQL*Plus® User's Guide and Reference [Electronic resource]. – Access mode : http://docs.Oracle.com/cd/B28359_01 /server.111/b31189/ch2.htm#i1132910.

74. Oracle Help Center.Database Data Warehousing Guide. [Electronic resource]. – Access mode : http://docs.Oracle.com/cd/B28359_01/server.111 /b28313/aggreg.htm.

75. Oracle Help Center. Database SQL Reference. Create database link [Electronic resource]. – Access mode : http://docs.Oracle.com/cd/ B19306_01 /server.102/b14200/statements_5005.htm.

76. Oracle history.Interactive Timeline. [Electronic resource]. – Access mode : http://Oracle.com.edgesuite.net/timeline/Oracle.

77. Oracle SQL Developer. Downloads [Electronic resource]. – Access mode : http://www.Oracle.com/technetwork/developer-tools/sql-developer /downloads/index.html.

78. Oracle® Database SQL Language Reference. Alter table [Electronic resource]. – Access mode : http://docs.Oracle.com/html/E26088_01 /statements_3001.htm.

79. Oracle® Database SQL Language Reference. Create Role [Electronic resource]. – Access mode : http://docs.Oracle.com/html /E26088_01/statements_6012.htm#i2066772.

80. Oracle® Database SQL Language Reference. Create sequence [Electronic resource]. – Access mode : http://docs.Oracle.com/html /E26088_01/statements_6015.htm.

81. Oracle® Database SQL Language Reference. Create table [Electronic resource]. – Access mode : http://Oracle.su/docs/11g/server.112 /e10592/statements_7002.htm.

82. Oracle® Database SQL Language Reference. Create User. [Electronic resource]. – Access mode : http://docs.Oracle.com/html/E26088_01 /statements_8003.htm.

83. SQL*Plus User's Guide and Reference. Defining Page and Report Titles and Dimensions [Electronic resource]. – Access mode : http://cs.mipt.ru/docs/comp/eng/develop/databases/Oracle/8i/doc/server.815 /a66736/ch43.htm.

84. SQL*Plus User's Guide and Reference. SQL*Plus Command Reference [Electronic resource]. – Access mode : http://docs.Oracle.com /cd/B10501_01/server.920/a90842/ch13.htm.

85. TimesTen In-Memory Database System Tables and Views Reference [Electronic resource]. – Access mode : https://docs.Oracle.com /database/121/TTSYS/systemtables.htm#TTSYS346.

Додатки

Додаток А

Вміст стандартного скрипта SCOTT.SQL корпорації Oracle

```
    SET TERMOUT OFF
    SET ECHO OFF
    GRANT CONNECT,RESOURCE,UNLIMITED TABLESPACE TO SCOTT
IDENTIFIED BY TIGER;
    ALTER USER SCOTT DEFAULT TABLESPACE USERS;
    ALTER USER SCOTT TEMPORARY TABLESPACE TEMP;
    CONNECT SCOTT/TIGER
    DROP TABLE DEPT;
    CREATE TABLE DEPT (DEPTNO NUMBER(2)CONSTRAINT PK_DEPT
PRIMARY KEY, DNAME VARCHAR2(14),  LOC VARCHAR2(13));

    DROP TABLE EMP;
    CREATE TABLE EMP (EMPNO NUMBER(4) CONSTRAINT PK_EMP
PRIMARY KEY, ENAME VARCHAR2(10), JOB VARCHAR2(9),
MGR NUMBER(4), HIREDATE DATE, SAL NUMBER(7,2),
COMM NUMBER(7,2), DEPTNO NUMBER(2) CONSTRAINT FK_DEPTNO
REFERENCES DEPT);
    INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');
    INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
    INSERT INTO DEPT VALUES (30,'SALES','CHICAGO');
    INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');

    INSERT INTO EMP VALUES (7369,'SMITH','CLERK',7902,
to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
    INSERT INTO EMP VALUES (7499,'ALLEN','SALESMAN',7698,
to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
    INSERT INTO EMP VALUES (7521,'WARD','SALESMAN',7698,
to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
    INSERT INTO EMP VALUES (7566,'JONES','MANAGER',7839,
to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);
```

```
    INSERT INTO EMP VALUES
(7654,'MARTIN','SALESMAN',7698, to_date('28-9-1981','dd-
mm-yyyy'),1250,1400,30);
    INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER',7839,
to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);
    INSERT INTO EMP VALUES (7782,'CLARK','MANAGER',7839,
to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);
    INSERT INTO EMP VALUES (7788,'SCOTT','ANALYST',7566,
to_date('13-JUL-87')-85,3000,NULL,20);
    INSERT INTO EMP VALUES (7839,'KING','PRESIDENT',NULL,
to_date('17-11-1981','dd-mm-yyyy'),5000,NULL,10);
    INSERT INTO EMP VALUES 7844,'TURNER','SALESMAN',7698,
to_date('8-9-1981','dd-mm-yyyy'),1500,0,30);
    INSERT INTO EMP VALUES (7876,'ADAMS','CLERK',7788,
to_date('13-JUL-87')-51,1100,NULL,20);
    INSERT INTO EMP VALUES (7900,'JAMES','CLERK',7698,
to_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);
    INSERT INTO EMP VALUES (7902,'FORD','ANALYST',7566,
to_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);
    INSERT INTO EMP VALUES (7934,'MILLER','CLERK',7782,
to_date('23-1-1982','dd-mm-yyyy'),1300,NULL,10);

    DROP TABLE BONUS;
    CREATE TABLE BONUS ( ENAME VARCHAR2(10),
JOB VARCHAR2(9), SAL NUMBER,  COMM NUMBER);
    DROP TABLE SALGRADE;
    CREATE TABLE SALGRADE (GRADE NUMBER,
LOSAL NUMBER, HISAL NUMBER);
    INSERT INTO SALGRADE VALUES (1,700,1200);
    INSERT INTO SALGRADE VALUES (2,1201,1400);
    INSERT INTO SALGRADE VALUES (3,1401,2000);
    INSERT INTO SALGRADE VALUES (4,2001,3000);
    INSERT INTO SALGRADE VALUES (5,3001,9999);
    COMMIT;
    SET TERMOUT ON
    SET ECHO ON
```

