

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**  
**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**  
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»**  
**Факультет інформатики та обчислювальної техніки**

**БАЗИ ДАНИХ**

**Методичні вказівки до виконання комп'ютерних практикумів**

для студентів спеціальності  
126 “Інформаційні системи та технології”  
кафедри технічної кібернетики  
всіх форм навчання

*Рекомендовано*  
*Вченою радою факультету*  
*інформатики та обчислювальної*  
*техніки НТУУ «КПІ ім.І.Сікорського»*  
*Протокол № \_\_\_ від \_\_.\_\_.\_\_\_\_р.*

Київ - 2020

Бази даних. Методичні вказівки до виконання комп'ютерних практикумів. [Текст] / Уклад. К.Б.Остапченко.- К.: НТУУ-“КПІ ім.І.Сікорського”, 2020. - 153с.

Методичні вказівки призначені для студентів спеціальності 126 «Інформаційні системи та технології» кафедри технічної кібернетики всіх форм навчання. В посібнику наведена тематика комп'ютерних практикумів, теоретичні відомості, вимоги до застосування програмних засобів виконання циклу робіт, контрольні завдання, список рекомендованої літератури.

Укладач

К.Б. Остапченко, к.т.н., доцент

Відповідальний редактор

Рецензент

*За редакцією укладачів*

## Зміст

Вступ.....	4
Організація проведення комп'ютерних практикумів .....	5
Комп'ютерний практикум № 1. МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ І ПРОЕКТУВАННЯ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ .....	15
Комп'ютерний практикум № 2. СТВОРЕННЯ СТРУКТУРИ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОМАНДАМИ МОВИ ВИЗНАЧЕННЯ ДАНИХ.....	36
Комп'ютерний практикум № 3. ОБРОБКА ІНФОРМАЦІЇ В БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОМАНДАМИ МОВИ МАНІПУЛЮВАННЯ ДАНИМИ.....	77
Комп'ютерний практикум № 4. ФОРМУВАННЯ ЗАПИТІВ НА ВИВЕДЕННЯ ІНФОРМАЦІЇ З БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ .....	98
Комп'ютерний практикум № 5. УПРАВЛІННЯ ДОСТУПОМ ДО БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОМАНДАМИ МОВИ УПРАВЛІННЯ ДАНИМИ.....	135
Підсумковий комп'ютерний практикум. РОЗРОБКА БАЗИ ДАНИХ ТА ЇЇ АДМІНІСТРУВАННЯ .....	150
Список рекомендованої літератури.....	153

## Вступ

Комп'ютерний практикум з дисципліни "Бази даних" виконують студенти спеціальності 126 «Інформаційні системи та технології» денної та заочної форм навчання.

Головне призначення циклу комп'ютерних практикумів (робіт) – закріплення отриманих теоретичних знань в області адміністрування баз даних гнучких комп'ютеризованих систем та оволодіння студентами засобами взаємодії з системою управління базою даних, вироблення раціональних прийомів програмування операцій з обробки даних за допомогою команд мови структурованих запитів SQL. Виконавши цикл робіт студенти отримують навички роботи в інтегрованому середовищі системи автоматизованого проектування баз даних та з інструментальними засобами взаємодії з промисловим сервером бази даних Oracle.

Виконання комп'ютерних практикумів базується на знаннях, отриманих студентом при вивченні дисциплін "Програмування", "Теорія алгоритмів", "Дискретна математика", "Комп'ютерні мережі". Крім того студент повинен мати навички роботи з ПЕОМ з використанням операційної системи Windows.

Мета та завдання циклу комп'ютерних практикумів відповідають робочій програмі кредитного модуля та пройшли апробацію на кафедрі технічної кібернетики факультету інформатики та обчислювальної техніки.

## Організація проведення комп'ютерних практикумів

### 1. Призначення комп'ютерних практикумів

Основними завданнями циклу комп'ютерного практикуму є:

- закріплення отриманих теоретичних знань в області адміністрування баз даних гнучких комп'ютеризованих систем;

- оволодіння студентами навичками взаємодії з системою управління базою даних, вироблення раціональних прийомів програмування операцій з обробки даних за допомогою команд мови структурованих запитів.

Виконавши цикл робіт практикумів студенти отримують навички роботи в інтегрованому середовищі системи автоматизованого проектування баз даних та з інструментальними засобами взаємодії з промисловим сервером бази даних.

№ з/п	Назва комп'ютерного практикуму	Кількість ауд.годин
КП1	<i>Комп'ютерний практикум 1: Моделювання предметної області і проектування бази даних інформаційної системи</i> Дидактичні матеріали: Розділ 2 Тема 2 Література: [1, с.45-61, 141-182; 2, с.151-181; 3, с.52-109; 4, с.15-30, 195-222; 6, с.531-550] Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування за теоретичними відомостями та отриманими практичними результатами комп'ютерного практикуму	4
КП2	<i>Комп'ютерний практикум 2: Створення структури бази даних командами мови визначення даних</i> Дидактичні матеріали: Розділ 4 Тема 1 Література: [4, 5, 7] Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування за теоретичними відомостями та отриманими практичними результатами комп'ютерного практикуму	2
КП3	<i>Комп'ютерний практикум 3: Обробка інформації в базі даних командами мови маніпулювання даними</i> Дидактичні матеріали: Розділ 4 Тема 2 Література: [4, 5, 7] Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування за	2

№ з/п	Назва комп'ютерного практикуму	Кількість ауд.годин
	теоретичними відомостями та отриманими практичними результатами комп'ютерного практикуму	
КП4	<i>Комп'ютерний практикум 4: Формування запитів на виведення інформації з бази даних</i> Дидактичні матеріали: Розділ 4 Тема 2 Література: [4, 5, 7] Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування за теоретичними відомостями та отриманими практичними результатами комп'ютерного практикуму	4
КП5	<i>Комп'ютерний практикум 5: Управління доступом до бази даних командами мови управління даними</i> Дидактичні матеріали: Розділ 4 Тема 3 Література: [5, 7] Завдання на СРС: підготуватися до надання відповідей на контрольні запитання комп'ютерного тестування за теоретичними відомостями та отриманими практичними результатами комп'ютерного практикуму	2
КПП	<i>Підсумковий комп'ютерний практикум: Розробка бази даних та її адміністрування</i> Дидактичні матеріали: комплексне контрольне завдання з перевірки знань і навичок із розробки бази даних та адміністрування взаємодії із сервером бази даних Література: [1-7]	4

## 2. Рекомендації щодо проведення комп'ютерних практикумів

Виконання кожного комп'ютерного практикуму передбачає попереднє вивчення відповідного методичного матеріалу, засвоєння мети та порядку роботи, опрацювання теоретичного матеріалу та підготовку проекту звіту з необхідними за змістом роботи даними виконання пунктів завдання. Студент допускається до виконання практикуму, якщо виконані ці попередні умови.

Після завершення виконання роботи студент подає звіт з комп'ютерного практикуму та захищає його через систему дистанційного навчання кафедри test.tc.kpi.ua, яка пропонує дати відповіді на питання з теоретичного матеріалу роботи і питання щодо обґрунтування отриманих практичних результатів виконаного завдання. Комп'ютерний практикум вважається зарахований, якщо отримано правильні відповіді на 60% питань.

Загальні рекомендації щодо оцінювання виконання практикумів:

1. Максимальна кількість балів:

- за тематичні комп'ютерні практикуми – 5 практикумів \* 7 балів = 35 балів;
- за підсумковий комп'ютерний практикум – 15 балів;

2. Максимальна кількість балів за роботу зменшується, якщо:

- не виконано попередню підготовку – 1 бали;
- студент не знає частини теоретичного матеріалу – 1 бал;
- студент не виконав пункт практичного завдання – 1 бал;
- вчасно не здано звіт з роботи – 2 бал.

При поданні синтаксису команд різних мов застосовується схема позначень Бекуса для їх запису, з використанням таких елементів:

[A] – позначення необов'язкового елемента A;

B ... – позначення про повторення попереднього елемента B;

A | B – позначення вибору одного з елементів переліку;

{A , B} – позначення множини елементів.

Наприклад, команда створення таблиці

```
CREATE TABLE [схема.]таблиці
```

```
( стовпець тип_даних [DEFAULT вираз] [обмеження_стовпця],
```

```
... [, обмеження_таблиці] );
```

трактується наступним чином: в команді обов'язковим є власна назва таблиці, хоча б один стовпець та тип даних – CREATE TABLE temp (id number(5)); , а схема, вираз та обмеження не є обов'язковими елементами та можуть бути відсутні при написанні команди.

Тобто при написанні команди не застосовуються символи [, ], ..., |, {, }, які є елементами опису синтаксису команди.

3. Інструментальні засоби проведення комп'ютерних практикумів

При проведенні комп'ютерних практикумів використовується програмне забезпечення сервера бази даних промислового застосування Oracle 11g, CASE-

систем автоматизованого проектування баз даних Oracle Data Modeler, а також інструментальні програмні засобами взаємодії з сервером бази даних задля адміністрування його роботи Oracle SQL Developer.

Інтерфейс підключення до сервера бази даних - протокол tcp, адреса tc.kpi.ua (зовн.ip:77.47.131.58, внутр.ip:10.18.80.10), порт 1521, сервіс base.

Ресурс завантаження рекомендованого інструментального засобу виконання комп'ютерних практикумів - <https://www.oracle.com/technical-resources/>.

Ресурси системи дистанційного навчання кафедри – <http://test.tc.kpi.ua>.

**Oracle SQL Developer Data Modeler** – це комплексне рішення, що дозволяє розробникам проектувати семантичні моделі взаємозв'язків об'єктів для наступного перетворення їх у повноцінні бази даних. Продукт підтримує інфологічне, серверне моделювання і моделювання типів даних, пропонуючи можливості багаторівневого проектування і побудови концептуальних діаграм сутностей і зв'язків. Користувачі можуть створювати, розширювати і модифікувати моделі, а також порівнювати їх із вже існуючими і виконувати реверс-інжиніринг баз і моделей даних. Oracle SQL Developer Data Modeler може входити як складова до інтегрованого програмного середовища адміністрування, проектування бази даних Oracle SQL Developer.

**Oracle SQL Developer** - це графічний інструмент для розробки баз даних. За допомогою SQL Developer можна переглядати об'єкти бази даних, запускати SQL-команди, редагувати і налагоджувати PL/SQL-програми. Ви також можете запустити будь-яку кількість наданих звітів, а також створювати і зберігати власні. SQL Developer підвищує продуктивність і спрощує підтримку вашої бази даних при виконанні завдань з її розвитку. SQL Developer може підключатися до будь-якої СУБД Oracle від версії 9.2.0.1 і може працювати на Windows, Linux, Mac OSX.

Для установки Oracle SQL Developer розархівуйте файл sqldeveloper.zip на жорсткий диск. Перейдіть в каталог, в якому знаходиться розпакований Oracle SQL Developer.



Для ініціалізації і налаштування Oracle SQL Developer запустіть перший раз файл sqldeveloper.exe. З'явиться екран пошуку налаштувань. Якщо ви встановлюєте Oracle SQL Developer вперше, то натисніть No. Для завантаження призначених для користувача налаштувань попередньої версії Oracle SQL Developer натисніть Yes. Виберіть шлях до призначених для користувача налаштувань і натисніть Ok.

З'явиться головний екран додатка, частина якого наведена на рис. 1. Установка завершена успішно.

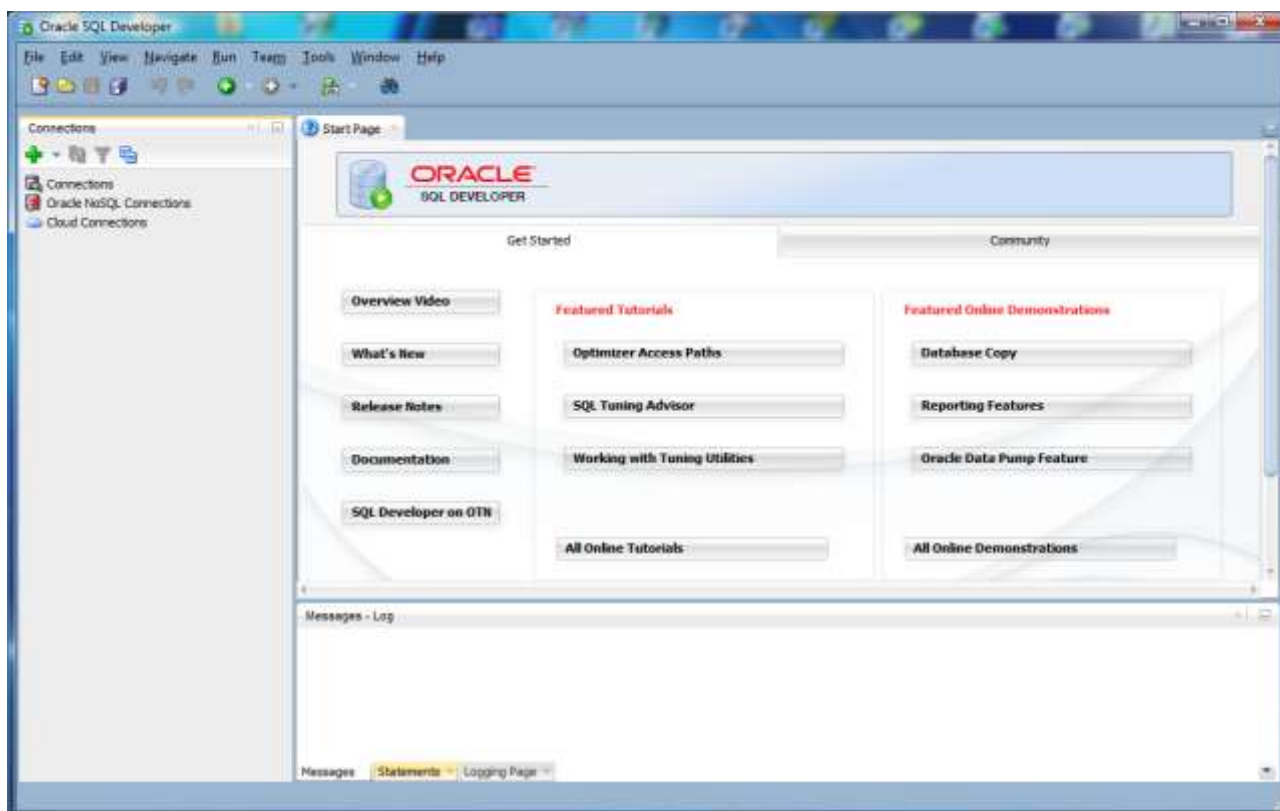


Рис. 1. Головне вікно Oracle SQL Developer

Для початку роботи з базою даних Oracle необхідно створити нове з'єднання. Для цього виберіть в головному меню програми пункт File / New. У діалоговому вікні виберіть пункт Database Connection і натисніть Ok.

Відкриється діалогове вікно New / Select Database Connection (рис. 2).

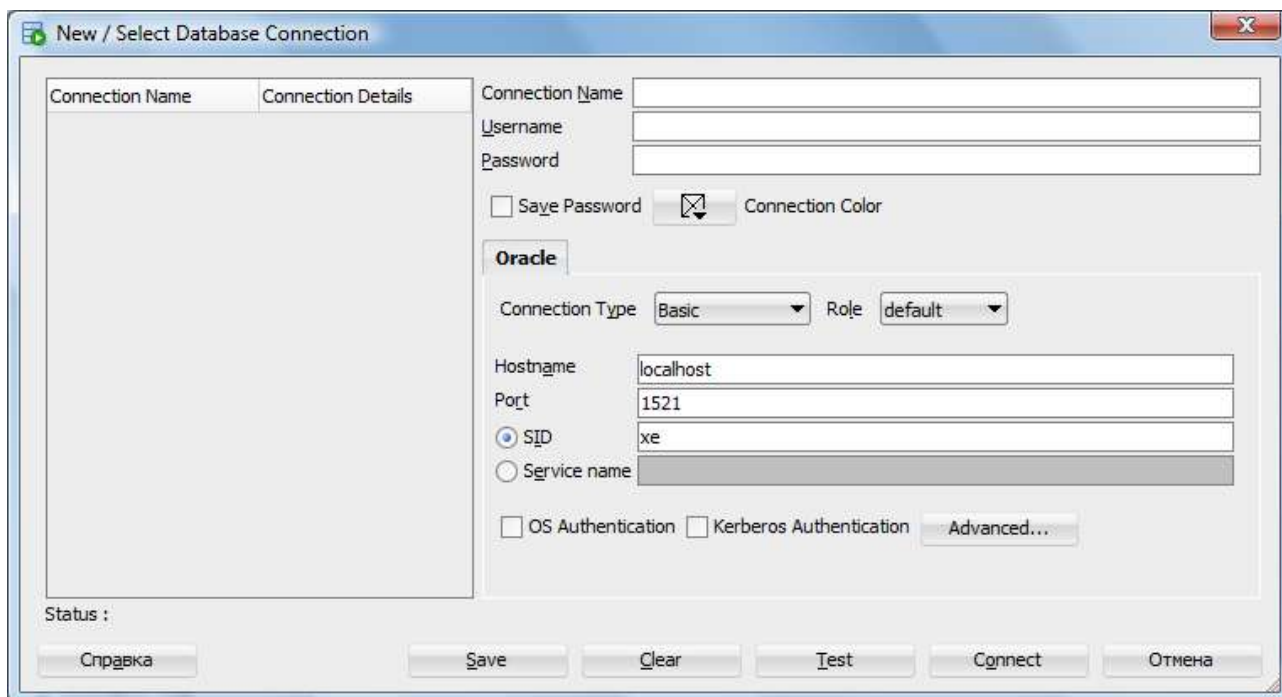


Рис. 2. Вікно створення нового з'єднання з базою даних

Спочатку налаштуйте підключення до бази даних. Для цього введіть в поля Connection Name назву з'єднання, ім'я користувача і пароль можна не вводити (в разі, якщо з цим підключенням буде працювати кілька користувачів). Потім встановіть Connection Type - Basic, Hostname - адреса або ім'я машини, де працює база даних, Service name - ім'я бази даних і натисніть Save (рис.3). Вийдіть з вікна створення підключення.

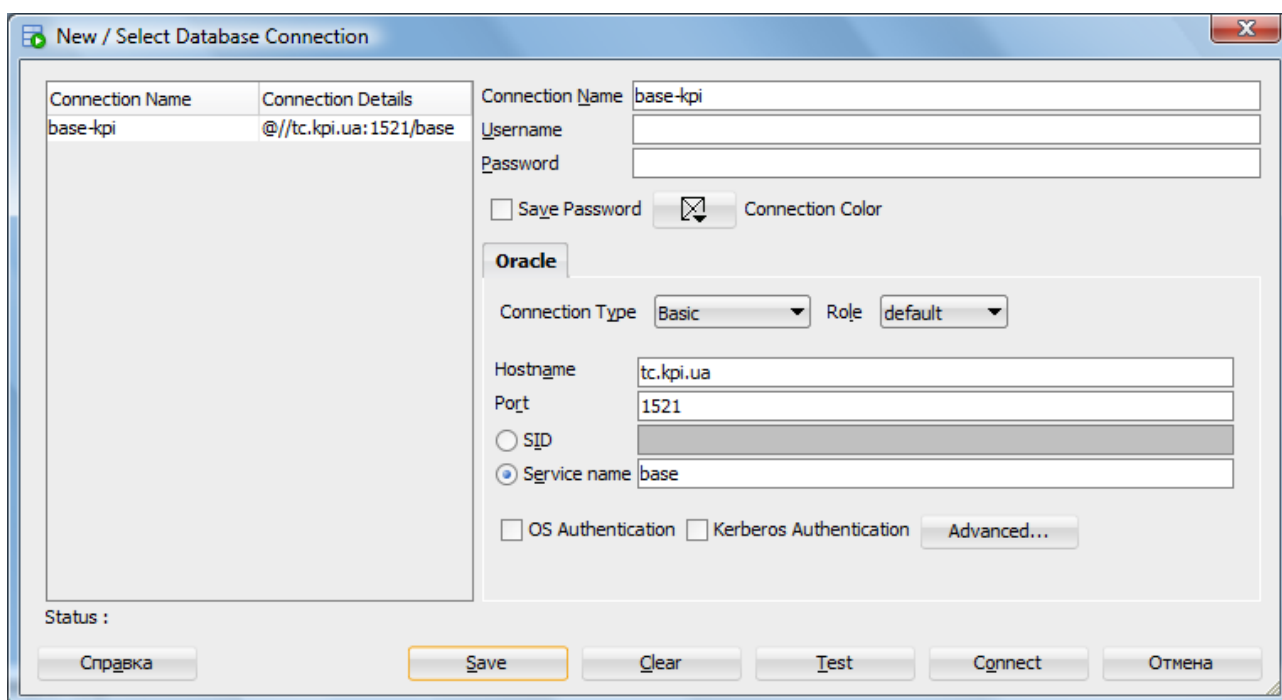


Рис. 3. Вікно створеного з'єднання з базою даних

Підключення до бази даних відбувається шляхом вибору в лівій навігаційній панелі головний екран додатка необхідного з'єднання та введення імені користувача і його паролю. В разі успішного підключення створюється робоче вікно поточного з'єднання під обраною назвою (рис. 4).

Вікно Oracle SQL Developer зазвичай використовує ліву навігаційну частину для пошуку і вибору об'єктів, праву частину для відображення інформації про вибрані об'єкти та головне меню з панелью інструментів.

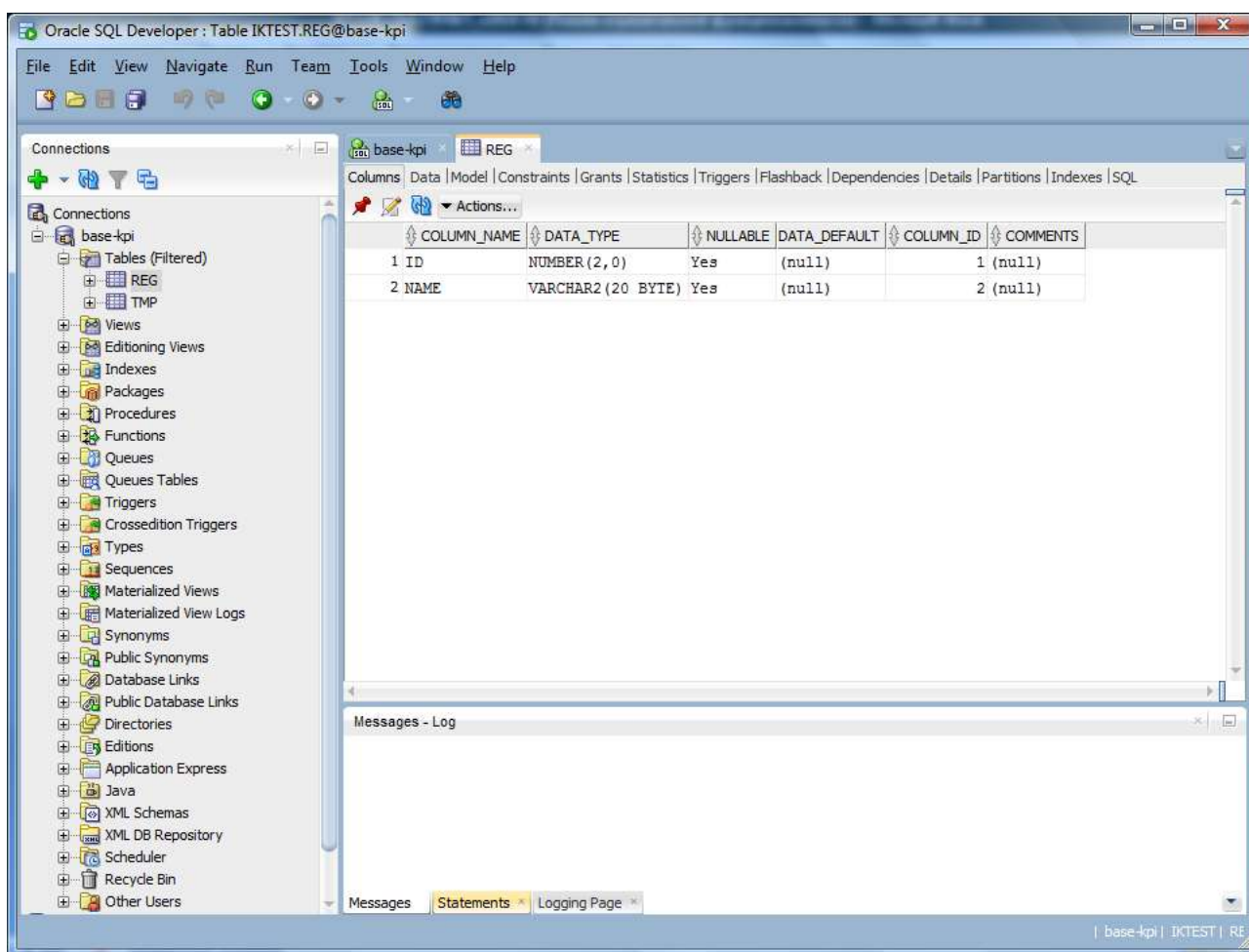


Рис. 4. Робоче вікно підключення до бази даних

Основна панель інструментів (розташована під головним меню) містить піктограми для виконання різних дій, які за замовчуванням включають наступне:

- 1) New - створює об'єкт бази даних;

- 2) Open - відкриває файл;
- 3) Save - зберігає будь-які зміни в обраному на даний момент об'єкті;
- 4) Save All - зберігає будь-які зміни на всіх відкритих об'єктах;
- 5) Back - переходить до області, яку було відвідано останньою;
- 6) Forward - переміщується до області після поточної у списку відвіданих панелей.

7) Open SQL Worksheet - відкриває робочий лист редактора команд SQL. Якщо не використовувати стрілку спадаючого меню, щоб вказати підключення до бази даних, яке потрібно використовувати, буде запропоновано вибрати з'єднання.

Ліва частина вікна Oracle SQL Developer містить навігаційні панелі, зокрема панель підключень Connections, піктограми для виконання дій та ієрархічне відображення дерева для поточної навігаційної панелі (рис. 5). Будь-які інші навігатори можуть бути обрані з пункту головного меню View, наприклад, панель звітів Reports.

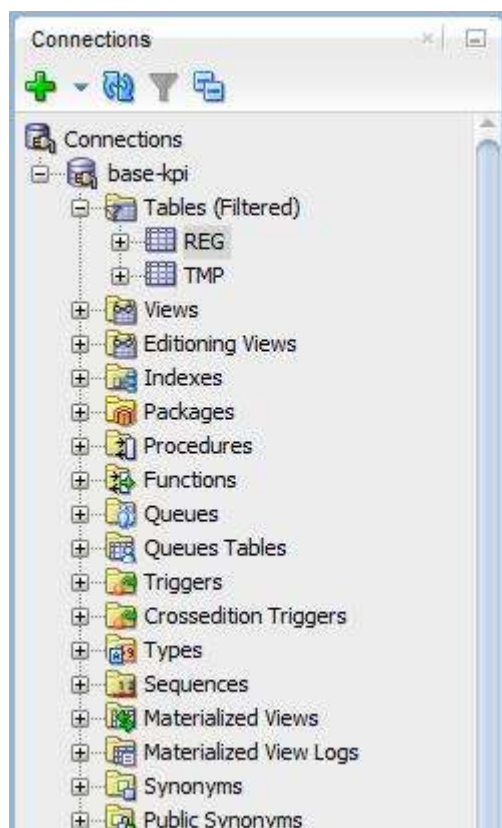


Рис. 5. Навігаційна панель Connections

Навігаційна панель Connections наводить перелік підключень до бази даних, які були створені. Щоб створити нове підключення до бази даних, імпортуйте XML-файл із визначеннями з'єднання або експортуйте чи відредагуйте поточні з'єднання, клацніть правою кнопкою миші вузол Connections та виберіть відповідний пункт меню.

Навігатор файлів (позначений значком папки у головному меню File / Open або View / Files для відтворення навігаційної панелі) відображає локальну файлову систему за допомогою стандартної ієрархії папок і файлів. Можна двічі клацнути або перетягнути файли, щоб відкрити їх, а також можна редагувати та зберігати файли. Наприклад, якщо відкрити .sql файл, він відображається у вікні робочого листа редактора команд SQL Worksheet (рис. 6). Навігатор файлів особливо корисний, якщо використовується система версій з Oracle SQL Developer.

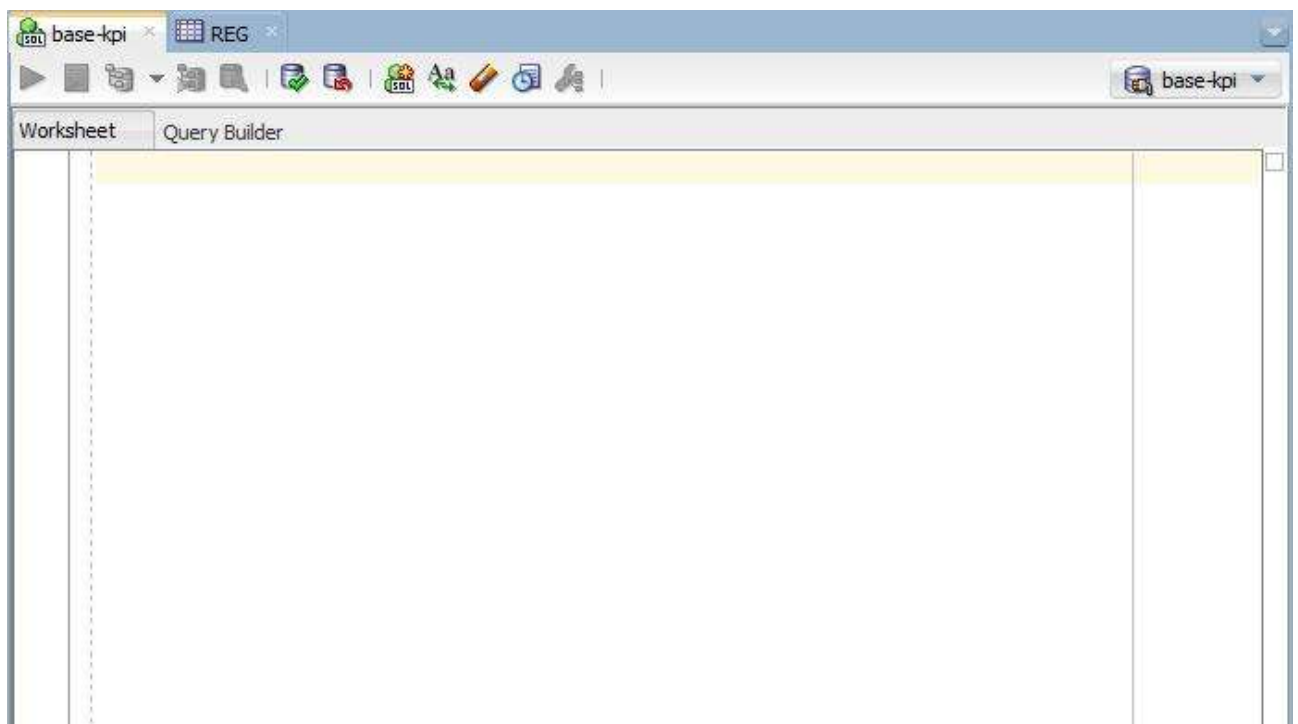
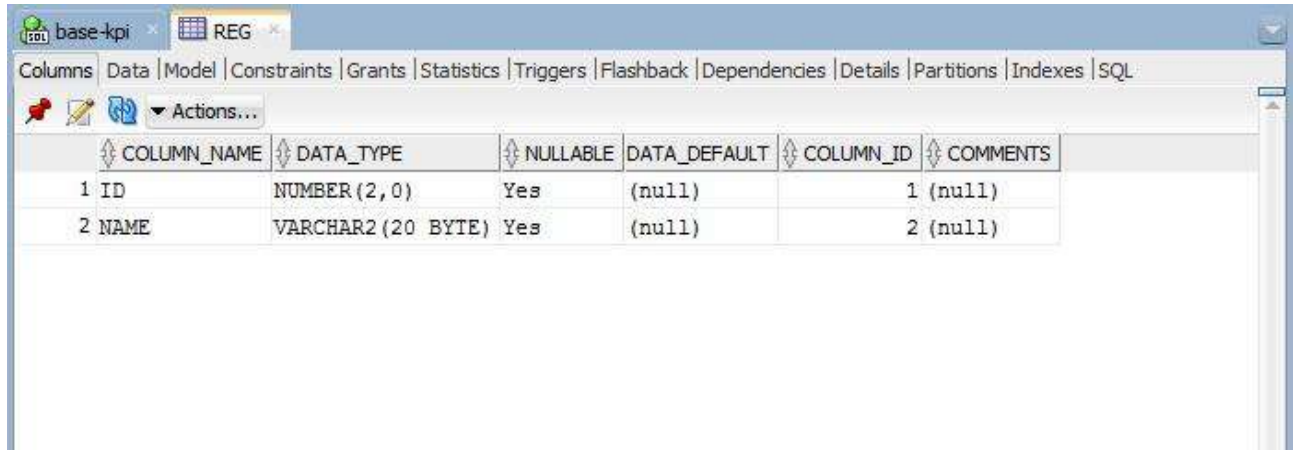


Рис. 6. Вікно робочого листа редактора команд та файлів SQL

Навігаційна панель звітів Reports перераховує інформаційні звіти, надані Oracle SQL Developer, такі як список таблиць без первинних ключів для кожного підключення до бази даних, а також будь-які визначені користувачем звіти.

