МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Східноєвропейський національний університет імені Лесі Українки

Кафедра прикладної математики та інформатики

Л. В. Булатецька, В. В. Булатецький

МОВА ЗАПИТІВ SQL

Текст лекцій
нормативної навчальної дисципліни
“Бази даних та розподілені
інформаційно-аналітичні системи”

Луцьк 2018

УДК 004.655
Б 90

Рекомендовано до друку науково-методичною радою
Східноєвропейського національного університету імені Лесі Українки
(протокол № 9 від 20 червня 2018 р.)

Рецензенти:

Федонюк А. А. – кандидат фізико-математичних наук, доцент, завідувач кафедри вищої математики та інформатики Східноєвропейського національного університету імені Лесі Українки;

Пех П. А. – кандидат технічних наук, доцент, завідувач кафедри комп'ютерної інженерії Луцького національного технічного університету

Б 90 Булатецька Л. В. Мова запитів SQL : *текст лекцій нормативної навчальної дисципліни "Бази даних та розподілені інформаційно-аналітичні системи" / Булатецька Леся Віталіївна, Булатецький Віталій Вікторович*. – Луцьк : СНУ імені Лесі Українки, 2018. – 92 с.

Текст лекцій призначений для студентів напряму підготовки 122 Комп'ютерні науки та є логічним доповненням до лекційного курсу нормативної навчальної дисципліни "Бази даних та розподілені інформаційно-аналітичні системи". Призначений для оволодіння теоретичними та практичними навиками написання запитів до бази даних мовою SQL. В розробці подано тексти лекцій та тестові завдання. Рекомендовано викладачам та студентам які викладають або вивчають технології баз даних у вищих навчальних закладах.

ВСТУП

“Бази даних та розподілені інформаційно-аналітичні системи” є нормативною навчальною дисципліною підготовки бакалавра галузі знань 12 Інформаційні технології, спеціальності 122 Комп’ютерні науки, освітньої програми Комп’ютерні науки та інформаційні технології. Метою викладання навчальної дисципліни “Бази даних та розподілені інформаційно-аналітичні системи” є сформувати у слухачів знання, вміння та навички, необхідні для ефективного використання засобів сучасних інформаційних систем (Систем керуваня базами даних) у своїй майбутній професійній діяльності, формування у слухачів знань, вмінь та навичок з проектування, розробки баз даних, використання сучасних мов запитів до баз даних, методів оптимізації, які застосовуються в процесі експлуатації бази даних, а також підвищення рівня теоретичних знань про основи баз даних.

В програмі навчальної дисципліни “Бази даних та розподілені інформаційно-аналітичні системи” велика увага приділяється теорії реляційних баз даних. При вивчені теорії реляційних баз даних неможливо обійтися без матеріалу, який відноситься до мови запитів SQL. Саме мові запитів SQL присвячено дану методичну розробку, так як насьогодні інтерфейси, засновані на SQL, підтримуються майже у всіх існуючих системах керування базами даних (СКБД), навіть таких, які не розроблялися як реляційні системи. У даній методичній розробці після невеликої історичної довідки і короткого вступу в структуру мови SQL, розглянуто типи даних, допустимі в мові SQL і в SQL-орієнтованих базах даних, а також мовні засоби визначення, маніпулювання, керування даними та управління транзакціями. Для кращого розуміння студентами теоретичного матеріалу, виклад супроводжується прикладами запитів. Для контролю якості знань студентів вкінці тексту лекцій подано тестові запитання.

1. ОСНОВИ SQL

1.1. Історія SQL

Перші розробки систем керування реляційними базами даних (реляційних СКБД) були виконані в компанії IBM на початку 1970-х років. Тоді ж було створено мову даних, призначену для роботи в цих системах. Експериментальна версія цієї мови називалася SEQUEL – від англ. Structured English QUEry Language (структурована англійська мова запитів). Проте офіційна версія була названа коротше – SQL (Structured Query Language). Тобто, SQL – це підмова даних, оскільки СКБД містить й інші мовні засоби.

У 1981 році IBM випускає реляційну СКБД SQL/DS. До цього часу компанія Relation Software Inc. (сьогодні це Oracle Corporation) вже випустила свою реляційну СКБД. Ці продукти відразу ж стали стандартом систем, призначених для керування базами даних. До складу цих продуктів увійшов і SQL, який фактично став стандартом для підмов даних. Виробники інших СКБД випустили свої версії SQL. У них були не тільки основні можливості продуктів IBM. Щоб отримати деяку перевагу для "своєї" СКБД, виробники вводили деякі розширення SQL. Разом з тим, почалися роботи зі створення загальновизнаного стандарту SQL.

Таблиця 1. Історія версій стандарту.