Додаток Б

Загальний перелік команд SQL*PLUS

Таблиця Б.1

Команди SQL*Plus та їх призначення

Команда	Опис
@	Виконує вказаний командний файл
/	Виконує поточну команду SQL або блок PL/SQL з буфера SQL
ACCEPT	Читає рядок з пристрою введення та запам'ятовує його у зазначеній змінній користувача
APPEND	Додає вказаний текст у кінець поточного рядка буфера
ARCHIVE LOG	Управляє режимами архівування оперативного логічного журналу
ATTRIBUTE	Визначає характеристики відображення для даного атрибута об'єктного типу колонки, наприклад, формат даних
BREAK	Визначає, де і як буде виконуватися форматування у звіті, або виводить поточні визначення розривів у звітах
BTITLE	Поміщає та форматує вказаний заголовок у кінці кожної сторінки звіту або виводить поточний опис BTITLE
CHANGE	Змінює текст поточного рядка буфера
CLEAR	Очищує поточне значення або установки для зазначених опцій – таких, як BREAKS або COLUMNS
COLUMN	Задає атрибути виводу для зазначеного стовпця або показує поточні установки виводу для одного або усіх стовпців
COMPUTE	Обчислює та друкує підсумкові рядки за групами обраних записів або показує всі описи COMPUTE
CONNECT	Підключення зазначеного користувача до ORACLE
COPY	Копіює дані з запиту в таблицю локальної або віддаленої БД
DEFINE	Визначає змінну користувача та привласнює їй CHAR-значення або показує значення та тип однієї або усіх змінних
DEL	Видаляє поточний рядок з буфера
DESCRIBE	Показує описи колонок для зазначеної таблиці, подання, синоніма
DISCONNECT	Виконує відкладені зміни в БД і відключає поточного користувача від ORACLE, але не виходить з SQL * PLUS
EDIT	Завантажує текстовий редактор ОС із зазначеним файлом або вмістом буфера SQL
EXECUTE	Виконує один оператор PL/SQL. Найчастіше, це виклик збереженої процедури або функції
EXIT	Виконує всі відкладені зміни в БД, завершує SQL * PLUS, і повертає управління ОС
GET	Завантажує файл ОС у буфер
HELP	Допомога по командам SQL * PLUS

Закінчення табл. Б.1

Команда	Опис
HOST	Виконує команду операційної системи без виходу з SQL * PLUS
INPUT	Додає один або декілька нових рядків після поточного рядка у буфер
LIST	Виводить одну або декілька рядків буфера
PASSWORD	Дозволяє змінити пароль користувача без відображення на екрані
PAUSE	Виводить порожній рядок і рядок з вказаним текстом, потім очікує натискання користувачем [Enter]
PRINT	Видає на екран значення зв'язаних змінних
PROMPT	Посилає вказане повідомлення або порожній рядок на екран
QUIT	Виконує усі відкладені зміни у БД, завершує SQL * PLUS, і повертає управління ОС
RECOVER	Команда відновлення БД, якщо база даних працювала у режимі ARCHIVELOG
REMARK	Починає коментар у командному файлі
REPFOOTER	Задає текст, що видається у кінці кожного звіту
REPHEADER	Задає текст, що видається на початку кожного звіту
RUN	Виводить і виконує поточну команду SQL або блок PL/SQL із буфера SQL
RUNFORM	Викликає додаток SQL * FORMS із SQL * PLUS
SAVE	Зберігає вміст буфера у файлі ОС
SET	Встановлює параметри оточення SQL * PLUS для поточного сеансу
SHOW	Виводить значення системних змінних SQL * PLUS
SHUTDOWN	Завершує роботу поточного екземпляра Oracle
SPOOL	Зберігає результати поточного сеансу у файлі ОС. Також виводить поточний статус буферизації
SQLPLUS	Запускає SQL * PLUS із ОС
START	Виконує вказаний командний файл
STARTUP	Запускає базу даних із SQL * Plus
STORE	Записує значення змінних середовища SQL*Plus у командний файл базової операційної системи
TIMING	Записує дані про загальний час виконання, виводить поточний заголовок часової області або виводить кількість активних часових областей
TTITLE	Поміщає і форматує вказаний заголовок на початку кожної сторінки звіту або виводить поточний опис TTITLE
UNDEFINE	Видаляє вказану змінну користувача
VARIABLE	Створює зв'язані змінні
WHENEVER SQLERROR	Повертає код помилки ORACLE
XQUERY	Виконує запит на мові XQUERY до бази даних (версії 10g і старше)