У правій частині вікна Oracle SQL Developer є вкладки та панелі для вибраних або відкритих об'єктів, як показано на рис.7, де відображається інформація про таблицю з назвою REG. Якщо тримати вказівник миші на назві вкладки, то відображається підказка про власника об'єкта та з'єднання з базою даних.



COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS
1 ID	NUMBER(2,0)	Yes	(null)	1	(null)
2 NAME	VARCHAR2(20 BYTE)	Yes	(null)	2	(null)

Рис. 7. Інформаційне вікно панелей обраного об'єкта бази даних

Для таблиць і представлень ця інформація згрупована під вкладками, які позначені вгорі. Наприклад, для таблиць вкладки - стовпці, дані (для перегляду та зміни самих даних), індекси, обмеження тощо; і можна натиснути заголовок стовпця під вкладкою, щоб сортувати рядки сітки за значеннями в цьому стовпці. Для більшості об'єктів вкладки включають SQL, який відображає оператор SQL для створення об'єкта.

Область повідомлень Messages-Log використовується для відповідної інформації зворотного зв'язку (наприклад, результати дії, або повідомлення про помилки чи попередження). Якщо ця область ще не видно, ви можете відобразити її, натиснувши в головному меню View / Log.

# Комп'ютерний практикум № 1.

## МОДЕЛЮВАННЯ ПРЕДМЕТНОЇ ОБЛАСТІ І ПРОЕКТУВАННЯ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ

### 1. Мета та основні завдання практикуму

Мета роботи: ознайомитися з процесом моделювання даних і отримати навички роботи в системі автоматизованого проектування бази даних.

Порядок виконання роботи:

- вивчити етапи розробки інформаційної системи;
- визначити основні компоненти опису даних в інформаційній системі;
- згідно умов контрольного завдання створити семантичну модель даних та проект реляційної бази даних;
- підготувати звіт за результатами проектування відповідно до умов контрольного завдання.

### 2. Основні теоретичні відомості

#### 2.1. Етапи розробки інформаційної системи

Для логічного й успішного створення об'єктів бази даних на сервері необхідно пройти весь цикл розробки інформаційної системи. Кожен етап циклу складається з визначених процедур, які необхідно виконати, якщо потрібно, щоб проект бази даних був ефективним. Проектування бази даних – це тільки один з етапів циклу розробки. Встановлено, що завдяки якісно створеному проекту бази даних, стає можливим реалізація надійної і високопродуктивної системи. У процесі розробки інформаційної системи доводиться вирішувати безліч питань – від контролю надмірності даних до удосконалювання взаємодії з кінцевим користувачем. Вирішуючи кожен з цих проблем в ретельно продуманому проекті бази даних, підвищується її продуктивність та й інформаційної системи в цілому.

Від зародження концепції до введення в експлуатацію розробка бази даних здійснюється в строгій відповідності із циклом розробки інформаційної

системи. Цей цикл заснований на системному підході до розробки бази даних за принципом «зверху-униз», який забезпечує перетворення інформаційних потреб предметної області системи в робочу базу даних. *Предметна область* – обумовлена область застосування конкретної інформації про об'єкт автоматизації, для якого розробляється інформаційна система.

Цикл розробки включає такі п'ять основних етапів:

- 1) Моделювання;
- 2) Проектування;
- 3) Кодування і документування;
- 4) Налагодження і впровадження;
- 5) Експлуатація.

Етап «*Моделювання*» припускає вивчення й аналіз потреб в інформації об'єкта автоматизації, де буде впроваджуватися інформаційна система з базою даних. З'ясування інформаційних потреб виробляється в ході взаємодії розробника майбутньої системи із замовником (представником об'єкта автоматизації) на основі бесід з його кінцевими користувачами задля встановлення термінів, понять, характеристик об'єктів (сутностей), що утворять предметну область об'єкта автоматизації. Також, корисним для розробки майбутньої системи є вивчення документації, у якій формулюються задачі функціонування об'єкта автоматизації (підприємства, виробництва, організації, бізнесу і т.п.) і прикладної системи. Результати аналізу представляються у виді семантичної/інфологічної моделі (Semantic/Logical Model), тобто словесні описи перетворюються в графічні представлення інформаційних потреб і правил роботи об'єкта автоматизації. Форма і семантика інфологічної моделі повинна бути придатна для обговорення і спільного удосконалювання з аналітиками й експертами замовника.

Етап «*Проектування*» припускає розробку структури бази даних. Сутності і зв'язки, представлені в інфологічній моделі, перетворюються в інформаційні компоненти бази даних (таблиці, стовпці, ключі, обмеження і т.і.), які утворюють так звану серверну/дatalogічну модель (Server/Data Model) бази



даних.

Етап «Кодування і документування» припускає створення дослідного зразка системи, написання і виконання команд для створення таблиць і допоміжних об'єктів бази даних, програмування інтерфейсів системи, а також розробку документації користувача, текстів довідок і посібників з експлуатації системи.

Етап «Налагодження і впровадження» передбачає передачу системи користувачу від замовника, спостереження за її продуктивністю, розширення можливостей і подальше її удосконалення.

## 2.2. Моделювання предметної області й інформаційних потреб системи

Моделювання – це першооснова процесу будь-якої розробки інформаційної системи. Метою моделювання є формалізація інформаційних потреб користувачів у виді подання правил взаємодії даних у рамках розроблювальних прикладних систем. При моделюванні необхідно враховувати такі ключові фактори, як:

- *продуктивність*. Якість (ефективність) проектування має найбільше значення для кінцевої продуктивності системи, ніж усі наступні зусилля з корекції і настроюванню системи.

- *інтегрованість складових*. Прикладні системи звичайно створюються командами розробників. При відсутності спільного (єдиного, узгодженого) вихідного завдання на проект кожен розробник працює виходячи з власних представлень про проєктовану систему. Якісний же проєкт забезпечує не тільки однаковість зовнішнього вигляду і поведіння системи, але й успішну інтеграцію готових прикладних частин системи одна з одною.

- *інтеграція з іншими системами*. Нову систему часто доводиться інтегрувати з вже існуючими системами і навіть з тими, котрі ще тільки створюються. Якісне проектування дозволяє поширювати вищевказані переваги на корпоративні (складні, комплексні) системи.

- *документація і взаємодія*. Одна з основних задач розробника полягає

також в тому, щоб довести кожне конструкторське рішення до інших розробників, користувачів через документування.

- *масштабованість/розширюваність*. Питання оцінки продуктивності системи повинні розглядатися на етапі проектування, а не під час експлуатації. Розробка проекту в невеликому контрольованому (тестовому) середовищі не дозволяють перевірити його поведінку в реальних ситуаціях з великими обсягами даних, що певним чином впливає на можливість встановлення недоліків проекту.

- *використання ефективних готових рішень*. Залучення в процес моделювання вже раніше апробованих рішень з відомими параметрами продуктивності дозволять скоротити загальний час проектування інформаційної системи.

Таким чином, основу проектування інформаційної системи складають моделі даних різного призначення. Тому моделями майбутньої інформаційної системи на різних етапах процесу проектування можуть бути такі їх різновиди:

- у виді описів у представленні користувача системи (технічне завдання/умови на розробку проекту);

- у виді інформаційних сутностей предметної області інформаційної системи (семантична модель);

- у виді табличної моделі бази даних (серверна модель) і файлів, блоків даних (фізична модель) на сервері бази даних.

Проте, кінцевим результатом моделювання предметної області інформаційної системи є створення моделі такого виду, яка б вирішувала задачі формалізації всіх правил роботи об'єкта автоматизації, була зрозуміла кінцевому користувачу замовника майбутньої інформаційної системи, а також містила досить докладну інформацію, на основі якої проектувальник розробника (адміністратор бази даних, програміст) міг би створити працездатну базу даних й ефективну інформаційну систему. Тому, в якості такої моделі частіше розглядають семантичну модель типу «сутність-зв'язок» - Entity Relation модель (ER-модель).

ER-модель створюється на основі словесних описів і документів, що відтворюють інформаційні потреби користувача і правила роботи системи у певній предметній області об'єкта автоматизації. Така модель є концептуальним (інфологічним) представленням предметної області інформаційної системи. Перевагами такої ER-моделі є:

- можливість обміну ідеями при розробці;
- ефективність збору і документування інформаційних потреб системи;
- зрозуміле графічне представлення системи;
- простота розробки і внесення удосконалень.

Процес моделювання є ітераційним й припускає повернення до попередніх етапів для перегляду раніше прийнятих рішень і включає наступні кроки:

- виділення семантичних об'єктів (понять) предметної області системи;
- класифікація семантичних об'єктів з виділенням сутностей і зв'язків з подальшим їх описом;
- побудова діаграми ER-типу з урахуванням всіх сутностей і зв'язків;
- формування унікального ідентифікатора сутностей;
- додавання не ключових атрибутів до сутностей;
- приведення моделі до нормалізованого стану.

ER-модель складається (структурно) із сутностей, атрибутів і зв'язків.

*Сутність* представляє об'єкт, факт або сукупність взаємозалежних даних предметної області, інформацію про які необхідно зберігати в системі. Прикладами сутностей є клієнти, замовлення чи службовці в системах обслуговування.

Для представлення сутностей у ER-моделі може бути використана одна з наступних систем позначень (рис. 1) – бінарна модель (Barker's notation): прямокутник з округленими кутами довільного розміру, унікальне ім'я сутності (заголовними буквами), синоніми сутності приводяться в дужках. На даний час існує багато різновидів систем позначень (Logical notation), які відрізняються способами відтворення сутностей, зв'язків, атрибутів в залежності від потреб

(обмежень) моделювання предметної області інформаційної системи. Проте, всі вони є підкласами моделі П.Чена, яким було запропоновано ER-моделювання.

Одну сутність можна відрізнити від іншої за допомогою її типу, яким у моделі слугує унікальне ім'я сутності, а також за набором атрибутів сутності.

*Атрибут* описує сутність (його властивість) і містить необхідну інформацію про неї. Сутність може описуватися декількома атрибутами. Наприклад, атрибутами сутності «КЛІЄНТ» можуть бути номер клієнта, прізвище, номер телефону, адреса. Якщо сутність не має атрибутів, то вона не входить у набір вимог до системи і не повинна з'являтися в моделі. Кожен атрибут сутності може бути обов'язковими (позначається символом «\*») чи необов'язковим (позначається символом «o»), з точки зору вимоги отримання певного значення атрибутом. Ця властивість називається *mandatory*. Таким чином, для представлення атрибута в моделі використовується наступна схема позначень (рис.1): унікальне ім'я атрибута (малими літерами) у сутності, властивість обов'язковості (*mandatory*), тип даних.

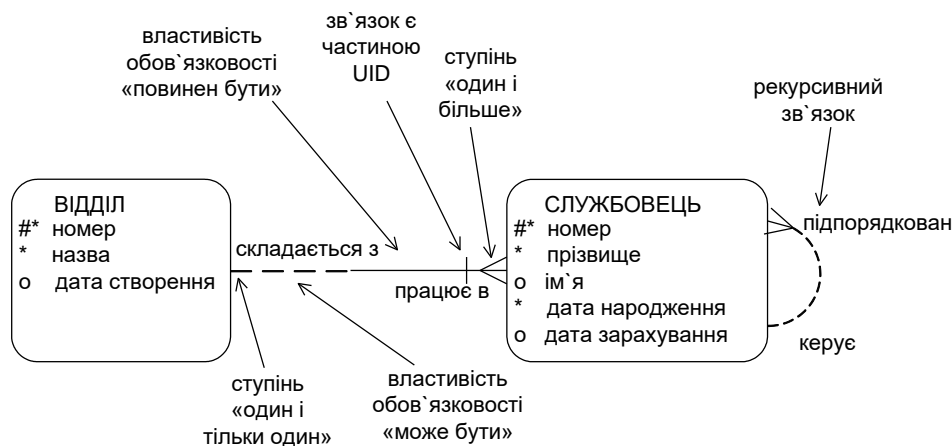


Рис. 1. Система позначень в ER-моделях

Дані із сутності повинні бути унікально пізнані та мати відмінності, тобто один екземпляр (примірник) сутності повинен відрізнитися від іншого. З цією метою в опис сутності вводиться унікальний ідентифікатор. *Унікальний ідентифікатор* (UID) – це будь-яка комбінація атрибутів, зв'язків, яка слугує для встановлення розходжень екземплярів даних у сутності. Кожен екземпляр

сутності повинний бути однозначно пізнаним. Атрибути, що входять у UID, позначаються символом «#», а вторинні (додаткові) UID – цим же символом у дужках – (#). Часто перші UID називають первинними.

Кожна сутність повинна мати зв'язок з іншими сутностями, який відтворює інформаційні потреби і правила взаємозалежності даних у прикладній системі. *Зв'язок* – це двох направлена асоціація між двома сутностями або асоціація сутності сама із собою. Якщо сутність зв'язана сама із собою, така асоціація називається рекурсивною. Рекурсивний зв'язок використовується для відображення ієрархічної організації сутності. Наприклад, коли один службовець перебуває в підпорядкуванні іншого службовця з однієї сутності.

Виходячи з визначення зв'язку як двонаправленої асоціації, кожен напрямок зв'язку характеризується в моделі:

- найменуванням (наприклад, “навчається в” або “прикріплений до”),
- властивістю необов'язковості (*optional* як альтернатива *mandatory*),
- властивістю (ступенем) кратності (*cardinality*) - «один і тільки один» або «один і більше».

Синтаксис представлення зв'язків у моделі наступний:

- а) вихідна сутність асоціюється спрямованим зв'язком з кінцевою сутністю;
- б) вид структури зв'язку – *вихідна сутність* {«може бути»|«повинна бути»} *ім'я зв'язку* {«один і тільки один»|«один і більше»} *кінцева сутність*.

Умовні позначення при відтворенні зв'язків:

- пунктирна лінія відображає необов'язковий елемент «може бути»;
- суцільна лінія відображає обов'язковий елемент «повинний бути»;
- одиночне закінчення лінії позначає ступінь «один і тільки один»;
- розгалужене закінчення лінії позначає ступінь «один і більше».

В залежності від значень ступенів (кратності) напрямків виділяють такі *типи зв'язків*:

- а) “один до одного”. Це зв'язок зі ступенем «один і тільки один» в обох

напрямок. Зв'язки сутностей такого типу зустрічаються рідко й у дійсності можуть виявитися тією же сутністю або атрибутом сутності. Тобто, такі зв'язки можуть бути використані, якщо потрібно відокремити певну підмножину необов'язкових атрибутів деякої сутності.

б) “багато до одного” (або “один до багатьох”). Це зв'язок зі ступенем «один і більше» в одному напрямку і «один і тільки один» в іншому напрямку. Такі зв'язки широко поширені і відтворюють переважну більшість інформаційних потреб і правил взаємозалежності даних у прикладній системі.

в) “багато до багатьох”. Це зв'язок зі ступенем «один і більше» в обох напрямках. Такі зв'язки менш поширені (наприклад, між сутностями “службовці” і їхні “навички”). І якщо виникає необхідність їх відображення в моделі, то представляються вони за допомогою додаткової сутності перетинання (проміжної між вихідною і кінцевою сутностями із зв'язок типу “багато до одного”).

В залежності від властивості обов'язковості напрямків виділяють наступні класи зв'язків:

а) повний зв'язок, якщо всі напрямки є обов'язковими, тобто визначені як «повинні бути»;

б) частковий зв'язок, якщо хоч один з напрямків є необов'язковим, тобто визначений як «може бути».

Прикладом опису зв'язку виконаного клієнтом замовлення товарів може бути така модель (рис.1):

- кожне ЗАМОВЛЕННЯ повинно бути для одного і тільки одного КЛІЄНТА;

- кожен КЛІЄНТ може зробити один і більш ЗАМОВЛЕНЬ.

Сутність може бути унікально пізнана також за допомогою зв'язку. У цьому випадку входження зв'язку в UID позначається поперечною рисою, тобто зв'язок є частиною первинного унікального ідентифікатора сутності. Зв'язок, що входить в UID, повинен бути обов'язковим і з ступенем «один і тільки один» у напрямку, що має відношення до UID. Наприклад, при замовленні товару

з'являється номер замовлення й унікальні номери рядків товарних позицій у замовленні. Але якщо зробити ще одне замовлення, номери рядків перестають бути унікальними. Отже, товарна позиція унікально ідентифікується своїм атрибутом «номер» і номером конкретного замовлення (див. табл.1).

Таблиця 1

Номер замовлення	Номер рядка	Товар
100	1	болт
100	2	шайба
101	1	гайка

### 2.3. Нормалізація моделі даних і умови цілісності бази даних

Перш, ніж приступити до роботи над структурою бази даних (її проекту), тобто перейти від ER-модель до серверної, необхідно усунути проблеми надмірності інформації шляхом нормалізації моделі даних таким чином, щоб вона задовольняла функціональним різним вимогам і відповідала різним проектним рішенням. Метою нормалізації є зведення до мінімуму надмірності (повторюваності) даних, зменшення проблеми забезпечення цілісності (узгодженості, несуперечності) даних, виявлення не потрібних сутностей, зв'язків.

Застосовують такі основні правила нормалізації:

- перша нормальна форма 1NF. Всі атрибути повинні мати одне значення і не повторюватися.

- друга нормальна форма 2NF. Всі атрибути повинні залежати (однозначно визначатися) від UID своєї сутності.

- третя нормальна форма 3NF. Атрибути, що не входять в UID, не можуть залежати від іншого атрибута, що не входить до UID.

Таким чином, виконуючи вимоги нормалізації в сутностях повинні бути присутні UID-атрибути, а зв'язки між сутностями встановлюватися на основі їхніх UID-атрибутів, які відображають спільні дані.

Умови цілісності бази даних, тобто обмеження спрямовані на її

збереження, дозволяють домогтися того, що користувачі виконують тільки ті операції, які зберігають базу даних правильною і узгодженою. Дотримання цих умов повинно контролюватися сервером бази даних або прикладною програмою інформаційної системи. У базі даних ER-модель і її компоненти (сутності, атрибути, зв'язки) перетворюються в серверну модель, яка включає в собі компоненти (об'єкти) бази даних - таблиці, стовпці, обмеження цілісності.

Таблиця бази даних це структура збереження інформації про одну однорідну групу об'єктів, що складається зі стовпців у базі даних.

Стовпець бази даних це атрибут таблиці, що включає в себе значення визначеної властивості.

Підтримка цілісності бази даних може розглядатися як механізм захисту даних від невірних (неприпустимих) змін у стовпцях бази даних. Цілісність забезпечується шляхом завдання обмежень. Обмеження цілісності бази даних подаються у вигляді визначення правил (умов) виконання зміни даних. Кожне правило, що накладає деяке обмеження на можливий стан бази даних, називається обмеженням цілісності (*integrity constraint*).

Умовам цілісності відповідають ключі трьох типів - первинні, унікальні і зовнішні, а також властивість обов'язковості (визначеності) значень атрибутів-стовпців, що утворюють ключі.

Кожен рядок таблиці унікально визначається за значенням одного чи декількох стовпців – обмеження *первинний ключ* PRIMARY KEY (PK). Визначення первинного ключа повинно бути таким, щоб він (його значення) не повторювався і не міг мати невизначені значення NULL. Первинний ключ, що складається з декількох стовпців, називається складеним чи складним первинним ключем. Комбінація значень стовпців у складеному первинному ключі повинна бути унікальною, хоча значення в кожному складовому окремому стовпці можуть повторюватися. Ніяка частина первинного ключа не може містити невизначені значення NULL.

Як первинний ключ може використовуватися будь-який унікальний ключ таблиці, а в цілому для таблиці визначається тільки один (єдиний) первинний



ключ.

Обмеження *унікальний ключ* UNIQUE (UK) – це стовпець або сполучення стовпців з неповторюваними значеннями. Таблиця може мати декілька унікальних ключів, що можуть претендувати на роль первинного ключа. Однак, унікальні ключі можуть містити невизначені значення NULL. Тому, для того щоб унікальний ключ використовувався в якості первинного для однозначного визначення рядків таблиці, необхідно, щоб його значення мали властивість обов'язковості (визначеності значень) NOT NULL. В результаті визначення первинного ключа рівнозначно визначенню ключа як UNIQUE з властивістю NOT NULL. Інші UK-ключі залишаються альтернативними унікальними ключами таблиці.

Обмеження *зовнішній ключ* FOREIGN KEY (FK) – це стовпець або набір стовпців в одній таблиці, що містить у своєму визначенні посилання на первинний ключ PK або унікальний ключ UK тієї ж чи іншої таблиці. Значення зовнішнього ключа є залежним від значення первинного чи унікального ключа таблиці-посилання і фактично відтворює дані, які слугують не фізичним, а логічним показником взаємозв'язку даних у різних таблицях. Отже, його значення повинно або збігатися зі значеннями відповідного первинного чи унікального ключа, або бути невизначеними NULL. Якщо зовнішній ключ є частиною первинного ключа таблиці, то він не може бути невизначеним.

Таким чином, у базі даних встановлюються такі типи обмежень цілісності:

1) цілісність сутностей – жодна частина первинного ключа не може мати значення NULL і, крім того, ключі повинні бути унікальними;

2) цілісність посилань – значення зовнішнього ключа повинні збігатися з первинним/унікальним ключем чи бути NULL;

3) цілісність стовпців – значення даних у стовпці повинні відповідати заданому типу даних (числовому NUMBER, рядковому з фіксованою довжиною CHAR, рядковому із змінною довжиною VARCHAR2, даті DATE) і володіти при необхідності властивістю обов'язковості значень NOT NULL. Інші

типи даних є похідними від цих базових і утворюються внаслідок встановлення правил домену як області визначення припустимих значень обраного базового типу;

4) цілісність, що задається користувачем – значення даних відповідають правилам, що встановлені в інформаційній системі (наприклад, ціна має тільки позитивне значення, номер замовлення повинен бути тільки цілим значенням і т.д.).

## 2.4. Проектування бази даних

Проектування бази даних – це перетворення інформаційної моделі системи в працездатне програмне рішення, яке відповідає умовам нормалізації даних. Метою проектування є створення надійних і ефективних систем на основі результатів попереднього аналізу і приведення їх до нормалізованого стану. В наслідок етапу моделювання, об'єкти-сутності, сприйняті кінцевим користувачем предметної області, повинні бути перетворені в об'єкти-компоненти бази даних, зрозумілі адміністратору бази даних. Результатом етапу проектування є проект структури бази даних, що включає визначення таблиць, стовпців, обмежень цілісності та додаткових об'єктів - індексів, представлень задля збереження і відтворення даних.

Перетворення ER-моделі системи в серверну модель (проект) бази даних полягає у виконання наступних кроків:

- 1) перетворення сутностей у таблиці;
- 2) перетворення атрибутів у стовпці;
- 3) перетворення унікальних ідентифікаторів у первинні ключі;
- 4) перетворення зв'язків у зовнішні ключі.

Додатково при проектуванні бази даних виконуються наступні дії:

1) рекомендується передбачити *індекси*, що є об'єктами бази даних, які забезпечують прямий і швидкий спосіб доступу до рядків таблиць. При бажанні можна створювати індекси по альтернативних унікальних ключах, по зовнішніх ключах і просто за стовпцями, які часто використовуються в умовах пошуку

даних для їх відтворення;

2) доцільно описати *представлення* даних, які можна визначити як віртуальні (не фізичні) таблиці, засновані на частині даних з однієї чи декількох інших таблиць/представлень. Представлення можуть обмежувати доступ до даних, забезпечувати вивід інформації в більш зручній формі і включати в собі заздалегідь підготовлені складні запити на відтворення даних;

3) виконати планування простору для фізичного збереження даних (формування умов утворення фізичної моделі бази даних), тобто кількість пам'яті, необхідної для розміщення даних таблиць у базі даних;

4) при необхідності перевизначити правила цілісності.

*Перетворення сутностей* у таблиці припускає створення і заповнення певного бланка для кожної таблиці. Бланк таблиці повинний містити ім'я таблиці і графи найменування стовпців, типи ключів (PK, UK, FK), властивості обов'язковості NOT NULL, інформацію про посилання зовнішніх ключів, типи даних і максимальну довжину стовпців.

Приклад заповнення бланка (проекта) таблиці на основі компонентів ER-моделі наведено у табл.2.

Таблиця 2

Таблиця EMPLOYEE

Найменування стовпця	Тип ключа	Тип властивості обов'язковості NOT NULL	Таблиця-посилання для FK	Стовпець таблиці - посилання для FK	Тип даних	Довжина
ID	PK				NUMBER	7
LAST_NAME		NOT NULL			CHAR	25
FIRST_NAME					CHAR	25
DEPT_ID	FK	NOT NULL	DEPARTMENT	ID	NUMBER	7

*Перетворення атрибутів* у стовпці означає присвоєння стовпцям імені

атрибутів з ER-моделі, при цьому обов'язкові атрибути, позначені символом «\*», перетворюються в стовпці з властивістю NOT NULL.

*Перетворення унікальних ідентифікаторів у первинні ключі* означає здійснення таких дій:

1) перетворити унікальний ідентифікатор, позначений у ER-моделі символом «#», у стовпці з позначенням їх як первинного ключа РК. Унікальний ідентифікатор, що містить тільки один атрибут, перетвориться в одно стовпчиковий РК, що буде мати властивості обов'язковості NOT NULL і унікальності;

2) позначити всі альтернативні унікальні ідентифікатори унікальним ключем УК із властивістю NOT NULL;

3) якщо унікальний ідентифікатор сутності включає зв'язок, що позначено поперечною рисою, то додати додатковий стовпець з визначенням зовнішнього ключа для кожного зв'язку і позначити його як РК і FK. Навіть, якщо стовпці з FK входять у первинний ключ, варто додавати їх наприкінці або праворуч від інших стовпців. Задайте унікальне ім'я для кожного стовпця з FK.

*Перетворення зв'язків у зовнішні ключі* виконується для кожного типу зв'язків окремо. Варто перетворювати найбільш застосовувані типи - “багато до одного” і “один до одного”, а інші слід привести до одного з них. Якщо зв'язок є частиною унікального ідентифікатора, то він перетвориться згідно наведеній вище процедури.

Для зв'язків “багато до одного” варто взяти первинний ключ на стороні “один” і включити його в таблицю на стороні “багато” як зовнішній ключ. Назву зовнішнього ключа утворити з імені таблиці на стороні “один”, знака “\_” і імені атрибута первинного ключа. Цей спосіб застосовується і до рекурсивних зв'язків. Для обов'язкових зв'язків слід позначити стовпці властивістю NOT NULL.

Зв'язок “багато до багатьох” перетвориться в додаткову таблицю-сутність, що зв'язується з таблицями-учасниками зв'язку додатковими зв'язками типу “багато до одного” з напрямком “один” з боку таблиць-учасників зв'язку і

напрямок “багато” з боку додаткової таблиці-сутності. Властивість обов'язковості первинного зв'язку перетвориться у властивість обов'язковості напрямку “багато” додаткових зв'язків. Далі додаткові зв'язки перетворяться за процедурою перетворення зв'язку типу “багато до одного”.

Для необов'язкових зв'язків “один до одного” варто включити додатковий стовпець як зовнішній ключ у таблицю на будь-якій стороні зв'язку і позначити його ключем UK.

Для обов'язкових зв'язків “один до одного” варто включити додатковий стовпець як унікальний зовнішній ключ (UK і FK) у таблицю на обов'язковій стороні зв'язку і позначити його властивістю NOT NULL. Властивість унікальності необхідна для забезпечення виконання умов зв'язку “один до одного”.

### 3. Порядок роботи із системою автоматизованого проектування бази даних Oracle SQL Developer Data Modeler

Oracle SQL Developer Data Modeler – це комплексне рішення, що дозволяє розробникам проектувати семантичні моделі взаємозв'язків об'єктів для наступного перетворення їх у повноцінні бази даних. Продукт підтримує інфологічне, серверне моделювання і моделювання типів даних, пропонуючи можливості багаторівневого проектування і побудови концептуальних діаграм сутностей і зв'язків. Користувачі можуть створювати, розширювати і модифікувати моделі, а також порівнювати їх із вже існуючими і виконувати реверс-інжиніринг баз і моделей даних. Oracle SQL Developer Data Modeler може входити як складова до інтегрованого програмного середовища адміністрування, проектування бази даних Oracle SQL Developer.

Oracle SQL Developer Data Modeler пропонує безліч функціональних можливостей для моделювання даних і баз даних, включаючи:

- візуальне моделювання взаємозв'язків між сутностями з підтримкою систем позначень - нотацій Баркера (Barker) і Бахмана (Bachman), щоб розробники могли переключатися між моделями для задоволення потреб

клієнтів для створення і збереження різних візуальних представлень моделей;

- прискорене перетворення ER-моделей у серверні моделі – трансформація усіх правил і рішень, зроблених на концептуальному рівні, у серверну модель, деталі в якій уточнюються й оновлюються;

- розділення серверної і фізичної моделей даних дозволяє розробникам створювати одну серверну модель для різних версій бази даних чи для різних баз даних, включаючи Oracle Database, IBM DB2, а також Microsoft SQL Server.

Порядок взаємодії із системою автоматизованого проектування Oracle DataModeler заснований на розглянутих раніше етапах моделювання і проектування баз даних. Послідовність процесу проектування передбачає виконання наступних дій:

1. Запуск і налаштування середовища проектування. Налаштування виконується за допомогою вибору пунктів меню «Tools / Preferences» і передбачає визначення місця зберігання репозитарію результатів проектування, вибору системи позначень у моделях, вибору способу моделювання типів даних та інше. У вікні “Preferences” потрібно обрати пункт “Data Modeler” та визначити папки для розташування елементів репозитарію результатів проектування, а саме “Default Design Directory”, “Default Import Directory”, “Default Save Directory”, “Default Reports Directory”. В якості місця для папок розташування обирайте власну папку на дисковому ресурсі спільного використання Z:\. Далі необхідно розкрити пункт “Data Modeler / Diagram / Logical Model” та обрати модель системи позначень “Notation Type” як модель “Barker”. Також для спрощення використання загальноприйнятими (стандартними) типами даних необхідно встановити спосіб моделювання типів даних через розкриття пункту “Data Modeler / Model” і встановлення значення для “Datatype” у “Logical” та позначку для “Nulls Allowed”. Можна також визначити прийнятні типи даних “Preferred Logical Types”, які будь найчастіше використані при моделюванні і проектуванні бази даних, наприклад, numeric, char, varchar, date.

2. Створення діаграми ER-моделі. Для проектування ER-моделі необхідно

запустити «Browser», в дереві якого відтворюються результати етапів моедлювання і проектування бази даних, зокрема, «Logical Model» – ER-модель, «Relational Models» – серверна модель. Перед створенням діаграми необхідно підготувати робочий простір з палітрою інструментів створення діаграми. Для цього на пункті «Logical Model» або викликати «Model Properties» встановити позначку «Visible» або правою кнопкою миші із впливаючого меню обрати пункт «Show». Внаслідок цих дій з'явиться робочий простір нової діаграми та меню палітри інструментів побудови елементів діаграми.

Після створення робочого простору нової діаграми переходять до введення необхідних за завданням сутностей. Для цього використовувати відповідну палітру інструментів. При створенні сутності ввести ім'я сутності (Name), позначення сутності в зв'язках (Short Name, Preferred Abbreviation) і ім'я сутності на діаграмі або його синоніму (Synonyms). Потім викликати властивості сутності (Property) і ввести опис атрибутів (Attributes), первинного й унікального ключів (UID). Після створення сутностей між ними встановити необхідні зв'язки. При створенні зв'язку необхідно визначити назву для кожного напрямку зв'язку (From і To), властивості обов'язковості і ступінь. Після завершення проектування створену діаграму зберегти за допомогою пункту меню «File / Save As», встановивши ім'я проекту. В проекті може знаходитися тільки одна ER- діаграма і декілька серверних (реляційних) діаграм, що їй відповідають. Тому видалити ER- діаграму означає видалення всього проекту, який зберігається у файлі типу dmd та папці з назвою проекту. В проекті можливо видаляти окремі його елементи – сутності зв'язки, серверні (реляційні) моделі, таблиці, тощо.