Рік	Назва	Зміни
1986	SQL-86	Перший варіант стандарту, прийнятий інститутом ANSI і схвалений ISO в 1987 році
1989	SQL-89	Трохи доопрацьований варіант попереднього стандарту
1992	SQL-92	Значні зміни (ISO 9075); рівень *Entry Level* стандарту SQL-92 був прийнятий як стандарт FIPS 127-2.
1999	SQL:1999	Додана підтримка регулярних виразів, рекурсивних запитів, підтримка тригерів, базові процедурні розширення, нескалярние типи даних і деякі об'єктно-орієнтовані можливості.
2003	SQL:2003	Введено розширення для роботи з XML -даними, віконні функції (застосовувані для роботи з OLAP -базами даних), генератори послідовностей і засновані на них типи даних.
2006	SQL:2006	Функціональність роботи з XML-даними значно розширена. З'явилася можливість спільно використовувати в запитах SQL і XQuery.
2008	SQL:2008	Покращені можливості віконних функцій, усунуті деякі неоднозначності стандарту SQL: 2003

У 1986 році Американський національний інститут стандартів (American National Standards Institute, ANSI) випустив офіційний стандарт

SQL-86, який у 1989 році був оновлений і отримав нову назву SQL-89. У 1992 році цей стандарт був названий SQL-92 (ISO / IEC 9075:1992). В даний час діє стандарт SQL: 2003 (ISO / IEC 9075X: 2003), прийнятий в 2003 році з невеликими модифікаціями внесеними пізніше. В таблиці 1 подана історія версій стандарту.

Будь-яка реалізація SQL в конкретній СКБД дещо відрізняється від стандарту. Так, багато СКБД (наприклад, Microsoft Access 2003, PostgreSQL 7.3) підтримують SQL-92 не повністю, а лише з деяким рівнем відповідності. Крім того, вони підтримують і елементи, які не входять в стандарт. Однак розробники СКБД прагнуть до того, щоб нові версії їх продуктів як можна більше відповідали стандарту SQL.

Мову SQL вважають декларативною (описовою) мовою, на відміну від мов, на яких пишуться програми. Це означає, що вирази на мові SQL описують, що потрібно зробити, а не яким чином.

Однак, останні версії SQL підтримують оператори управління обчисленнями, властиві процедурним мовам управління (оператори умовного переходу і циклу). Тому SQL зараз це не чисто декларативна мова.

Крім вибірки, додавання, зміни та видалення даних з таблиць, SQL дозволяє виконувати всі необхідні дії по створенню, модифікації і забезпечення безпеки баз даних. Всі ці можливості розподілені наступними компонентами SQL:

- DML (Data Manipulation Language – мова маніпулювання даними) призначена для підтримки бази даних: вибору (SELECT), додавання (INSERT), зміни (UPDATE) і видалення (DELETE) даних з таблиць;
- DDL (Data Definition Language – мова визначення даних) призначена для створення, модифікації і видалення таблиць і всієї бази даних;
- DCL (Data Control Language – мова керування даними) призначена для забезпечення захисту бази даних від різного роду пошкоджень;
- TCL (Transaction Control Language) – мова управління транзкціями.

Ключові слова у висловах на мові SQL можуть записуватися як великими, так і малими літерами, в один чи кілька рядків. В кінці виразу повинна стояти крапка з комою (;). Однак в багатьох системах, які забезпечують введення, редагування і відправку на виконання SQL-виразів, у випадку введення одного такого виразу допускається не вказувати ознаку закінчення (;). Якщо ж потрібно виконати блок з декількох SQL-виразів, то вони обов’язково повинні бути розділені крапкою з комою.

Усі ключові слова SQL (команди, оператори та ін.) Є зарезервованими і не повинні використовуватися в якості імен таблиць, стовпців, змінних і т. п.

1.2. Типи даних

Дані – це сукупна інформація, що зберігається в базі даних у вигляді одного з декількох різних типів. За допомогою типів даних встановлюються

основні правила для даних, що містяться в конкретному стовпці таблиці, у тому числі розмір пам'яті, що виділяється для них.

Основні типи даних – символьні, числові, дата та час.

Символьні дані. Складаються з послідовності символів, що входять у визначений набір символів створеної БД. Найчастіше використовуються набори символів ASCII, UTF, тощо.

SQL визначає два первинних символьних типи: із строго заданою кількістю символів (*character*) та змінною кількістю символів (*character varying*).

Для визначення даних символьного типу використовується наступний синтаксис:

CHARACTER (n) або CHAR (n) (рядок фіксованої довжини), де n – максимальна кількість символів, що містяться в рядку. Якщо (n) не зазначено, то передбачається, що рядок складається з одного символу. Якщо у стовпець типу CHARACTER (n) вводиться m < n символів, то позиції що залишилися заповнюються пробілами.

CHARACTER VARYING (n) АБО VARCHAR (n) (рядок змінної довжини) застосовується тоді, коли дані, що вводяться мають різну довжину і небажано доповнювати їх пробілами. При цьому зберігається тільки та кількість символів, яку ввів користувач. В даному випадку задання максимальної кількості символів обов'язкове.

CHARACTER і CHARACTER VARYING можуть брати участь в одних і тих же стрічкових операціях.

Крім цього, у конкретних реалізаціях SQL існують типи для збереження текстових даних "необмеженої" (визначається архітектурою ЕОМ та особливістю ОС) довжини, наприклад MEMO чи TEXT.