Вміст скрипта SCOTT_XE2.SQL навчальної бази даних

```
-- SCRIPT FOR TRAINING DATABASE
SET TERMOUT ON
SET ECHO OFF

REM ************ Table DEPT **********************************
DROP TABLE DEPT CASCADE CONSTRAINTS;
CREATE TABLE DEPT
      (DEPTNO NUMBER(2)  PRIMARY KEY
                  CHECK (MOD(DEPTNO,10)=0 AND DEPTNO > 0),
         DNAME VARCHAR2(14) UNIQUE,
         LOC VARCHAR2(13) NOT NULL) ;

INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK');
INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS');
INSERT INTO DEPT VALUES (30,'SALES','CHICAGO');
INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON');
INSERT INTO DEPT VALUES (50,'TEACHING','BOSTON');
INSERT INTO DEPT VALUES (60,'FINANCE','NEW YORK');
INSERT INTO DEPT VALUES (70,'CONTROL','ATLANTA');
-- Rows with error
INSERT INTO DEPT VALUES (80,'CONTROL','ATLANTA');
INSERT INTO DEPT VALUES (85,'SECURITY','ATLANTA');
COMMIT;

REM ************ Table EMP **********************************
DROP TABLE EMP CASCADE CONSTRAINTS;
CREATE TABLE EMP
      (EMPNO NUMBER(4) CONSTRAINT PK_EMP PRIMARY KEY,
         ENAME VARCHAR2(10),
         JOB VARCHAR2(10) DEFAULT 'CLERK',
         MGR NUMBER(4),
         HIREDATE DATE DEFAULT SYSDATE,
         SAL NUMBER(7,2) NOT NULL CHECK (SAL >= 500 AND SAL <= 6000),
         COMM NUMBER(7,2),
         DEPTNO NUMBER(2) CONSTRAINT FK_DEPTNO REFERENCES DEPT,
       CONSTRAINT CH_COMM CHECK (COMM<= 2*SAL));

INSERT INTO EMP VALUES
(7369,'SMITH','CLERK',7902,to_date('17-12-1980','dd-mm-yyyy'),800,NULL,20);
```

```
INSERT INTO EMP VALUES
(7499,'ALLEN','SALESMAN',7698,to_date('20-2-1981','dd-mm-yyyy'),1600,300,30);
INSERT INTO EMP VALUES
(7521,'WARD','SALESMAN',7698,to_date('22-2-1981','dd-mm-yyyy'),1250,500,30);
INSERT INTO EMP VALUES
(7566,'JONES','MANAGER',7839,to_date('2-4-1981','dd-mm-yyyy'),2975,NULL,20);
INSERT INTO EMP VALUES
(7654,'MARTIN','SALESMAN',7698,to_date('28-9-1981','dd-mm-
yyyy'),1250,1400,30);
INSERT INTO EMP VALUES
(7698,'BLAKE','MANAGER',7839,to_date('1-5-1981','dd-mm-yyyy'),2850,NULL,30);
INSERT INTO EMP VALUES
(7782,'CLARK','MANAGER',7839,to_date('9-6-1981','dd-mm-yyyy'),2450,NULL,10);
INSERT INTO EMP VALUES
(7788,'SCOTT','ANALYST',7566,to_date('13-7-87','dd-mm-yyyy')-
85,3000,NULL,20);
INSERT INTO EMP VALUES
(7839,'KING','PRESIDENT',NULL,to_date('17-11-1981','dd-mm-
yyyy'),5000,NULL,10);
INSERT INTO EMP VALUES
(7844,'TURNER','SALESMAN',7698,to_date('8-9-1981','dd-mm-yyyy'),1500,0,30);
INSERT INTO EMP VALUES
(7876,'ADAMS','CLERK',7788,to_date('13-7-87','dd-mm-yyyy')-51,1100,NULL,20);
INSERT INTO EMP VALUES
(7900,'JAMES','CLERK',7698,to_date('3-12-1981','dd-mm-yyyy'),950,NULL,30);
INSERT INTO EMP VALUES
(7902,'FORD','ANALYST',7566,to_date('3-12-1981','dd-mm-yyyy'),3000,NULL,20);
INSERT INTO EMP VALUES
(7934,'MILLER','CLERK',7782,to_date('23-1-1982','dd-mm-yyyy'),1250,NULL,10);
INSERT INTO EMP VALUES
(7401,'SMITH','CLERK',7782,to_date('17-10-1983','dd-mm-yyyy'),1000,NULL,10);
INSERT INTO EMP VALUES
(7402,'MARTIN','PROGRAMMER',7566,to_date('01-05-1985','dd-mm-
yyyy'),2000,800,20);
INSERT INTO EMP VALUES
(7415,'DAVE','SALESMAN',7698,to_date('11-11-1990','dd-mm-
yyyy'),1300,NULL,30);
INSERT INTO EMP VALUES
(7501,'MARTIN','MANAGER',7839,to_date('01-07-1982','dd-mm-
yyyy'),3000,200,50);
INSERT INTO EMP VALUES
```

```
        (7502,'MARTIN','TEACHER',7501,to_date('10-07-1983','dd-mm-
yyyy'),2500,NULL,50);
        INSERT INTO EMP VALUES
        (7701,'CLARK','MANAGER',7839,to_date('09-06-1981','dd-mm-yyyy'),2450,200,70);
        INSERT INTO EMP VALUES
        (7712,'BILL','ENGINEER',7701,to_date('09-06-1986','dd-mm-yyyy'),2000,150,70);
        INSERT INTO EMP VALUES
        (7713,'BLAKE','ENGINEER',7701,to_date('19-06-1989','dd-mm-
yyyy'),1600,NULL,70);

        -- Rows with error
        INSERT INTO EMP VALUES
        (8001,'GOODMAN','ENGINEER',7701,to_date('19-06-1992','dd-mm-
yyyy'),NULL,NULL,70);
        INSERT INTO EMP VALUES
        (8002,'BOLT','ENGINEER',7701,to_date('15-07-1992','dd-mm-yyyy'),1000,2100,70);
        INSERT INTO EMP VALUES
        (8003,'SANDERS','ENGINEER',7701,to_date('19-07-1989','dd-mm-
yyyy'),1600,NULL,90);
        COMMIT;
        REM ************ Table SALGRADE ***********************************
        DROP TABLE SALGRADE;
        CREATE TABLE SALGRADE
            (GRADE NUMBER NOT NULL,
               LOSAL NUMBER  NOT NULL,
               HISAL NUMBER  NOT NULL);
        INSERT INTO SALGRADE VALUES (1,700,1200);
        INSERT INTO SALGRADE VALUES (2,1201,1400);
        INSERT INTO SALGRADE VALUES (3,1401,2000);
        INSERT INTO SALGRADE VALUES (4,2001,3000);
        INSERT INTO SALGRADE VALUES (5,3001,9999);
        COMMIT;
        REM ************ Table BONUS ***********************************
        DROP TABLE BONUS;
        CREATE TABLE BONUS
                (
                ENAME VARCHAR2(10),
                JOB VARCHAR2(10),
                SAL NUMBER,
                COMM NUMBER
                );
```

```
REM ************ Table VACATION **********************************
DROP SEQUENCE NUM_VACATION;
CREATE SEQUENCE NUM_VACATION;
DROP TABLE VACATION;
CREATE TABLE VACATION
        (ORDNUMBER      NUMBER PRIMARY KEY,
         NORDER         VARCHAR2(10) NOT NULL,
         EMPNO                  NUMBER(4) NOT NULL,
         HDATE_BEGIN    DATE NOT NULL,
         HDATE_END      DATE NOT NULL,
        CONSTRAINT     FK_EMPNO FOREIGN KEY (EMPNO)
                       REFERENCES EMP(EMPNO)
                       ON DELETE CASCADE,
        CONSTRAINT     VK_UN  UNIQUE (EMPNO, HDATE_BEGIN),
        CONSTRAINT     DT_CHK
                       CHECK (HDATE_BEGIN < HDATE_END));

-- Row with error
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',
NULL, to_date('02.02.2010', 'dd.mm.yyyy'), to_date(' 23.02.2010', 'dd.mm.yyyy'));
-- -------------

 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7369,
to_date('02.02.2010', 'dd.mm.yyyy'), to_date(' 23.02.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7401,
to_date('09.02.2010', 'dd.mm.yyyy'), to_date(' 02.03.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7415,
to_date('01.03.2010', 'dd.mm.yyyy'), to_date(' 22.03.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7499,
to_date('16.03.2010', 'dd.mm.yyyy'), to_date(' 06.04.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7501,
to_date('10.04.2010', 'dd.mm.yyyy'), to_date(' 28.04.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7502,
to_date('30.04.2010', 'dd.mm.yyyy'), to_date(' 18.05.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7521,
to_date('10.05.2010', 'dd.mm.yyyy'), to_date(' 31.05.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-1',  7566,
to_date('09.06.2010', 'dd.mm.yyyy'), to_date(' 30.06.2010', 'dd.mm.yyyy'));
 INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7654,
to_date('02.07.2010', 'dd.mm.yyyy'), to_date(' 20.07.2010', 'dd.mm.yyyy'));
```

```
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7698,
to_date('23.07.2010', 'dd.mm.yyyy'), to_date(' 10.08.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7701,
to_date('09.08.2010', 'dd.mm.yyyy'), to_date(' 30.08.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7712,
to_date('21.08.2010', 'dd.mm.yyyy'), to_date(' 14.09.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7713,
to_date('27.08.2010', 'dd.mm.yyyy'), to_date(' 20.09.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7782,
to_date('05.09.2010', 'dd.mm.yyyy'), to_date(' 23.09.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7839,
to_date('10.09.2010', 'dd.mm.yyyy'), to_date(' 04.10.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7844,
to_date('14.09.2010', 'dd.mm.yyyy'), to_date(' 05.10.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7900,
to_date('24.09.2010', 'dd.mm.yyyy'), to_date(' 15.10.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7902,
to_date('05.10.2010', 'dd.mm.yyyy'), to_date(' 26.10.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '10H-2',  7934,
to_date('06.10.2010', 'dd.mm.yyyy'), to_date(' 27.10.2010', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-1',  7782,
to_date('28.02.2011', 'dd.mm.yyyy'), to_date(' 21.03.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-1',  7839,
to_date('03.03.2011', 'dd.mm.yyyy'), to_date(' 30.03.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-1',  7844,
to_date('08.03.2011', 'dd.mm.yyyy'), to_date(' 29.03.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-1',  7900,
to_date('18.03.2011', 'dd.mm.yyyy'), to_date(' 05.04.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-1',  7902,
to_date('02.04.2011', 'dd.mm.yyyy'), to_date(' 20.04.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-1',  7934,