3. Перетворення ER-моделі в серверну модель бази даних. Виконується в автоматичному режимі за допомогою «Engineer to Relational Model» у палітрі інструментів «Logical Model» або в її спливаючого меню. Налаштування режиму перетворення виконується у закладах утвореного вікна та дерева елементів моделі, які будуть перетворюватися. Запуск перетворення виконується кнопкою «Engineer».

4. Перевірка результатів перетворення і внесення змін і доповнень у серверну модель проекту бази даних здійснюється за допомогою вибору “Relational Models” у вікні «Browser». У дереві проектів розкрити опис реляційних таблиць і викликати відповідне вікно ”Table Properties” для докладного перегляду і редагування компонентів серверної моделі проекту. Для перегляду і внесення змін до графічного представлення серверної моделі бази даних необхідно обрати закладки “Relational” де буде побудована діаграма. Виклик закладки здійснюється обранням позначки “Visible” у “Model Properties” або пункту “Show” із спливаючого меню реляційної моделі. Після завершення проектування створену модель необхідно зберегти за допомогою пункту меню «File / Save». Результати проектування можна подати у виді звіту у форматі pdf (або html) за допомогою пункту меню «File / Report».

5. Генерація файлів з командами створення об'єктів бази даних за сформованим описом компонентів проекту виконується в пункті меню «View / DDL File Editor» або у серверній діаграмі кнопкою “Generate DDL” палітри інструментів. Перед запуском процесу генерації необхідно вказати реляційну модель та тип бази, для якої будуть генеруватися команди, та опції самого процесу генерації. Результати генерації можна переглянути у вікні, що з'явилося, після завершення процесу. Збереження результатів виконується кнопкою “Save” у папці розташування результатів проектування.

6. Для верифікації результатів проектування бази даних обов'язково передбачається зворотний процес відновлення ER-моделі з існуючої серверної моделі бази даних – процес реверс-інжиніринг. Цей процес передбачає виконання в «Browser» опису реляційних таблиць за бланками таблиць з наступним їх перетворенням у сутності ER-моделі:

– на пункті «Relational Models» виклик правою кнопкою миші підпункту «New Relational Model» і заповнення опису структури таблиці, виходячи з представлених у бланку таблиці даних;

– на пункті «Relational» в дереві проекту виклик правою кнопкою миші підпункту “Engineer to Logical Model” або у серверній діаграмі кнопкою



“Engineer to Logical Model” з палітри інструментів виклик вінка процесу перетворення. Настроювання режиму перетворення виконується у закладах утвореного вікна та дерева елементів моделі, які будуть перетворюватися. Запуск перетворення виконується кнопкою «Engineer».

#### 4. Контрольне завдання

1. Розглядається об'єкт автоматизації – постачальна компанія товарів, предметна область якої складається з наступних правил:

- компанія постачає товари на замовлення своїх клієнтів з реєстру;
- замовлення може складатися з декількох товарів, але повинно включати хоча б один товар;
- клієнт може подати (зробити) декілька замовлень, але кожне замовлення повинно надходити тільки від одного клієнта;
- замовлення може прийняти тільки один службовець компанії, який може оформляти (підготувати) декілька замовлень, що надійшли від клієнтів;
- службовець повинен працювати завжди на одному й тому ж робочому місці у компанії, а робоче місце може використовуватися тільки одним службовцем;
- декілька службовців можуть мати свого керівника, який у свою чергу може мати тільки одного начальника або директора, директор не є підпорядкованою особою.

Постає завдання по моделюванню і проектуванню бази даних постачальної компанії згідно наступного порядку:

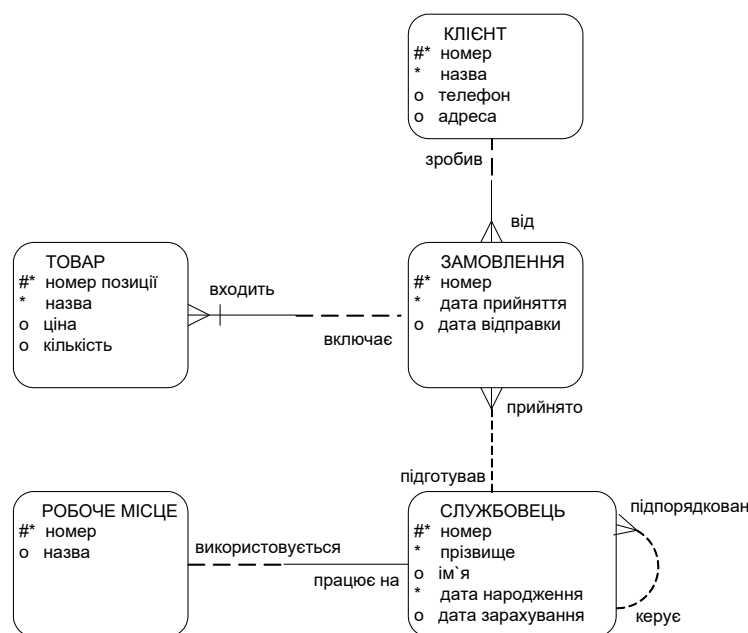
1.1) Створити проект бази даних по наведеній діаграмі сутностей. Перевірити відповідність поданої діаграми сутностей опису предметної області об'єкта автоматизації. При необхідності внести до діаграми зміни.

1.2) Виконати всі етапи процесу проектування, побудувавши ER-діаграму та перетворити її в діаграму серверної моделі бази даних з описом всіх таблиць. При позначенні компонентів проекту бази використовувати латинські букви, а також наступні формати даних NUMBER (NUMERIC) - для номерів, цін,

кількості, CHAR - для телефону, VARCHAR2 (VARCHAR) - для назв, прізвища, ім'я, DATE - для дат. В назвах сутностей і таблиць застосувати наступні позначення (повні назви і скорочення): клієнт – klient, kli; замовлення – zamov, zam; товар – tovar, tov; службовець – sluzbov, slu; робоче місце – robmisc, rob.

1.3) У сгенерованій серверній моделі стовпці таблиць упорядкувати відповідно до атрибутів діаграми сутностей. Виконати генерування Report-звіту про результати проектування серверної моделі.

1.4) Подати у звіті діаграми ER-моделі, серверної моделі та описи проектів таблиць у вигляді бланків таблиць на підставі даних, отриманих з Report-файлу. Надати пояснення про відповідність поданої діаграми сутностей опису предметної області об'єкта автоматизації.



2. Виконати завдання по зворотному проектуванню моделі предметної області об'єкта автоматизації – реєстр службовців установи, що складається з відділів, згідно наступного порядку:

2.1) Побудувати серверну модель проекту бази, поданої у вигляді наведених бланків таблиць. Додати до серверної моделі опис індексу за стовпцями NAME, LAST\_NAME.

2.2) Відновити ER-модель із серверної моделі бази даних. Порівняти

побудовану серверну модель і утворену ER-модель.

2.3) За аналізом утвореної ER-моделі побудувати опис предметної області об'єкта автоматизації у вигляді правил за прикладом п.1.

2.4) Включити у звіт утворену ER-модель, опис предметної області та надати пояснення про головну відмінність у представленні діаграм ER-моделі від серверної моделі.

Таблиця відділів DEPARTMENT

Найменування стовця	Тип ключа	Тип властивості обов'язковості NOT NULL	Таблиця-посилання для FK	Стовпець таблиці-посилання для FK	Тип даних	Довжина
ID	PK	NOT NULL			NUMBER	7
NAME		NOT NULL			VARCHAR2	25

Таблиця службовців EMPLOYEE

Найменування стовця	Тип ключа	Тип властивості обов'язковості NOT NULL	Таблиця-посилання для FK	Стовпець таблиці-посилання для FK	Тип даних	Довжина
ID	PK	NOT NULL			NUMBER	7
LAST_NAME		NOT NULL			VARCHAR2	25
FIRST_NAME					VARCHAR2	25
DEPT_ID	FK	NOT NULL	DEPARTMENT	ID	NUMBER	7

**Комп'ютерний практикум № 2.****СТВОРЕННЯ СТРУКТУРИ БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОМАНДАМИ МОВИ ВИЗНАЧЕННЯ ДАНИХ****1. Мета та основні завдання практикуму**

Мета роботи: ознайомитися з командами мови визначення даних при проектуванні бази даних і набути навички роботи з інструментальними засобами взаємодії з промисловим сервером бази даних Oracle

Порядок виконання роботи:

- вивчити команди мови визначення даних;
- визначити основні компоненти опису даних в інформаційній системі;
- згідно умов контрольного завдання створити інформаційні об'єкти в базі даних;
- підготувати звіт за результатами створення бази даних.

**2. Основні теоретичні відомості****2.1. Основні види структур даних у базі даних**

База даних може містити кілька різних структур даних. Кожна структура повинна бути описана на стадії проектування бази даних, щоб на етапі кодування її можна було створити. Структурами даних є наступні інформаційні об'єкти бази даних наведені у табл.3

Таблиця 3

Структура	Призначення
Таблиця (Table)	Зберігати дані
Представлення (View)	Логічно представляти підмножини даних з однієї чи декількох таблиць
Послідовність (Sequence)	Генерувати значення для первинного ключа
Індекс (Index)	Підвищувати продуктивність запитів до таблиць

Особливості створення таблиць як об'єктів бази даних:

- 1) Таблиці можуть створюватися в будь-який момент – навіть під час

роботи користувачів з базою даних.

2) Заздалегідь задавати розмір таблиці не потрібно. В остаточному підсумку він визначається розміром простору, займаного базою даних у цілому. Важливо оцінити, який простір буде використовувати таблиця згодом.

3) Структуру таблиці можна оперативно змінити.

## 2.2. Опис команди створення таблиць

Створити таблицю для збереження даних можна за допомогою команди CREATE TABLE мови визначення даних Data Definition Language (DDL). Команди DDL є підмножиною команд мови SQL і використовуються для створення, зміни чи видалення структур бази даних. Ці команди негайно впливають на базу даних і, крім того, заносять службову інформацію в словник даних бази. Мова SQL є галузевим стандартом мови реляційних баз даних. Всі основні промислові системи управління базами даних (СУБД) підтримують мову SQL у будь-якій формі.

Для створення таблиць користувач повинен мати привілей CREATE TABLE і виділену область збереження даних (табличний простір), де можна створювати об'єкти. Надаючи користувачам зазначену привілею, адміністратор бази даних використовує команди мови керування даними Data Control Language (правила їх застосування вивчаються в комп'ютерному практикумі №5).

Узагальнений синтаксис команди створення таблиці:

```
CREATE TABLE [схема.]таблиця
( стовпець тип_даних [DEFAULT вираз] [обмеження_стовпця], ...
[, обмеження_таблиці] ) ;
```

де

*схема* – ім'я користувача бази даних, власника таблиці, у схемі якого буде зберігатися дані цієї таблиці,

*таблиця* – найменування таблиці,

DEFAULT *вираз* – визначає значення за замовчуванням, використовуване

при відсутності значення в команді вставки нового рядка,

*стовпець* – ім'я стовпця,

*тип\_даних* – тип даних і довжина стовпця,

*обмеження\_стовпця* – правило цілісності даних як частина визначення стовпця (якщо декілька обмежень, то їх визначення відокремлені один від одного “”),

*обмеження\_таблиці* – правило цілісності даних як частина визначення таблиці (якщо декілька обмежень, то їх визначення відокремлені один від одного “;”).

Будь-яка команда мови SQL закінчується вказівкою відправлення її на виконання. Така вказівка подається у виді символу “;” наприкінці команди або у виді символу “/” на початку наступної рядка при введенні команди.

До команди або до послідовності команд можна надавати коментар у вигляді тексту, який поділяється на рядковий або блоковий.

Рядковий коментар починається с двох символів мінус “--”, а далі йде текст коментаря до звершення цього рядка. На наступний рядок дія коментаря не переходить. Блоковий коментар починається з символів “/\*”, далі йде коментар на декількох рядках, який повинен закінчитися символами “\*/”. Якщо коментар розташовано усередині команди, то при її виконанні коментар не вважається помилкою і опускається.

Схема – це іменованій, за назвою користувача-власника, набір об'єктів бази даних. Об'єктами схеми є логічні структури, що безпосередньо посилаються на дані в базі даних. До об'єктів схеми відносяться таблиці (TABLE), представлення (VIEW), синоніми (SYNONYM), послідовності (SEQUENCE), збережені у базі програмні модулі (PROCEDURE, FUNCTION, TRIGGER), індекси (INDEX), кластери і зв'язки бази даних.

Таблиці, посилання на які застосовуються в обмеженнях у означені команди, повинні існувати в цій же базі даних. Якщо таблиця не належить користувачу, що визначив обмеження, ім'я власника повинне бути задано як префікс до найменування таблиці, на яку посилається дане обмеження.

Параметр DEFAULT дозволяє призначити значення стовпцю за замовчуванням. Цей параметр виключає появу невизначених значень при вставці рядка без конкретного значення для даного стовпця. Значенням за замовчуванням може бути рядок, вираз чи певна системна SQL-функція, наприклад, SYSDATE - поточна дата і час у СУБД або USER - ім'я поточного користувача бази даних, але не ім'я іншого стовпця або псевдо стовпця (NEXTVAL, CURRVAL і т.д.). Тип значення за замовчуванням повинний збігатися з типом даних стовпця.

Найменування таблиць і ім'я стовпців бази даних повинні відповідати стандартним правилам присвоєння імен всім об'єктам бази даних:

1) Імена повинні починатися з букви і можуть включати від 1 до 30 символів;

2) Імена повинні містити тільки символи A-Z, a-z, 0-9, (підкреслення), \$ і # (символи "\$" і "#" припустимі, але не рекомендуються). Прописні і малі літери в іменах не розрізняються;

3) Імена не повинні дублювати ім'я іншого об'єкта, що належить цьому ж користувачу сервера бази даних;

4) Імена не повинні збігатися із зарезервованими словами СУБД.

Рекомендації з присвоєння імен:

- для таблиць і інших об'єктів бази даних краще використовувати описові імена, що мають визначене значення. Наприклад, первинні ключі рекомендується іменувати скороченим ім'ям таблиці і суфіксом \_PK (DEPT\_PK), зовнішні ключі – скороченими іменами “дочірньої” і “батьківської” таблиць і суфіксом \_FK (EMP\_DEPT\_FK), унікальні ключі – скороченими іменами таблиці, ключового стовпця і суфіксом \_UK (EMP\_LAST\_NAME\_UK), обмеження NOT NULL – скороченими іменами таблиці, стовпця і суфіксом \_NN (EMP\_LAST\_NAME\_NN), індекси – скороченими іменами таблиці, індексованого стовпця і суфікса \_I (EMP\_LAST\_NAME\_I);

- привласнюйте однакові імена ідентичним об'єктам, що знаходяться в

різних таблицях. Наприклад, стовпець номера відділу називається DEPT\_ID і в таблиці S\_EMP, і в таблиці S\_REGION.

### 2.3. Типи даних у базі даних

Існує достатня розмаїтість типів даних у базі для адекватного представлення значень різних властивостей. Основними типами даних є символ, число, дата і бінарне число. У СУБД Oracle встановлені такі подання основних типів (див. табл.4).

Таблиця 4

Тип даних	Опис
VARCHAR2( <i>розмір</i> )	Символьні значення змінної довжини, що не перевищує заданого <i>розміру</i> . Мінімальна довжина дорівнює 1, максимальна – 4000. Значення розміру обов'язательно.
CHAR( <i>розмір</i> )	Символьні значення фіксованої довжини, рівної <i>розміру</i> . Довжина за замовчуванням складає 1, максимальна – 255. Значення розміру за замовчуванням 1.
NUMBER	Число з крапкою, що плаває, з точністю 38 значущих цифр.
NUMBER( <i>p,s</i> )	Числове значення, що має максимальну точність <i>p</i> від 1 до 38 і максимальний масштаб <i>s</i> ; точність – це загальна кількість десяткових цифр, а масштаб – кількість цифр праворуч від десяткової крапки.
DATE	Значення дати і часу між 1 січня 4712 до н.е. і 31 грудня 4712 н.е.
LONG / CLOB	Символьні значення змінної довжини розміром до 2 гігабайтів. У таблиці допускається тільки один стовпець типу LONG. В останній версіях СУБД Oracle використовується тип CLOB для символьних значень розміром до 4 гігабайт.
RAW, LONG RAW / BLOB	Аналогічні, відповідно, типам даних VARCHAR2 і LONG, але використовуються для збереження байт-орієнтованих або двійкових даних, що не повинні інтерпретуватися сервером Oracle. В останній версіях СУБД Oracle використовується тип BLOB для двійкових даних розміром до 4 гігабайт.

Приклад синтаксису визначення стовпців при створенні таблиці:

```
CREATE TABLE friend (
```



```
id NUMBER(5),
date_meeting DATE DEFAULT sysdate, ...
last_name VARCHAR2(25), ... );
```

#### 2.4. Опис обмежень бази даних

Обмеження діють на рівні бази даних в цілому як настанови (правила).

Вони використовуються в наступних цілях:

1) Для реалізації правил забезпечення цілісності даних на рівні таблиці при операціях вставки, оновлення і видалення рядків. Якщо обмеження задане, то успішне виконання операції без його дотримання неможливо;

2) Для запобігання видалення таблиці, якщо вона залежить від інших таблиць;

3) Для оснащення прикладними (специфічними до процесів обробки даних) правилами інформаційних систем, які взаємодіють з базою даних.

Типи обмежень цілісності даних наведені в табл.5.

Опис всіх обмежень зберігається в словнику даних СУБД. Присвоєння обмеженням імен, що несуть якесь суттєве значення, полегшує посилання на них. Імена обмежень повинні відповідати стандартним правилам присвоєння імен об'єктам. Якщо обмеженню явно не привласнюється ім'я, то СУБД Oracle створює його у форматі SYS\_Cn, де n – ціле число, що забезпечує унікальність імені обмеження в базі.

Таблиця 5

Обмеження	Опис
NOT NULL	Означає, що даний стовпець не може містити невизначених значень.
UNIQUE	Указує, що чи стовпець набір стовпців містить значення, що повинні бути унікальними для всіх рядків таблиці.
PRIMARY KEY	Унікально ідентифікує кожен рядок таблиці.
FOREIGN KEY	Встановлює і підтримує відношення між даним стовпцем і стовпцем таблиці, на яку робиться посилання, за допомогою правила зовнішнього ключа.
CHECK	Задає довільну умову, що повинне перевірятися.

Звичайно обмеження створюються одночасно зі створенням таблиці. Але додавати їх можна і після створення таблиці. Крім того, обмеження можуть бути тимчасово заборонені, тобто призупинені у дії.

Обмеження можуть бути задані у спосіб на одному з двох рівнів опису у команді створення таблиці (див. табл.6) або окремою командою, що вносить зміни до вже існуючій таблиці (див. п.2.6.3).

Таблиця 6

Рівень	Опис
Стовпець	Посилається на один стовпець і описується в межах характеристик відповідного стовпця. Дозволяє задати правило цілісності будь-якого типу.
Таблиця	Посилається на один або декілька стовпців і описується окремо від визначення стовпців у даній таблиці. Дозволяє задати будь-які обмеження, крім NOT NULL.

Узагальнений синтаксис подання обмеження у спосіб на рівні стовпця у команді створення таблиці (в узагальненому синтаксисі команди створення таблиці подане як *обмеження\_стовпця*):

```
стовпець тип_даних [DEFAULT вираз]
[ [CONSTRAINT ім'я_обмеження] тип_обмеження] ...
```

На рівні стовпця (у кінці визначення стовпця) може бути задано декілька обмежень до цього стовпця, розділених пробілом.

Приклад синтаксису:

```
CREATE TABLE friend (
  id NUMBER(5) PRIMARY KEY, ... );
або
CREATE TABLE friend (
  id NUMBER(5) CONSTRAINT friend_pk PRIMARY KEY, ... );
```

Узагальнений синтаксис подання обмеження у спосіб на рівні таблиці у команді створення таблиці (в узагальненому синтаксисі команди створення таблиці подане як *обмеження\_таблиці*):

```
стовпець тип_даних [DEFAULT вираз], ...
```

[ [CONSTRAINT ім'я\_обмеження] тип\_обмеження (стовпець, ...) ], ...

На рівні таблиці (у кінці визначення таблиці) може бути задано декілька обмежень, розділених комою, як до різних стовпців, так і до одного стовпця.

Приклад синтаксису:

```
CREATE TABLE friend (
  id NUMBER(5), ... ,
  PRIMARY KEY (id), ... );
```

або

```
CREATE TABLE friend (
  id NUMBER(5), ... ,
  CONSTRAINT friend_pk PRIMARY KEY(id), ... );
```

2.4.1.Обмеження NOT NULL забороняє невизначені значення в стовпці. Стовпці, не зв'язані обмеженням NOT NULL, можуть містити невизначені значення NULL за замовчуванням. Це обмеження може бути задано тільки у спосіб на рівні стовпця, а не на рівні таблиці.

Приклад синтаксису:

```
CREATE TABLE friend ( ... ,
  phone VARCHAR2(15) NOT NULL, ... ,
  last_name VARCHAR2(25)
  CONSTRAINT friend_last_name_nn NOT NULL, ...);
```

У цьому прикладі обмеження NOT NULL задане для стовпця phone, що містить символічні рядки змінної довжини, але не більш 15 символів. Оскільки ім'я обмеженню не визначено, то СУБД створить ім'я самостійно. Також обмеження NOT NULL задане для стовпця last\_name, що містить рядок змінної довжини не більш 25 символів, але ім'я обмеження задане явно – friend\_last\_name\_nn.

2.4.2.Обмеження UNIQUE визначає стовець або набір стовпців як деякий унікальний ключ. У таблиці не існує двох рядків з однаковим значенням

цього ключа. Невизначені значення можливі, якщо унікальний ключ заснований на одному стовпці, тобто не є складеним.

Обмеження UNIQUE можуть застосовуватися при поданні у спосіб як на рівні стовпця, так і таблиці. Складений унікальний ключ створюється тільки за допомогою визначення у спосіб на рівні таблиці. Для унікального ключа автоматично створюється унікальний індекс.

Приклад синтаксису визначення у спосіб на рівні стовпця:

```
CREATE TABLE friend ( ... ,  
    phone VARCHAR2(15) UNIQUE, ... ,  
    last_name VARCHAR2(25) CONSTRAINT s_emp_uk UNIQUE, ... );
```

Приклад синтаксису визначення у спосіб на рівні таблиці:

```
CREATE TABLE friend ( ... ,  
    phone VARCHAR2(15), ... ,  
    last_name VARCHAR2(25), ... ,  
    UNIQUE (phone), ... ,  
    CONSTRAINT s_emp_uk UNIQUE (last_name), ... );
```

Обмеження UNIQUE не є аналогічним до обмеження PRIMARY KEY і не є його синонімом.

2.4.3. Обмеження PRIMARY KEY встановлює (визначає) первинний ключ таблиці. Кожна таблиця може мати тільки один первинний ключ. Обмеження PRIMARY KEY – це стовпець або набір стовпців, за значеннями яких однозначно ідентифікується кожен рядок у таблиці. Це обмеження вимагає унікальності значень конкретного стовпця або набору стовпців і гарантує відсутність у них невизначених значень.

Обмеження PRIMARY KEY можуть бути задані у спосіб як на рівні стовпця, так і таблиці. Складений первинний ключ створюється тільки на рівні таблиці. Для стовпця, зв'язаного обмеженням PRIMARY KEY, автоматично створюється унікальний індекс.

Приклад синтаксису визначення у спосіб на рівні стовпця:

```
CREATE TABLE s_emp ( ... ,  
id NUMBER(7) CONSTRAINT s_emp_pk PRIMARY KEY, ...);
```

Приклад синтаксису визначення у спосіб на рівні таблиці:

```
CREATE TABLE s_item ( ... ,  
ord_id NUMBER(7), id NUMBER(7), ... ,  
CONSTRAINT s_item_pk PRIMARY KEY(ord_id, id), ...);
```

2.4.4.Обмеження FOREIGN KEY, яке спрямоване на забезпечення в базі даних цілісності посилань, задає (позначає) стовпець або набір стовпців як зовнішній ключ і встановлює зв'язок його значень із значеннями первинного або унікального ключа тій же самій таблиці або іншій таблиці.

Значення зовнішнього ключа повинне збігатися з існуючим значенням первинного/унікального ключа таблиці-посилання у визначенні зовнішнього ключа або бути невизначеним (NULL-значенням).

Обмеження FOREIGN KEY можуть бути задані у спосіб як на рівні стовпця, так і таблиці. Складений зовнішній ключ створюється тільки за допомогою визначення у спосіб на рівні таблиці.

Зовнішні ключі засновані на значеннях даних і тому є тільки логічними, а не фізичними покажчиками. Зовнішній ключ, що є частиною первинного ключа, не може бути з невизначеним значенням, оскільки жодна частина первинного ключа не може мати значення NULL.

Зовнішній ключ призначається на стовпець в “дочірній” (підпорядкованій, пов’язаній) таблиці-визначення, а таблиця, що містить стовпець, на який робиться посилання, слугує за “батьківську” таблицю-посилання. Зовнішній ключ описується комбінацією наступних ключових слів (фраз):

1) FOREIGN KEY задає стовпець у “дочірній” таблиці-визначення обмеження при його поданні у спосіб на рівні таблиці;

2) REFERENCES вказує “батьківську” таблицю-посилання і стовпець у неї, який повинен мати обмеження первинного або унікального ключів;

3) ON DELETE CASCADE визначає, що якщо видаляється рядок у “батьківській” таблиці-посилання, то залежні рядки в “дочірній” таблиці-визначення також будуть вилучені (залежність встановлюється за однаковим значенням зовнішнього ключа у “дочірній” таблиці-визначенні та первинного/унікального ключа у “батьківській” таблиці-посиланні). Якщо параметр ON DELETE CASCADE відсутній, то рядок “батьківської” таблиці-посилання не може бути вилучений, якщо існують залежні рядки в “дочірній” таблиці-визначенні.

Приклад синтаксису подання обмеження зовнішнього ключа у спосіб на рівні стовпця (для dept\_id) і на рівні таблиці (для region\_id):

```
CREATE TABLE s_emp ( ... ,
    region_id NUMBER(2), ... ,
    dept_id NUMBER(7)
    CONSTRAINT s_emp_dept_fk
    REFERENCES s_dept (id) ON DELETE CASCADE, ... ,
    CONSTRAINT s_emp_region_fk FOREIGN KEY (region_id)
    REFERENCES s_region (id), ... );
```

Примітка. Потрібно чітко розуміти контекст використання словосполучення “зовнішній ключ”, “первинний ключ”, “унікальний ключ”. Якщо, річ йде про застосування правила обмеження цілісності, то потрібно використовувати означення “обмеження цілісності зовнішній ключ” і т.д. Проте, якщо річ йде про стовпець (або його значення), який має встановлене визначення обмеження цілісності, то можливе використання скороченого означення “зовнішній ключ” і т.д., але більш точним означенням є – “стовпець-зовнішній ключ”, стовпець з обмеженням цілісності зовнішній ключ і т.д. Доцільно використовувати і більш точні означення для “дочірній”, “батьківській” таблиці, “залежні” рядки.

2.4.5.Обмеження CHECK визначає умову, якій повинен задовольняти

кожен рядок таблиці. Ця умова може містити ті ж конструкції, що й у запитах до таблиці, за винятком наступного:

- посилань на псевдо стовпці CURRVAL (поточне значення послідовності значень), NEXTVAL (вибір наступного значення послідовності значень), LEVEL (номер рівня ієрархії отриманих даних) і ROWNUM (номер рядка отриманих даних);

- звертань до функцій SYSDATE, UID, USER і USERENV;

- запитів з посиланнями на інші значення в інших рядках.

Обмеження CHECK може бути задане як на рівні стовпця, так і таблиці. Синтаксис умови цього обмеження може використовувати будь-який стовпець таблиці, а не тільки той, для якого задане дане обмеження.

Приклад синтаксису:

```
CREATE TABLE s_item ( ...  
price NUMBER(7,2)  
CONSTRAINT s_item_price_ck  
CHECK (price>=1000 AND price<=2000), ...
```

## 2.5. Створення таблиці на основі бланку проекту таблиці

Власне таблиці створюються за допомогою команди CREATE TABLE на основі відомостей з бланку проекту таблиці, заповненого при проектуванні бази даних. З метою документування і для можливого повторного використання можна зберегти синтаксис команди у спеціальному файлі команд (командному файлі).

Порядок створення таблиці на основі бланку проекту:

1. Створіть командний файл. Почніть із синтаксису команди CREATE TABLE і вкажіть ім'я таблиці.

2. Перенесіть із бланку проекту таблиці в командний файл імена стовпців, типи даних і їхню довжину. Визначення стовпців розділите комами.

3. Обмеження NOT NULL задайте у спосіб на рівні стовпців у всіх випадках, крім стовпців, зв'язаних обмеженням PRIMARY KEY.

4. Задайте обмеження PRIMARY KEY у спосіб на рівні стовпця, якщо воно відноситься тільки до одного стовпця, або у спосіб на рівні таблиці, якщо воно відноситься до декількох стовпців.

5. Задайте обмеження UNIQUE, CHECK і FOREIGN KEY.

6. Збережіть і виконайте командний файл.

Розглянемо приклад створення командного файлу. Необхідно створити таблицю бази даних S\_DEPT на основі бланку проекту таблиці, наведеного у табл.7.

Таблиця 7

Таблиця S\_DEPT

Найменування стовпця	Тип ключа	NOT NULL	Таблиця-посилання FK	Стовпець таблиці-посилання FK	Тип даних	Довжина
ID	PK				NUMBER	7
NAME	UK	NOT NULL			CHAR	25
REGION_ID	FK, UK		REGION	ID	NUMBER	7

```
CREATE TABLE s_dept
( id NUMBER(7) CONSTRAINT s_dept_pk PRIMARY KEY,
  name VARCHAR2(25) CONSTRAINT s_dept_name_nn NOT NULL,
  region_id NUMBER(7)
  CONSTRAINT s_dept_s_region_fk REFERENCES s_region (id),
  CONSTRAINT s_dept_uk UNIQUE (name, region_id) );
```

Обмеження S\_DEPT\_ID\_PK визначене у спосіб на рівні стовпця робить стовпець ID первинним ключем таблиці S\_DEPT. Це обмеження означає, що в таблиці не може бути двох відділів з однаковими номерами і номер відділу не може бути невизначеним (NULL).

Обмеження S\_DEPT\_NAME\_NN визначене у спосіб на рівні стовпця означає, що кожен відділ у цій таблиці має визначене ім'я.



Обмеження S\_DEPT\_S\_REGION\_FK на рівні стовпця означає, що номер регіону, введений у таблицю S\_DEPT, має відповідне (таке ж) значення в таблиці S\_REGION. Перш, ніж задавати це обмеження, необхідно створити таблицю S\_REGION, а також обмеження PRIMARY KEY або UNIQUE для стовпця ID цієї таблиці.

Обмеження S\_DEPT\_UK визначене у спосіб на рівні таблиці визначає стовпці NAME і REGION\_ID як складений унікальний ключ, так щоб жодна комбінація імені відділу й імені регіону не могла з'явитися в цій таблиці більш одного разу.

## 2.6. Створення таблиці за образом (шаблоном) іншої таблиці

Альтернативним методом створення таблиці є використання фрази AS у команді CREATE TABLE. Він дозволяє за описом структури існуючої в базі даних таблиці створювати нову таблицю як її копію й одночасно вставити до неї рядки із існуючої, повернуті в результаті виконання підзапиту в синтаксисі фрази AS.

Синтаксис альтернативного методу створення таблиці за шаблоном:

```
CREATE TABLE таблиця [(стовпець, ...)] AS підзапит ;
```

де

*стовпець* – ім'я стовпця,

*підзапит* – команда SELECT, що визначає стовпці і рядки вже існуючої таблиці, які будуть введені і вставлені в нову таблицю після її створення.

Особливості застосування альтернативного методу створення таблиць:

- таблиця буде створена із зазначеними іменами стовпців і в цю таблицю будуть вставлені рядки, які повернуті в результаті виконання підзапиту;

- визначення стовпця може містити тільки його ім'я, значення за замовчуванням і обмеження цілісності стовпця. Стовпці нової таблиці успадковують з відповідних стовпців таблиці підзапиту тільки обмеження NOT NULL;

- якщо явно задане визначення стовпців, то їхня кількість повинна

збігатися з кількістю стовпців підзапиту. Якщо визначення стовпців не задано, то їхні імена в створюваній таблиці будуть такими, як і в таблиці підзапиту.

Приклад синтаксису:

```
CREATE TABLE s_dept_duplicate AS  
SELECT * FROM s_dept;
```

В результаті виконання даної команди буде створена повна копія таблиці `s_dept`, як за структурою, так і за наповненням даними.

```
CREATE TABLE s_dept_copy (id_copy, name_copy) AS  
SELECT id, name FROM s_dept WHERE id=100;
```

В результаті виконання даної команди буде створена часткова копія таблиці `s_dept` із даними, що відповідають умові `id=100` підзапиту.