Бітовий тип даних. Використовується для визначення бітових рядків, тобто послідовності двійкових цифр (бітів), кожна з яких може мати значення 0 або 1. Дані бітового типу визначаються за допомогою наступного синтаксису:

BIT [VARYING] (<довжина>)

Числові типи даних. Дані числового типу визначаються точністю і довжиною дробової частини. Точність задає спільну кількість значущих десяткових цифр числа, в яку входить довжина як цілої частки, так і дробової, але без врахування самої десяткової крапки. Масштаб вказує кількість дробових десяткових розрядів числа.

Синтаксис:

SMALLINT

{ INTEGER | INT }

BIGINT

{ NUMERIC | DECIMAL | DEC } [(<точність>[,<масштаб>])

Типи NUMERIC і DECIMAL призначені для зберігання чисел в десятковому синтаксисі. За замовчуванням довжина дробової частки

дорівнює нулю, а точність, що приймається за умовчанням, залежить від версії SQL. Тип INTEGER (INT) використовується для зберігання великих додатних або від'ємних цілих чисел. Тип SMALLINT – для зберігання невеликих додатних або від'ємних цілих чисел. В цьому випадку використання зовнішньої пам'яті істотно зменшується.

Дійсні числа або числа з плаваючою крапкою подаються за допомогою мантиси, помноженої на певний степінь десятки (порядку), наприклад: 10E3, +5.2E6, -0.2E-4.

Для визначення даних дійсного типу використовується Синтаксис:

{ FLOAT | REAL } [(<точність>)]

DOUBLE PRECISION

Параметр точність задає кількість значущих цифр мантиси. Точність типів REAL і DOUBLE PRECISION залежить від конкретної реалізації.

Тип даних "дата/час". Даний тип використовується для визначення часу з деякою встановленою точністю. Стандарт SQL підтримує наступний синтаксис:

{ DATE | TIME | TIMESTAMP } [<точність>][WITH TIME ZONE]

Тип даних DATE використовується для зберігання календарних дат, що включають поля YEAR (рік), MONTH (місяць) і DAY (день). Тип даних TIME – для зберігання відміток часу, що включають поля HOUR (години), MINUTE (хвилини) і SECOND (секунди). Тип даних TIMESTAMP – для спільного зберігання дати і часу. Параметр <точність> задає кількість дробових десяткових знаків, що визначають точність збереження значення в полі SECOND. Якщо цей параметр опускається, то за замовчуванням його значення, для стовпців типу TIME, встановлюється рівним нулю (тобто зберігаються цілі секунди), тоді як для полів типу TIMESTAMP він встановлюється рівним 6-и (тобто відмітки часу зберігаються з точністю до мілісекунд). Наявність ключової фрази WITH TIME ZONE визначає використання полів TIMEZONE HOUR і TIMEZONE MINUTE, тим самим задаються година і хвилини зміщення зонального часу по відношенню до універсального координатного часу (час за Гринвічем). Дані типу INTERVAL використовуються для представлення періодів часу.

1.3. Невизначені значення

Якщо в клітинку (поле) таблиці ви вводите якісь значення (літери, цифри, пробіли або ще що-небудь), то ці дані набувають так звані певні значення. З визначеними значеннями, що належать тому чи іншому типу, можна виконувати якісь операції. Однак ви можете створити запис (рядок) таблиці, нічого не вводячи в її поля (стовпці). Такий запис нічого не містить. Запис у таблиці, яка створена, але в яку не введені конкретні значення, називається порожньою. У кожній комірці порожнього запису міститься так зване невизначене значення, що позначається через NULL.

Отже, кожен раз, коли ви додаєте в таблицю новий запис, не вводячи в неї яких-небудь конкретних значень, СКБД встановлює у її полях значення NULL.

Значення NULL ви можете інтерпретувати так, як вам хочеться, наприклад, “ще не введено”, “поки не відомо”. Важливо розуміти, що число 0 або порожній рядок є цілком визначеними значеннями і відрізняються від NULL, хоча візуально порожній рядок і невизначене значення можуть відображатися однаково – у вигляді порожньої комірки таблиці. Багато функцій SQL повертають, в залежності від обставин, якесь певне значення або NULL. Це необхідно враховувати при складанні SQL-виразів, оскільки комбінація NULL і інших конкретних значень може дати той чи інший результат в залежності від конкретної операції.

При порівнянні будь-якого певного значення з NULL результат дорівнює false. Те ж саме відбувається при порівнянні двох величин, значеннями яких є NULL, тобто порівняння двох невизначених величин дає в результаті false.

У багатьох випадках значення NULL є дуже корисними. Так, наприклад, ви можете створити таблицю і ввести в неї відомості, якими ви маєте в своєму розпорядженні в даний момент, залишивши інші значення поки невизначеними. Потім, по мірі надходження нової інформації, ви можете змінити значення NULL на ті, які стали відомі.