to_date('17.04.2011', 'dd.mm.yyyy'), to_date(' 05.05.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7369,
to_date('01.06.2011', 'dd.mm.yyyy'), to_date(' 22.06.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7713,
to_date('18.06.2011', 'dd.mm.yyyy'), to_date(' 28.06.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7401,
to_date('22.06.2011', 'dd.mm.yyyy'), to_date(' 10.07.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7415,
to_date('09.07.2011', 'dd.mm.yyyy'), to_date(' 27.07.2011', 'dd.mm.yyyy'));
```

```
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7499,
to_date('21.07.2011', 'dd.mm.yyyy'), to_date(' 11.08.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7501,
to_date('27.07.2011', 'dd.mm.yyyy'), to_date(' 17.08.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7502,
to_date('05.08.2011', 'dd.mm.yyyy'), to_date(' 23.08.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7521,
to_date('10.08.2011', 'dd.mm.yyyy'), to_date(' 28.08.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7566,
to_date('14.08.2011', 'dd.mm.yyyy'), to_date(' 04.09.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7654,
to_date('24.08.2011', 'dd.mm.yyyy'), to_date(' 17.09.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7698,
to_date('04.09.2011', 'dd.mm.yyyy'), to_date(' 28.09.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7701,
to_date('05.09.2011', 'dd.mm.yyyy'), to_date(' 23.09.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-3',  7713,
to_date('18.09.2011', 'dd.mm.yyyy'), to_date(' 29.09.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '11H-2',  7712,
to_date('26.09.2011', 'dd.mm.yyyy'), to_date(' 14.10.2011', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7712,
to_date('28.03.2012', 'dd.mm.yyyy'), to_date(' 15.04.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7713,
to_date('07.04.2012', 'dd.mm.yyyy'), to_date(' 28.04.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7782,
to_date('17.04.2012', 'dd.mm.yyyy'), to_date(' 11.05.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7839,
to_date('02.05.2012', 'dd.mm.yyyy'), to_date(' 26.05.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7844,
to_date('12.05.2012', 'dd.mm.yyyy'), to_date(' 30.05.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7900,
to_date('23.05.2012', 'dd.mm.yyyy'), to_date(' 10.06.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7902,
to_date('04.06.2012', 'dd.mm.yyyy'), to_date(' 25.06.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-1',  7934,
to_date('18.06.2012', 'dd.mm.yyyy'), to_date(' 09.07.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7369,
to_date('05.07.2012', 'dd.mm.yyyy'), to_date(' 26.07.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7401,
to_date('10.07.2012', 'dd.mm.yyyy'), to_date(' 03.08.2012', 'dd.mm.yyyy'));
```

```
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7415,
to_date('17.07.2012', 'dd.mm.yyyy'), to_date(' 10.08.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7499,
to_date('22.07.2012', 'dd.mm.yyyy'), to_date(' 09.08.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7501,
to_date('29.07.2012', 'dd.mm.yyyy'), to_date(' 16.08.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7502,
to_date('05.08.2012', 'dd.mm.yyyy'), to_date(' 26.08.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7521,
to_date('20.08.2012', 'dd.mm.yyyy'), to_date(' 10.09.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7566,
to_date('25.08.2012', 'dd.mm.yyyy'), to_date(' 15.09.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7698,
to_date('09.09.2012', 'dd.mm.yyyy'), to_date(' 27.09.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '12H-2',  7701,
to_date('24.09.2012', 'dd.mm.yyyy'), to_date(' 12.10.2012', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-1',  7701,
to_date('12.04.2013', 'dd.mm.yyyy'), to_date(' 03.05.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-1',  7782,
to_date('13.05.2013', 'dd.mm.yyyy'), to_date(' 03.06.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-1',  7839,
to_date('24.05.2013', 'dd.mm.yyyy'), to_date(' 17.06.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-1',  7844,
to_date('03.06.2013', 'dd.mm.yyyy'), to_date(' 21.06.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-1',  7900,
to_date('14.06.2013', 'dd.mm.yyyy'), to_date(' 05.07.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-1',  7902,
to_date('24.06.2013', 'dd.mm.yyyy'), to_date(' 06.07.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7934,
to_date('05.07.2013', 'dd.mm.yyyy'), to_date(' 23.07.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7369,
to_date('17.08.2013', 'dd.mm.yyyy'), to_date(' 04.09.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7401,
to_date('27.08.2013', 'dd.mm.yyyy'), to_date(' 17.09.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7415,
to_date('06.09.2013', 'dd.mm.yyyy'), to_date(' 30.09.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7499,
to_date('16.09.2013', 'dd.mm.yyyy'), to_date(' 04.10.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7501,
to_date('27.09.2013', 'dd.mm.yyyy'), to_date(' 18.10.2013', 'dd.mm.yyyy'));
```

```
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7502,
to_date('08.10.2013', 'dd.mm.yyyy'), to_date(' 01.11.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7521,
to_date('19.10.2013', 'dd.mm.yyyy'), to_date(' 06.11.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-3',  7902,
to_date('24.10.2013', 'dd.mm.yyyy'), to_date(' 05.11.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7566,
to_date('29.10.2013', 'dd.mm.yyyy'), to_date(' 19.11.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7654,
to_date('08.11.2013', 'dd.mm.yyyy'), to_date(' 22.12.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '13H-2',  7698,
to_date('18.11.2013', 'dd.mm.yyyy'), to_date(' 06.12.2013', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-1',  7566,
to_date('10.03.2014', 'dd.mm.yyyy'), to_date(' 28.03.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-1',  7654,
to_date('20.03.2014', 'dd.mm.yyyy'), to_date(' 10.04.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-1',  7698,
to_date('04.04.2014', 'dd.mm.yyyy'), to_date(' 16.04.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-1',  7701,
to_date('14.04.2014', 'dd.mm.yyyy'), to_date(' 02.05.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-1',  7712,
to_date('29.04.2014', 'dd.mm.yyyy'), to_date(' 28.05.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-1',  7713,
to_date('09.05.2014', 'dd.mm.yyyy'), to_date(' 12.06.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-1',  7782,
to_date('24.05.2014', 'dd.mm.yyyy'), to_date(' 11.06.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-2',  7839,
to_date('03.06.2014', 'dd.mm.yyyy'), to_date(' 30.06.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-2',  7844,
to_date('18.06.2014', 'dd.mm.yyyy'), to_date(' 12.07.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-2',  7900,
to_date('28.06.2014', 'dd.mm.yyyy'), to_date(' 16.07.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-2',  7902,
to_date('13.07.2014', 'dd.mm.yyyy'), to_date(' 03.08.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-2',  7934,
to_date('23.07.2014', 'dd.mm.yyyy'), to_date(' 13.08.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-3',  7369,
to_date('15.09.2014', 'dd.mm.yyyy'), to_date(' 09.10.2014', 'dd.mm.yyyy'));
INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-3',  7401,
to_date('22.09.2014', 'dd.mm.yyyy'), to_date(' 10.10.2014', 'dd.mm.yyyy'));
```

```
        INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-3',  7415,
to_date('02.10.2014', 'dd.mm.yyyy'), to_date(' 23.10.2014', 'dd.mm.yyyy'));
        INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-3',  7499,
to_date('09.10.2014', 'dd.mm.yyyy'), to_date(' 02.11.2014', 'dd.mm.yyyy'));
        INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-3',  7501,
to_date('19.10.2014', 'dd.mm.yyyy'), to_date(' 06.11.2014', 'dd.mm.yyyy'));
        INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-3',  7502,
to_date('26.10.2014', 'dd.mm.yyyy'), to_date(' 16.11.2014', 'dd.mm.yyyy'));
        INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-4',  7698,
to_date('04.11.2014', 'dd.mm.yyyy'), to_date(' 16.11.2014', 'dd.mm.yyyy'));
        INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-3',  7521,
to_date('05.11.2014', 'dd.mm.yyyy'), to_date(' 29.11.2014', 'dd.mm.yyyy'));
        INSERT INTO VACATION  VALUES (NUM_VACATION.NEXTVAL, '14H-5',  7712,
to_date('29.11.2014', 'dd.mm.yyyy'), to_date(' 09.12.2014', 'dd.mm.yyyy'));

        COMMIT;

        SET TERMOUT ON
        SET ECHO ON
```