## 2.7. Внесення змін у створені структури таблиць бази даних

Внесення змін у створене раніше подання таблиці виконується за допомогою наступних команд:

1) `ALTER TABLE` – модифікація структури існуючої таблиці. Ця команда дозволяє додавати стовпці, змінювати довжину стовпців, додавати і видаляти обмеження, а також дозволяти і забороняти (призупиняти) їхню дію;

2) `DROP TABLE` – видалення всіх рядків і структури таблиці;

3) `RENAME` – зміна імені об'єкта бази даних;

4) `TRUNCATE` – видалення усіх рядків таблиці і звільнення пам'яті, що зайнята таблицею.

Після виконання кожної такої команди відбувається автоматична фіксація транзакції (одиниця роботи бази даних, що змінює її стан). Відкрит транзакції (повернення у стан до виконання команди) на рівні команди `DDL` неможливий, тобто скасувати її результати не можна. Тому користуватися такими командами необхідно надзвичайно обережно.

2.7.1. Додавання стовпця. Додати стовпець у таблицю можна командою `ALTER TABLE` із фразою `ADD`.

Синтаксис команди:

ALTER TABLE *таблиця*

ADD (*стовпець тип\_даних* [DEFAULT *вираз*]

[*обмеження\_стовпця*], ...);

де

*таблиця* – ім'я таблиці,

*стовпець* – ім'я нового стовпця,

*тип\_даних* – тип даних і довжина нового стовпця,

DEFAULT *вираз* – визначення значення нового стовпця по-замовчання,

*обмеження\_стовпця* - обмеження цілісності (може бути декілька) для нового стовпця.

За допомогою ADD можна додавати стовпці, але не видаляти їх з таблиці. Вказати, у якому місці таблиці повинен з'явитися стовець, неможливо. Новий стовець автоматично стає в таблиці останнім.

Приклад синтаксису команди додавання нових стовпців:

ALTER TABLE s\_dept

ADD ( time\_zone CHAR(4) DEFAULT 'UTC2' NOT NULL,

name\_zone VARCHAR2(10) );

2.7.2.Зміна стовпця. Змінити визначення стовпця таблиці можна командою ALTER TABLE із фразою MODIFY. Можна змінювати такі параметри стовпця, як тип даних, розмір, значення за замовчуванням і обмеження NOT NULL.

Синтаксис команди:

ALTER TABLE *таблиця*

MODIFY (*стовпець тип\_даних* [DEFAULT *вираз*]

[*обмеження\_стовпця*], ...);

За допомогою даної команди можливо:

- збільшити довжину або точність числових значень стовпця;
- зменшити довжину стовпця, якщо він містить невизначені значення або

таблиця не містить рядків;

- змінити тип даних, якщо стовпець містить невизначені значення;
- змінити тип даних з CHAR на VARCHAR2 чи навпаки, якщо стовпець містить невизначені значення або якщо не міняється довжина;
- змінити значення за замовчуванням тільки для наступних вставок рядків у таблицю;
- задавати обмеження NOT NULL тільки у випадку, якщо стовпець не містить значень.

Приклад синтаксису команди:

```
ALTER TABLE s_dept  
MODIFY ( name_zone VARCHAR2(15) );
```

2.7.3. Додавання і видалення обмежень. Додавати і видаляти обмеження для існуючих таблиць можна за допомогою фраз ADD і DROP у команді ALTER TABLE.

Синтаксис команди додавання обмеження:

```
ALTER TABLE таблиця  
ADD [CONSTRAINT обмеження] тип (стовпець, ...);
```

де

*обмеження* - ім'я обмеження,

*тип* - тип обмеження,

*стовпець* - ім'я стовпця (або декількох стовпців), до якого відноситься обмеження.

Як впливає із синтаксису команди, ім'я обмеження можна не задавати, хоча це і рекомендується. Якщо обмеженню не привласнене ім'я, то система робить це сама.

Приклад синтаксису:

```
ALTER TABLE s_dept  
ADD CONSTRAINT s_dept_uk2 UNIQUE (name);  
ALTER TABLE s_dept
```

```
ADD UNIQUE (name_zone);
```

Обмеження NOT NULL для існуючого стовпця можна додати тільки за допомогою команди ALTER TABLE із фразою MODIFY, а видаляти, вказавши NULL для існуючого стовпця.

Приклад синтаксису:

```
ALTER TABLE s_dept
MODIFY time_zone NOT NULL;
ALTER TABLE s_dept
MODIFY time_zone NULL;
```

Синтаксис команди видалення обмеження:

```
ALTER TABLE таблиця
DROP CONSTRAINT обмеження |
PRIMARY KEY | UNIQUE (стовпець, ...) |
FOREIGN KEY (стовпець, ...) [CASCADE] ;
```

де

*стовпець* - ім'я стовпця, до якого відноситься обмеження,

*обмеження* - ім'я обмеження.

Якщо потрібно видалити обмеження, то ім'я цього обмеження можна одержати з представлень USER\_CONSTRAINTS і USER\_CONS\_COLUMNS словника даних СУБД Oracle, а потім використовувати команду ALTER TABLE із фразою DROP. Але можливо видалення обмеження без встановлення його назви. Для цього необхідно здійснити видалення обмеження командою ALTER TABLE із фразою DROP, вказавши його тип PRIMARY KEY, UNIQUE, FOREIGN KEY.

Якщо у фразі DROP використовується параметр CASCADE, одночасно видаляються і всі “залежні” обмеження зовнішнього ключа від первинного або унікального обмеження.

Якщо обмеження видалене як об'єкт, то з цього моменту воно вже не

підтримується сервером бази даних і недоступно в словнику даних.

Приклад синтаксису:

```
ALTER TABLE s_dept  
DROP PRIMARY KEY CASCADE;  
ALTER TABLE s_dept  
DROP CONSTRAINT s_dept_s_region_fk;
```

Можливо додавати, видаляти, забороняти і дозволяти обмеження, але не змінювати їхню структуру.

2.7.4. Дозвіл і заборона дії обмежень. За допомогою команди ALTER TABLE із фразами ENABLE і DISABLE можна дозволяти і забороняти дію обмежень, не видаляючи їх з бази даних.

Синтаксис команди:

```
ALTER TABLE таблиця  
DISABLE | ENABLE CONSTRAINT обмеження [CASCADE] ;
```

Якщо обмеження дозволене, то воно поширюється на всі дані в таблиці. Усі дані в таблиці повинні відповідати обмеженню. Якщо дозволене обмеження UNIQUE або PRIMARY KEY, то автоматично створюються унікальні індекси для UNIQUE і PRIMARY KEY. Режим CASCADE використовується для одночасної заборони або відновлення дії всіх “залежних” обмежень.

Приклад синтаксису:

```
ALTER TABLE s_dept  
DISABLE CONSTRAINT s_dept_s_region;  
ALTER TABLE s_dept  
ENABLE CONSTRAINT s_dept_s_region;
```

Фрази ENABLE і DISABLE можна використовувати також у команді CREATE TABLE та ALTER TABLE при створенні обмеження, вказавши їх після фрази CONSTRAINT. За замовчуванням створюване обмеження

автоматично вводиться в дію, тобто знаходиться в стані ENABLE. В разі необхідності створення обмеження без введення його у дію застосовується фраза DISABLE.

Приклад синтаксису:

```
CREATE TABLE s_reg ( id NUMBER(5)
CONSTRAINT s_reg_pk PRIMARY KEY DISABLE );
ALTER TABLE s_dept
ADD CONSTRAINT s_dept_time_zone_ck
CHECK (time_zone like 'UTC%') DISABLE;
```

2.7.5.Видалення таблиці. Команда DROP TABLE видаляє визначення таблиці з бази даних. В результаті виконання цієї команди з бази даних віддаляються всі дані таблиці і всі створені до цієї таблиці обмеження і індекси. Якщо використовується параметр CASCADE CONSTRAINTS, то видаляються і всі “залежні” від обмеження первинного або унікального ключа цієї таблиці обмеження зовнішнього ключа у інших таблицях.

Синтаксис команди:

```
DROP TABLE таблиця [CASCADE CONSTRAINTS];
```

З таблиці видаляються всі дані. Усі представлення, синоніми, збережені процедури, функції і пакети залишаються як об’єкти бази, але в недійсному стані. Усі ще незафіксовані транзакції фіксуються. Видалити таблицю може тільки той, хто її створив (власник схеми), або користувач із привілеєм DROP ANY TABLE (адміністратор бази).

Приклад синтаксису:

```
DROP TABLE s_dept CASCADE CONSTRAINTS;
```

Після виконання команди DROP TABLE відкот (повернення в попередній стан) неможливий. Якщо виконується команда DROP TABLE, то сервер бази даних не запитує ніяких підтверджень. Якщо Ви є власником таблиці або маєте досить високий рівень привілеїв, то таблиця буде вилучена негайно.

Усі команди DDL перед їх виконанням фіксують попередню транзакцію (виконують неявний COMMIT), роблячи зміни постійними і незмінними.

2.7.6.Перейменування й усікання таблиці. Команди RENAME і TRUNCATE TABLE також є командами DDL. Команда RENAME використовується для перейменування таблиць, представлень, послідовностей або синонімів, а команда TRUNCATE TABLE - для видалення всіх рядків таблиці і звільнення пам'яті, відведеної під дані таблиці.

Синтаксис команди RENAME:

RENAME *старе\_ім'я\_таблиці* TO *нове\_ім'я\_таблиці* ;

Перейменувати об'єкт може тільки його власник.

Синтаксис команда TRUNCATE:

TRUNCATE TABLE *таблиця* ;

Щоб зробити усікання таблиці, необхідно бути її власником або мати системні привілеї на рівні DELETE TABLE.

Командою DELETE також можна видалити всі рядки таблиці, але вона не звільняє пам'ять, відведену під таблицю, тобто виділена раніше пам'ять залишається за таблицею.

## 2.8. Опис індексів бази даних

2.8.1.Загальні відомості про індекси. Індекс – це об'єкт бази даних, за допомогою якого можна прискорити пошук рядків у таблиці за рахунок використання значення покажчика рядків, а також забезпечити виконання правил обмеження цілісності у базі. Індекси створюються або вручну, або автоматично. Якщо не створено індексу по стовпцю, то таблиця буде переглядатися цілком при виконанні будь-якого запиту на пошук значень рядків за цим стовпцем. Індекси створюються для зменшення потреби в дискових операціях введення-виведення за рахунок використання *B*-дерева для швидкого пошуку даних. Індекси автоматично використовуються і підтримуються сервером бази даних. Якщо індекс створений, то від



користувача надалі ніяких прямих дій не потрібно.

Індекси фізично і логічно незалежні від таблиці. Це означає, що індекси можуть бути створені або видалені в будь-який час існування таблиці. Це не вплине на рядки таблиці й інші індекси.

Кожний індекс складається із значень стовпців, за якими побудовано індекси, та показчиків на рядки, що містять ці значення. Показчик прямо вказує на відповідний рядок у таблиці, усуваючи необхідність повного перегляду таблиці. *B*-дерево - це двійкова самоврівноважена пошукова структура, яка дозволяє урівноважити час доступу до рядків таблиці, тобто зробити час доступу до заданого значення приблизно однаковою незалежно від того, в якому місці таблиці знаходиться даний рядок.

Існує два типи індексів. Перший – це унікальний індекс. Сервер бази даних створює його автоматично, якщо для стовпця в таблиці задане обмеження PRIMARY KEY або UNIQUE. Користувач же може створювати індекси другого типу – не унікальні індекси. Наприклад, для збільшення швидкості обробки запиту із з'єднанням таблиць можна створити індекс по стовпцю з обмеженням FOREIGN KEY.

Різновиди індексів бази даних наведені в табл.8.

Таблиця 8

Різнovid	Опис
Унікальний	Забезпечує унікальність значень для зазначених стовпців
Не унікальний	Прискорює виконання пошукових запитів рядків
Одностовпцовий	В індексі використано тільки один стовпець
Складений	Індекс може містити до 16 стовпців для прискорення запитів або перевірки унікальності. Стовпці не обов'язково повинні бути суміжними

Різновиди індексів не виключають один одного. Наприклад, можна створити унікальний складений індекс.

2.8.2.Створення індексу. Індекс по одному чи декільком стовпцям створюється за допомогою команди CREATE INDEX.

Узагальнений синтаксис команди:

```
CREATE [UNIQUE] INDEX індекс  
ON таблиця (стовпець [, стовпець] ... ) ;
```

де

*індекс* – ім'я індексу,

*таблиця* – ім'я таблиці,

*стовпець* – ім'я стовпця в таблиці.

Фраза UNIQUE визначає унікальний тип створюваного індексу.

Приклад створення індексу для прискорення доступу до стовпця LAST\_NAME у таблиці S\_EMP:

```
CREATE INDEX s_emp_last_name_i ON s_emp (last_name) ;
```

Приклад створення унікального індексу для встановлення обмеження на припустимі значення у стовпці LAST\_NAME у таблиці S\_EMP:

```
CREATE UNIQUE INDEX s_emp_last_name_i ON s_emp (last_name) ;
```

Збільшення кількості індексів для таблиці не завжди прискорює запити. Кожна операція маніпулювання з даними мови DML (команди INSERT, UPDATE, DELETE), виконувана над таблицею з індексами, вимагає їх відновлення (перебудування). Чим більше індексів пов'язано з таблицею, тим більше зусиль потрібно від сервера на відновлення всіх індексів після операції DML.

Індекс варто створювати у випадку, якщо:

- 1) стовпець часто використовується у фразі WHERE або умові з'єднання команди пошуку (вибірки) даних SELECT;
- 2) стовпець має широкий діапазон значень;
- 3) стовпець містить велику кількість невизначених значень;
- 4) два чи більше стовпців часто використовуються разом у фразі WHERE або умові з'єднання команди SELECT;
- 5) передбачається, що велика частина запитів до таблиці великого розміру буде вибирати менш 10-15% рядків.

Необхідно пам'ятати, що для забезпечення унікальності варто задати

обмеження UNIQUE у визначенні таблиці і тоді унікальний індекс буде створений автоматично.

Індекс не слід створювати, якщо:

- 1) таблиця невеликого розміру;
- 2) стовпці не дуже часто використовуються в умові запити команди SELECT;
- 3) велика частина запитів буде вибирати більш 10-15% рядків;
- 4) таблиця часто оновлюється.

2.8.3.Видалення індексу. Індекс неможливо змінити. Його можна тільки спочатку видалити, а потім створити заново. Для видалення визначення індексу зі словника даних бази використовується команда DROP INDEX. Щоб видалити індекс, необхідно бути його власником або мати привілей DROP ANY INDEX.

Синтаксис команди видалення:

DROP INDEX *індекс* ;

де

*індекс* - ім'я індексу.

## 2.9. Опис створення представлення

2.9.1.Загальні відомості про представлення. Представлення – це об'єкт бази даних, за допомогою якого можна подати підмножину або проєкцію даних з однієї чи декількох таблиць. Така підмножина даних може бути обмежена як за кількістю стовпців, так і рядків, які задовольняють умові запити команди SELECT до таблиць представлення.

За структурою, представлення – це логічна таблиця, створена на основі фізичної таблиці або іншого представлення. Представлення не містить власних даних, а швидше є “вікном”, через яке можна переглядати або змінювати підмножину даних у таблиці. Представлення зберігається в словнику даних бази як команда SELECT.

Переваги використання представлення:

- обмежується доступ до бази даних внаслідок того, що представлення можуть відтворювати визначену підмножину даних;

- за допомогою простих запитів користувач може отримати такі ж результати, як із допомогою складних. Наприклад, використання представлень дозволяє здійснювати вибірку даних з декількох таблиць без будь яких знань про оператор з'єднання таблиць JOIN;

- забезпечується незалежність даних для користувачів, які посилають випадкові запити, від прикладних програм. Одно представлення може використовуватися для вибірки даних з декількох таблиць.

- використання представлень дозволяє забезпечити доступ до даних за різними критеріями для різних груп користувачів.

2.9.2. Створення представлення. Створити представлення для подання підмножини даних можна за допомогою команди CREATE VIEW мови визначення даних DDL.

Для створення представлення користувач повинен мати відповідний привілей CREATE VIEW.

Узагальнений синтаксис команди:

```
CREATE [OR REPLACE]
[FORCE | NOFORCE] VIEW [схема.]представлення
[( стовпець, ... )] AS запит_даних
[WITH CHECK OPTION [CONSTRAINT обмеження] ]
[WITH READ ONLY];
```

де

*схема* – ім'я користувача бази даних, власника представлення,

*представлення* – найменування об'єкта в базі,

*стовпець* – ім'я стовпця для даних, які вибрані у запиті для подання у представленні,

*запит\_даних* – команда SELECT запиту, для утворення підмножини даних,

фраза `OR REPLACE` – режим зміни визначення представлення без його видалення і створення заново, тобто внесення змін до існуючого представлення та заміна наданих привілеїв,

фраза `FORCE` – режим створення представлення незалежно від існування таблиць запиту даних у базі,

фраза `NOFORCE` – режим створення представлення тільки при умові існування таблиць запиту даних у базі (є режимом за замовчуванням),

фраза `WITH CHECK OPTION` - режим створення представлення, для якого додавати або змінювати можна тільки ті рядки, які доступні у ньому,

*обмеження* – ім'я, яке встановлюється обмеженню `CHECK OPTION`,

фраза `WITH READ ONLY` – режим заборони застосування до представлення операцій `DML`.

Запит, яким визначено представлення, може містити команду `SELECT` зі складним синтаксисом, що включає з'єднання, групування та інші підзапити. Але запит не може містити фразу впорядкування даних `ORDER BY`. Для зміни визначення існуючого представлення можна застосовувати режим `OR REPLACE`.

Існує два види представлення: прості та складні. Основна їх відмінність пов'язана з можливістю виконання команд мови `DML`. Їх порівняльні характеристики подані у табл.9.

Таблиця 9

Характеристика	Прості представлення	Складні представлення
Кількість таблиць	Тільки одна	Одна і більше
Містить функції	Ні	Так
Містить групи даних ( <code>DISTINCT</code> або групові функції)	Ні	Так
Операції <code>DML</code> над представленням	Так	Ні

Вказати імена стовпців можна шляхом включення псевдонімів стовпців таблиці у під запити представлення. Це неявний спосіб завдання назв стовпців представлення, які унаслідуються від запиту. Другий спосіб вказування імен

стовпців представлення – це явно вказати у поданні представлення після його назви. При явному визначенні стовпців у представленні, їх кількість повинна співпадати з кількістю стовпців у запиті даних.

Приклади синтаксису команди:

- створення представлення для вибору підмножини рядків з таблиці S\_EMP:

```
CREATE VIEW v1_emp AS  
SELECT * FROM s_emp WHERE last_name like 'A%';
```

- створення представлення для вибору підмножини стовпців з таблиці S\_EMP з відтворенням їх під новими виразами:

```
CREATE VIEW v2_emp (emp_id, emp_last_name) AS  
SELECT id, last_name FROM s_emp;
```

- створення представлення з даних двох таблиць S\_DEPT і S\_REGION:

```
CREATE VIEW v_region_dept (region_name, dept_name) AS  
SELECT r.name, d.name FROM region r, dept d WHERE d.region_id=r.id;
```

Команди мови DML можуть виконуватися над представленнями у відповідності до наступних правил:

1) Видалення рядка з представлення можливе, якщо представлення не містить нічого з наступного:

- умова з'єднання;
- групова функція;
- групування даних фразою GROUP BY;
- оператор не повторення даних DISTINCT;

2) Дані в представленні можуть бути змінені, якщо представлення не містить нічого з наведеного вище та нічого із наступного:

- стовпці, які описані як вирази, наприклад, salary\*12;
- псевдо-стовпець ROWNUM;

3) Додавання нових даних через представлення можливе, якщо воно не містить нічого із наведеного вище та у таблиці запиту немає стовпців з

обмеженням цілісності NOT NULL, які не включені у поданні представлення. Всі необхідні значення повинні бути присутні в представленні. Додавання даних через представлення фактично означає додавання в таблицю запиту напряду.

При роботі з представленнями необхідно слідкувати за тим, щоб результати операцій DML залишалися в межах значень, визначених у поданні представлення – домена представлення. Тобто любі зміни у даних, виконані через представлення, повинні бути доступні через звернення до представлення.

Приклад виконання операцій DML через представлення:

```
CREATE OR REPLACE VIEW emp_dept41
AS SELECT * FROM s_emp WHERE dept_id=41
WITH CHECK OPTION;
UPDATE emp_dept41 SET dept_id=42 WHERE dept_id=41;
```

При виконанні DML виникла помилка:

```
ORA-01402: view WITH CHECK OPTION where-clause violation
```

Ні один рядок не може бути змінено тому, що при зміні dept\_id=42 представлення вже не може “бачити” ці дані. Отже, якщо задано режим WITH CHECK OPTION, в представленні можуть бути відтворені тільки службовці з відділу 41 і змінювати номер відділу цих службовців через представлення неможливо.

Можна зробити так, щоб виконання операцій DML через представлення було неможливим. Для цього представлення створюється з режимом WITH READ ONLY.

2.9.3.Видалення представлення. Для видалення представлення застосовується команда DROP VIEW. Ця команда видаляє визначення представлення з бази даних та не змінює таблиці запиту, на підставі яких воно було створено. Інші представлення, які були створені на базі видаленого представлення стають недіючими і не можуть бути використаними. Видалити представлення може тільки той користувач бази, хто його створив або

користувач з привілеєм DROP ANY VIEW.

Синтаксис команди:

DROP VIEW *представлення*;

## 2.10. Опис створення послідовностей

2.10.1. Загальні відомості про послідовність. Для автоматичної генерації номерів-ідентифікаторів рядків таблиці застосовуються генератори послідовностей. Послідовність – це об'єкт бази даних, який створюється одним користувачем, а може використовуватись спільно декількома користувачами.

Типовим застосуванням послідовності – створення значення для первинного ключа – стовпця таблиці з обмеженням цілісності первинний ключ. Ці значення повинні бути унікальними для кожного рядка таблиці. Послідовність генерується та збільшується (або зменшується) власне СУБД. Числові значення послідовності зберігаються і генеруються незалежно від стану таблиць. Отже, одна і та ж послідовність може застосовуватися до декількох таблиць. Також, послідовність може застосовуватися для створення значень унікального ключа.

2.10.2. Створення послідовність. Послідовність для автоматичної генерації чисел створюється командою CREATE SEQUENCE.

Синтаксис команди:

CREATE SEQUENCE *послідовність*

[INCREMENT BY *n*]

[START WITH *n*]

[MAXVALUE *n* | NOMAXVALUE]

[MINVALUE *n* | NOMINVALUE]

[CYCLE | NOCYCLE]

[CACHE *n* | NOCACHE] ;

де

послідовність – ім'я генератора чисел послідовності,



INCREMENT BY  $n$  – інтервал  $n$  між двома послідовними номерами,  $n$  є цілим числом. Якщо ця фразу пропущено, то приріст при генерації чисел дорівнює 1;

START WITH  $n$  – перше генероване число в послідовності. Якщо ця фразу пропущено, то послідовність починається з 1;

MAXVALUE  $n$  – максимальне значення, яке може генерувати послідовність. NOMAXVALUE задає максимальне значення по замовченню  $10^{27}$ ;

MINVALUE  $n$  – мінімальне значення послідовності. NOMINVALUE задає мінімальне значення по замовченню 1;

CYCLE – задає продовження циклічного генерування чисел після досягнення максимального або мінімального значення. По замовченню встановлено NOCYCLE – немає циклічної генерації;

CACHE  $n$  – задає кількість чисел, які СУБД попередньо розподіляє і зберігає в оперативній пам'яті для прискорення доступу до чисел послідовності. По замовченню СУБД зберігає в кеші 20 значень. NOCACHE – немає попереднього розподілу в кеші.

Якщо послідовність використовується для генерації значень первинних ключів, не застосовується режим CYCLE.

Створена послідовність документується в словнику даних, оскільки послідовність є об'єктом бази даних. Перевірити параметри послідовності можна шляхом вибірки даних із представлення словника даних СУБД Oracle USER\_SEQUENCES.

Приклад синтаксису команди:

```
CREATE SEQUENCE s_dept_id  
INCREMENT BY 1  
START WITH 51  
MAXVALUE 9999999  
NOCACHE NOCYCLE;
```

2.10.3. Використання послідовності. Числа створеної послідовності можна використовувати в якості порядкових номерів для таблиць. Звернення до чисел послідовності здійснюється за допомогою псевдо-стовпців NEXTVAL і CURRVAL.

Псевдо-стовпець NEXTVAL використовується для вибірки наступного вільного числа послідовності. Назву NEXTVAL необхідно доповнювати ім'ям послідовності. При посиланні на *послідовність*.NEXTVAL, генерується наступне число, яке зберігається в CURRVAL.

Псевдо-стовпець CURRVAL використовується для посилання на поточне число, яке генероване у послідовності. Перш, ніж вперше звернутися до CURRVAL, необхідно використати NEXTVAL для генерації числа у поточній сесії користувача послідовності. Назву CURRVAL необхідно доповнювати ім'ям послідовності. При посиланні на *послідовність*.CURRVAL, видається останнє значення, генероване сесією цього користувача послідовності.

Команди SQL, де використовуються NEXTVAL і CURRVAL:

- список даних команди SELECT, яка не є частиною підзапиту;
- список даних підзапиту SELECT в команді INSERT;
- фраза VALUES команди INSERT;
- фраза SET команди UPDATE.

Команди SQL, де не використовуються NEXTVAL і CURRVAL:

- список даних запиту SELECT в представленні;
- команда SELECT з оператором DISTINCT;
- команда SELECT з фразами GROUP BY, HAVING, ORDER BY;
- фраза WHERE команди SELECT, DELETE, UPDATE;
- підзапит у команді SELECT, DELETE, UPDATE;
- фраза DEFAULT команди CREATE TABLE, ALTER TABLE.

Приклади команд використання послідовності:

- генерування значення послідовності у відкритій сесії користувача

```
SELECT s_dept_pk_seq.nextval FROM DUAL;
```

(DUAL – системна таблиця СУБД Oracle, застосовується для визначення

значень псевдо-стовпці, константних виразів, одно-рядкових функцій за допомогою команди SELECT, завжди повертає один рядок)

- визначення останнього значення, яке генероване поточною сесією користувача послідовності

```
SELECT s_dept_pk_seq.currval FROM dual;
```

(поверне однакове значення, що й останнє звернення до s\_dept\_pk\_seq.nextval)

```
SELECT s_dept_pk_seq.nextval, s_dept_pk_seq.currval FROM dual;
```

(згенерує нове і поверне таке ж значення, що й звернення до s\_dept\_pk\_seq.nextval)

- використання значення послідовності для визначення первинного ключа

```
INSERT INTO s_dept (id,name) VALUES (s_dept_pk_seq.nextval,'FIOT');
```

Кешування послідовності до пам'яті прискорює доступ до його значень. Кеш-пам'ять заповнюється при першому зверненні до послідовності. У відповідь на кожний наступний запит видається одне з чисел, що знаходяться у кеш-пам'яті. Якщо останнє з цих значень використане, запит наступного значення призводить до запису у пам'ять чергової партії значень.

Хоч генератори послідовностей видають числа без пропуску, але ця дія виконується незалежно від операцій COMMIT і ROLLBACK. Отже, при відкоту команди, яка містить звернення до послідовності, число втрачається. Значення послідовності, що знаходяться в кеш-пам'яті також втрачаються при збоях СУБД. Також, якщо одна і та ж послідовність використовується в декількох таблицях, то пропуски у числах можуть бути у цих таблицях. Побачити наступне вільне значення послідовності, не збільшивши його, можна тільки у випадку створення послідовності у режимі NOCACHE.

Якщо послідовність досягла верхньої межі MAXVALUE, додаткові значення не генеруються і виникає помилка. Щоб продовжувати користуватися послідовністю, можна змінити її параметри за допомогою команди ALTER SEQUENCE.

Синтаксис команди зміни параметрів послідовності:

```
ALTER SEQUENCE послідовність
[INCREMENT BY n]
[MAXVALUE n | NOMAXVALUE]
[MINVALUE n | NOMINVALUE]
[CYCLE | NOCYCLE]
[CACHE n | NOCACHE] ;
```

Для зміни параметрів необхідно бути власником послідовності або мати відповідний привілей ALTER SEQUENCE. Команда ALTER SEQUENCE впливає тільки на числа, що генеруються після введення змін. Крім того, нове значення MAXVALUE не може бути менше, ніж поточне значення послідовності. Змінити параметр START WITH командою ALTER SEQUENCE не можливо. Щоб почати послідовність з іншого числа, необхідно її видалити, а потім створити заново.

2.10.4.Видалення послідовності. Для видалення послідовності із словника даних використовується команда DROP SEQUENCE. Щоб видалити послідовність, необхідно бути її власником або мати привілей DROP ANY SEQUENCE.

Синтаксис команди:

```
DROP SEQUENCE послідовність;
```

## 2.11. Перевірка створення структур об'єктів у базі даних

2.11.1.Словник даних СУБД - це один з найбільш важливих компонентів сервера бази даних. Він створюється під час генерації бази і складається з набору системних таблиць і представлень (віртуальних таблиць), що забезпечують незмінну службу (довідкову) інформацію з бази даних про її об'єкти. При кожному використанні бази даних відновлення й обслуговування словника даних здійснюється сервером. Власником усіх таблиць словника

даних у СУБД Oracle є системний користувач SYS, супер-адміністратор бази даних, в схемі якого було утворено базу. Пряме звертання до системних таблиць бази даних використовується рідко, тому що інформацію, яку вони містять, досить складно зрозуміти. Тому звичайні користувачі працюють зі словником даних через так названі системні представлення, що дозволяють одержувати інформацію у форматі простому для розуміння.

Вмістом словника даних є:

- імена користувачів сервера бази даних;
- привілеї, надані користувачам;
- імена об'єктів бази даних (таблиць, представлень, індексів і т.д.);
- обмеження цілісності даних таблиць;
- облікова інформація (наприклад, про те, хто звертався до конкретних об'єктів бази даних або їх оновлював).

Словник даних забезпечує довідковою інформацією всіх користувачів бази даних. Це джерело службової інформації для кінцевих користувачів, розроблювачів додатків і адміністраторів бази даних. Особливе значення має він і для сервера, тому що саме в ньому зберігається і перевіряється інформація про саму базу даних.

Запитувати дані зі словника даних можна за допомогою команди SELECT мови запитів SQL. Можливість доступу до різних представлень словника даних залежить від наданого адміністратором рівня привілеїв.

Імена представлень словника даних відповідають їх призначенню. Існує чотири категорії представлень, кожна з яких позначається особливим префіксом (див. табл.10).

Таблиця 10

Префікс	Опис
USER_	Включає тільки об'єкти, власником яких є користувач. Зокрема, представлення з цим префіксом дозволяють користувачу одержати дані про створені ним таблиці і надані йому привілеї
ALL_	Крім об'єктів, що належать самому користувачу, включає об'єкти, на доступ до яких йому надано право
DBA_	Дозволяє користувачам з рівнем привілеїв системних

	адміністраторів звертатися до будь-якого об'єкта в базі даних
V\$	Дозволяє виводити інформацію про продуктивність сервера бази даних і блокування. Доступний тільки адміністратору бази даних

Для деяких представлень словника даних (наприклад, синонімів представлень з довгими іменами) вищевказані префікси не використовуються. Найбільше уживані представлення наведені у табл.11

Таблиця 11

Ім'я представлення	Опис
DICTIONARY	Перелік усіх таблиць, представлень і синонімів словника даних
TABLE_PRIVILEGES	Права на доступ до об'єктів, надані й отримані користувачем, а також об'єкти, що належать користувачу
USER_OBJECTS	Перелік всіх об'єктів користувача
USER_TABLES	Перелік усіх таблиць користувача
USER_TAB_COLUMNS	Перелік усіх стовпців таблиць користувача
USER_INDEXES	Перелік всіх індексів користувача
USER_IND_COLUMNS	Перелік усіх стовпців індексів користувача
USER_CONSTRAINTS	Перелік всіх обмежень користувача
USER_CONS_COLUMNS	Перелік усіх стовпців обмежень користувача
USER_VIEWS	Перелік усіх представлень користувача
USER_SEQUENCES	Перелік усіх послідовностей користувача
IND	Синонім представлення індексів користувача USER_INDEXES
TAB	Синонім представлення таблиць користувача USER_TABLES

Представлення DICTIONARY дозволяє одержати список усіх представлень словника даних, доступних користувачу, з коротким описом у колонці примітки. Звертатися до цього представлення можна і за синонімом DICT:

```
SELECT * FROM DICTIONARY ;
```

```
SELECT * FROM DICT ;
```

Опис будь-якого стовпця таблиці або представлення словника даних можна одержати за допомогою представлення DICT\_COLUMNS:

```
SELECT column_name, comments
```

```
FROM dict_columns  
WHERE table_name = 'USER_OBJECTS' ;
```

Запит списку типів об'єктів, що належать користувачу:

```
SELECT DISTINCT object_type FROM user_objects ;
```

Примітка. Назва об'єктів у словнику бази зберігається завжди у верхньому регістрі.