2. ПРОСТА ВИБІРКА ДАНИХ МОВИ SQL

Припустимо, що реляційна база даних, що складається з однієї або декількох таблиць, створена і ви до неї вже підключилися. У цьому випадку типовим практичним завданням є отримання потрібних даних. Мова SQL надає для цього широкі можливості.

В результаті виконання виразу на мові SQL (SQL-вирази) створюється таблиця, яка або містить запитані дані, або порожня, якщо даних, що відповідають запиту, не знайшлося. Ця таблиця, називається результатною, і існує тільки під час сеансу роботи з базою даних. Інакше кажучи, вона не зберігається на жорсткому диску комп’ютера, і тому її ще називають віртуальною.

Для прикладу, розглянемо фрагмент бази даних “Книжковий магазин” (рис.1). Дана база даних призначена для обліку розповсюдження книг в книжному магазині. В даній базі міститься інформація про наявні книги в книжковому магазині, інформація про їх авторів та видавців, а також відомості про продажі та покупців (таб. 2 – таб. 7).

Authors	**BookSales**	**Buyers**	**Publishers**	**TitleAuthors**	**Titles**
AUID *(PK)*	BookSalesID *(PK)*	BuyerID *(PK)*	PubID *(PK)*	ISBN *(PK) (FK)*	ISBN *(PK)*
Surname	Qtr	Name	Name	AUID *(PK) (FK)*	Title
Name	Sales	CompanyName	Company		PubID *(FK)*
Lastname	SalesRep	Address	Address		YearPub
Address	ISBN *(FK)*	City	City		Description
Date_of_birth	Units	Region	Region		Notes
Contracted	BuyerID *(FK)*	Zip	Zip		Subject
		Phone	Phone		Comments
		Fax	Fax		
			Comments		

Рис. 1. Фрагмент бази даних “Книжковий магазин”

Зв’язок між таблицями здійснюється за допомогою таких пар полів з типом зв’язку “один-до-багатьох” відповідно:

1. Authors.AUID – TitleAuthors.AUID
2. Titles.ISBN – TitleAuthors.ISBN
3. Publishers.PubID – Titles.PubID
4. Titles.ISBN – BookSalesID.ISBN
5. Buyers. BuyerID – BookSalesID. BuyerID

Таблиця 2. Відомості про назви книг (таблиця Titles)

Назва поля	**Тип поля**	**Опис поля**
ISBN	Integer	Універсальний ідентифікаійний номер книги
Title	Varchar	Назва книги
PubID	Integer	Ідентифікаійний номер видавництва
YearPub	Date	Дата видавництва
Description	Varchar	Короткий опис книги
Notes	Varchar	Примітки
Subject	Varchar	Тематика книги
Comments	Varchar	Коментарі

Таблиця 3. Список авторів (таблиця Authors)

Назва поля	**Тип поля**	**Опис поля**
AUID	Integer	Ідентифікаійний номер автора
Surname	Varchar	Прізвище автора
Name	Varchar	Ім’я автора
Lastname	Varchar	По батькові автора
Address	Varchar	Адреса автора
Date_of_birth	Date	Дата народження
Contracted	Numeric	Номер договору

Таблиця 4. Таблиця, яка пов'язує Titles та Authors (таблиця TitleAuthors)

Назва поля	**Тип поля**	**Опис поля**
ISBN	Integer	Універсальний ідентифікаійний номер книги
AUID	Integer	Ідентифікаійний номер автора

Таблиця 5. Відомості про книжкові продажі (таблиця BookSales)

Назва поля	**Тип поля**	**Опис поля**
BookSalesID	Integer	Ідентифікаійний номер продажу
Qtr	Numeric	Квартал
Sales	Numeric	Ціна
SalesRep	Varchar	Ім'я торгового представника
ISBN	Integer	Ідентифікаійний номер книги
Units	Numeric	Кількість проданих книг
BuyerID	Integer	Ідентифікаійний номер покупця

Таблиця 6. Відомості про покупців (таблиця Buyers)

Назва поля	**Тип поля**	**Опис поля**
BuyerID	Integer	Ідентифікаійний номер покупця
Surname	Varchar	Прізвище покупця
Name	Varchar	Ім'я покупця
Lastname	Varchar	По батькові покупця
CompanyName	Varchar	Назва компанії, яку представляє покупець
Address	Varchar	Адреса (вулиця, будинок, квартира)
City	Varchar	Місто
Region	Varchar	Область
Zip	Varchar	Поштовий код
Telephone	Varchar	Номер телефону
Fax	Varchar	Фах

Таблиця 7. Відомості про видавців (таблиця Publishers)

Назва поля	**Тип поля**	**Опис поля**
PubID	Integer	Ідентифікаійний номер видавництва
Name	Varchar	Назва видавництва
Company	Varchar	Назва компанії
Address	Varchar	Адреса (вулиця, будинок, квартира)
City	Varchar	Місто
Region	Varchar	Регіон
Zip	Varchar	Поштовий код
Phone	Varchar	Номер телефону
Fax	Varchar	Фах
Comments	Varchar	Коментарі

2.1.Основний SQL-вираз для вибірки даних. Оператор SELECT_FROM

Основний SQL-вираз для вибірки даних, має вигляд:

SELECT списокСтовпців FROM списокТаблиць;

Оператори SELECT (вибрати) і FROM (з) в SQL-виразі, що визначають вибірку даних, є обов'язковими, тобто жоден з них не можна пропустити. В результаті виконання цього запиту створюється віртуальна таблиця, що містить зазначені стовпці і всі записи вихідної таблиці.