Призначення додаткових параметрів у команді Create Table

Необов'язкові параметри команди CREATE TABLE, а саме: PCTFREE, PCTUSED, INITRANS, MAXTRANS, TABLESPACE, STORAGE, RECOVERABLE та UNRECOVERABLE – характеризують простір, що розподіляється системою при роботі з таблицею.

• PCTFREE визначає відсоток простору блоку, який резервується для модифікації таблиці. Значення за замовчуванням дорівнює 10. Допустимі значення від 1 до 99.

• PCTUSED визначає мінімальний відсоток використання простору блоку, при якому в нього вводяться дані. За замовчуванням, дорівнює 40, тобто якщо в блоці зайнято менше 40% простору, в нього вводяться дані при виконанні операції вставки. Сума значень параметрів PCTFREE та PCTUSED не повинна перевищувати 100.

• INITRANS задає розмір пам'яті, який виділяється всередині блоку даних для запису транзакцій. Значення за замовчуванням дорівнює 1. Дозволені цілі числа від 1 до 255MAXTRANS використовується для визначення максимального числа одночасних транзакцій, які можуть оновлювати блок даних таблиці. Його значення за замовчуванням залежить від розміру блоку даних.

• TABLESPACE визначає назву табличного простору, в якій буде розміщено таблицю. При відсутності параметра таблиця розміщується у табличному просторі, який заданий за замовчуванням для користувача – власника поточної схеми.

• STORAGE визначає обсяг зовнішньої пам'яті, що виділяється під таблицю. Для великих таблиць доцільно явно виділяти необхідну пам'ять, щоб зменшити число запитів на динамічне виділення простору.

• RECOVERABLE і UNRECOVERABLE слугують для керування записом у журнальний файл контрольної інформації.

Призначення інших параметрів таке:

• CLUSTER вказує прив'язку стовпців таблиці до кластеру. Зазвичай стовпцями кластера є стовпці, що утворюють первинний ключ.

• CACHE вказує на те, що блоки таблиці позначаються у системному кеші як найбільш використовувані. Параметр рекомендується використовувати для невеликих таблиць-довідників для здійснення швидкого перетворення кодів у значення. За замовчуванням, використовується NOCACHE.

Додаток Д

Системні привілеї Oracle

Таблиця Д.1

Системні привілеї Oracle та їх призначення

Системний привілей	Дозволяє
ALTER ANY CLUSTER	Змінювати будь-який кластер у будь-якій схемі
ALTER ANY INDEX	Змінювати будь-який індекс у будь-якій схемі
ALTER ANY PROCEDURE	Змінювати будь-яку збережену процедуру, функцію або пакет у будь-якій схемі
ALTER ANY ROLE	Змінювати будь-яку роль у базі даних
ALTER ANY SEQUENCE	Змінювати будь-яку послідовність у будь-якій схемі
ALTER ANY SNAPSHOT	Змінювати будь-який знімок у будь-якій схемі
ALTER ANY TABLE	Змінювати будь-яку таблицю в будь-якій схемі
ALTER ANY TRIGGER	Включати, виключати й компілювати тригери в будь-якій схемі
ALTER DATABASE	Змінювати базу даних
ALTER PROFILE	Змінювати будь-який профіль у базі даних
ALTER RESOURCE COST	Установлювати вартості системних ресурсів
ALTER ROLLBACK SEGMENT	Змінювати сегменти відкоту
ALTER SESSION	Видавати речення ALTER SESSION
ALTER SYSTEM	Видавати речення ALTER SYSTEM
ALTER TABLESPACE	Змінювати табличні простори
ALTER USER	Змінювати інших користувачів: пароль; метод ідентифікації; квоти; табличні простори; призначати за замовчуванням, профілі й роли; призначати квоти на будь-який табличний простір
ANALYZE ANY	Аналізувати будь-яку таблицю, індекс або кластер у будь-якій схемі
AUDIT ANY	Виконувати аудит по будь-якому об'єкті в будь-якій схемі
AUDIT SYSTEM	Виконувати аудит речень SQL
BECOME USER	Ставати іншим користувачем. Використовується утилітою Import під час завантаження даних у схему іншого користувача
BACKUP ANY TABLE	Виконувати резервне копіювання будь-якої таблиці в будь-якій схемі за допомогою утиліти Export
COMMENT ANY TABLE	Створювати коментар по будь-якій таблиці, поданню або стовпцю в будь-якій схемі
CREATE ANY CLUSTER	Створювати кластер у будь-якій схемі
CREATE ANY INDEX	Створювати будь-який індекс у будь-якій схемі