2.11.2.Перевірка обмежень для таблиці. Повний список обмежень для таблиці користувача можна одержати за допомогою запиту до USER\_CONSTRAINTS.

Приклад синтаксису для таблиці s\_emp:

```
SELECT      constraint_name,      constraint_type,      search_condition,  
r_constraint_name  
FROM user_constraints  
WHERE table_name = 'S_EMP' ;
```

Переглянути імена стовпців, на які поширюються відповідні обмеження, можна за допомогою представлення словника даних USER\_CONS\_COLUMNS. Цей запит особливо корисний для обмежень, імена яких привласнені системою.

Приклад синтаксису для таблиці s\_emp:

```
SELECT constraint_name, column_name  
FROM user_cons_columns  
WHERE table_name = 'S_EMP';
```

### 3. Порядок роботи з інструментальними засобами взаємодії із сервером бази даних

Будь-які інструментальні засоби і прикладні додатки здійснюють взаємодію із сервером бази даних через мову команд запитів SQL. До складу програмного забезпечення сервера бази даних Oracle входить спеціалізований консольний додаток Oracle SQL\*Plus та програмне середовище взаємодії і

адміністрування сервера бази даних Oracle SQL Developer.

**Oracle SQL\*Plus** – це інтерпретатор команд із мови SQL для виконання дій по створенню інформаційних об'єктів бази даних, їхньої модифікації і наповнення даними.

**Oracle SQL Developer** – це графічний інструмент для розробки і адміністрування баз даних. За допомогою SQL Developer можна переглядати об'єкти бази даних, запускати SQL-команди, редагувати і налагоджувати PL/SQL-програми. Ви також можете запустити будь-яку кількість наданих звітів, а також створювати і зберігати власні. SQL Developer підвищує продуктивність і спрощує підтримку вашої бази даних при виконанні завдань з її розвитку. SQL Developer може підключатися до будь-якої СУБД Oracle від версії 9.2.0.1 і може працювати на Windows, Linux, Mac OSX.

Порядок взаємодії з інструментальними засобами взаємозв'язку із сервером Oracle:

1. Запуск і реєстрація користувача в базі даних. Необхідно обрати попередньо створене підключення до бази (наприклад, base), ввести ім'я користувача - ідентифікатор студента в групі (наприклад, ik3101), пароль.

2. Введення команди і запуск її на виконання здійснюється в робочому листі вікна інструментального засобу. Команда може розташовуватися на декількох рядках. Але повинна завершуватися символом «;» на поточному рядку або «/» на наступній рядку.

3. Виконання командного файлу (попередньо збереженого на локальному диску) здійснюється вказівкою символу «@» і наведенням повної специфікації файлу.

4. Вихід з інструментального засобу здійснюється закриттям його вікна.

Oracle SQL Developer це програмний засіб для взаємодії з сервером бази даних у вигляді GUI-додатків. Тому він підтримує роботу, як у командному рядку робочого листа додатку, так і у меню-орієнтованому інтерфейсі з автоматичною генерацією виконуваних SQL-команд. Робота в такому засобі здійснюється в інтерактивному режимі за допомогою вікон.



Основним вікном у Oracle SQL Developer є навігаційне вікно з деревом доступних об'єктів бази даних. Правою кнопкою миші видаються дозволені операції модифікації обраного об'єкта бази даних.

З основного меню додатку викликається SQL Worksheet - робочий лист редактора команд SQL. Виклик редактора рядків команд здійснюється для введення як одиничних команд SQL без завершального символу «;», так і їх послідовності, які розділені завершальним символом виконання команди «;». Запуск команд на виконання здійснюється від поточного положення маркера до кінця послідовності команд кнопкою F5 або тільки поточної команди за допомогою кнопки F9 або Ctrl-Enter.

Контроль виконання команд виконується переглядом пунктів дерева у навігаційному вікні. Можна зберегти введену послідовність команд у вигляді файлу з типом sql за допомогою пункту головного меню «File / Save».

#### 4. Контрольне завдання

1. Побудувати базу даних згідно наступного порядку:

1.1) Створити таблиці бази даних, яка описує службовців, що працюють у відділах підприємства за наведеними бланками екземплярів таблиць. Визначити обмеження цілісності для DEPARTMENT на рівні таблиці, для EMPLOYEE на рівні стовпця, а для REGION окремо від визначення структури таблиці за допомогою окремих команд. Утворити імена первинних, зовнішніх і унікальних ключів згідно рекомендованих правил.

Таблиця регіонів REGION

Найменування стовпця	Тип ключа	NOT NULL	Таблиця посилання FK	Стовпець таблиці-посилання FK	Тип даних	Довжина
ID	PK				NUMBER	7
NAME		NOT NULL	встановити значення стовпця за замовчуванням 'Новий регіон'		VARCHAR2	25

Таблиця відділів DEPARTMENT

Найменування стовпця	Тип ключа	NOT NULL	Таблиця-посилання FK	Стовпець таблиці-посилання FK	Тип даних	Довжина
ID	PK				NUMBER	7
NAME		NOT NULL			VARCHAR2	25
REGION_ID	FK, UK		REGION	ID	NUMBER	7

Таблиця службовців EMPLOYEE

Найменування стовпця	Тип ключа	NOT NULL	Таблиця-посилання FK	Стовпець таблиці-посилання FK	Тип даних	Довжина
ID	PK				NUMBER	7
LAST_NAME		NOT NULL			VARCHAR2	25
FIRST_NAME					VARCHAR2	25
USER_ID	UK	NOT NULL			CHAR	12
DEPT_ID	FK	NOT NULL	DEPARTMENT	ID	NUMBER	7
START_DATE	встановити поточну дату як значення стовпця за замовчуванням				DATE	
COMISSION	Дозволяються тільки значення від 10 до 50				NUMBER	7

1.2) Побудувати індекси для зовнішніх ключів і додаткові пошукові індекси для строкових стовпців (NAME, LAST\_NAME). Перевірити результати побудови структур даних в базі через запити до його словника даних і дати пояснення про результати їх побудови.

Усі завдання п.1 з командами створення відповідних структур даних у базі та результатами їх перевірки представити у виді командного файлу, який подати у звіті з коментарями-поясненнями.

2. Внести зміни до об'єктів побудованої бази даних згідно наступного порядку:

2.1) Створити таблицю WORKER як копію таблиці EMPLOYEE. Порівняти опис структури таблиці й обмежень WORKER з EMPLOYEE і надати пояснення про наявні відмінності.

2.2) Додати первинний ключ для таблиці WORKER, використовуючи стовпець ID. Явно вказати, що обмеження повинно вступити в дію негайно.

2.3) Додати обмеження цілісності зовнішній ключ для стовпця DEPT\_ID у таблиці WORKER на таблицю DEPARTMENT.

2.4) Додати стовпець DEPT\_ID з обмеженням цілісності зовнішній ключ у таблиці DEPARTMENT на таблицю DEPARTMENT.

2.5) Додати в таблицю WORKER стовпець TITLE з типом VARCHAR2 і розміром 30.

2.6) Видалити таблицю EMPLOYEE, а потім DEPARTMENT, зберігши WORKER без змін. Пояснити отриманий результат.

2.7) Створити послідовність DEPT\_PK\_SEQ для генерації значення первинного ключа таблиці DEPARTMENT. Перше число послідовності – 76, максимальне – 80, приріст значень – 1.

2.8) Напишіть командний файл, який для послідовності DEPT\_PK\_SEQ виконує наступні дії:

- генерацію нового значення числа послідовності;
- виведення поточного значення послідовності;

- виведення інформації про послідовність із словника даних бази – ім'я послідовності, розмір кешу, максимальне значення, шаг приросту і останнє генероване число.

Порівняйте отримані дані цих дій та надайте до них пояснення.

Усі завдання п.2 подати як командний файл, який подати у звіті з коментарями-поясненнями.

### Комп'ютерний практикум № 3.

## ОБРОБКА ІНФОРМАЦІЇ В БАЗІ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОМАНДАМИ МОВИ МАНІПУЛЮВАННЯ ДАНИМИ

### 1. Мета та основні завдання практикуму

Мета роботи: ознайомитися з командами мови маніпулювання даними при виконанні обробки інформації і набути навички роботи з управління транзакціями сервера бази даних Oracle

Порядок виконання роботи:

- вивчити команди мови маніпулювання даними;
- визначити основні компоненти процесу керування транзакціями в інформаційній системі;
- згідно умов контрольного завдання наповнити базу даних необхідною інформацією;
- підготувати звіт з результатами обробки даних у базі даних.

### 2. Основні теоретичні відомості

Мова маніпулювання даними Data Manipulation Language (DML) є складовою мови взаємодії із сервером бази даних – мови структурованих запитів SQL. При кожній операції додавання, зміни або видалення даних з бази виконується певна команда DML. Сукупність команд DML, результати дії яких переводять базу даних з одного фіксованого стану (цілісного стану даних) до іншого називають транзакцією або логічною одиницею роботи сервера бази даних. Управління транзакцією означає її фіксацію (переведення в постійний, незмінний стан), або відкот (повернення до попереднього фіксованого стану на початку транзакції).

Таким чином, команди мови DML складаються з команд маніпулювання даними у рядках таблиць бази даних та команд управління транзакціями.

## 2.1. Вставка рядків у таблицю

Додати новий рядок у таблицю можна за допомогою команди INSERT.

Синтаксис команди:

```
INSERT INTO таблиця [(стовпець [, стовпець ...])]
```

```
VALUES (значення [, значення ...]) ;
```

де

*таблиця* – ім'я таблиці,

*стовпець* – імена стовпців таблиці, у які вносяться значення,

*значення* – відповідні значення стовпців.

Ця команда з фразою VALUES додає в таблицю тільки один рядок. Якщо, існує можливість вставити новий рядок, який містить значення до кожного стовпця, то їх перерахування у команді INSERT необов'язково. Однак в такому випадку послідовність самих значень повинна відповідати стандартної послідовності стовпців у цій таблиці (визначеній при її створенні).

Приклад вставки рядка без вказівки імен стовпців:

```
INSERT INTO s_dept VALUES (11,'Фінансовий відділ',2) ;
```

Але для адекватності сприйняття список стовпців у команді INSERT рекомендується вказувати. Символьні значення і значення дат наводять з прикінцевими лапками, цифрові значення в лапки не беруться.

2.1.1.Методи вставки невизначених значень у стовпці таблиці, що застосовуються у базі даних, наведені у табл.12.

Таблиця 12

Метод	Опис
Неявний	Опустити стовпець у списку стовпців
Явний	Задати ключове слово NULL у списку фрази VALUES
	Задати пустий рядок (") у списку фрази VALUES (тільки для символічних значень і дат)

Використання явного методу введення невизначених значень можливий, якщо стовпець припускає такі значення.

Приклад уведення нового відділу без номера регіону. Відсутність номера

регіону в команді INSERT означає неявне введення невизначеного значення для цього стовпця в новому рядку:

```
INSERT INTO s_dept (id, name) VALUES (12, 'Відділ кадрів');
```

Приклад явного введення невизначеного значення в рядок шляхом вказівки ключового слова NULL замість значення:

```
INSERT INTO s_dept VALUES (13, 'Адміністративний відділ', NULL);
```

2.1.2. Вставка спеціальних значень за допомогою функцій SQL. Для вставки в таблицю спеціальних значень можна використовувати функції або псевдо стовпці. Використовуйте функцію USER для введення імені поточного користувача і функцію SYSDATE для введення поточної дати і часу.

Приклад запису інформації про слухача курсу в таблицю s\_emp.

Ім'я поточного користувача вказується в стовпці user\_id, а поточні (системні) дата і час – у стовпці start\_date:

```
INSERT INTO s_emp (id, first_name, last_name, user_id, comission, start_date)
```

```
VALUES (26, 'Олександр', 'Петров', USER, NULL, SYSDATE);
```

Перевірка результатів вставки рядка в таблицю виконується за допомогою команди SELECT:

```
SELECT id, last_name, first_name, user_id, start_date, comission
FROM s_emp WHERE id=26;
```

Якщо потрібно вставити певне значення дати, то використовується неявне перетворення символічного подання дати у встановленому форматі за замовченням 'DD-MON-YY' (число-місяць-рік). У цьому форматі стандартним значенням для сторіччя вважається поточне сторіччя. Оскільки дата містить і інформацію про час, то стандартним значенням часу є північ 00:00:00.

Якщо потрібно задати інше століття, конкретний час або застосувати явне перетворення символічного подання дати за певним форматом, то застосовується відповідна функція перетворення TO\_DATE('значення', 'формат\_значення'). Припустимі формати значення дати і

часу наведені у табл.13

Таблиця 13

Формат значення дати і часу	Приклад значення
DD-MON-YY	01-DEC-96
DD-MON-YYYY	01-DEC-1996
DD-MON-YY HH24:MI	01-DEC-1996 18:30
DD/MM/YYYY	01/12/1996
DD/MM/YYYY HH24:MI	01/12/1996 18:30

Приклад запису інформації про слухача курсу в таблицю s\_emp.

Поточне ім'я користувача вказується в стовпці user\_id, а час і дата (8 годин 00 хвилин 1 січня 1996 р.) - у стовпці start\_date:

```
INSERT INTO s_emp (id, first_name, last_name, user_id, commision,
start_date)
```

```
VALUES (26, 'Donna', 'Smith', USER, NULL,
```

```
TO_DATE('01-JAN-96 08:00', 'DD-MON-YY HH24:MI') );
```

або

```
INSERT INTO s_emp (id, first_name, last_name, user_id, commision,
start_date)
```

```
VALUES (26, 'Donna', 'Smith', USER, NULL,
```

```
TO_DATE('01/01/1996 08:00', 'DD/MM/YYYY HH24:MI') );
```

Мова відтворення результату функції перетворення залежить від налаштування інструментального засобу взаємодії з базою даних Oracle. Для цього потрібно змінити значення мови відтворення дати (Data Language) на сторінці налаштувань інструментального засобу Oracle SQL Developer за пунктами меню Tools-Preferences та навігаційного дерева параметрів Database-NLS.

## 2.2. Інтерактивна обробка значень для вставки за допомогою командного файлу

Інструментальний засіб взаємодії з базою даних Oracle дозволяє створювати інтерактивні командні файли, коли користувач запрошується для



введення значень, які оброблюються і тимчасово зберігаються для подальшої їх опрацюваннями командами DML. Для створення такого командного файлу у його окремі команди включаються змінні підстановки. Ця змінна у командному файлі виступає в якості контейнера, в якому тимчасового зберігається її значення. Вона дозволяє гнучко замінювати значення в фразах команд DML, імена стовпців та вирази. Для створення інтерактивних командних файлів, вони повинні включати: поодинокі амперсанди &, змінні підстановки, команди інструментального засобу взаємодії – ACCEPT, DEFINE, UNDEFINE.

2.2.1.Визначення змінної підстановки. Визначити змінну підстановки можна до її використання командами DML. Для визначення і призначення значення змінній застосовуються дві команди – ACCEPT, DEFINE. Ці команди створюють змінну підстановки, якщо вона ще не існує, або перевизначає вже існуючу змінну.

Команда DEFINE використовується тільки для автоматичного визначення змінної для поточного сеансу (сесії з'єднання з базою даних) користувача або на час виконання командного файлу.

Синтаксис команди:

DEFINE [змінна\_підстановки [=значення] ]

де

*змінна\_підстановки* – ім'я змінної підстановки (без вказівки &),

*значення* – текстове значення змінної типу CHAR.

Команда вказує змінну підстановки та присвоює їй значення типу CHAR, або перелічує значення та тип однієї або всіх змінних сеансу користувача.

Приклади застосування команди:

- визначення значень двох змінних

```
DEFINE dept_name="кафедра технічної кібернетики"
```

```
DEFINE dept_id=1
```

- виведення значення та типу змінної dept\_name

```
DEFINE dept_name
```

результат виконання -

```
DEFINE DEPT_NAME = "кафедра технічної кібернетики" (CHAR)
```

- виведення значень та типів всіх змінних поточного сеансу користувача

```
DEFINE
```

результат виконання -

```
DEFINE _DATE = "15.03.20" (CHAR)
```

```
DEFINE _CONNECT_IDENTIFIER = "base" (CHAR)
```

```
DEFINE _USER = "IKTEST" (CHAR)
```

```
DEFINE _PRIVILEGE = "UNKNOWN" (CHAR)
```

```
DEFINE _SQLPLUS_RELEASE = 0401000000 (NUMBER)
```

```
DEFINE _EDITOR = "notepad" (CHAR)
```

```
DEFINE _O_VERSION = "Oracle Database 11g Enterprise Edition Release  
11.2.0.1.0 - 64bit Production With the Partitioning, OLAP, Data Mining and Real  
Application Testing options" (CHAR)
```

```
DEFINE _O_RELEASE = 1102000000 (NUMBER)
```

```
DEFINE _PWD = "C:\oracle\sqldeveloper\sqldeveloper\bin" (CHAR)
```

```
DEFINE DEPT_ID = 1 (BINARY_FLOAT)
```

```
DEFINE DEPT_NAME = "кафедра технічної кібернетики" (CHAR)
```

Для відміни визначення змінної та видалення її значення застосовується команда UNDEFINE.

Синтаксис команди відміни змінної:

```
UNDEFINE змінна_підстановки.
```

Запрошення до введення даних і присвоєння значень змінним підстановки можна здійснити за допомогою команди ACCEPT інструментального засобу. У команді ACCEPT символ & перед змінною підстановки не ставиться. Для переносу команди на наступну рядок використовується тире.

Синтаксис команди:

```
ACCEPT змінна_підстановки [тип_даних]
```

[FORMAT *формат*] [PROMPT '*запрошення*'] [HIDE]

де

*змінна\_підстановки* – ім'я змінної підстановки (без вказівки &),

*тип\_даних* – тип даних значення змінної – NUMBER, CHAR, DATE, BINARY\_FLOAT, BINARY\_DOUBLE,

*формат* – модель формату даних, наприклад, 9,999 (числове значення з фіксованою крапкою), A10 (рядкове значення довжиною 10 символів),

*запрошення* – текст рядка запрошення на введення значення для змінної підстановки,

HIDE – опція запобігання відображення даних при введенні користувачем, наприклад, значення паролю.

Приклад застосування команди:

```
ACCEPT dept_name PROMPT 'Введіть назву відділу:'
```

```
DEFINE dept_name
```

Результат виконання буде наступним -

Введіть назву відділу: кафедра технічної кібернетики

```
DEFINE DEPT_NAME = "кафедра технічної кібернетики" (CHAR)
```

2.2.2. Використання значень змінній підстановки. Задати змінні підстановки в командах DML можна за допомогою амперсанда & та назви змінної. Привласнювати значення змінної кожен раз при її використанні не потрібно. Якщо змінної на момент її використання в команді DML не визначено, то інструментальний засіб запросить користувача ввести її значення та автоматично визначить її для можливого подальшого застосування.

Приклад використання значення за допомогою змінній підстановки. Потрібно визначити змінну із значенням назви відділу та вивести її під заголовком department\_name для перевірки.

```
DEFINE dept_name="кафедра технічної кібернетики"
```

```
SELECT '&dept_name' AS department_name FROM DUAL;
```

Результат виконання буде наступним -

DEPARTMENT\_NAME

кафедра технічної кібернетики

Текстові значення та дати змінної підстановки при виведенні завжди повинні бути заключні в апострофи. Змінні підстановки можуть бути застосовані при виведенні їх значень також і як імена стовпців, виразів та текстових рядків.

Для того, щоб інструментальний засіб виводив вираз команди із змінною підстановки до її заміни відповідним значенням застосовується команда SET VERIFY ON. Якщо потрібно вивести тільки вираз команди після підстановки значення застосовується команда SET VERIFY OFF.

Для виклику та виконання командного файлу із змінними підстановки застосовується команда START або @ з вказівкою назви командного файлу.

2.2.3. Вставка значень за допомогою змінних підстановки. Команда INSERT дозволяє користувачу вводити значення в інтерактивному режимі за допомогою змінних підстановки інструментального засобу Oracle. Позначаються такі змінні за допомогою символу &.

Запрошення до введення даних і присвоєння значень змінним підстановки можна здійснити за допомогою команди ACCEPT інструментального засобу. Ця команда не є командою мови DML. Якщо для змінної підстановки така команда не визначена, то інструментальний засіб Oracle запросить введення значення при першому звертанні до такої змінній.

Приклад використання введеного значення за допомогою змінній підстановки. Потрібно запросити назву відділу та вивести його під заголовком department\_name для перевірки.

```
ACCEPT dept_name PROMPT 'Введіть назву відділу:'
```

```
SELECT '&dept_name' AS department_name FROM DUAL;
```

Результат виконання буде наступним -

Введіть назву відділу: кафедра технічної кібернетики

DEPARTMENT\_NAME

кафедра технічної кібернетики

У командах DML (INSERT, UPDATE, DELETE, SELECT) змінні підстановки разом з & для дат і символічних значень повинні бути укладені в апострофи. Команду із змінними підстановки можна зберігати у файлі для наступного повторного використання. При кожному виконанні такого командного файлу інструментальний засіб Oracle буде пропонувати ввести нові значення змінних.

Приклад вставки за допомогою змінної підстановки.

Потрібно записати інформацію про новий відділ у таблицю s\_dept. Користувач повинний ввести номер відділу, назву відділу і номер регіону з використанням запрошення до введення значень:

```
ACCEPT department_id PROMPT 'Ведіть номер відділу:'
```

```
ACCEPT department_name PROMPT 'Введіть назву відділу:'
```

```
ACCEPT region_id PROMPT 'Введіть номер регіону:'
```

```
INSERT INTO s_dept (id, name, region_id)
```

```
VALUES (&department_id, '&department_name', &region_id);
```

Результат виконання буде наступним -

Ведіть номер відділу: 61

Введіть назву відділу: кафедра технічної кібернетики

Введіть номер регіону: 2

### 2.3. Копіювання рядків з іншої таблиці

Команду INSERT можна використовувати для вставки в таблицю нових рядків, значення яких копіюються з вже існуючих у інших таблицях бази даних. В цьому випадку замість фрази VALUES використовується підзапит у виді команди SELECT. Кількість стовпців у списку команди INSERT повинне збігатися з кількістю виведених значень у підзапиті.

Синтаксис команди:

INSERT INTO *таблиця* ( *стовпець* [, *стовпець* ...] ) *підзапит* ;

де

*таблиця* – ім'я таблиці,

*стовпець* – ім'я стовпця в таблиці, значення якого необхідно ввести,

*підзапит* – команда SELECT, що повертає рядки із іншої таблиці або таблиць, значення яких будуть введені до *таблиці*.

Приклад копіювання рядків з таблиці s\_emp у таблицю s\_emp\_copy:

```
INSERT INTO s_emp_copy (id, last_name, comission, title, start_date)
```

```
SELECT id, last_name, comission, title, start_date
```

```
FROM s emp WHERE start_date<'01-JAN-94';
```

#### 2.4. Оновлення рядків

Існуючі рядки таблиці можна оновити новими значеннями за допомогою команди UPDATE.

Синтаксис команди:

```
UPDATE таблиця SET стовпець = значення [, стовпець = значення ...]
```

```
[ WHERE умова ] ;
```

де

*таблиця* – ім'я оновлюваної таблиці,

*стовпець* – ім'я оновлюваного стовпця таблиці,

*значення* – значення або підзапит на одержання значення для цього стовпця,

*умова* – задає рядки, які необхідно змінити від час оновлення таблиці.

Складається умова з імен стовпців, виразів, констант, підзапитів і операторів їх порівняння. Якщо в команді UPDATE відсутня фраза WHERE, то змінюються усі рядки таблиці.

Щоб переконатися в успішному оновленні таблиці, можна виконати запит SELECT на вибірку оновлених рядків.

Приклад оновлення рядків. Необхідно перевести службовця з персональним номером 2 у відділ номер 10 з підвищенням зарплати на 10:

```
UPDATE s_emp SET dept_id=10, comission=comission+10 WHERE id=2 ;
```

Перевірка оновлення таблиці -

```
SELECT id, last_name, dept_id, comission FROM s_emp WHERE id=2 ;
```

Результат виконання запиту -

<u>id</u>	<u>last_name</u>	<u>dept_id</u>	<u>comission</u>
2	Петров	10	2550

## 2.5. Видалення рядків

Існуючі в таблиці рядки видаляються командою DELETE.

Синтаксис команди:

```
DELETE FROM таблиця [ WHERE умова ] ;
```

де

*таблиця* – ім'я таблиці.

*умова* – задає рядки, які необхідно видалити. Умова складається з імен стовпців, виразів, констант, підзапитів і операторів їх порівняння.

Якщо фраза WHERE відсутня, то видаляються всі рядки таблиці. Щоб переконатися у тому, що видалення відбулося, необхідно зробити запит на вибірку вилучених даних командою SELECT.

Приклад видалення рядків про службовців, прийнятих на роботу після січня 1996 року:

```
DELETE FROM s_emp  
WHERE start_date>TO_DATE('31.01.1996', 'DD.MM.YYYY') ;
```

Перевірка видалення рядків таблиці:

```
SELECT * FROM s_emp  
WHERE start_date>TO_DATE('31.01.1996', 'DD.MM.YYYY') ;
```

## 2.6. Помилки, викликані порушенням обмежень цілісності, при виконанні команд вставки, оновлення і видалення

Сервер бази даних при вставці, оновленні і видаленні рядків з таблиці перевіряє можливість їхнього виконання згідно умов обмеження цілісності і

видає відповідне повідомлення у випадку виявлення помилок.

Приклад видачі повідомлення про помилку при спробі оновлення рядку новим значенням зовнішнього ключа. Спробуємо змінити номер відділу для всіх службовців відділу з номером 10:

```
UPDATE s_emp SET dept_id=60 WHERE dept_id=10 ;
```

Текст повідомлення про встановлену помилку:

ORA-02291: integrity constraint (S\_EMP\_DEPT\_FK) violated - parent key not found

Якщо зазначене значення 60 не існує в “батьківській” таблиці s\_dept, що є таблицею-посиланням у визначенні зовнішнього ключа dept\_id у “дочірній” таблиці s\_emp, то сервер бази даних видає повідомлення про помилку ORA-02291, інформуючи про те, що через встановлення обмеження цілісності зовнішнього ключа S\_EMP\_DEPT\_FK, такого значення 60 первинного ключа у “батьківській” таблиці не виявлено.

Приклад видачі повідомлення про помилку при спробі видалення рядка, на значення якого встановлене обмеження цілісності первинного ключа. Спробуємо видалити відділ с номером 10:

```
DELETE s_dept WHERE id=10 ;
```

Текст повідомлення про встановлену помилку:

ORA-02292: integrity constraint (S\_EMP\_DEPT\_FK) violated - child record found

Якщо зазначене значення 10 первинного ключа id “батьківської” таблиці s\_dept, що є таблицею-посиланням у визначенні зовнішнього ключа у “дочірній” таблиці s\_emp, використовується як значення цього зовнішнього ключа, то сервер бази даних видає повідомлення про помилку ORA-02292, інформуючи про те, що через встановлення обмеження цілісності зовнішнього ключа S\_EMP\_DEPT\_FK, таке значення 10 виявлене у зовнішньому ключі “дочірній” таблиці.



## 2.7. Обробка транзакцій

Сервер бази даних забезпечує погодженість даних на основі механізму транзакцій. Транзакції забезпечують велику гнучкість, більш широкий спектр засобів управління при зміні даних, а також їхню погодженість у випадку помилок при виконанні команд або збої системи.

Транзакції складаються з команд DML, що вносять у дані одну погоджену зміну. Наприклад, переказ коштів між двома рахунками обов'язково включає дебетування одного рахунку і кредитування іншого на ту ж саму суму. Успішне або невдале виконання повинно бути одночасним для обох дій у рахунках. Тобто, залишати в базі дані про кредитування без даних про дебетування не припустимо.

Тип транзакції визначається належністю команди, що її утворює, до типу мови структурованих запитів. Типи транзакцій наведені у табл.14.

Таблиця 14

Тип	Опис
Маніпулювання даними (DML)	Складається з довільної кількості команд DML, сприйнятих сервером бази даних як одна логічна одиниця роботи
Визначення даних (DDL)	Складається з однієї команди DDL
Керування даними (DCL)	Складається з однієї команди DCL

Транзакція починається, коли сервер бази даних одержує першу команду, що змінює стан бази, і закінчується, коли відбувається щось одне з наступних подій:

- 1) Виконання команди управління транзакцією COMMIT або ROLLBACK;
- 2) Виконання команди DDL (наприклад, CREATE) або команда DCL;
- 3) Виявлення певних помилок, наприклад, взаємного блокування;
- 4) Завершення користувачем сеансу роботи з базою даних;
- 5) Програмно-апаратний збій або аварійне зупинення системи.

Після завершення однієї транзакції автоматично починається наступна

після успішного виконання команди DML, DDL або DCL. Результати виконання команд DDL і DCL фіксуються автоматично. Отже, ці команди неявно завершують транзакцію.

2.7.1. Команди управління транзакціями. Для явного управління логікою транзакцій використовуються команди COMMIT, SAVEPOINT і ROLLBACK (див. табл.15).

Таблиця 15

Команда	Опис
COMMIT	Завершує поточну транзакцію, роблячи постійними всі зроблені зміни даних
SAVEPOINT <i>ім'я</i>	Встановлює у поточній транзакції маркер збереження (savepoint) – точку відкоту
ROLLBACK [TO SAVEPOINT <i>ім'я</i> ]	Припиняє поточну транзакцію, скасовуючи всі зроблені зміни в даних до встановленого маркеру

Неявна обробка транзакцій виконується у випадках, наведених у табл.16.

Таблиця 16

Статус	Причина
Автоматична фіксація	Команда DDL чи DCL
	Нормальне завершення сеансу роботи в базі даних без явного виконання команди COMMIT чи ROLLBACK
Автоматичний відкіт	Аварійне припинення сеансу роботи або системний збій у базі даних

2.7.2. Фіксація змін. Кожна зміна даних, що виконана в ході транзакції, є тимчасовою, доти транзакція не буде зафіксована.

До виконання фіксації транзакції дані будуть знаходитися в наступному стані:

1) команди по обробці (маніпулюванню) даних змінюють тільки буфер бази даних. Отже, попередній стан даних може бути відновлено;

2) поточний користувач може переглянути результати своїх команд по обробці даних, виконуючи запити до таблиць бази;

3) інші користувачі не можуть бачити результати команд обробки даних, виконуваних поточним користувачем. СУБД забезпечує погодженість читання, і кожен користувач бачить дані такими, якими вони були на момент виконання останньої команди COMMIT;

4) змінені рядки таблиць блокуються і інші користувачі не мають можливості їх змінювати.

Усі внесені зміни фіксуються за допомогою команди COMMIT, у результаті чого дані переходять у наступний стан:

1) змінені дані записуються в базу даних. Попередній стан даних втрачається;

2) усі користувачі можуть бачити результати виконаних команд;

3) блокування знімається зі змінених рядків, інші користувачі одержують можливість вносити в них чергові зміни;

4) усі маркери збереження видаляються.

Приклад створення нового відділу, до якого переводиться службовець з номером 2. Усі зміни даних по завершенні переведення фіксуються:

```
INSERT INTO s_dept (id, name, region_id) VALUES (54, 'Education', 1) ;
```

```
UPDATE s_emp SET dept_id=54 WHERE id=2 ;
```

```
COMMIT ;
```

2.7.3.Скасування змін. Усі незафіксовані зміни скасовуються командою ROLLBACK, у результаті чого дані переходять у наступний стан:

1) зроблені зміни втрачаються. Відновлюється попередній стан даних;

2) розблоковуються рядки, з якими виконувалися операції, і інші користувачі одержують можливість вносити в них зміни.

Приклад скасування змін при видаленні записів. Під час видалення запису з таблиці test помилково стерті усі дані з цієї таблиці, які потрібно відновити:

```
DELETE FROM test ;
ROLLBACK ;
```

2.7.4. Скасування змін до маркеру збереження. Команда SAVEPOINT дозволяє створювати в поточній транзакції точки відкоту, що розбивають її на одиниці меншого розміру. Після цього за допомогою команди ROLLBACK TO можна скасовувати незафіксовані зміни до зазначеного маркеру. Створювати можна кілька маркерів збереження в одній транзакції. Повторне ж створення маркеру збереження з тим же ім'ям, викликає видалення створення попереднього такого маркеру.