Оператор SELECT здійснює проекцію відношення, зазначеного у виразі FROM, на задану множину атрибутів (стовпців), зазначеного у виразі SELECT. Так, наприклад, якщо вихідна таблиця R містить стовпці A1, A2, ..., An (іншими словами, таблиця представляє деяке відношення R (A1, A2, ..., An) над атрибутами A1, A2, ..., An), то оператор: SELECT Al, A2, ..., Ak FROM R; реалізує проекцію R [A1, A2, ..., Ak] цього відношення на атрибути A1, A2, ..., Ak (k = 1, 2, ..., k).

У виразі FROM вказується список імен таблиць бази даних, з яких потрібно вибрати дані. У простому випадку списокТаблиць містить лише одне ім'я таблиці. Якщо ж таблиць декілька, то їх імена у списку розділяються комами. Якщо у виразі FROM вказано більше однієї таблиці, то результатна таблиця виходить з декартового добутку, перерахованих в списку таблиць. Список стовпців – це перелік імен стовпців, розділених комою, як вони визначені в таблиці, зазначеної у виразі FROM.

Простим прикладом оператора SQL може служити:

SELECT AUID FROM Authors;

Зрозуміло, можна вказати всі або тільки деякі стовпці. Якщо ви хочете отримати всі стовпці таблиці, то замість списку стовпців достатньо вказати символ (*). Якщо у виразі FROM зазначено кілька таблиць, то у виразі SELECT імена стовпців повинні містити префікси, що вказують, до якої саме таблиці вони відносяться. Префікс відокремлюється від імені стовпця крапкою.

Оператор SELECT, що відображає всі стовпчики таблиці Authors, виглядає в такий спосіб:

SELECT * FROM Authors;

Якщо нам потрібно вибрати тільки декілька стовпців:

SELECT Name, AUID FROM Authors;

Коли в запиті використовується більше однієї таблиці, гарною практикою програмування є написання перед іменем стовпчика ім'я таблиці і розміщення між ними точки (.). Доти, поки ім'я стовпчика залишається унікальним серед імен інших стовпчиків, що беруть участь у запиті, SQL не вимагає від вас обов'язкової вказівки імені таблиці. Проте, використання конструкції "таблиця.Стовпчик" є гарною навичкою.

Заголовки стовпців в результатній таблиці можна переозначити, призначивши для них так звані псевдоніми. Для цього в списку стовпців після відповідного стовпця слід написати вираз виду: AS заголовок_стовпця. Наприклад:

SELECT Name As NameAuthors, DateofBirth As Birthday FROM Authors;

Псевдоніми також можна задати і для кожної таблиці після ключового слова FROM. Для цього достатньо вказати псевдонім через пробіл відразу після імені відповідної таблиці. Псевдоніми таблиць, коротші, ніж їхні імена, їх зручно використовувати в складних запитах. Наприклад:

SELECT T1. Name, T2.Title FROM Authors T1, Titles T2;

2.2. Оператори для уточнення запиту. Порядок виконання операторів SQL-виразу

Основний SQL-вираз для вибірки даних, має вигляд:

SELECT списокСтовпців FROM списокТаблиць;

Такий запит повертає таблицю, отриману з зазначеної в операторі FROM (або з декартового добутку зазначених таблиць, якщо їх декілька), шляхом виділення в ній тільки тих стовпців, які визначені в операторі SELECT.

Для уточнення запиту на вибірку даних служить ряд додаткових операторів:

- WHERE (де) – вказує записи, які повинні увійти в результатну таблицю (фільтр записів);
- GROUP BY (групувати по) – групує записи по значеннях певних стовпців;
- HAVING (що мають, за умови) – вказує групи записів, які мають увійти до результатної таблиці (фільтр груп);
- ORDER BY (сортувати по) – сортує (впорядковує) записи.

Ці оператори не є обов'язковими. Їх можна зовсім не використовувати, або використовувати лише деякі з них, або всі одразу. Якщо застосовуються декілька операторів, то в SQL-виразі вони виконуються у певному порядку.