Системний привілей	Дозволяє
CREATE ANY PROCEDURE	Створювати збережені процедури, функції та пакети в будь-якій схемі
CREATE ANY SEQUENCE	Створювати послідовність у будь-якій схемі
CREATE ANY SNAPSHOT	Створювати знімок у будь-якій схемі. Вимагає також привілею CREATE ANY TABLE
CREATE ANY SYNONYM	Створювати синонім у будь-якій схемі
CREATE ANY TABLE	Створювати таблицю в будь-якій схемі. Власник схеми, що містить таблицю, повинен мати квоту у відповідному табличному просторі
CREATE ANY TRIGGER	Створювати в будь-якій схемі тригер, асоційований з будь-якою таблицею в будь-якій схемі
CREATE ANY VIEW	Створювати подання в будь-якій схемі
CREATE CLUSTER	Створювати кластери у своїй схемі
CREATE DATABASE LINK	Створювати особисті зв'язки баз даних у своїй схемі
CREATE INDEX	Створювати індекс у своїй схемі по будь-якій таблиці в цій же схемі
CREATE PROCEDURE	Створювати збережені процедури, функції та пакети у своїй схемі
CREATE PROFILE	Створювати профілі
CREATE PUBLIC DATABASE LINK	Створювати загальні зв'язки баз даних
CREATE PUBLIC SYNONYM	Створювати загальні синоніми
CREATE ROLE	Створювати ролі
CREATE ROLLBACK SEGMENT	Створювати сегменти відкоту
CREATE SESSION	З'єднуватися з базою даних
CREATE SEQUENCE	Створювати послідовності у своїй схемі
CREATE SNAPSHOT	Створювати знімки у своїй схемі. Вимагає також привілею CREATE TABLE
CREATE SYNONYM	Створювати синоніми у своїй схемі
CREATE TABLE	Створювати таблиці у своїй схемі. Потрібна також UNLIMITED TABLESPACE або квота на табличний простір
CREATE TABLESPACE	Створювати табличні простори
CREATE TRIGGER	Створювати тригери у своїй схемі

Системний привілей	Дозволяє
CREATE USER	Створювати користувачів, призначати квоту на будь-який табличний простір, установлювати за замовчуванням тимчасові табличні простори, призначати профіль як частину CREATE USER
CREATE VIEW	Створювати подання у своїй схемі
DELETE ANY TABLE	Видаляти рядки з будь-якої таблиці, подання або знімка в будь-якій схемі
DROP ANY CLUSTER	Видаляти будь-який кластер у будь-якій схемі
DROP ANY INDEX	Видаляти будь-який індекс у будь-якій схемі
DROP ANY PROCEDURE	Видаляти збережені процедури, функції й пакети в будь-якій схемі
DROP ANY ROLE	Видаляти будь-яку роль у базі даних
DROP ANY SEQUENCE	Видаляти будь-яку послідовність в у будь-якій схемі
DROP ANY SNAPSHOT	Видаляти будь-який знімок у будь-якій схемі
DROP ANY SYNONYM	Видаляти будь-який синонім у будь-якій схемі, за винятком загальних синонімів
DROP ANY TABLE	Видаляти будь-яку таблицю в будь-якій схемі
DROP ANY TRIGGER	Видаляти будь-який тригер у будь-якій схемі
DROP ANY VIEW	Видаляти будь-яке подання у будь-якій схемі
DROP PROFILE	Видаляти будь-який профіль у базі даних
DROP PUBLIC DATABASE LINK	Видаляти загальні зв'язки баз даних
DROP PUBLIC SYNONYM	Видаляти загальні синоніми
DROP ROLLBACK SEGMENT	Видаляти сегменти відкоту
DROP TABLESPACE	Видаляти табличні простори
DROP USER	Видаляти інших користувачів
EXECUTE ANY PROCEDURE	Виконувати будь-яку збережену процедуру, функцію або пакет або звертатися до будь-якої загальної пакетованої змінної в будь-якій схемі
FORCE ANY TRANSACTION	Форсувати підтвердження або відкот будь-якої розподіленої транзакції в локальній базі даних. Форсувати збій розподіленої транзакції
FORCE TRANSACTION	Форсувати підтвердження або відкот своєї розподіленої транзакції в локальній базі даних
GRANT ANY PRIVILEGE	Призначати системні привілеї, навіть ті, котрими користувач не володіє
GRANT ANY ROLE	Призначати будь-яку роль у базі даних

Закінчення табл. Д.1

Системний привілей	Дозволяє
INSERT ANY TABLE	Вставляти рядки в будь-яку таблицю, подання або знімок у будь-якій схемі
LOCK ANY TABLE	Блокувати будь-яку таблицю або подання у будь-якій схемі
MANAGE TABLESPACE	Переводити табличний простір в он-лайн і оф-лайн, виконувати резервне копіювання табличного простору
READUP	Опитувати дані, що мають клас доступу вище, ніж мітка сесії. Цей привілей доступний тільки в Trusted ORACLE
RESTRICTED SESSION	З'єднуватися з базою даних, запущеною в режимі STARTUP RESTRICT
SELECT ANY SEQUENCE	Звертатися до будь-якої послідовності в будь-якій схемі
SELECT ANY TABLE	Опитувати будь-яку таблицю, подання або знімок у будь-якій схемі
UNLIMITED TABLESPACE	Використовувати необмежену пам'ять у будь-якому табличному просторі. Цей привілей перекриває будь-які квоти. Якщо треба відкликати цей привілей від користувача, то його об'єкти залишаються, але подальший розподіл пам'яті в табличних просторах буде дозволено тільки при наявності квоти на конкретний табличний простір. Цей привілей не можна призначати ролям
UPDATE ANY TABLE	Оновляти рядки в будь-якій таблиці, поданні або знімку в будь-якій схемі
TRUNCATE ANY	Усікати будь-яку таблицю або кластер у будь-якій схемі
WRITEDOWN	Створювати, змінювати (alter) і видаляти об'єкти схем, а також вставляти, оновляти та видаляти рядки, що мають клас доступу нижче, ніж у мітки сесії. Цей привілей доступний тільки в Trusted ORACLE
WRITEUP	Створювати, змінювати (alter) і видаляти об'єкти схем, а також вставляти, оновляти та видаляти рядки, що мають клас доступу вище, ніж у мітки сесії. Цей привілей доступний тільки в Trusted ORACLE