Приклад скасування змін до точки відкоту. Створюється маркер збереження update\_done між двома командами для можливого повернення до стану виконання першої команди:

```
UPDATE s_emp SET comission=comission*1.1 WHERE dept_id=54 ;
SAVEPOINT update_done ;
INSERT INTO s_region (id, name) VALUES (8, 'Central')
ROLLBACK TO update_done ;
```

2.7.5. Відкот на рівні команди. Якщо при виконанні команди виявлена помилка, то частина транзакції може бути скасована шляхом неявного відкоту. Якщо в ході транзакції помилка виникла в одній команді DML, то скасовується результат тільки цієї команди, тобто здійснюється відкіт на рівні команди. Однак зміни, внесені під час цієї транзакції попередніми командами DML, зберігаються. Вони можуть бути зафіксовані або скасовані у явній формі.

СУБД виконує неявну команду COMMIT до і після виконання будь-якої команди мови визначення даних DDL. Тому, навіть, якщо команда DDL виконана невдало, скасувати зміни, зроблені попередньою командою, неможливо, тому що сервер вже їх зафіксував.

Завершувати явно транзакції слід командою COMMIT або ROLLBACK.

### 3. Порядок роботи з інструментальними засобами взаємодії із сервером бази даних

Будь-які інструментальні засоби і прикладні додатки здійснюють взаємодію із сервером бази даних через мову команд запитів SQL. До складу програмного забезпечення сервера бази даних Oracle входить спеціалізований консольний додаток Oracle SQL\*Plus та програмне середовище взаємодії і адміністрування сервера бази даних Oracle SQL Developer.

**Oracle SQL\*Plus** – це інтерпретатор команд із мови SQL для виконання дій по створенню інформаційних об'єктів бази даних, їхньої модифікації і наповнення даними.

**Oracle SQL Developer** – це графічний інструмент для розробки і адміністрування баз даних. За допомогою SQL Developer можна переглядати об'єкти бази даних, запускати SQL-команди, редагувати і налагоджувати PL/SQL-програми. Ви також можете запустити будь-яку кількість наданих звітів, а також створювати і зберігати власні. SQL Developer підвищує продуктивність і спрощує підтримку вашої бази даних при виконанні завдань з її розвитку. SQL Developer може підключатися до будь-якої СУБД Oracle від версії 9.2.0.1 і може працювати на Windows, Linux, Mac OSX.

Порядок взаємодії з інструментальними засобами взаємозв'язку із сервером Oracle:

1. Запуск і реєстрація користувача в базі даних. Необхідно обрати попередньо створене підключення до бази (наприклад, base), ввести ім'я користувача - ідентифікатор студента в групі (наприклад, ik3101), пароль.

2. Введення команди і запуск її на виконання здійснюється в робочому листі вікна інструментального засобу. Команда може розташовуватися на декількох рядках. Але повинна завершуватися символом «;» на поточному рядку або «/» на наступній рядку.

3. Виконання командного файлу (попередньо збереженого на локальному диску) здійснюється вказівкою символу «@» і наведенням повної специфікації файлу.

4. Вихід з інструментального засобу здійснюється закриттям його вікна.

Oracle SQL Developer це програмний засіб для взаємодії з сервером бази даних у вигляді GUI-додатків. Тому він підтримує роботу, як у командному рядку робочого листа додатку, так і у меню-орієнтованому інтерфейсі з автоматичною генерацією виконуваних SQL-команд. Робота в такому засобі здійснюється в інтерактивному режимі за допомогою вікон.

Основним вікном у Oracle SQL Developer є навігаційне вікно з деревом доступних об'єктів бази даних. Правою кнопкою миші видаються дозволені операції модифікації обраного об'єкта бази даних.

З основного меню додатку викликається SQL Worksheet - робочий лист редактора команд SQL. Виклик редактора рядків команд здійснюється для введення як одиничних команд SQL без завершального символу «;», так і їх послідовності, які розділені завершальним символом виконання команди «;». Запуск команд на виконання здійснюється від поточного положення маркера до кінця послідовності команд кнопкою F5 або тільки поточної команди за допомогою кнопки F9 або Ctrl-Enter.

Контроль виконання команд виконується переглядом пунктів дерева у навігаційному вікні. Можна зберегти введену послідовність команд у вигляді файлу з типом sql за допомогою пункту головного меню «File / Save».

#### 4. Контрольне завдання

1. Виконати вставку даних до таблиць REGION, DEPARTMENT, EMPLOYEE, що були створені в комп'ютерному практикумі №2, згідно наступного порядку:

1.1) відновіть таблицю EMPLOYEE відповідно до завдання попереднього практикуму. Перегляньте опис у словнику даних таблиць REGION, DEPARTMENT, EMPLOYEE для з'ясування імен стовпців, у які обов'язкове введення значень при вставці;

1.2) перегляньте обмеження, що відносяться до кожної таблиці (первинні, зовнішні, унікальні ключі і т.д.), через виконання типового запиту до словника

даних бази. Внесіть зміни в обмеження таблиці DEPARTMENT, замінивши унікальний ключ стовпця REGION\_ID на унікальний складений ключ із стовпців NAME, REGION\_ID;

1.3) додайте рядок даних у таблицю REGION з номером регіону 44 і назвою «Київ». Додайте рядок даних у таблицю DEPARTMENT з номером відділу 10 і назвою «факультет інформатики й обчислювальної техніки». У команді вставки стовпці не вказуйте. Створіть командний файл lab3z1\_3.sql з командами вставки, а потім перевірте результат внесення даних. Поясніть отриманий результат і, якщо завдання не виконано, знайдіть спосіб його успішного виконання;

1.4) створіть командний файл lab3z1\_4.sql, який запитує значення персонального номера студента, прізвища і номера відділу і вводить новий рядок з цими значеннями в таблицю EMPLOYEE. З його допомогою додайте два рядки у таблицю. У першому випадку введіть власне прізвище з персональним номером 200 і номером відділу 10, а в другому – введіть прізвище іншого студента вашої групи з персональним номером 201 і номером відділу 11. Перевірте внесення даних у таблицю. Поясніть отриманий результат;

1.5) введіть у таблицю DEPARTMENT відомості про факультет прикладної математики з номером 10. Поясніть отриманий результат;

1.6) створіть командний файл lab3z1\_6.sql, що додає в таблицю DEPARTMENT відомості про факультет прикладної математики (ФПМ) з номером 37, електроніки (ФЭ) з номером 54 та авіаційних і космічних систем (ФАКС) з номером 75. Виконаєте командний файл lab3z1\_4.sql і створіть у таблиці EMPLOYEE рядки з даними про студентів з персональним номером 201 з ФЭ, з персональним номером 202 з ФАКС, з персональним номером 203 із ФПМ. Перевірте внесення даних у таблиці і зробіть їх постійними.

2. Виконати зміну і видалення даних з таблиць DEPARTMENT і EMPLOYEE згідно наступного порядку:

2.1) змініте назву факультету під номером 75 на будь-яку іншу назву;

2.2) змініте прізвище студента з персональним номером 202 на власну;

2.3) перевірте правильність зроблених змін у таблицях;

2.4) видалите дані про факультет під номером 54 і дайте пояснення отриманому результату;

2.5) видалите інформацію про студентів, що навчаються на факультеті під номером 54, і знову зробіть видалення даних про факультет з номером 54. Поясніть отриманий результат;

2.6) перевірте правильність зроблених змін у таблицях і зробіть їх постійними.

3. Виконати управління транзакцією з даними у таблицях DEPARTMENT і EMPLOYEE згідно наступного порядку:

3.1) виконайте командний файл lab3z1\_6.sql для відновлення в таблиці DEPARTMENT даних про факультет з номером 54. Перевірте правильність внесених змін;

3.2) створіть маркер збереження в транзакції;

3.3) видалите всі дані з таблиці EMPLOYEE. Перевірте стан таблиці;

3.4) скасуйте результати останнього видалення, але збережіть результати відновлення факультету з номером 54. Перевірте стан таблиць DEPARTMENT і EMPLOYEE і порівняйте з попереднім. Зафіксуйте ці зміни і зробіть їх постійними.

3.5) створіть командний файл lab3z3\_5.sql для вставки рядка в таблицю DEPARTMENT з використанням послідовності DEPT\_ID\_SEQ. Перевірте стан послідовності, якщо вона не придатна до використання, то відновте її відповідно до завдання практикуму №2, зробивши її некешованою, нециклічною. У командному файлі виконайте запит значення назви відділу з наступною вставкою рядка відділу у таблицю DEPARTMENT. За допомогою командного файлу додайте два відділи – Адміністративний, Навчальний. Перевірте внесені дані та зробіть їх постійними.

3.6) За допомогою командного файлу lab3z3\_5.sql додайте до таблиці DEPARTMENT це чотири відділи: Бухгалтерія, Склад, Канцелярія,



Дослідницький. Перевірте внесені дані, зробіть їх постійними та поясніть отриманий результат. Якщо дані не внесені, то змініть параметри послідовності для успішного виконання завдання.

Усі команди контрольного завдання подати у виді єдиного командного файлу, який представити у звіті з необхідними поясненнями.

## Комп'ютерний практикум № 4.

# ФОРМУВАННЯ ЗАПИТІВ НА ВИВЕДЕННЯ ІНФОРМАЦІЇ З БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ

### 1. Мета та основні завдання практикуму

Мета роботи: ознайомитися з формами команди формування запитів для організації виведення інформації з бази даних.

Порядок виконання роботи:

- вивчити форми команди формування запитів;
- визначити основні компоненти процедури формування запитів до бази даних;
- згідно умов контрольного завдання наповнити базу даних необхідною інформацією і дослідити можливості різних форм запитів на виведення інформації;
- підготувати звіт з результатами обробки даних у базі даних.

### 2. Основні теоретичні відомості

#### 2.1. Специфікація форм запитів на виведення даних

Для виведення даних з бази даних використовується команда SELECT мови структурованих запитів SQL. Основними способами вибірки даних з бази є наступні форми запитів:

- вибірка всіх рядків таблиці;
- обмежений упорядкований вибір рядків таблиці;
- вибірка рядків з'єднання декількох таблиць;
- групова вибірка;
- вибірка з вкладеним підзапитом;
- ієрархічна вибірка.

Узагальнений синтаксис команди формування запиту на виведення інформації є наступним:

```
SELECT [{DISTINCT|ALL}] { * | {стовпець, вираз, (підзапит)} [AS
```

*псевдонім*] } [ , ... ]

FROM { *таблиця* | (*підзапит*) [*табличний\_псевдонім*] } [ , ... ]

[ WHERE *умова\_вибірки* ]

[ GROUP BY *вираз\_групи* [HAVING *умова\_групи*] ] |

[ [ START WITH *умова\_коренева* ] CONNECT BY *умова\_ієрархії* ]

[ ORDER BY { *стовпець, вираз, псевдонім* } [ASC|DESC], ... ] ;

де

DISTINCT – виключає дублювання значень рядків,

ALL – включає всі значення рядків (у т.ч. і дублікати),

\* – вибирає всі стовпці із джерела інформації – таблиць, підзапитів,

*стовпець* – вибирає заданий стовпець,

*вираз* – функція обробки значень групи рядків, арифметичний вираз з стовпців, константа,

*псевдонім* – дає обраному стовпцю або виразу із стовпців таблиці інше ім'я у запиті,

*таблиця* – вказує таблицю джерела інформації, що містить задані стовпці,

*підзапит* – вказує іншу команду SELECT, що вибирає дані, які пов'язані з поточними даними головного запиту і подаються як таблиця джерела інформації із назвою псевдоніма;

*умова\_вибірки* – обмежує запит рядками, що задовольняють заданій умові (складається з імен стовпців, виразів, підзапитів, констант, пов'язаних операторами порівняння),

*вираз\_групи* – задає список стовпців, на основі однакових значень яких групуються рядки (утворюється група рядків),

*умова\_групи* – включає в результат тільки ті групи, для яких виконується умова,

*умова\_коренева* – задає умову відбору корневих (початкових) вершин дерева ієрархічної вибірки підпорядкованих рядків таблиці,

*умова\_ієрархії* – задає умову підпорядкування та вибірки всіх наступних вершин в утворюваній ієрархії рядків таблиці,

ORDER BY – задає послідовність (порядок) виведення обраних рядків,  
ASC – задає сортування рядків по зростанню значень,  
DESC – задає сортування рядків по спаданню значень.

## 2.2. Вибірка всіх рядків таблиці

У найпростішій формі команда SELECT повинна включати наступне:

- фразу SELECT з переліком необхідних стовпців. Зірочка (\*) означає вибір усіх стовпців таблиці або всіх джерел інформації запита;
- фразу FROM із вказівкою хоча б однієї запитуваної таблиці.

Приклад виведення вмісту всіх стовпців і рядків таблиці s\_dept (вибірка всіх даних):

```
SELECT * FROM s_dept ;
```

Приклад виведення усіх рядків перерахованих стовпців таблиці s\_dept (вибірка проекції даних):

```
SELECT id, name FROM s_dept ;
```

Стовпці в команді SELECT вказуються в послідовності, у якій повинен здійснюватися їхнє виведення. Коми між іменами стовпців обов'язкові.

2.2.1. Псевдоніми стовпців. Результат виконання запиту завжди подається як таблиця із заголовком стовпців, а при виводі результатів запиту в якості заголовків стовпців звичайно використовуються їх імена. Але іноді такі заголовки часто важкі для розуміння і навіть безглузді (у випадку застосування виразів або однорядкових функцій до стовпців запитуваної таблиці). Змінити заголовок можна за допомогою псевдоніма стовпця.

Псевдонім застосовуються у фразі SELECT відразу поза іменем стовпця і відокремлюється від нього або пробілом або ключовим словом AS. За замовчуванням такі альтернативні заголовки виводяться в символах верхнього регістра і не можуть містити пробілів. Якщо псевдонім повинен містити пробіли його значення береться у подвійні лапки ("псевдонім").

Приклад виведення прізвища, заробітної плати і суми компенсаційних

виплат за рік для кожного службовця. Обсяг виплат за рік обчислюється шляхом додатка до заробітної плати щомісячної премії в розмірі 100у.о. і множення суми на 12. Потрібно позначити цей стовпець як ANNUAL\_SALARY або Зарплата за рік.

```
SELECT last_name, salary, 12*(salary+100) AS ANNUAL_SALARY  
FROM s_emp ;
```

або

```
SELECT last_name, salary, 12*(salary+100) "Зарплата за рік"  
FROM s_emp ;
```

2.2.2.Оператор конкатенації. Оператор конкатенації (||) дозволяє з'єднувати стовпці з іншими стовпцями, арифметичними виразами або постійними значеннями для створення символічних виразів. Стовпці, зазначені по обох сторонах цього оператора, поєднуються в один стовпець.

Приклад виведення імені і прізвища службовців під заголовком Employees:

```
SELECT first_name || ' ' || last_name AS Employees FROM s_emp ;
```

2.2.3.Запобігання виведення дублікатів рядків. При відсутності явних вказівок СУБД включає в результат запиту усі рядки, не видаляючи їхні дублікати (режим ключового слова ALL). Ключове слово DISTINCT, що вказується відразу за фразою SELECT, виключає дублювання рядків (з однаковими значеннями). За замовченням після фрази SELECT розуміється слово ALL.

Приклад виведення всіх неповторюваних назв відділів з таблиці s\_dept:

```
SELECT DISTINCT name FROM s_dept ;
```

Після слова DISTINCT можна вказувати декілька стовпців. У цьому випадку воно буде відноситися до всіх обраних стовпців.

2.3. Вибірка обмеженої кількості рядків у заданому порядку значень

При вибірці даних з бази даних користувач може обмежити кількість вихідних рядків і задати їх порядок виведення.

2.3.1.Сортування рядків результату запиту. Сортування рядків визначається за допомогою фрази ORDER BY. Для сортування необхідно задати стовпець, вираз із стовпців або позицію стовпця в списку фрази SELECT запиту.

Приклад виведення з таблиці s\_emp прізвища, номера відділу і дати початку роботи кожного службовця. Результат сортується за прізвищем:

```
SELECT last_name, dept_id, start_date FROM s_emp  
ORDER BY last_name ;
```

За замовчуванням рядки сортуються в порядку зростання (додаткове слово ASC):

- виведення числових значень здійснюється від менших до більших значень – наприклад, 1–999;

- виведення дат починається з більш ранніх – наприклад, 01-JAN-92 передує 01-JAN-95;

- виведення символічних значень здійснюється за абеткою – наприклад, від А до Z;

- невизначені значення при сортуванні по зростанню виводяться останніми, а при сортуванні по спаданню - першими.

Порядок сортування, прийнятий за замовчуванням, міняється на протилежний за допомогою додаткового слова DESC після імені стовпця у фразі ORDER BY.

Приклад виведення з таблиці s\_emp прізвища, номера відділу і дати початку роботи кожного службовця. Результат сортується таким чином, щоб службовці, найняті останніми, очолювали список:

```
SELECT last_name, dept_id, start_date FROM s_emp  
ORDER BY start_date DESC ;
```

У фразі ORDER BY можна вказувати псевдонім стовпця або його позицію. Він особливо корисний при сортуванні за довгим виразом. Замість повторного введення виразу можна вказати його позицію в списку фрази SELECT:

```
SELECT last_name, salary*12 FROM s_emp ORDER BY 2 ;
```

Сортувати результат можна і по декількох стовпцях. Граничною кількістю стовпців сортування є кількість стовпців таблиці. Стовпці вказуються в фразі ORDER BY через кому. Для зміни порядку сортування по якому-небудь стовпцю на зворотний варто задати слово DESC після його імені чи позиції. Сортувати можна і по стовпцях, що не входять у список SELECT, явно вказавши його ім'я.

Приклад виведення прізвища, номера відділу і заробітної плати всіх службовців. Результат сортується по номерах відділів, а усередині відділів - у порядку спадання значення заробітної плати:

```
SELECT last_name, dept_id, salary FROM s_emp
ORDER BY dept_id ASC, salary DESC;
```

2.3.2.Обмеження кількості рядків. Обмежити набір рядків, які виводяться в результаті запиту, можна за допомогою фрази WHERE. Фраза WHERE вказується відразу за фразою FROM і задає умову, яка повинна бути виконана. Умова складається з імен стовпців, виразів, підзапитів, констант і операторів їх порівняння.

Оператори порівняння поділяються на дві категорії: логічні й предикати SQL. У фразі WHERE вони використовуються для порівняння виразів у наступному форматі:

*WHERE вираз оператор\_порівняння вираз*

Приклад простої умови з логічним оператором порівняння:

```
WHERE dept_id = 42
```

Рядки символів, використані як константи в умові у фразі WHERE повинні бути укладені в апострофи. Числові константи в апострофи не укладаються.

Логічні оператори порівняння, що застосовуються для перевірки умов наведені у табл.17.

Таблиця 17

Логічний оператор	Значення
=	рівно
>	більше
>=	більше або рівно
<	менше
<=	менше або рівно

Крім логічних операторів, застосовуються чотири оператори-предикати SQL, наведених в табл.18.

Таблиця 18

Оператор-Предикат	Значення
BETWEEN <i>значення1</i> AND <i>значення2</i>	між двома значеннями (включно)
IN( <i>список</i> )	збігається з якимсь зі значень у списку
LIKE <i>значення шаблону</i>	відповідає символному шаблону
IS NULL	є невизначеним значенням

Складні вирази умови запиту конструюються з простих порівнянь і булевих операторів зв'язування:

- 1) AND – якщо обидві частини умови вірні, то результат вірний;
- 2) OR – якщо хоча б одна частина умови вірна, то результат вірний;
- 3) NOT – повертає протилежне значення умови.

Іноді простіше знайти рядки, що не задовольняють умові, ніж рядки, що йому задовольняють. Зробити це можна за допомогою операторів заперечення порівняння, наведених у табл.19.

Таблиця 19

Оператор заперечення	Опис
!=	нерівно
<>	нерівно



Оператор заперечення	Опис
NOT <i>стовпець</i> = значення	нерівно
NOT <i>стовпець</i> > значення	не більше
NOT BETWEEN <i>значення1</i> AND <i>значення2</i>	не є між двома заданими значеннями
<i>стовпець</i> NOT IN ( <i>список</i> )	не входить до списку значень
<i>стовпець</i> NOT LIKE <i>значення шаблону</i>	не подібно заданому значенню
<i>стовпець</i> IS NOT NULL	не є невизначеним значенням

Для порівняння відомого значення з невизначеним значенням варто використовувати оператор порівняння IS NULL або IS NOT NULL. При порівнянні невизначених значень за допомогою інших операторів результат завжди буде «НЕВІРНО». Наприклад, COMM != NULL завжди «НЕВІРНО», тому що невизначене значення не може дорівнювати або не дорівнювати іншому значенню, навіть невизначеному. Слід зазначити, що при такій перевірці стану помилки не виникає. Просто результатом завжди буде «НЕВІРНО».

Невизначені значення перевіряються завжди за допомогою оператора-предиката IS NULL.

Приклад виведення прізвища і розміру зарплати для всіх службовців, що їх одержують, тобто мають визначені значення зарплати:

```
SELECT last_name, comission FROM s_emp
WHERE comission IS NOT NULL ;
```

Оператор-предикат BETWEEN дозволяє виводити рядки, які містять значення із заданого діапазону значень. Діапазон задається верхньою і нижньою границями.

Приклад виведення імені, прізвища і дати прийняття на роботу службовців, найнятих між 9 травня і 17 червня 1991 року включно:

```
SELECT first_naine, last_name, start_date FROM s_dept
WHERE start_date BETWEEN '09-MAY-91' AND '17-JON-91' ;
```

Значення, зазначені в операторі BETWEEN, входять у діапазон. Нижня

границя повинна бути зазначена першою.

Для перевірки приналежності значень до заданого списку використовується оператор-предикат IN.

Приклад виведення номера, назви відділу і номера регіону для відділів у регіонах з номерами 1 і 3:

```
SELECT id, name, region id FROM s_dept WHERE region_id IN (1,3) ;
```

Якщо в списку присутні символічні рядки або дати, вони повинні бути укладені в апострофи - 'текстовий рядок', '01-02-2019'.

Шукане значення не завжди достовірно відомо. Оператор-предикат LIKE дозволяє робити пошук за деяким символічним шаблоном на співпадіння значень. Така операція називається пошуком за мета символами. Для створення шуканого рядку для шаблону збігу можна використовувати два «спецсимволи»:

- 1) % - представляє будь-яку послідовність символів;
- 2) \_ - представляє будь-який одиночний символ.

Приклад виведення всіх прізвищ службовців на першу букву "М":

```
SELECT last_name FROM s_emp WHERE last_name LIKE 'M%' ;
```

Приклад виведення всіх прізвищ службовців, що не містять букви "а":

```
SELECT last_name FROM s_ernp WHERE last_name NOT LIKE '%a%' ;
```

Приклад виведення всіх прізвищ службовців, що містять першу "К", третю і четверту "ав" і будь-яку другу та всі інші:

```
SELECT last_name FROM s_ernp WHERE last_name LIKE 'K_ав%' ;
```

2.3.3.Вибірка даних за декількома умовами. Іноді виникає потреба у складних критеріях пошуку, що включають кілька умов. Для створення складних логічних виразів користуються операторами AND і OR.

У двох наступних прикладах умови однакові, але використовуються різні оператори. Тому різниця результатів дуже велика.

Приклад виведення прізвища, заробітної плати і номера відділу тих

працівників відділу 41, які найняті 1 травня 1991 року:

```
SELECT last_name, salary, dept_id FROM s_emp  
WHERE dept_id =41 AND start_date='01-MAY-91' ;
```

Приклад виведення прізвища, заробітної плати і номера відділу всіх працівників відділу 41 або найнятих 1 травня 1991 року:

```
SELECT last_name, salary, dept_id FROM s_emp  
WHERE dept_id =41 OR start_date='01-MAY-91' ;
```

Оператор OR є менш обмежуючим, тому кількість вихідних рядків може зрости.

Оператори AND і OR можна з'єднати в тому самому логічному виразі. Порядок обчислення сукупності умов залежить від операторів, що з'єднують ці умови. Якщо поруч задано два оператори того самого пріоритету, вони обробляються по черзі.

Спочатку виконується кожен оператор AND, а потім - кожен оператор OR. Пріоритет оператора AND вище. Правило обчислення пріоритету наступне:

- 1) Всі оператори порівняння =, >, >=, <, <=, IN, LIKE, IS NULL, BETWEEN;
- 2) AND;
- 3) OR.

У випадку заперечення виразу оператори порівняння також виконуються першими.

Стандартний порядок виконання операцій можна змінити, уклавши частину виразу в дужки. СУБД насамперед обчислює вирази, укладені в дужки.

#### 2.4. Вибірка даних із з'єднанням декількох таблиць

Якщо потрібні дані з більш, ніж однієї таблиці бази даних, використовується вибірка-з'єднання. Рядки однієї таблиці з'єднуються з рядками іншої попарно або відповідно до спільних значень у певних стовпцях – стовпцях первинних і зовнішніх ключів.

З'єднання - це запит, який поєднує рядки з двох або більше таблиць. СУБД виконує з'єднання кожного разу, коли у фразі FROM з'являється кілька таблиць запиту. У списку вибору запиту можна вибрати будь-які стовпці з будь-якої із цих таблиць. Якщо будь-яка з цих двох таблиць має спільну назву стовпця, тоді необхідно ідентифікувати всі посилання на ці стовпці з іменами таблиць.

Більшість запитів з'єднання містять щонайменше одну умову приєднання або у фразі FROM, або у фразі WHERE.

Для виконання з'єднання СУБД поєднує пари рядків, кожен з яких містить один рядок з кожної таблиці, для якого умова приєднання оцінюється як «ВІРНО». Стовпці з умови приєднання не потрібно також відображати у списку вибору запиту.

Застосовується такі основних типи з'єднання:

- декартовий добуток (CROSS JOIN);
- просте (внутрішнє) з'єднання (INNER JOIN ON);
- еквіз'єднання (INNER JOIN USING);
- природне з'єднання (NATURAL JOIN)
- не-еквіз'єднання (JOIN ON);
- зовнішнє з'єднання (OUTER JOIN ON);
- рекурсивне з'єднання.

Зовнішнє і рекурсивне з'єднання є розширенням простого внутрішнього з'єднання.

Узагальнений синтаксис вибірки даних із з'єднанням декількох таблиць:

```
SELECT таблиця.стовпець [ , ... ]
FROM таблиця [псевдонім] { CROSS JOIN | NATURAL JOIN |
  { INNER | { LEFT|RIGHT|FULL } OUTER }
JOIN з'єднувальна_таблиця [псевдонім]
  { ON умова_приєднання [{AND|OR} умова_приєднання [ ... ] ]
  | USING (стовпець_приєднання [ , ... ] ) }
} [ ... ]
```

[ WHERE умова\_вибірки ]

де

*таблиця.стовпець* – стовпець вибірки даних з джерела *таблиця*,

*таблиця* – таблиця джерела вибірки даних,

*з'єднувальна\_таблиця* – друге джерело вибірки даних,

*умова\_приєднання* – умова поєднання рядків з джерел вибірки даних,

*стовпець\_приєднання* – однойменний стовпець поєднання рядків з джерел вибірки даних.

Якщо стовпці, за якими здійснюється з'єднання таблиць мають співпадаючі ім'я, то замість умови з ON можна записати USING.

2.4.1.Декартовий добуток. Якщо умова з'єднання опущена, результатом запиту буде декартовий (cartesian) добуток двох таблиць, який включає всі комбінації рядків. Усі рядки першої таблиці з'єднуються з усіма рядками другої таблиці.

Приклад декартового добутку:

```
SELECT name, last_name FROM s_dept, s_emp ;
```

або у новому форматі стандарту мови SQL

```
SELECT name, last_name FROM s_dept CROSS JOIN s_emp ;
```

Декартовий добуток двох таблиць, кожна зі 100 рядків, має 10 000 рядків.

Отже, декартовий добуток завжди генерує багато рядків і рідко є корисним.

2.4.2.Просте (внутрішнє) з'єднанням. Для відтворення даних із двох або більш пов'язаних таблиць при поєднанні пари рядків необхідно задати просту умову з'єднання у фразі WHERE або ON.

Синтаксис простого внутрішнього з'єднання:

```
SELECT таблиця1.стовпець [, таблиця2.стовпець [, ... ] ]
```

```
FROM таблиця1, таблиця2 [, ... ]
```

```
WHERE таблиця1.стовпець1 = таблиця2.стовпець2 [ {AND|OR} ... ] ;
```

або у новому форматі стандарту мови SQL

```
SELECT таблиця1.стовпець [, таблиця2.стовпець [, ... ] ]
FROM таблиця1 [INNER] JOIN таблиця2
ON таблиця1.стовпець1 = таблиця2.стовпець2 [ {AND|OR} ... ]
[ ... ];
```

де

*таблиця1.стовпець*, *таблиця2.стовпець* – стовпці таблиць, з яких здійснюється вибірка даних, що відповідають умові простого внутрішнього з'єднання;

*таблиця1.стовпець1* = *таблиця2.стовпець2* – умова, що поєднує пари рядків з таблиць внутрішнього з'єднання (або задає їх взаємозв'язок).

Для спрощення читання команд SELECT, що припускають з'єднання таблиць, і розширення можливостей доступу до бази даних варто вказувати імена таблиць перед іменами стовпців. Якщо більш, ніж в одній таблиці, є стовпці з однаковими іменами, вказівка імені таблиці перед ім'ям стовпця обов'язкова. Мінімальна кількість умов з'єднання таблиць дорівнює кількості таблиць, що з'єднуються, мінус один. Отже, для з'єднання чотирьох таблиць вимагається, принаймні, три умови. Це правило може не відноситися до таблиць зі складеним первинним ключем – у цьому випадку для однозначної ідентифікації рядка потрібно більш одного стовпця.

Приклад внутрішнього з'єднання для визначення назви відділу, у якому працює службовець:

```
SELECT s_dept.name, s_emp.last_name
FROM s_dept, s_emp
WHERE s_dept.id = s_emp.dept_id ;
```

або

```
SELECT s_dept.name, s_emp.last_name
FROM s_dept INNER JOIN s_emp
ON s_dept.id = s_emp.dept_id ;
```

Для визначення назви відділу, у якому працює службовець, значення зовнішнього ключа стовпця *dept\_id* у таблиці *s\_emp* порівнюється зі

значеннями первинного ключа `id` таблиці `s_dept`. Відношення між таблицями `s_dept` і `s_emp` являє собою з'єднання по однаковим значенням їх стовпців первинного і зовнішнього ключа відповідно.

Щоб уникнути неоднозначності читання іменам стовпців у фразі `WHERE` повинні передувати імена таблиць. Без такого префікса стовець `id` може належати як таблиці `s_dept`, так і таблиці `s_emp`. Для виконання запити необхідно додати префікси. Якщо однойменних стовпців у таблицях немає, то для розрізнення стовпців вказівка імен таблиць не потрібна. Однак, завдяки префіксам збільшується продуктивність виконання запити.

Крім з'єднання у фразі умови до `WHERE` можна задавати й інші критерії відбору. Оскільки з'єднання необхідне для відбору відповідностей, то додаткова умова має додаватися за допомогою оператора `AND`. Використання псевдонімів таблиць дозволяє зменшити обсяг коду `SQL`, що скорочує витрату пам'яті на подання та обробку запити.

Приклад виведення прізвища, номера відділу і назви відділу співробітника Петрова:

```
SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
FROM s_emp, s_dept
WHERE s_emp.dept id = s_dept.id
AND INITCAP(s_emp.last_name)='Петров' ;
```

або

```
SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
FROM s_emp INNER JOIN s_dept
ON s_emp.dept id=s_dept.id
WHERE INITCAP(s_emp.last_name)='Петров' ;
```

Розрізнення стовпців за допомогою імен таблиць може віднімати дуже багато часу - особливо, якщо імена в таблицях довгі. Тому замість імен використовуються псевдоніми таблиць. Аналогічно псевдонімам стовпців

псевдоніми таблиць дозволяють привласнити таблиці інше ім'я для конкретного запиту SELECT. Уживши псевдонім таблиці один раз, необхідно продовжувати його використовувати для правильного визначення кожного стовпця.

Приклад виведення найменування клієнта, номера регіону і назви регіону для всіх клієнтів. Використовуються псевдоніми стовпців, а для спрощення посилань на таблиці - псевдоніми таблиць:

```
SELECT c.name AS "Customer Name", c.region_id AS "Region ID",
       r.name AS "Region Name"
FROM s_customer c, s_region r
WHERE c.region_id = r.id ;
```

або

```
SELECT c.name AS "Customer Name", c.region_id AS "Region ID",
       r.name AS "Region Name"
FROM s_customer c JOIN s_region r
ON c.region_id = r.id ;
```

Псевдоніми таблиць можуть містити до тридцяти символів, але чим вони коротше, тим краще. Якщо псевдонім таблиці використовується для вказівки таблиці у фразі FROM, то цей же псевдонім повинен використовуватися замість імені цієї таблиці у всій команді SELECT. Варто вибирати осмислені псевдоніми. Дія псевдоніма таблиці поширюється лише на поточну виконувану команду SELECT.