Запит даних з таблиці із використанням всіх перерахованих операторів уточнення запиту має наступний вигляд:

SELECT списокСтовпців FROM Ім'яТаблиці
WHERE УмоваПошуку
GROUP BY стовпецьГрупування
HAVING УмоваПошуку
ORDER BY УмоваСортування;

Порядок подання операторів в SQL-виразі не співпадає з порядком їх виконання. Отже, перераховані оператори SQL-виразу виконуються в наступному порядку, передаючи один одному результат у вигляді таблиці:

- FROM – вибирає таблицю з бази даних; якщо зазначено декілька таблиць, то виконується їх декартовий добуток і результуюча таблиця передається для обробки наступному оператору;

- WHERE – з таблиці вибираються записи, що відповідають умові пошуку, і відкидаються всі інші;
- GROUP BY – створюються групи записів, відібраних за допомогою оператора WHERE (якщо він присутній в SQL-виразі); кожна група відповідає якому-небудь значенню стовпця групування. Стовпець групування може бути будь-яким стовпцем таблиці, заданої в операторі FROM, а не тільки тим, що вказаний в SELECT;
- HAVING – обробляє кожну із створених груп записів, залишаючи тільки ті з них, які задовольняють умові пошуку; цей оператор використовується тільки разом з оператором GROUP BY;
- SELECT – вибирає з таблиці, отриманої в результаті використання перерахованих операторів, тільки вказані стовпці;
- ORDER BY – сортує записи таблиці. При цьому в умові сортування можна звертатися лише до тих стовпців, які вказані в операторі SELECT.

2.3. Оператор WHERE

За допомогою оператора WHERE Ви можете контролювати вміст результуючого набору оператора SELECT_FROM. При цьому, оператор WHERE ви можете використовувати в одному із таких двох випадків:

- використовувати WHERE для обмеження вмісту результуючого набору;
- використовувати WHERE для зв’язування двох або декількох таблиць у єдиний результуючий набір.

Використання WHERE для обмеження результуючого набору. Умови пошуку в операторі WHERE (де) є логічними виразами, тобто приймають одне з двох можливих значень – true або false.

При складанні логічних виразів використовуються спеціальні ключові слова і символи операцій порівняння, які називають предикатами. Вирази з оператором WHERE використовуються не тільки при вибірці даних (тобто з оператором SELECT), але і при вставці, модифікації і видаленні записів. Нижче наведено список всіх предикатів:

- предикати порівняння: (=), (<), (>), (<>), (<=), (> =);
- BETWEEN;
- IN, NOT IN;
- LIKE, NOT LIKE;
- IS NULL;
- ALL, SOME, ANY;
- EXISTS;
- UNIQUE;
- DISTINCT;
- OVERLAPS;
- MATCH;
- SIMILAR.

У пропозиції WHERE ви можете використовувати декілька логічних операторів AND і OR

Предикати порівняння: (=), (<), (>), (<>), (<=), (> =).

SELECT Name, Region FROM Publishers WHERE Region = ‘Vol’

Приведений вище оператор SQL повертає в результуючому наборі *підмножину* даних. Це означає, що результуючий набір містить не всі рядки таблиці Publishers, а тільки ті з них, що задовольняють критерію пропозиції WHERE.

BETWEEN. Предикат BETWEEN (між) дозволяє задати вираз перевірки входження якого-небудь значення в діапазон, який визначається граничними значеннями. Наприклад:

SELECT PubID, Name, Region, City FROM Publishers WHERE PubID BETWEEN 10 AND 15;

Тут ключове слово AND являє собою логічний вираз І. Граничні значення (у прикладі це 10 і 15) входять в діапазон. Причому перше граничне значення повинно бути не більше другого.

Еквівалентним наведеному є вираз з предикатами порівняння:

WHERE PubID > = 10 AND PubID <= 15;

Крім даних числового типу, у виразах з BEETWEEN можна використовувати дані наступних типів: символьні, бітові, дати-часу. Так наприклад, щоб вибрати записи, в яких імена авторів знаходяться в діапазоні від А до Н, можна використовувати такий вираз:

SELECT Name FROM Avtors WHERE Name BETWEEN ‘A’ AND ‘H’;

IN і NOT IN. Предикати IN (в) і NOT IN (не в) використовуються для перевірки входження якого-небудь значення в заданий список значень. Наприклад, якщо вам потрібний список усіх видавців у областях Волинській, Львівській і Київській, то в пропозиції WHERE ви можете використовувати ключове слово IN, за яким у квадратних дужках повинні випливати необхідні значення, розділені комами.

SELECT PubID, Name, City, Region FROM Publishers WHERE Region IN (‘Vol’, ‘Lviv’, ‘Kyiv’)

Якщо потрібно отримати дані про всіх видавців крім тих що проживають в областях Волинській, Львівській й Київській, то можна використовувати предикат NOT IN:

SELECT PubID, Name, City, Region FROM Publishers WHERE Region NOT IN (‘Vol’, ‘Lviv’, ‘Kyiv’)

LIKE і NOT LIKE. Предикати LIKE (схожий) і NOT LIKE (не схожий) використовуються для перевірки часткової відповідності символьних рядків. Наприклад, вам потрібно одержати всі рядки, у яких поле Region містить букву “і” у будь-якій позиції.

Критерій часткової відповідності задається за допомогою двох символів-масок: значка відсотка (%) і підкреслення (_). Знак відсотка означає будь-який набір символів, в тому числі і порожній, а символ підкреслення –

будь-який одиночний символ. Тоді ви можете використовувати такий оператор SQL SELECT_FROM.