Додаток Е

Основні подання словника даних Oracle

Таблиця Е.1

Основні подання словника даних Oracle та їх призначення

Подання	Коментар
ALL_OBJECTS	Об'єкти, доступні користувачеві
ALL_SEQUENCES	Опис послідовностей, доступних користувачеві
ALL_SNAPSHOTS	Усі моментальні копії, доступні користувачеві
ALL_SOURCE	Вихідний текст об'єктів, доступних користувачеві
ALL_SYNONYMS	Усі синоніми, доступні користувачеві
ALL_TABLES	Опис таблиць, доступних користувачеві
ALL_TAB_COLUMNS	Стовпці таблиць, подань і кластерів, доступних користувачеві
ALL_TAB_COMMENTS	Коментарі до таблиць і подань, доступні користувачеві
ALL_TAB_PRIVS	Привілей на об'єкти, які отримав користувач безпосередньо, через роль або як PUBLIC
ALL_TAB_PRIVS_MADE	Привілей користувача та привілей на його об'єкти
ALL_TAB_PRIVS_RECD	Привілей на об'єкти, які отримав користувач безпосередньо, через роль або як PUBLIC
ALL_TRIGGERS	Тригери, доступні користувачеві
ALL_TRIGGER_COLS	Використання стовпців у користувальницьких тригерах, у тригерах для його таблиць або у всіх тригерах, якщо він має привілей CREATE ANY TRIGGER
ALL_USERS	Інформація про всіх користувачів бази даних
ALL_VIEWS	Текст подань, доступних користувачеві
AUDIT_ACTIONS	Коди типів аудиторських дій
CODE_PIECES	Використовується для створення подань _OBJECT_SIZE
CODE_SIZE	Використовується для створення подань _OBJECT_SIZE
COLUMN_PRIVILEGES	Привілей на стовпці, які належать користувачеві, котрі він видав або отримав безпосередньо, через роль або як користувач PUBLIC
DBA_2PC_NEIGHBORS	Інформація від запитів затриманих транзакцій, що знову надійшли та відпрацьовали
DBA_2PC_PENDING	Транзакції, в яких відбувся збій під час фази підготовки
DBA_AUDIT_EXISTS	Протокол, створений командою AUDIT EXISTS
DBA_AUDIT_OBJECT	Журнал протоколу команд над об'єктами. Створюється у файлі CATAUDIT.SQL
DBA_AUDIT_SESSION	Журнал протоколу команд входу та виходу з ORACLE
DBA_AUDIT_STATEMENT	Синонім для USER_AUDIT_STATEMENT
DBA_AUDIT_TRAIL	Журнал протоколу всієї системи

Подання	Коментар
DBA_BLOCKERS	Усі сеанси, що тримають блокування, яке очікує хтось інший
DBA_CATALOG	Усі таблиці, подання, синоніми й послідовності, що належать користувачеві
DBA_CLUSTERS	Опис усіх кластерів
DBA_CLU_COLUMNS	Відповідність стовпців таблиць стовпцям кластера
DBA_COL_COMMENTS	Коментарі до стовпців усіх таблиць і подань
DBA_COL_PRIVS	Привілей на всі стовпці бази даних
DBA_CONSTRAINTS	Визначення правил цілісності для всіх таблиць бази даних
DBA_CONS_COLUMNS	Інформація про стовпці у визначеннях правил цілісності, створених користувачем
DBA_DATA_FILES	Файли бази даних
DBA_DB_LINKS	Усі зв'язки бази даних
DBA_DDL_LOCKS	Усі блокування DDL у базі даних і всі пов'язані з ними запити до блокувань DML
DBA_DEPENDENCIES	Залежності (від) всіх об'єктів бази даних
DBA_DML_LOCKS	Усі блокування DML у базі даних і всі пов'язані з ними запити до блокувань DML
DBA_ERRORS	Поточні помилки для всіх збережених об'єктів
DBA_EXP_FILES	Опис експортних файлів
DBA_EXP_OBJECTS	Об'єкти, які експортувалися
DBA_EXP_VERSION	Номер версії останнього експорту
DBA_EXTENTS	Екстенти всіх сегментів бази даних
DBA_FREE_SPACE	Вільні екстенти в табличних просторах, доступних користувачеві
DBA_INDEXES	Опис індексів, доступних користувачеві
DBA_IND_COLUMNS	Стовпці індексів користувача або його індексовані таблиці
DBA_LOCKS	Усі блокування та затримки в базі даних, а також усі запити, що до них надходять
DBA_OBJECTS	Усі об'єкти бази даних
DBA_OBJECT_SIZE	Розмір об'єктів PL/SQL бази даних
DBA_OBJ_AUDIT_OPTS	Параметри аудиторства для всіх таблиць і подань
DBA_PRIV_AUDIT_OPTS	Параметри аудиторства для привілеїв
DBA_ROLES	Усі ролі в базі даних
DBA_ROLE_PRIVS	Ролі, видані користувачам або іншим ролям
DBA_ROLLBACK_SEGS	Опис сегментів відкоту бази даних
DBA_SEGMENTS	Розподіл простору для всіх сегментів бази даних
DBA_SEQUENCES	Опис всіх послідовностей у базі даних
DBA_SNAPSHOTS	Усі моментальні копії в базі даних

Подання	Коментар
DBA_SNAPSHOT_LOGS	Усі журнали моментальних копій у базі даних
DBA_SOURCE	Вихідний текст усіх збережених об'єктів
DBA_STMT_AUDIT_OPTS	Параметри системного аудиторства
DBA_SYNONYMS	Усі синоніми в базі даних
DBA_SYS_PRIVS	Системні привілеї, видані користувачам або ролям
DBA_TABLES	Опис усіх таблиць бази даних
DBA_TABLESPACES	Опис усіх табличних просторів у базі даних
DBA_TAB_COLUMNS	Стовпці всіх таблиць, подань і кластерів
DBA_TAB_COMMENTS	Коментарі до таблиць і подань бази даних
DBA_TAB_PRIVS	Привілею на об'єкти всієї бази даних
DBA_TRIGGERS	Опис усіх тригерів бази даних
DBA_TRIGGERS_COLS	Використання стовпців у тригерах користувача
DBA_TS_QUOTAS	Квоти всіх користувачів у табличному просторі
DBA_USERS	Інформація про всіх користувачів бази даних
DBA_VIEWS	Текст усіх подань бази даних
DBA_WAITERS	Усі сеанси, що очікують або володіють блокуваннями
DBMS_ALERT_INFO	Таблиця сигналів тривоги, що регіструються
DBMS_LOCK_ALLOCATED	Таблиця користувальницьких блокувань
DEPTREE	Дерево залежності об'єктів
DICTIONARY	Опис таблиць і подань словника даних (синонім DICT)
DICT_COLUMNS	Опис стовпців таблиць і подань словника даних
ERROR_SIZE	Використовується для створення подань _OBJECT_SIZE
GLOBAL_NAME	Містить один рядок із глобальним іменем поточної бази даних
IDEPTREE	Відсортований, сформатований варіант DEPTREE
INDEX_HISTOGRAM	Містить статистику команди ANALYZE INDEX VALIDATE STRUCTURE
INDEX_STATS	Містить статистику команди ANALYZE INDEX VALIDATE STRUCTURE
LOADER_COL_INFO	Інформація про стовпці для SQL*LOADER
LOADER_CONSTRAINT_INFO	Інформація про обмеження цілісності для SQL*LOADER
LOADER_INDCOL_INFO	Інформація про індексовані стовпці для SQL*LOADER
LOADER_IND_INFO	Інформація про індекси для SQL*LOADER
LOADER_PARAM_INFO	Інформація про параметри для SQL*LOADER
LOADER_TAB_INFO	Інформація про таблиці для SQL*LOADER
LOADER_TRIGGER_INFO	Інформація про тригери для SQL*LOADER