Псевдоніми таблиць використовуються не тільки щоб уникнути неоднозначності, а ще й для збільшення продуктивності виконання запитів.

2.4.3. Еквіз'єднання. Відношення між таблицями 1 і 2 є еквіз'єднанням, якщо один стовпець таблиці 1 відповідає точно такому ж стовпцю в таблиці 2 як за назвою, так і за значеннями. Таке відношення визначається оператором рівності (=).

Синтаксис еквіз'єднання:



```
SELECT таблиця1.стовпець [, таблиця2.стовпець [, ... ] ]
FROM таблиця1, таблиця2 [, ... ]
WHERE таблиця1.стовпець = таблиця2.стовпець [ {AND|OR} ... ] ;
```

або у новому форматі стандарту мови SQL

```
SELECT таблиця1.стовпець [, таблиця2.стовпець [, ... ] ]
FROM таблиця1 [INNER] JOIN таблиця2
USING (стовпець [, ... ] ) [ ... ] ;
```

Легко помітити, що в еквіз'єднанні таблиць входять дублікати стовпців, за якими проводилося з'єднання. Для виключення цих дублікатів можна створити природне з'єднання тих же таблиць за умовою збігу значень усіх однойменних стовпців.

Синтаксис команди природного з'єднання:

```
SELECT таблиця1.стовпець [, таблиця2.стовпець [, ... ] ]
FROM таблиця1 NATURAL JOIN таблиця2 [ ... ] ;
```

Для попереднього прикладу визначення назви відділу, у якому працює службовець, значення зовнішнього ключа стовпця `dept_id` у таблиці `s_emp` повинно порівнюватися зі значеннями первинного ключа таблиці `s_dept`, яке у випадку еквіз'єднання повинно мати таке ж ім'я `dept_id`. Тобто, відношення між таблицями `s_emp` і `s_dept` буде являти собою еквіз'єднання, якщо значення стовпця `dept_id` в обох таблицях будуть рівні.

Приклад з'єднання таблиць службовців і відділів для виведення прізвища службовця, номера і назви відділу:

```
SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
FROM s_emp, s_dept
WHERE s_emp.dept_id = s_dept.dept_id ;
```

або

```
SELECT s_emp.last_name, s_emp.dept_id, s_dept.name
FROM s_emp JOIN s_dept
USING (dept_id) ;
```

Використання природного з'єднання доцільне, коли одна з таблиць команди є доповненням іншої, наприклад, за допомогою встановлення однакового первинного або унікального ключа. Приклад з'єднання таблиць службовців `s_emp(emp_id primary key,last_name,work_place_id unique foreign key)` і їх робочих місць `s_work_place(work_place_id primary key,name)`, які мають серед інших однаковими тільки колонки `work_place_id`:

```
SELECT s_emp.emp_id, s_emp.last_name,
       s_work_place.work_place_id, s_work_place.name
FROM s_emp NATURAL JOIN s_work_place ;
```

2.4.4. Не-еквіз'єднання. Відношення між таблицями 1 і 2 не є еквіз'єднанням, якщо жоден стовпець таблиці 1 не відповідає точно стовпцю в таблиці 2. Таке відношення визначається оператором, що не є оператором рівності (=).

Синтаксис не-еквіз'єднання:

```
SELECT таблиця1.стовпець [, таблиця2.стовпець [, ... ] ]
FROM таблиця1, таблиця2 [, ... ]
WHERE таблиця1.стовпець1  $\theta$ -оператор таблиця2.стовпець2
      [ {AND|OR} ... ] ;
```

або у новому форматі стандарту мови SQL

```
SELECT таблиця1.стовпець [, таблиця2.стовпець [, ... ] ]
FROM таблиця1 JOIN таблиця2
ON таблиця1.стовпець1  $\theta$ -оператор таблиця2.стовпець2
   [ {AND|OR} ... ] [ ... ] ;
```

де

$\theta$ -оператор – всі оператори порівняння, окрім (=).

Можуть використовуватися всі інші оператори, крім (=), наприклад, `<=` і `>=`, але найпростіший оператор - це BETWEEN. Варто пам'ятати про те, що

після оператора BETWEEN вказується спочатку нижня границя діапазону, а потім - верхня.

Приклад створення не-еквів'єднання для обчислення категорії службовця по заробітній платі. Заробітна плата повинна бути між будь-якою парою нижнього і верхнього значень діапазону зарплат:

```
SELECT e.last_name, e.commission, s.grade
FROM s_emp e, s_salgrade s
WHERE e.commission BETWEEN s.lo_sal AND s.hi_sal ;
```

2.4.5.Зовнішнє з'єднання. Це вибірка записів при відсутності прямої відповідності. Якщо рядки не задовольняють умовам з'єднання, то вони не включаються в результат запиту. Однак, відсутній рядок або рядки можуть включатися в результат запиту, якщо в умові з'єднання використовується оператор зовнішнього з'єднання. Цей оператор являє собою знак плюс у дужках (+) і вказується на тій "стороні" з'єднання, де не вистачає потрібної для встановлення з'єднання відповідної інформації, або використовується фраза LEFT, RIGHT, для вказівки "сторони", з якої завжди обирається рядок, який не задовольняє умові з'єднання. Оператор (+) імітує на "стороні" його застосування створення одного або декількох рядків з невизначеним значенням (NULL), до яких можна приєднати одне чи декілька рядків з протилежної "сторони", що містять необхідну інформацію.

Синтаксис зовнішнього з'єднання:

1 форма

```
SELECT таблиця1.стовпець, таблиця2.стовпець, ...
FROM таблиця1, таблиця2
WHERE таблиця1.стовпець1 = таблиця2.стовпець2(+) ;
```

або у новому форматі стандарту мови SQL

```
SELECT таблиця1.стовпець, таблиця2.стовпець, ...
FROM таблиця1 LEFT [OUTER] JOIN таблиця2
ON таблиця1.стовпець1 = таблиця2.стовпець2 ;
```

## 2 форма

```
SELECT таблиця1.стовпець, таблиця2.стовпець, ...
FROM таблиця1, таблиця2
WHERE таблиця1.стовпець1 (+) = таблиця2.стовпець2 ;
```

або у новому форматі стандарту мови SQL

```
SELECT таблиця1.стовпець, таблиця2.стовпець, ...
FROM таблиця1 RIGHT [OUTER] JOIN таблиця2
ON таблиця1.стовпець1 = таблиця2.стовпець2 ;
```

## 3 форма

```
SELECT таблиця1.стовпець, таблиця2.стовпець, ...
FROM таблиця1 FULL [OUTER] JOIN таблиця2
ON таблиця1.стовпець1 = таблиця2.стовпець2 ;
```

де

(+) – символ зовнішнього з'єднання (у наборі це комбінація символів), де відсутні дані прямої відповідності.

Символ зовнішнього з'єднання вказується після імені таблиці, у якій немає відповідних до заданої умови з'єднання рядків. Символ зовнішнього з'єднання може використовуватися на будь-якій стороні виразу умови у фразі WHERE – на стороні, де недостатньо інформації, але не по обидва боки. Він завжди повертає рядки з таблиці, для якої в іншій таблиці немає відповідного рядка.

Умова, що передбачає зовнішнє з'єднання, не може включати оператор IN або бути пов'язаним з іншими умовами за допомогою оператора OR.

У новому форматі стандарту мови SQL дозволено використовувати у фразі FROM з обох боків ознаку зовнішнього з'єднання, у новому форматі запису це фраза FROM *таблиця1 FULL [OUTER] JOIN таблиця2*.

LEFT [OUTER] JOIN – вказує, що рядки будуть повертатися з таблиці, що знаходиться зліва від ключового слова JOIN. Якщо рядок, який повертається з лівої таблиці, не має відповідного рядка в правій таблиці, то він все одно вибирається. В цьому випадку в стовпцях для значень з правої таблиці будуть

встановлені значення NULL. Рекомендується всюди, де це можливо, використовувати лівобічні зовнішні з'єднання (LEFT OUTER), щоб не змішувати лівосторонні і правосторонні з'єднання. Взагалі ключове слово OUTER (зовнішній) можна опускати - цей тип використовується тільки для фраз LEFT, RIGHT і FULL JOIN і передбачається за замовчуванням.

RIGHT [OUTER] JOIN – схожа з LEFT OUTER, але рядки будуть повертатися з таблиці, що знаходиться праворуч від ключового слова JOIN.

FULL [OUTER] JOIN – вказує, що повертатися будуть всі рядки з обох таблиць незалежно від того, чи збігаються рядки в таблицях. Всім стовпцям, для яких немає відповідних значень в з'єднаній таблиці, присвоюються значення NULL.

Приклад виведення для кожного клієнта s\_customer його найменування, а також прізвища й ідентифікаційного номера торгового представника s\_emp. У список виводу повинні включатися найменування навіть тих клієнтів, які не мають торгового представника:

```
SELECT e.last_name, e.id, c.name
FROM s_emp e, s_customer c
WHERE e.id(+) = c.sales_rep_id ORDER BY e.id ;
```

або

```
SELECT e.last_name, e.id, c.name
FROM s_emp e RIGHT OUTER JOIN s_customer c
ON e.id = c.sales_rep_id ORDER BY e.id ;
```

або

```
SELECT e.last_name, e.id, c.name
FROM s_emp e, s_customer c
WHERE c.sales_rep_id = e.id(+) ORDER BY e.id ;
```

або

```
SELECT e.last_name, e.id, c.name
FROM s_customer c LEFT OUTER JOIN s_emp e
ON e.id = c.sales_rep_id ORDER BY e.id ;
```

2.4.6. Рекурсивне з'єднання таблиці із собою. У ряді випадків виникає необхідність одночасної обробки даних будь-якої таблиці та однієї або декількох її віртуальних копій, створюваних на час виконання запиту. Тимчасову копію таблиці можна сформувати, вказавши ім'я псевдоніма за ім'ям таблиці у фразі FROM.

За допомогою псевдонімів таблиці можна представити таблицю, як дві окремі таблиці. Це дозволяє з'єднувати рядки таблиці з її ж рядками. Для цього у таблиці повинен існувати зовнішній ключ, побудований на первинний або унікальний ключ цієї таблиці. А умова з'єднання повинна включати порівняння значень стовпців цих ключів.

Приклад субпідпорядкованого з'єднання. У цьому прикладі для імітації двох таблиць у фразі FROM використаний псевдонім тієї ж таблиці службовців s\_emp, в якій є стовець номера службовця (id) і стовець номера менеджера-керівника службовця (manager\_id). Умова з'єднання визначає виведення прізвищ службовця і його менеджера, якщо номер менеджера службовця збігається з номером службовця, який є менеджером:

```
SELECT worker.last_name || ' works for ' || manager.last_name
FROM s_emp worker, s_emp manager
WHERE worker.manager_id = manager.id ;
```

або

```
SELECT worker.last_name || ' works for ' || manager.last_name
FROM s_emp worker INNER JOIN s_emp manager
ON worker.manager_id = manager.id ;
```

## 2.5. Групова вибірка даних

Багато запитів до бази даних не потребують тій мірі деталізації, яку забезпечують раніше розглянуті типи багаторядкових SQL-запитів. Іноді потрібно дізнатися лише одне або кілька значень, які підсумовують інформацію, що міститься в базі даних, тобто збирають статистичну

інформацію.

У SQL існує ряд спеціальних стандартних функцій (агрегатних SQL-функцій групової обробки), кожна з яких оперує сукупністю значень стовпця деякої таблиці і створює єдине значення. Слід зазначити, що тут стовпець - це стовпець, який може містити дані не тільки з стовпця фізичної таблиці, але і дані, отримані шляхом функціонального перетворення і зв'язування операторами арифметичних операцій значень з одного або декількох стовпців.

При цьому вираз, що визначає стовпець такої таблиці, може бути як завгодно складним, але не повинен містити агрегатних SQL-функцій (вкладеність таких функцій не допускається). Однак з агрегатних SQL-функцій можна складати будь-які вирази.

На відміну від багаторядкових запитів групова вибірка припускає обробку за допомогою групової агрегатної функції підмножини рядків і видачі тільки одного статистичного рядка на відповідну підмножину (групу) рядків. Групові функції використовуються, як у списку фрази SELECT, так і в фразі HAVING при груповій вибірці даних.

За замовчуванням усі рядки таблиці розглядаються як одна спільна група. Для розбиття таблиці на менші групи рядків використовується фраза GROUP BY команди SELECT. Крім того, для відбору з утворених не всіх груп, що виводяться, використовується фраза HAVING.

Синтаксис групової вибірки:

SELECT *стовпець, групова\_функція*

FROM *таблиця*

[WHERE *умова*]

GROUP BY *вираз\_групи*

[HAVING *умова\_групи*]

[ORDER BY *стовпець*];

де

*умова* – задає вираз умови підмножини рядків джерела даних *таблиця*, з яких будуть утворюватися групи рядків;

*вираз\_групи* – задає *стовпець*, на основі значення якого утворюються групи рядків;

*умова\_групи* – відбирає і включає у кінцевий результат тільки ті групи, для яких виконана наведена умова;

*стовпець, групова\_функція* – атрибути списку статистичної інформації, яка утворюється в наслідок перетворення рядків відібраних груп.

2.5.1. Групові функції. Кожна групова функція допускає один аргумент. У табл.20 наведені можливі варіанти синтаксису таких функцій.

Таблиця 20

Функція	Опис
AVG(DISTINCT ALL n)	Середнє значення n без врахування невизначених значень
COUNT(DISTINCT ALL  <i>вираз</i>  *)	Кількість рядків тільки з визначеними результатами обчислення виразу. При аргументі "*" підраховуються всі рядки, включаючи повторювані і рядки з невизначеними значеннями
MAX(DISTINCT ALL  <i>вираз</i> )	Максимальне значення виразу
MIN(DISTINCT ALL  <i>вираз</i> )	Мінімальне значення виразу
STDDEV(DISTINCT ALL n)	Стандартне відхилення n без врахування невизначених значень
SUM(DISTINCT ALL n)	Сума значень n без врахування невизначених значень
VARIANCE(DISTINCT ALL n)	Дисперсія n без врахування невизначених значень

Якщо задано слово DISTINCT, функція враховує лише неповторювані значення; при наявності слова ALL розглядаються всі значення, включаючи повторювані. Варіант ALL приймається за замовчуванням, тому задавати його немає необхідності. Якщо у функції аргументом є вираз, то припустимими типами даних для нього є CHAR, VARCHAR2, NUMBER і DATE. Усі групові функції, окрім COUNT(\*), ігнорують невизначені значення. Для підстановки певного значення замість невизначеного у виразі використовується функція NVL(*вираз*, *значення*).



Фраза GROUP BY використовується для розбивки рядків таблиці на групи. Потім для одержання зведеної статистичної інформації для кожної групи можна використовувати групові функції. Якщо у фразу SELECT включено групову функцію з аргументом, то одержати результат за різними групами можна тільки у випадку, якщо у фразі GROUP BY заданий відповідний стовпець. Якщо список стовпців відсутній, видається повідомлення про помилку. Фраза WHERE дозволяє виключити деякі рядки до початку розбивки на групи. У фразі GROUP BY повинен бути заданий хоча б один стовпець. Не можна використовувати позиційні позначення або псевдоніми стовпців.

Фраза HAVING грає таку ж роль для груп, що і фраза WHERE для рядків. Вона використовується для виключення груп точно так же, як WHERE використовується для виключення рядків. Ця фраза включається в команду лише при наявності фрази GROUP BY, а вираз в HAVING повинне приймати єдине значення для групи.

За замовчуванням рядки сортуються в порядку зростання у відповідності зі списком фрази GROUP BY. Змінити порядок виведення груп можна за допомогою фрази ORDER BY.

Приклад виведення всіх можливих кредитних рейтингів і кількості клієнтів у кожній категорії. Стовпець повинен мати заголовок "# Cust":

```
SELECT credit_rating, COUNT(*) "# Cust"  
FROM s_customer GROUP BY credit_rating ;
```

Приклад виведення всіх посад, крім віце-президента 'VP', і відповідної загальної заробітної плати за місяць. Список сортується за загальною заробітною платні:

```
SELECT title, SUM(salary) AS PAYROLL FROM s_emp  
WHERE title NOT LIKE 'VP%'  
GROUP BY title  
ORDER BY SUM(salary) ;
```

2.5.2.Недійсні запити з груповими функціями. Якщо в одній і тій же команді SELECT зазначені як окремі елементи даних (region\_id), так і групові функції (COUNT), то потрібна фраза GROUP BY, що описує ці окремі елементи (у даному випадку - region\_id). Якщо фраза GROUP BY опущена, видається повідомлення про помилку "not a single-group group function" ("це не одногрупова функція").

Приклад недійсного запиту:

```
SELECT region_id, COUNT(name) FROM s_dept;
```

Для виправлення цієї помилки варто додати фразу GROUP BY, щоб region\_id було ознакою групи:

```
SELECT region_id, COUNT(name) FROM s_dept GROUP BY region_id ;
```

Любий стовпець у списку фрази SELECT, який не є аргументом групової функції, повинен бути поданий у фразі GROUP BY.

Фраза WHERE для виключення груп не використовується. Замість цього для обмеження кількості груп варто використовувати фразу HAVING.

Приклад обмеження груп:

```
SELECT dept_id, AVG(salary) FROM s_emp  
GROUP BY dept_id HAVING AVG(salary)>2000 ;
```

2.5.3.Групи всередині груп. Зведені результати по групах і підгрупах можна одержувати шляхом вказівки більш, ніж одного стовпця в фразі GROUP BY. Порядок сортування, використовуваний за замовчуванням, визначається порядком стовпців у фразі GROUP BY.

Приклад виведення кількості службовців по посадах усередині відділів:

```
SELECT dept_id, title, COUNT(*) FROM s_emp GROUP BY dept_id, title ;
```

Приклад виведення кількості службовців по відділах для кожної посади.

```
SELECT title, dept_id, COUNT(*) FROM s_emp GROUP BY title, dept_id ;
```

Фраза HAVING задає умову відбору груп для виведення. Отже, на групи

накладається подальше обмеження, засноване на вже зведеній інформації. Якщо використовується фраза HAVING, СУБД виконує наступні дії:

- групує рядки;
- застосовує групову функцію;
- робить виведення тих груп, які задовольняють умові фрази HAVING.

Утворення груп і обчислення групових функцій відбуваються до того, як до груп зі списку SELECT застосовується обмеження, задане в фразі HAVING.

Приклад виведення посади і загальної заробітної плати для всіх посад із заробітною платою більш 5000 на місяць, крім віце-президентів. Вихідні рядки сортуються по заробітній платі:

```
SELECT title, SUM(salary) AS zarplata FROM s_emp
WHERE title NOT LIKE 'VP%'
GROUP BY title HAVING SUM(salary)>5000
ORDER BY zarplata ;
```

## 2.6. Вибірка з вкладеним підзапитом

Вкладений підзапит - це команда запиту SELECT, укладена в круглі дужки і вбудована в фразу SELECT, WHERE, HAVING іншого запиту SELECT чи інших фраз, які використовують фразу WHERE. Вкладений підзапит може містити в своїй фразі SELECT, WHERE, HAVING інший вкладений підзапит. Вкладений підзапит створюється для того, щоб при відборі рядків таблиці, сформованої головним (основним) запитом, можна було використовувати дані з інших таблиць.

Підзапити дуже корисні при написанні загального запиту на вибірку значень за невідомим значенням умови на момент його формування. За допомогою вкладеного в головний запит підзапита можна знаходити невідомі значення.

Синтаксис вибірки із вкладеним підзапитом:

```
SELECT список_стовпців_головного_запиту
FROM таблиця [, ( SELECT список_стовпців_підзапиту
```

FROM *таблиця\_підзапиту* WHERE *умова\_підзапиту* )  
 WHERE *вираз\_оператор* ( SELECT *список\_стовпців\_підзапиту*  
 FROM *таблиця\_підзапиту* [WHERE *умова\_підзапиту*] ) ;

де

*оператор* – оператор порівняння значення *виразу* із значеннями, які повертає підзапит.

Існують прості і корельовані вкладені підзапити.

Прості вкладені підзапити використовуються для подання множини значень, дослідження яких має здійснюватися в будь-якому операторі порівняння головного запиту, тобто їх результат не пов'язаний з множиною значень головного запиту. Прості підзапити включаються у фразу WHERE, HAVING головного запиту за допомогою операторів порівняння - однорядкових (<, >, =, <>, >=, <=) і багаторядкових (IN, NOT IN).

Прості вкладені підзапити обробляються системою "знизу доверху". Першим обробляється вкладений підзапит самого нижнього рівня. Множина значень, отриманих в результаті його виконання, використовується при реалізації підзапиту більш високого рівня і т.д. до головного запиту. Тобто, в кінці виконання результат підзапитів використовується головним запитом.

Корельовані вкладені підзапити використовуються для подання множини значень, результат утворення яких залежить від значень, отриманих із зовні від іншого підзапиту (головного запиту). Обробка корельованого підзапиту, повинна повторюватися для кожного значення, що надходить від зовнішнього підзапиту, а не виконуватися тільки один раз. Корельовані вкладені підзапити включаються у фразу WHERE, HAVING головного запиту за допомогою багаторядкових операторів порівняння (IN, NOT IN) та оператора-квантора існування EXISTS або у фразу SELECT головного запиту.

Запити з корельованими вкладеними підзапитах обробляються системою в зворотному порядку "зверху до низу". Спочатку вибирається перший рядок, сформований головним запитом, і з нього вибираються значення тих стовпців, які використовуються у вкладеному підзапиті. Якщо ці значення задовольняють

умовам вкладеного підзапиту, то обраний рядок головного запиту включається в результат і т.д.

У підзапиті не можна використовувати фразу ORDER BY. На кожен запит дозволяється тільки одна фраза ORDER BY, і якщо вона використовується, то повинна бути останньою у головному запиті SELECT.

2.6.1.Однорядкові підзапити. Однорядковий підзапит повертає з вкладеної команди SELECT тільки один рядок і за способом виконання є простим. У підзапитах цього типу використовується однорядковий оператор порівняння.

Приклад виведення прізвищ співробітників, що мають таку ж посаду як і Петров:

```
SELECT last_name, title FROM s_emp
WHERE title = (SELECT title FROM s_emp WHERE last_name='Петров');
```

Можна виводити дані з головного запиту, використовуючи групову функцію в підзапиті для повернення одного рядка. Підзапит укладається в дужки і ставиться після оператора порівняння.

Приклад виведення прізвища, посади і зарплати всіх співробітників з оплатою нижче середньої:

```
SELECT last_name, title, salary FROM s_emp
WHERE salary < (SELECT AVG(salary) FROM s_emp);
```

Одна з розповсюджених помилок – повернення однорядковим підзапитом більш, ніж одного рядка. У цьому випадку СУБД видає повідомлення “single-row subquery returns more than one row”. Для виправлення такої помилки слід однорядковий оператор порівняння замінити на багаторядковий IN.

2.6.2.Багаторядкові підзапити. Підзапити, що повертають більш одного рядка, називають багаторядковими. У них варто використовувати багаторядковий оператор порівняння IN або NOT IN. Ці оператори очікують

одного або декількох значень. Якщо виконання багаторядкового підзапиту залежить від значення стовпця з головного запиту, то такий підзапит є кореляційним, інакше його слід віднести до простого.

Приклад простого багаторядкового підзапиту виведення назви і статусу постачальників `supplier`, які здійснюють поставки `delivery` продукту з номером 11:

```
SELECT supplier_name, supplier_status FROM supplier
WHERE supplier_id IN
(SELECT supplier_id FROM delivery WHERE product_id = 11);
```

або

```
SELECT s.supplier_name, s.supplier_status FROM supplier s
JOIN (SELECT supplier_id FROM delivery WHERE product_id = 11) d
USING (supplier_id) ;
```

Приклад виведення списку службовців, приписаних до відділу 10 або регіону 2:

```
SELECT last_name, first_name, title FROM s_emp
WHERE dept_id IN (SELECT id FROM s_dept
WHERE id=10 OR region_id=2) ;
```

або

```
SELECT e.last_name, e.first_name, e.title FROM s_emp e,
(SELECT id FROM s_dept WHERE id=10 OR region_id=2) d
WHERE e.dept_id=d.id ;
```

Підзапити можна використовувати не тільки в фразі `WHERE` головного запиту, але й у фразі `HAVING` та `FROM`. СУБД виконує підзапрос першим і повертає результат у фразу `HAVING` головного запиту.

2.6.3.Корельовані підзапити. Корельовані підзапити можуть повертати один і більш рядків. Тому у них використовується багаторядковий оператор

порівняння IN або NOT IN, а також оператор-квантора існування EXISTS або у фразі SELECT головного запиту предикат-квантор оператор порівняння (=). Тобто такі підзапити очікують одного або декількох значень. Крім того, виконання багаторядкового підзапиту залежить від значення стовпців з головного запиту. Тому доцільно у структурі головного запиту і під запиту використовувати псевдоніми для таблиць і підзапитів, які подаються як віртуальні таблиці.

Приклад кореляційного підзапиту у фразі SELECT для отримання впорядкованого списку службовців і назв підрозділів, де вони працюють:

```
SELECT e.last_name, e.first_name, e.title,
       (SELECT d.name FROM s_dept d WHERE d.id = e.dept_id) AS dept_name
FROM s_emp e ORDER BY dept_name, e.last_name ;
```

Приклад кореляційного під запиту у фразі WHERE виведення назви і статусу постачальників supplier, які здійснюють поставки delivery продукту з номером 11:

```
SELECT s.supplier_name, s.supplier_status FROM supplier s
WHERE 11 IN
       (SELECT d.product_id FROM delivery d
        WHERE d.supplier_id = s. supplier_id) ;
```

Кореляційний підзапит з квантором існування EXISTS вважається дійсним тільки тоді, коли результат обчислення виразу підзапиту (SELECT \* FROM ... WHERE ...) є непустою множиною, тобто коли існує будь-який рядок в таблиці, зазначеної у фразі FROM підзапиту, який задовольняє умові WHERE підзапиту. Практично цей підзапит завжди буде корельованою множиною.

Приклад кореляційного підзапиту з квантором існування для виведення назви і статусу постачальників supplier, які здійснюють поставки delivery продукту з номером 11:

```
SELECT s.supplier_name, s.supplier_status
```

```

FROM supplier s
WHERE EXISTS(SELECT d.product_id FROM delivery d
WHERE d.supplier_id = s. supplier_id AND d.product_id = 11) ;

```

Хоча цей приклад тільки показує інший спосіб формулювання запиту для завдання, розв'язуваної і іншими шляхами (за допомогою оператора IN або з'єднання), EXISTS являє собою одну з найбільш важливих можливостей SQL. Фактично будь-який запит, який виражається через IN, може бути альтернативним способом сформульовано також за допомогою EXISTS. Однак зворотне не є вірним.

Приклад кореляційного підзапиту з квантором існування для виведення назви і статусу постачальників supplier, які не здійснюють поставки delivery продукту з номером 11:

```

SELECT s.supplier_name, s.supplier_status
FROM supplier s
WHERE NOT EXISTS(SELECT d.product_id FROM delivery d
WHERE d.supplier_id = s. supplier_id AND d.product_id = 11) ;

```

2.6.4. Функції в підзапиті. Природно, що функції в підзапиті найчастіше використовуються в кореляційних підзапитах.

Приклад виведення постачальників supplier продуктів з номером 11, які здійснюють поставки delivery цих продуктів за мінімальну ціну:

```

SELECT s.supplier_name, s.supplier_status, d.price
FROM supplier s JOIN delivery d
ON d.supplier_id = s. supplier_id
WHERE d.price = ( SELECT MIN(x.price) FROM delivery x
WHERE x.product_id = d.product_id ) ;

```

## 2.7. Ієрархічна вибірка

Якщо таблиця містить ієрархічні (підпорядковані) дані у випадку рекурсивного зв'язку, то можна вибирати рядки за ієрархічним порядком їх



розташування або підпорядкування.

Синтаксис ієрархічної вибірки:

```
SELECT список_стовпців FROM таблиця
[WHERE умова]
[START WITH умова_кореневих_вершин]
CONNECT BY умова_зв'язку_вершин ;
```

Фраза START WITH визначає умову формування кореневої вершини дерева ієрархічної вибірки рядків. Условие в этой фразе не должно содержать подзапросов. Якщо фраза START WITH не задана, то всі рядки будуть кореневими.

Фраза CONNECT BY – умову вибірки (підпорядкування) всіх наступних вершин, тобто встановлюється зв'язок між “батьківськими” рядками і “дочірніми” в утвореній ієрархії. Для того, щоб в умові зв'язку визначити значення виразу, який відноситься до “батьківського” рядка, використовують оператор PRIOR. Умова в цій фразі не повинна містити підзапитів.

Умова фрази WHERE обмежує всю вибірку, до якої застосовуються умови утворення ієрархії.

У цих запитах не можна використовувати фрази ORDER BY і GROUP BY. Для сортування однорангових вузлів у дереві ієрархічної вибірки рядків можна використовувати фразу ORDER SIBLINGS BY.

Приклад (прямої вибірки ієрархічних даних) виведення керівників та підпорядкованих службовців с таблиці s\_emp(id primary key,last\_name,manager\_id references s\_emp(id)):

```
SELECT id AS worker, last_name, manager_id AS manager FROM s_emp
START WITH manager_id is NULL
CONNECT BY manager_id = PRIOR id ;
```

Приклад (зворотної вибірки ієрархічних даних) виведення службовців з прізвищем ‘Петров’ і їхніх керівників:

```
SELECT id AS worker, last_name, manager_id AS manager FROM s_emp  
START WITH last_name = 'Петров'  
CONNECT BY PRIOR manager_id = id ;
```

3. Порядок роботи з інструментальними засобами взаємодії із сервером бази даних

Будь-які інструментальні засоби і прикладні додатки здійснюють взаємодію із сервером бази даних через мову команд запитів SQL. До складу програмного забезпечення сервера бази даних Oracle входить спеціалізований консольний додаток Oracle SQL\*Plus та програмне середовище взаємодії і адміністрування сервера бази даних Oracle SQL Developer.

**Oracle SQL\*Plus** – це інтерпретатор команд із мови SQL для виконання дій по створенню інформаційних об'єктів бази даних, їхньої модифікації і наповнення даними.

**Oracle SQL Developer** – це графічний інструмент для розробки і адміністрування баз даних. За допомогою SQL Developer можна переглядати об'єкти бази даних, запускати SQL-команди, редагувати і налагоджувати PL/SQL-програми. Ви також можете запустити будь-яку кількість наданих звітів, а також створювати і зберігати власні. SQL Developer підвищує продуктивність і спрощує підтримку вашої бази даних при виконанні завдань з її розвитку. SQL Developer може підключатися до будь-якої СУБД Oracle від версії 9.2.0.1 і може працювати на Windows, Linux, Mac OSX.

Порядок взаємодії з інструментальними засобами взаємозв'язку із сервером Oracle:

1. Запуск і реєстрація користувача в базі даних. Необхідно обрати попередньо створене підключення до бази (наприклад, base), ввести ім'я користувача - ідентифікатор студента в групі (наприклад, ik3101), пароль.

2. Введення команди і запуск її на виконання здійснюється в робочому листі вікна інструментального засобу. Команда може розташовуватися на декількох рядках. Але повинна завершуватися символом «;» на поточному

рядку або «/» на наступній рядку.

3. Виконання командного файлу (попередньо збереженого на локальному диску) здійснюється вказівкою символу «@» і наведенням повної специфікації файлу.

4. Вихід з інструментального засобу здійснюється закриттям його вікна.

Oracle SQL Developer це програмний засіб для взаємодії з сервером бази даних у вигляді GUI-додатків. Тому він підтримує роботу, як у командному рядку робочого листа додатку, так і у меню-орієнтованому інтерфейсі з автоматичною генерацією виконуваних SQL-команд. Робота в такому засобі здійснюється в інтерактивному режимі за допомогою вікон.

Основним вікном у Oracle SQL Developer є навігаційне вікно з деревом доступних об'єктів бази даних. Правою кнопкою миші видаються дозволені операції модифікації обраного об'єкта бази даних.

З основного меню додатку викликається SQL Worksheet - робочий лист редактора команд SQL. Виклик редактора рядків команд здійснюється для введення як одиничних команд SQL без завершального символу «;», так і їх послідовності, які розділені завершальним символом виконання команди «;». Запуск команд на виконання здійснюється від поточного положення маркера до кінця послідовності команд кнопкою F5 або тільки поточної команди за допомогою кнопки F9 або Ctrl-Enter.