SELECT PubID, Name, City, Region FROM Publishers WHERE Region LIKE (‘%i%’)

Якщо вам потрібно вибрати записи про видавців, що проживають в Луцьку, на вулиці Потапова, а стовпець Адреса містить назву вулиці, номер будинку і номер квартири, то для цього підійде наступний вираз:

SELECT PubID, Name, Address FROM Publishers WHERE Address LIKE ‘%Potapova%’ AND City=‘Lutsk’;

Якщо ви хочете вибрати всіх видавців, що проживають в Луцьку, крім тих, що проживають на вулиці Потапова, то скористайтеся таким виразом:

SELECT PubID, Name, Address FROM Publishers WHERE Address NOT LIKE ‘%Potapova%’ AND City=‘Lutsk’;

Можливо, вам буде потрібно вибрати записи, що містять символи відсотка і / або підкреслення. Тоді необхідно, щоб такі символи сприймалися інтерпретатором SQL не як символи-маски. Щоб знак відсотка або підкреслення сприймався буквально, перед ним необхідно вказати спеціальний керуючий символ. Цей символ можна визначити довільно, лише б він не зустрічався в якості елемента даних. У наступному прикладі показано, як це можна зробити:

… WHERE percent LIKE ‘20 #% ‘ESCAPE’ # ‘;

Тут за ключовим словом ESCAPE вказується символ, який використовується в якості керуючого. Таким же способом можна відключити і сам керуючий символ.

У Microsoft Access в якості символів-масок використовується знак зірочка (*), що означає будь-який набір символів і знак запитання (?) – будь-який одиночний символ.

IS NULL. Предикат IS NULL використовується для виявлення записів, в яких той чи інший стовпець не має значення. Наприклад, для отримання записів про видавців, для яких не вказана адреса, можна використовувати наступний вираз:

SELECT PubID, Name, Address FROM Publishers WHERE Address IS NULL;

Для отримання записів, в яких стовпець Адреса містить визначені значення (тобто відмінні від NULL), можна використовувати аналогічний вираз, але з логічним оператором NOT (не):

SELECT PubID, Name, Address FROM Publishers WHERE Address IS NOT NULL;

Не слід використовувати предикати порівняння з NULL, такі як

Адреса = NULL.

Логічні оператори. Логічні вирази в операторах WHERE можуть бути складними, тобто складатися з двох і більше простих виразів, з'єднаних між собою логічними операторами AND і/або OR. Оператор AND виконує роль логічного оператора І, а оператор OR – оператора АБО. Логічний оператор NOT застосовується до одного виразу (можливо і до складного), розташованому праворуч від нього. Цей оператор міняє значення виразу на протилежне.

Наприклад, якщо вам потрібний список усіх видавців у областях Волинській, Львівській і Київській.

SELECT PubID, Name, City, Region FROM Publishers WHERE Region= 'Vol' OR Region= 'Lviv' OR Region= 'Kyiv'

Зверніть увагу, що тут використовується логічний оператор OR (АБО), а не AND (І), оскільки нам потрібні видавці, які проживають або в 'Vol', або в 'Lviv' або в 'Kyiv'. Якби замість оператора OR ми використали AND, то отримали б порожню таблицю, так як у вихідній таблиці немає жодного запису, в якій один і той же стовпець мав би різні значення.

Наступний SQL-вираз еквівалентний розглянутому раніше. Він заснований на використанні оператора IN :

SELECT PubID, Name, City, Region FROM Publishers WHERE Region IN ('Vol', 'Lviv', 'Kyiv');

Якщо потрібно отримати дані про всіх видавців, які не проживають ні в 'Vol', ні в 'Lviv', ні в 'Kyiv', то можна використати такий SQL-вираз:

SELECT PubID, Name, City, Region FROM Publishers WHERE NOT (Region= 'Vol' OR Region= 'Lviv' OR Region= 'Kyiv');

Цей вираз еквівалентний наступним двом:

SELECT PubID, Name, City, Region FROM Publishers WHERE Region<> 'Vol' OR Region<>'Lviv'AND Region<> 'Kyiv';

і:

SELECT PubID, Name, City, Region FROM Publishers WHERE Region NOT IN ('Vol', 'Lviv', 'Kyiv');

Використання WHERE для зв'язування двох або декількох таблиць у єдиний результуючий набір. У пропозиції WHERE можна порівнювати стовпчики, що належать різним таблицям. При цьому ви можете задати критерій, що буде зв'язувати дві або декілька таблиць у єдиний результуючий набір. Така форма пропозиції WHERE має такий синтаксис:

SELECT table1. column, table2. column FROM tablel, table2

WHERE tablel. column = table2. column

Тут table1 і table2 – різні таблиці однієї бази даних, а column – стовпчик, що міститься в обох таблицях.