Продовження табл. Е.1

Подання	Коментар
PARSED_PIECES	Використовується для створення подань _OBJECT_SIZE
PARSED_SIZE	Використовується для створення подань _OBJECT_SIZE
PUBLIC_DEPENDENCY	Залежності між об'єктами
RESOURCE_COST	Вартість кожного ресурсу
ROLE_ROLE_PRIVS	Інформація про ролі, призначені іншим ролям
ROLE_SYS_PRIVS	Інформація про системні привілеї, призначені ролям
ROLE-TAB-PRIVS	Інформація про об'єктні привілеї, призначені ролям
SESSION-PRIVS	Привілей, які користувач має в даний момент
SESSION-ROLES	Ролі, включені для користувача в даний момент
SOURCE-SIZE	Використовується для створення подань -OBJECT_SIZE
STMT_AUDIT_OPTION_MAP	Таблиця опису кодів типів параметрів протоколювання
SYSTEM_PRIVILEGE_MAP	Таблиця опису кодів системних привілеїв
TABLE_PRIVILEGES	Привілей на об'єкти, до яких користувач отримав привілеї; видав, для яких він є власником; або привілей, виданий користувачеві PUBLIC
USER_AUDIT_OBJECT	Записи протокольного журналу для команд, що звертаються до об'єкта
USER_AUDIT_.SESSION	Записи протокольного журналу про входи й виходи в систему
USER_AUDIT_STATEMENT	Записи протокольного журналу про наступні команди: GRANT, REVOKE, AUDIT, NOAUDIT і ALTER SYSTEM
USER_AUDIT_TRAIL	Записи протокольного журналу, стосовні до користувача
USER_CATALOG	Усі таблиці, подання, синоніми та послідовності, що належать користувачеві (синонім CAT)
USER_CLUSTERS	Опис кластерів користувача (синонім CLU)
USER_CLU_COLUMNS	Відповідність стовпців таблиць стовпцям кластера
USER_COL_COMMENTS	Коментарі до стовпців користувальницьких таблиць і подань
USER_COL_PR1VS	Привілей на стовпці, для яких користувач є власником, видав або отримав привілеї
USER_COL_PRIVS_MADE	Привілей на стовпці, для яких користувач є власником
USER_COL_PRIVS_RECD	Привілей на стовпці, для яких користувач є власником, видав або отримав привілеї
USER_CONSTRAINTS	Визначення правил цілісності для таблиць користувача
USER_CONS_COLUMNS	Інформація про стовпці у визначеннях правила цілісності, створених користувачем
USER.DB_LINKS	Зв'язки бази даних, що належать користувачеві
USER_DEPENDENCIES	Залежності об'єктів користувача

Подання	Коментар
USER_ERRORS	Поточні помилки для всіх об'єктів, що належать користувачеві
USER_EXTENTS	Екстенти сегментів, виділені об'єктам, що належать користувачеві
USER_FREE_SPACE	Вільні екстенти в табличних просторах, доступних користувачеві
USER_INDEXES	Опис індексів, доступних користувачеві (синонім IND)
USER_IND_COLUMNS	Стовпці індексів користувача або його індексовані таблиці
USER_OBJECTS	Об'єкти, що належать користувачеві (синонім OBJ)
USER_OBJECT_SIZE	Розмір об'єктів PL/SQL, що належать користувачеві
USER_OBJ_AUDIT_OPTS	Параметри аудиторства для користувальницьких таблиць і поданні
USER_RESOURCE_LIMITS	Обмеження на ресурси, доступні поточному користувачеві
USER_ROLE_PRIVS	Ролі, видані поточному користувачеві
USER_SEGMENTS	Розподіл простору для сегментів об'єктів користувача
USER_SEQUENCES	Опис послідовностей, створених користувачем (синонім SEQ)
USER_SNAPSHOTS	Усі моментальні копії, доступні користувачеві
USER_SNAPSHOT_LOGS	Усі журнали моментальних копій, що належать користувачеві
USER_SOURCE	Вихідний текст збережених об'єктів, що належать користувачеві
USER_SYNONYMS	Усі приватні синоніми користувача (синонім SYN)
USER_SYS_PRIVS	Системні привілеї, видані поточному користувачеві
USER_TABLES	Опис таблиць користувача (TABS)
USER_TABLESPACES	Опис доступних табличних просторів
USER_TAB_COLUMNS	Стовпці всіх таблиць, подань і кластерів(синонім COLS)
USER_TAB_COMMENTS	Коментарі до таблиць і подань, що належать користувачеві
USER_TAB_PRIVS	Привілей на об'єкти, для яких користувач є власником, видав або отримав привілеї
USER_TAB_PRIVS_MADE	Усі привілеї на об'єкти, що належать користувачеві

Зміст

НАВЧАЛЬНЕ ВИДАННЯ

Тарасов Олександр Васильович
Федько Віктор Васильович

КЛІЄНТ-СЕРВЕРНІ ТЕХНОЛОГІЇ СКБД ORACLE. МОВА SQL ORACLE

Навчально-практичний посібник
для самостійної підготовки студентів
з навчальної дисципліни
"Організація баз даних та знань"
для студентів напряму підготовки
6.050101 "Комп'ютерні науки"

Відповідальний за випуск *Пономаренко В. С.*

Відповідальний редактор *Оленич М. М.*

Редактор *Ганцевич Н. І.*

Коректор *Маркова Т. А.*

План 2015 р. Поз. № 40-П.
Підп. до друку 08.10.2015 р. Формат 60 × 90 1/16. Папір офсетний. Друк цифровий.
Ум. друк. арк. 24,0. Обл.-вид. арк. 30,0. Тираж 400 пр. Зам. № 173.

Видавець і виготівник – ХНЕУ ім. С. Кузнеця, 61166, м. Харків, просп. Леніна, 9-А

Свідоцтво про внесення суб'єкта видавничої справи до Державного реєстру
ДК № 4853 від 20.02.2015 р.