Контроль виконання команд виконується переглядом пунктів дерева у навігаційному вікні. Можна зберегти введену послідовність команд у вигляді файлу з типом sql за допомогою пункту головного меню «File / Save».

#### 4. Контрольне завдання

1. Виконати виведення даних з таблиці EMPLOYEE згідно наступного порядку:

1.1) підготуйте таблицю EMPLOYEE до виведення даних, виконавши наступні дії:

- скасувати дію перевірконого обмеження цілісності стовпця

## COMISSION;

- створити послідовність EMPL\_SEQ з параметрами початку номерів з 1, максимальне число – 9999999, кешування 5 чисел, не циклічна послідовність;

- ввести список з 10 нових студентів, що навчаються на різних факультетах 10, 37, 54, 75, заповнивши всі стовпці таблиці визначеними даними - заробіток в діапазоні 10–300, дата зарахування – з 2000 по 2003 р., та застосувавши послідовність для створення номерів;

1.2) напишіть запит для виведення прізвищ (стовпець LAST\_NAME) і заробітної плати (COMISSION) тих студентів, заробіток яких не знаходиться в інтервалі між 100 і 250 грн.;

1.3) отримайте список прізвищ, імен та заробітної плати студентів з факультетів за номерами 10, 37, 54. У списку об'єднайте ім'я з прізвищем і позначте його як стовпець “ПІБ”, заробітну плату – як стовпець “Зарплата” і відсортуйте список по прізвищах за абеткою;

1.4) отримайте список прізвищ і дат зарахування (START\_DATE) тих студентів, прийнятих в 2001 році, і прізвища яких починаються з букви “А”.

2. Виконати виведення даних з таблиць REGION, DEPARTMENT, EMPLOYEE згідно наступного порядку:

2.1) підготуйте таблиці DEPARTMENT і REGION до вибірки даних за умовою з'єднання таблиць, додавши необхідні відомості:

- додати регіон 50 – Крим, 51 – Одеса;  
- встановити для факультетів 10, 37 код регіону 44, для факультету 54 – код регіону 50, а для факультету 75 – NULL-значення коду регіону.

2.2) Підготуйте таблицю DEPARTMENT до вибірки підпорядкованих даних, зробивши наступні зміни:

- додати підрозділи з номером 1, назвою “НТУУ-КПІ” з кодом регіону 44, з номером 2, назвою “КНУ” з кодом регіону 44,  
- підпорядкувати всі наявні факультети до підрозділу з номером 1,  
- додати підрозділи з номер 101 назвою “кафедра ТК” з кодом регіону 44, з номер 102 назвою “кафедра ОТ” з кодом регіону 44, які підпорядковані

факультету з номером 10.

2.3) напишіть запит, що містить дані про ім'я, прізвище, номер і назву факультету усіх студентів. Відсортуйте список по факультетах і прізвищах за абеткою;

2.4) напишіть запит, що містить дані про прізвище, заробіток, назви факультету і регіону тих студентів, які навчаються в регіоні на букву "К". Відсортуйте список за прізвищем в зворотному порядку;

2.5) напишіть запити, що містять дані назв факультету і регіону, які нададуть інформацію:

- про усі факультети навчання і відповідні їм регіони,
- про усі регіони і відповідні їм факультети,
- про тільки ті факультети, для яких встановлені регіони.

Знайдіть відмінності в отриманих результатах;

2.6) виведіть назви тих підрозділів, до складу яких входить або підпорядкований підрозділ з номером 101.

2.7) виведіть назви всіх підрозділів за ієрархічним порядком підпорядкування.

3. Виконати формування звітних (зведених) даних з таблиць REGION, DEPARTMENT, EMPLOYEE згідно наступного порядку:

3.1) підрахуйте кількість факультетів у регіонах і подайте результат у виді списку з коду регіону і кількості факультетів. Надайте відповідь, чи встановлена кількість у всіх регіонах;

3.2) визначте мінімальну і максимальну заробітну плату по кожному факультеті. Подайте результат у виді списку з найменування факультету, мінімальної і максимальної зарплат для всіх факультетів;

3.3) визначте розмір мінімальної зарплати студентів на тих факультетах, де студенти зараховані після 2001 року з мінімальною заробітною платою по факультеті серед цих студентів менше 100 грн. У список включити код факультету і розмір мінімальної зарплати, а результат відсортувати за розміром мінімальної зарплати;

3.4) визначте розмір мінімальної зарплати студентів на тих факультетах, де розмір мінімальної заробітної плати по факультеті складає менше 100 грн. і є студенти, які зараховані після 2001 року. У список включити код факультету і розмір мінімальної зарплати, а результат відсортувати за розміром мінімальної зарплати.

Усі команди контрольного завдання подати у виді єдиного командного файлу, який представити у звіті з необхідними поясненнями.

## Комп'ютерний практикум № 5.

# УПРАВЛІННЯ ДОСТУПОМ ДО БАЗИ ДАНИХ ІНФОРМАЦІЙНОЇ СИСТЕМИ КОМАНДАМИ МОВИ УПРАВЛІННЯ ДАНИМИ

### 1. Мета та основні завдання практикуму

Мета роботи: ознайомитися з формами команд управління доступом до бази даних і вивчити систему безпеки сервера бази даних

Порядок виконання роботи:

- вивчити форми команд управління доступом;
- визначити основні поняття моделі безпеки сервера бази даних;
- згідно умов контрольного завдання здійснити керування доступом до об'єктів власної схеми і дослідити можливості надання привілеїв доступу різного рівня до даних новим користувачем, створеним в системі управління базою даних;
- підготувати звіт з результатами виконання завдання.

### 2. Основні теоретичні відомості

#### 2.1. Основні означення моделі безпеки сервера бази даних

У багатокористувацькому середовищі однією із функцій сервера бази даних є необхідність забезпечення безпеки збереження і надання доступу до даних. Тому, модель безпеки СУБД повинна дозволяти та забезпечувати:

- керування доступом до бази даних;
- надавання доступу до визначених об'єктів бази даних;
- контролювання виданих й отриманих привілеїв у словнику бази даних;
- створювання синонімів для об'єктів бази даних.

Отже, модель безпеки бази даних може скластися з двох частин:

- безпеки системи;
- безпеки даних.

*Безпека системи* охоплює доступ до бази даних і користування базою даних на системному рівні, тобто контроль за ім'ям і паролем користувача,

дисковим простором, наданим користувачу, за виконуваними системними операціями, що дозволені користувачу.

*Безпека даних* включає доступ до об'єктів бази даних, їхнє використання і дії, що може виконувати користувач з цими об'єктами.

*Привілеї* – це права на виконання визначених команд SQL у базі даних. Найвищим рівнем користувача у СУБД є адміністратор бази даних. Він має право встановлювати іншим користувачам доступ до бази даних і призначати привілеї їх доступу до її об'єктів. Таким чином, користувачам для роботи в базі даних необхідні системні привілеї для одержання доступу до бази даних і привілеї на об'єкти для маніпулювання їх вмістом у базі даних. Користувачам може бути також надано право на передачу отриманих привілеїв іншим користувачам або ролям, що слугують іменованими групами пов'язаних привілеїв.

## 2.2. Системні привілеї

Існує багато видів привілеїв, що надаються користувачам або ролям. Системні привілеї звичайно надаються адміністратором бази даних.

До загальних системних привілеїв, що має в своєму розпорядженні і може передавати іншим користувачам адміністратор бази даних, віднесені привілеї, подані у табл.21.

Таблиця 21

Системний привілеї	Дозволені операції
CREATE USER	Дозволяє реєструвати інших користувачів (необхідна для ролі адміністратора бази даних)
ALTER USER	Дозволяє змінювати пароль користувача
DROP USER	Дозволяє видаляти інших користувачів
DROP ANY TABLE	Дозволяє видаляти таблиці з будь-якої схеми
BACKUP ANY TABLE	Дозволяє копіювати будь-яку таблицю в будь-якій схемі за допомогою утиліти експорту
UNLIMITED TABLESPACE	Дозволяє надати необмежений табличний простір для розміщення об'єктів бази даних

### 2.2.1. Створення користувача. Адміністратор бази даних реєструє нового



користувача бази даних і надає йому ряд системних привілеїв. Від цих привілеїв залежить, що може робити даний користувач на рівні системи бази даних. Для реєстрації користувача адміністратор бази даних використовує команду CREATE USER.

Скорочений синтаксис команди створення користувача:

```
CREATE USER користувач IDENTIFIED BY пароль
[DEFAULT TABLESPACE табличний_простір]
[TEMPORARY TABLESPACE тимчасовий_простір]
[QUOTA розмір ON табличний_простір]
[PROFILE DEFAULT] ;
```

де

*користувач* – ім'я користувача бази даних,

*пароль* – пароль, з яким користувач повинний входити в систему,

*табличний\_простір* – іменованний простір для розміщення об'єктів схеми нового користувача (наприклад, USERS),

*розмір* – значення зовнішньої пам'яті, що надається в зазначеному табличному просторі при розміщенні об'єктів користувача. Задається в кілобайтах (наприклад, 10K) або мегабайтах (10M),

*тимчасовий\_простір* – іменованний простір для тимчасового розміщення даних (наприклад, TEMP).

2.2.2.Надання системних привілеїв користувачу. Після виконання команди створення користувача новий користувач ніяких привілеїв не має. Зареєструвавши користувача, адміністратор бази даних повинний обов'язково надати йому певні системні привілеї.

Основними системними привілеями користувача є привілеї, наведені у табл.22.

Таблиця 22

Системний привілей	Авторизовані операції
CREATE SESSION	Дозволяє почати сеанс роботи з базою даних
CREATE TABLE	Дозволяє створювати таблиці в схемі користувача
CREATE SEQUENCE	Дозволяє створювати послідовності в схемі користувача
CREATE VIEW	Дозволяє створювати представлення в схемі користувача
CREATE PROCEDURE	Дозволяє створювати програмні процедури або функції в схемі користувача
CREATE SYNONYM	Дозволяє створювати синоніми інформаційних об'єктів в схемі користувача

Адміністратор бази даних надає системні привілеї користувачам за допомогою команди GRANT. Користувач може користатися наданими привілеями відразу після їхнього одержання.

Скорочений синтаксис команди надання системних привілеїв:

GRANT *привілей* [, *привілей...*] TO *користувач* [, *користувач ...*]

[WITH ADMIN OPTION] ;

де

*привілей* – наданий системний привілей,

*користувач* – ім'я користувача,

WITH ADMIN OPTION – дозволяє користувачу, що одержав системну привілею, передавати її іншим користувачам або ролям.

2.2.3. Створення ролі. Роль – це іменована група взаємозв'язаних привілеїв, що можуть бути надані одночасно користувачу або іншій ролі. Цей метод спрощує процес надання і скасування привілеїв.

Користувач може мати доступ до декількох ролей, а одна й та сама роль може бути надана багатьом користувачам. Звичайно ролі створюються для забезпечення роботи різних за функціональністю інформаційних систем із базою даних.

Процедура створення і надання ролі передбачає виконання таких дій:

- створення ролі адміністратором бази даних;

- визначення набору привілеїв для цієї ролі;
- надання ролі користувачам.

Синтаксис команди створення ролі:

```
CREATE ROLE роль ;
```

де

*роль* – ім'я створюваної ролі.

Коли роль створена, адміністратор (або інший користувач, кому надані такі привілеї) бази даних може використовувати команду GRANT для визначення набору привілеїв цієї ролі, а потім надання ролі користувачам.

Синтаксис команди визначення набору привілеїв ролі:

```
GRANT привілей [, привілей...] TO роль [, роль ...] ;
```

Синтаксис команди присвоєння ролі користувачам:

```
GRANT роль [, роль ...] TO користувач [, користувач ...] ;
```

Приклад надання двом користувачам привілеїв ролі менеджера на створення таблиці і представлення:

```
CREATE ROLE manager ;
```

```
GRANT create table, create view TO manager ;
```

```
GRANT manager TO user1, user2 ;
```

При створенні бази даних системою управління підготовлена до використання системна роль CONNECT, що забезпечує користувачів привілеями підключення до бази даних. Надання цієї ролі здійснюється командою:

```
GRANT CONNECT TO користувач ;
```

Системна роль CONNECT за своєю дією еквівалентна системній привілеї CREATE SESSION. Надання привілеїв на створення основних інформаційних об'єктів у власній схемі відбувається за допомогою інших ролей (наприклад, RESOURCE) або безпосередньо командами присвоєння системних привілеїв користувачеві.

2.2.4.Зміна пароля і параметрів схеми користувача. Кожен користувач має пароль, наданий йому адміністратором бази даних при реєстрації. Змінити свій пароль можна за допомогою команди ALTER USER.

Синтаксис команди зміни пароля:

ALTER USER *користувач* IDENTIFIED BY *пароль* ;

де

*користувач* – ім'я користувача;

*пароль* – новий пароль.

Крім зміни свого пароля ця команда надає й інші можливості, але для їхнього використання потрібно мати привілею ALTER USER, надану адміністратором бази даних.

Зміна наданої квоти для розміщення об'єктів схеми нового користувача виконується за допомогою наступної форми цієї команди:

ALTER USER *користувач* QUOTA *розмір* ON *табличний\_простір* ;

де

*розмір* – значення зовнішньої пам'яті, що надається в зазначеному табличному просторі при розміщенні об'єктів користувача. Задається в кілобайтах (наприклад, 10К) або мегабайтах (наприклад, 10М).

Проте, щоб надати користувачу необмежені права на використання всього табличного простору необхідно призначити йому системний привілей UNLIMITED TABLESPACE за допомогою наступної команди:

GRANT UNLIMITED TABLESPACE TO *користувач* ;

2.2.5.Видалення користувача. Невикористовуванні (або які стали недійсними) в базі даних користувачі можуть бути видалені із СУБД за допомогою команди DROP USER.

Синтаксис команди видалення користувача:

DROP USER *користувач* [CASCADE] ;

СУБД не виконує видалення користувача, якщо його схема містить інформаційні об'єкти – таблиці, представлення, індекси і т.д. Необхідно

попередньо видалити ці об'єкти зі схеми користувача або в команді видалення вказати фразу CASCADE. У цьому випадку спочатку виконується видалення всіх об'єктів зі схеми користувача, а потім видалення подання самого користувача із системи.

### 2.3. Привілеї на об'єкти бази даних

Адміністратор бази даних може дозволити користувачу виконувати конкретні дії над окремими таблицями, представленнями, послідовностями або програмними процедурами/функціями, надаючи йому визначені привілеї на ці об'єкти. Для різних типів об'єктів ці привілеї різні. Так, над даними таблиць можна виконувати дії з маніпулювання, над представлення – формувати запити на виведення даних, а над програмними процедурами/функціями – виконувати їх.

Крім того, власник об'єкта – користувач, до схеми якого належить або у схемі, якого створено об'єкт, має на нього всі види привілеїв. Тобто, по відношенню до об'єктів власної схеми, користувач виступає у правах адміністратора об'єктів бази даних.

Щоб надати доступ до своїх об'єктів іншим користувачам, необхідно використовувати об'єктну форму команди GRANT.

Синтаксис команди надання об'єктних привілеїв:

GRANT {*привілей* [, *привілей* ...] | ALL} [(стовпці)]

ON *об'єкт*

TO {*користувач* [, *користувач* ...] | *роль* | PUBLIC}

[WITH GRANT OPTION] ;

де

*привілей* – наданий привілей на об'єкт,

ALL – усі види привілеїв на об'єкт,

*стовпці* – стовпець таблиці або представлення, на який надаються привілеї,

*об'єкт* – об'єкт, на який даються привілеї,

фраза TO – вказує суб'єкт надання привілеїв – користувач, роль або PUBLIC – привілеї на об'єкт надаються всім користувачам бази,

WITH GRANT OPTION – дозволяє утримувачеві привілеїв на об'єкт, передавати їх іншим користувачам або ролям.

У табл.23 наведені можливі види привілеїв на об'єкти бази даних.

Таблиця 23

Об'єктні привілеї	Таблиця	Представлення	Послідовність	Процедура/ Функція
ALTER	+		+	
DELETE	+	+		
EXECUTE				+
INDEX	+			
INSERT	+	+		
REFERENCES	+			
SELECT	+	+	+	
UPDATE	+	+		

Привілеї INDEX і REFERENCES ролям не надаються.

Щоб надавати іншим користувачам об'єктні привілеї, користувач, що надає ці привілеї, повинен мати об'єкт у власній схемі або одержати ці привілеї з правом передачі - WITH GRANT OPTION. Власник об'єкта може надавати будь-які привілеї на свій об'єкт будь-якому користувачеві або ролі. Власник об'єкта автоматично одержує всі привілеї на свої об'єкти під час їх створення.

Приклад надання користувачам Sue і Rich привілеї на вибірку даних з таблиці s\_emp:

```
GRANT select ON s_emp TO sue, rich ;
```

Приклад надання привілеї оновлення на визначені стовпці в таблиці s\_dept користувачу scott і ролі managers:

```
GRANT update (name, region_id) ON s_dept TO scott, managers ;
```

Користувач, що одержав привілеї за допомогою опції WITH GRANT OPTION, може надавати їх іншим користувачам. Об'єктний привілеї, наданий

іншому користувачу на підставі дії опції WITH GRANT OPTION, скасовується тоді, коли скасовується дія опції або самий привілей разом з опцією для того користувача, хто його отримував.

Приклад надання користувачем alice користувачу scott доступу до своєї таблиці з привілеями на вибірку даних і додавання рядків із правом передавання цих привілеїв іншим користувачам.

```
GRANT select, insert ON s_dept TO scott WITH GRANT OPTION ;
```

За допомогою ключового слова PUBLIC власник таблиці може надати доступ до неї всім користувачам.

Приклад надання користувачем scott всім користувачам права на вибірку даних з таблиці s\_dept користувача alice:

```
GRANT select ON slice.s_dept TO PUBLIC ;
```

#### 2.4. Перегляд наданих привілеїв

Надані ролі або користувачу привілеї можна переглянути в словнику даних через визначені у табл.24 представлення.

Таблиця 24

Таблиці словника даних	Опис
ROLE_SYS_PRIVS	Системні привілеї, надані ролям
ROLE_TAB_PRIVS	Об'єктні привілеї на таблиці, надані ролям
USER_ROLE_PRIVS	Ролі, доступні користувачу
USER_TAB_PRIVS	Надані привілеї на власні і чужі об'єкти
USER_TAB_PRIVS_MADE	Надані користувачем привілеї на власні об'єкти
USER_TAB_PRIVS_RECD	Надані користувачу привілеї на чужі об'єкти
USER_COL_PRIVS_MADE	Надані користувачем привілеї на стовпці власних об'єктів
USER_COL_PRIVS_RECD	Надані користувачу привілеї на стовпці чужих об'єктів
USER_USERS	Інформація про поточного користувача
ALL_USERS	Інформація про всіх користувачів бази

#### 2.5. Скасування привілеїв

Скасування привілеїв або ролей, наданих іншим користувачам або ролям,

здійснюється за допомогою команди REVOKE. Команда REVOKE скасовує зазначені привілеї чи ролі для всіх користувачів або ролей, яким раніше були надані ці привілеї чи ролі.

Синтаксис команди скасування системних привілеїв:

REVOKE {*привілей* [, *привілей* ...] | ALL PRIVILEGES}

FROM {*користувач* [, *користувач* ...] | *роль* / PUBLIC}

де

*привілей* – системний привілей або надана роль,

ALL PRIVILEGES – усі можливі системні привілеї,

*користувач* – користувач, у якого скасовують привілеї,

PUBLIC – видаляє зазначені привілеї або ролі у всіх користувачів.

Синтаксис команди скасування об'єктних привілеїв:

REVOKE {*привілей* [, *привілей* ...] | ALL PRIVILEGES} ON *об'єкт*

FROM {*користувач* [, *користувач* ...] | *роль* / PUBLIC}

[CASCADE CONSTRAINTS]

де

CASCADE – використовується для скасування обмежень, накладених на об'єкт за допомогою привілеї REFERENCES для збереження цілісності посилань.

Приклад:

REVOKE select, insert ON s\_dept FROM scott ;

## 2.6. Створення синоніма об'єкта

Для посилання на таблицю, що належить іншому користувачу, необхідно додати до імені таблиці префікс у виді імені користувача, що створив таблицю, і розділової крапки. Створення синоніма усуває необхідність вказівки схеми в імені об'єкта і забезпечує альтернативне ім'я для таблиці, представлення, послідовності, процедури або інших об'єктів бази даних. Цей спосіб особливо корисний для довгих імен об'єктів - наприклад, імен представлень.

Синтаксис команди створення синоніма:



CREATE [PUBLIC] SYNONYM *синонім* FOR [схема.]об'єкт ;

де

PUBLIC – створює синонім, доступний усім користувачам,

*синонім* – ім'я створюваного синоніма,

*схема* – назва користувача, у схемі якого створено або знаходиться об'єкт,

*об'єкт* – назва об'єкта, для якого створюється синонім (таблиця, представлення).

Ім'я синоніма повинно відрізнятися від імен всіх інших об'єктів даного користувача.

Приклад створення користувачем власного синоніма `alice_dept` для таблиці `s_dept`, що належить користувачу `alice`:

```
CREATE SYNONYM alice_dept FOR alice.s_dept ;
```

Адміністратор бази даних може створювати публічний синонім, доступний усім користувачам бази даних.

Приклад створення публічного синоніму для таблиці `s_dept`, що належить користувачу `alice`:

```
CREATE PUBLIC SYNONYM s_dept FOR alice.s_dept ;
```

Для видалення синоніма використовується наступна команда:

```
DROP SYNONYM синонім ;
```

Публічний синонім може видаляти тільки адміністратор бази даних.

### 3. Порядок роботи з інструментальними засобами взаємодії із сервером бази даних

Будь-які інструментальні засоби і прикладні додатки здійснюють взаємодію із сервером бази даних через мову команд запитів SQL. До складу програмного забезпечення сервера бази даних Oracle входить спеціалізований консольний додаток Oracle SQL\*Plus та програмне середовище взаємодії і

адміністрування сервера бази даних Oracle SQL Developer.

**Oracle SQL\*Plus** – це інтерпретатор команд із мови SQL для виконання дій по створенню інформаційних об'єктів бази даних, їхньої модифікації і наповнення даними.

**Oracle SQL Developer** – це графічний інструмент для розробки і адміністрування баз даних. За допомогою SQL Developer можна переглядати об'єкти бази даних, запускати SQL-команди, редагувати і налагоджувати PL/SQL-програми. Ви також можете запустити будь-яку кількість наданих звітів, а також створювати і зберігати власні. SQL Developer підвищує продуктивність і спрощує підтримку вашої бази даних при виконанні завдань з її розвитку. SQL Developer може підключатися до будь-якої СУБД Oracle від версії 9.2.0.1 і може працювати на Windows, Linux, Mac OSX.

Порядок взаємодії з інструментальними засобами взаємозв'язку із сервером Oracle:

1. Запуск і реєстрація користувача в базі даних. Необхідно обрати попередньо створене підключення до бази (наприклад, base), ввести ім'я користувача - ідентифікатор студента в групі (наприклад, ik3101), пароль.

2. Введення команди і запуск її на виконання здійснюється в робочому листі вікна інструментального засобу. Команда може розташовуватися на декількох рядках. Але повинна завершуватися символом «;» на поточному рядку або «/» на наступній рядку.

3. Виконання командного файлу (попередньо збереженого на локальному диску) здійснюється вказівкою символу «@» і наведенням повної специфікації файлу.

4. Вихід з інструментального засобу здійснюється закриттям його вікна.

Oracle SQL Developer це програмний засіб для взаємодії з сервером бази даних у вигляді GUI-додатків. Тому він підтримує роботу, як у командному рядку робочого листа додатку, так і у меню-орієнтованому інтерфейсі з автоматичною генерацією виконуваних SQL-команд. Робота в такому засобі здійснюється в інтерактивному режимі за допомогою вікон.

Основним вікном у Oracle SQL Developer є навігаційне вікно з деревом доступних об'єктів бази даних. Правою кнопкою миші видаються дозволені операції модифікації обраного об'єкта бази даних.

З основного меню додатку викликається SQL Worksheet - робочий лист редактора команд SQL. Виклик редактора рядків команд здійснюється для введення як одиничних команд SQL без завершального символу «;», так і їх послідовності, які розділені завершальним символом виконання команди «;». Запуск команд на виконання здійснюється від поточного положення маркера до кінця послідовності команд кнопкою F5 або тільки поточної команди за допомогою кнопки F9 або Ctrl-Enter.

Контроль виконання команд виконується переглядом пунктів дерева у навігаційному вікні. Можна зберегти введену послідовність команд у вигляді файлу з типом sql за допомогою пункту головного меню «File / Save».

#### 4. Контрольне завдання

1. Виконати управління доступом до бази даних, згідно наступного порядку:

- 1.1) змініть пароль власної схеми і перереєструйтеся в базі даних;
- 1.2) перегляньте власні привілеї в словнику даних;
- 1.3) створіть нового користувача з ім'ям new\_ікххуу (ххуу – ідентифікатор студента у групі), табличним простором USERS, тимчасовим простором TEMP і квотою в 10M для USERS;
- 1.4) створіть роль з ім'ям role\_ікххуу та призначте їй системні привілеї доступу до бази даних, створення синоніма і таблиці;
- 1.5) призначте новому користувачу створену роль;
- 1.6) перевірте в словнику даних усі виконані призначення та стан власних привілеїв. Поясніть отриманий результат.

2. Виконати дії по наданню новому користувачу прав доступу до таблиць REGION, DEPARTMENT, EMPLOYEE старого користувача, згідно наступного порядку:

2.1) створіть нове підключення до бази з поточними параметрами з'єднання з базою старого користувача. Зареєструйтеся новим користувачем і виконайте запит до словника бази про існування нового користувача;

2.2) надайте новому користувачу право доступу на вибірку даних до вказаних таблиць REGION, DEPARTMENT, EMPLOYEE старого користувача;

2.3) створіть у схемі нового користувача повні копії вказаних таблиць;

2.4) надайте старому користувачу усі права доступу на таблиці нового користувача;

2.5) перевірте в словнику даних усі виконані призначення для старого і нового користувачів. Порівняйте їх.

3. Виконати дії по використанню наданих прав доступу до таблиць REGION, DEPARTMENT, EMPLOYEE старого користувача, згідно наступного порядку:

3.1) новим користувачем додайте новий рядок у таблицю REGION нового користувача, а потім додайте цей же рядок у таблицю REGION старого користувача. Поясніть отриманий результат;

3.2) старим користувачем додайте нові рядки в таблиці DEPARTMENT і EMPLOYEE нового користувача. Перевірте стан цих таблиць старим користувачем, а потім новим користувачем. Поясніть отриманий результат;

3.3) зробіть внесені зміни у таблицях старого і нового користувачів постійними. Поясніть ваші дії;

3.4) старим користувачем створіть синонім на таблицю REGION, що належить новому користувачу, та представлення на підмножину даних з цієї таблиці. За їх допомогою виконайте запити всієї інформації. Порівняйте результат.

3.5) видалите створений синонім. Повторно виконайте запит з використанням синоніму. Поясніть отриманий результат;

3.6) скасуйте привілеї вибірки даних, надані старому користувачу на таблицю REGION нового користувача. Виконайте старим користувачем запит даних з цієї таблиці за допомогою представлення. Поясніть отриманий

результат;

3.7) старим користувачем видалите всі дані з таблиць нового користувача в довільному порядку, а потім спробуйте видалити раніше створеного користувача без опції CASCADE, а потім з цією опцією. Поясніть отриманий результат.

Усі команди контрольного завдання подати у виді єдиного командного файлу, який представити у звіті з необхідними поясненнями.

## **Підсумковий комп'ютерний практикум. РОЗРОБКА БАЗИ ДАНИХ ТА ЇЇ АДМІНІСТРУВАННЯ**

### **1. Мета та основні завдання практикуму**

Мета роботи: перевірка накопичених знань і навичок із розробки бази даних та адміністрування взаємодії із сервером бази даних

Порядок виконання роботи:

1. Використовуючи інструментальний засіб взаємодії із сервером бази даних написати командний файл виконання контрольного завдання практикуму. Як інструментальний засіб застосувати Oracle SQL Developer у командному режимі робочого листа додатка.

2. Виконати команди створеного командного файлу, створити всі передбачені завданням об'єкти бази даних, сформувавши необхідний інформаційний стан цих об'єктів.

3. З використанням системи дистанційного навчання кафедри підготувати звіт з результатами виконання контрольного завдання.

4. На наступному занятті отримати індивідуальне завдання в системі дистанційного навчання кафедри і виконати проектування та адміністрування бази самостійно. Результати завантажити в систему дистанційного навчання.

### **2. Контрольне завдання**

1. Зареєструватися в базі даних під власним шифром студента (ікххуу, де хх-номер групи, уу-номер студента у списку групи).

Створити нового користувача з ім'ям newікххуу, де хх-номер групи, уу-номер студента у списку групи, з табличним простором для розміщення об'єктів бази USERS з квотою в 5М, тимчасовим табличним простором TEMP.

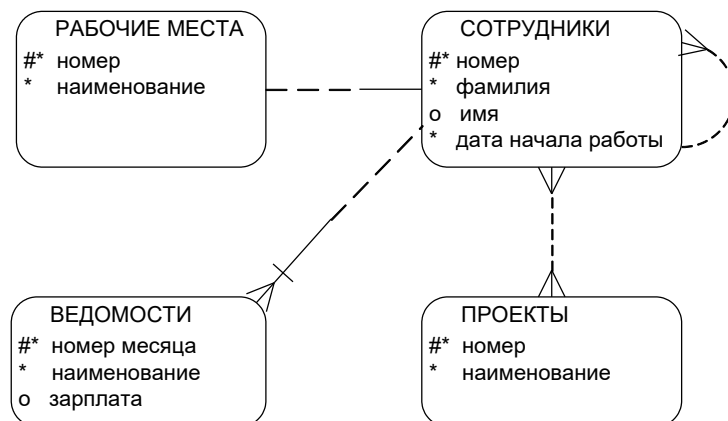
Надати новому користувачу системні привілеї в наступному порядку - доступу до бази даних, створення синоніму, створення таблиці.

Надати новому користувачу об'єктні привілеї доступу на вибірку даних з

однієї з таблиць (REGION, DEPARTMENT, EMPLOYEE) старого користувача.

## 2. Зареєструватися в базі даних під ім'ям нового користувача.

Командами мови DDL створити базу даних, представлену ER-діаграмою.



На значення атрибутів сутностей встановити наступні обмеження:

Атрибут сутності	Формат даних	Значення за замовчуванням
Номер	NUMBER(4)	
Назва	VARCHAR2(20)	
Дата	DATE	установити системну дату як значення стовпця за замовчуванням
Номер місяця	NUMBER(2)	
Зарплата	NUMBER(7,2)	дозволяються значення більше 10
Прізвище	VARCHAR2(15)	
Ім'я	VARCHAR2(15)	

Визначити кількість таблиць і зовнішніх ключів, які будуть утворені в базі даних.

Новим користувачем створити синонім з наступними ім'ям STAB на таблицю старого користувача (REGION, DEPARTMENT, EMPLOYEE) в залежності від наданої об'єктної привілеї. Встановити новим користувачем надану об'єктну привілею.

## 3. Внести в одну із створених таблиць бази даних з колонками «номер»,

«назва» інформацію з доступної новому користувачеві таблиці старого користувача. В команді не використовувати явно назву схеми старого користувача, а колонки номер і назва вказати явно.

Побудувати до бази даних по одному прикладу запитів наступного виду:

- вибірка всіх рядків таблиці
- обмежена вибірка рядків таблиці
- упорядкована вибірка
- вибірка рядків з'єднання таблиць
- групова вибірка
- вибірка з вкладеним підзапитом



## Список рекомендованої літератури

1. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных/Учебник для высших учебных заведений. – СПб: Корона, 2004. – 736с.
2. Пасічник В.В., Резніченко В.А. Організація баз даних та знань/Підручник для ВНЗ. – К.: Видавнича група ВНУ, 2006. – 384 с.
3. Гайна Г.А. Основи проектування баз даних/Навчальний посібник для ВНЗ. – К: КНУБА, 2005. – 204 с.
4. Кирилов В., Громов Г. Введение в реляционные базы данных/Учебное пособие для высших учебных заведений. – СПб: ВНУ-Петербург, 2009. – 454с.
5. Гудов А.М., Шмакова Л.Е. Введение в язык структурированных запросов SQL/Учебное пособие. – Кемерово: Кемеровский госуниверситет, 2001.- 118с.
6. К.Дж. Дейт. Введение в системы баз данных. – М: Вильямс, 2005. - 1328с.
7. М. Грабер. Введение в SQL. – М: Лори, 1996. – 375с.