SELECT Titles.Title, Publishers.Name FROM Publishers, Titles WHERE Publishers.PubID = Titles.PubID;

Наведений вище запит SQL створює результуючий набір, що містить назву книги й ім'я її видавця. Це досягається за допомогою пропозиції

WHERE, що вказує SQL вибрати з обох таблиць тільки ті рядки, у яких поля PubID збігаються. Майте на увазі, що все це робиться без яких-небудь програмних процедур, спеціального індексування або команд сортування. Всю роботу бере на себе SQL. Крім того, цей оператор SQL містить декілька нових елементів, що потребують подальшого розгляду.

Порівнювані в пропозиції WHERE стовпчики (Publishers. PubID і Titles. PubID) не включені в пропозицію SELECT нашого оператора. Для того, щоб використовувати стовпчик у пропозиції WHERE, необовязково включати його в пропозицію SELECT, оскільки цей стовпчик вже існує у фізичній таблиці.

Об'єднання таблиць за допомогою пропозиції WHERE завжди повертає неоновлюваний результуючий набір. Ви не зможете змінювати дані в колонках представлення, отриманого таким способом. Для того, щоб зв'язати таблиці й одночасно зберегти можливість відновлення фізичних таблиць, що лежать в основі об'єднаного представлення, ви повинні використовувати пропозицію JOIN, яку ми розглянемо пізніше.

У одному операторі SELECT_FROM ви можете об'єднувати версії пропозиції WHERE що зв'язують і обмежують:

SELECT Titles.Title, Publishers.Name FROM Titles, Publishers WHERE Titles.PubID = Publishers.PubID AND Publishers PubID BETWEEN 5 AND 10;

Наведений вище оператор SQL вибирає тільки ті записи, для яких значення колонок PubID збігаються і знаходяться між 5 і 10.

За допомогою пропозиції WHERE можна зв'язувати більш двох таблиць. При цьому стовпчик, що зв'язує таблиці tablel і table2, не обов'язково повинний збігатися зі стовпчиком, що зв'язує таблиці table2 і table3. Наприклад:

SELECT Titles.PubID, Titles.Title, Publishers.Name, Authors.Name FROM Titles, Publishers, Authors WHERE Titles.PubID = Publishers.PubID AND Titles.AUID = Authors.AUID

У цьому прикладі таблиці Publishers і Titles пов'язані за допомогою стовпчика PubID, а таблиці Titles і Authors – за допомогою стовпчика AUID. Після установки зв'язку обрані стовпчики відображаються в результуючому наборі.

2.4. Агрегатні функції мови SQL

Стандарти мови SQL визначають базовий набір функцій, що повинен бути присутнім у всіх SQL-сумісних системах. Ці функції називаються *агрегатними* і використовуються для швидких обчислень над збереженими в стовпчиках числовими даними.

- COUNT – повертає число рядків і використовується для визначення загального числа рядків у результуючому наборі (це єдина агрегатна функція, що можна застосовувати для нечисловых стовпчиків);
- SUM – повертає суму всіх значень у стовпчику;

- AVG – повертає середнє для всіх значень у стовпчику;
- MAX – повертає найбільше значення в стовпчику;
- MIN – повертає найменше значення в стовпчику.

Такий оператор SQL демонструє всі п’ять агрегатних функцій.

SELECT COUNT(Units) AS UnitCount, AVG(Units) AS UnitAvg, SUM(Units) AS UnitSum, MIN(Units) AS UnitMin, MAX(Units) AS UnitMax FROM BookSales;

У одному операторі SELECT_FROM можна одночасно використовувати пропозицію WHERE і функції, що агрегують. Такий запит показує, як за допомогою пропозиції WHERE можна обмежити кількість рядків, що беруть участь у розрахунках із використанням функцій, що агрегують.

SELECT COUNT(Units) AS UnitCount, AVG(Units) AS UnitAvg, SUM(Units) AS UnitSum, MIN(Units) AS UnitMin, MAX(Units) AS UnitMax FROM BookSales WHERE Qtr =1;

2.5. Оператор GROUP BY

Оператор GROUP BY (групувати по) використовується для групування записів за значеннями одного або декількох стовпців. Якщо в SQL-вирзі використовується оператор WHERE, то оператор GROUP BY знаходиться і виконується після нього. Якщо WHERE не використовується, то групуються всі записи вихідної таблиці.

Припустимо, що на основі таблиці Publishers треба згрупувати дані по регіонах. Для цього можна скористатися наступним SQL-виразом:

SELECT Name, Company, Region FROM Publishers GROUP BY Region;

При цьому записи з однаковими назвами регіонів розташовані поруч один з одним (в одній групі).

Наприклад, ви хочете створити набір даних, що містить назву книги і сумарна кількість її проданих примірників. Це можна зробити за допомогою такого запиту:

SELECT Title, SUM(Units) AS UnitsSold FROM BookSales, Titles Where Titles.ISBN= BookSales.ISBN GROUP BY Titles.ISBN;

Пропозиція GROUP BY потребує, щоб усі числові стовпчики в списку полів вибірки оператора SELECT були аргументами якої-небудь агрегатної функції мови SQL (SUM, AVG, MIN, MAX, COUNT). Крім того, у цьому випадку в списку полів вибірки оператора SELECT не можна використовувати зірочку (*).

Оператор GROUP BY збирає записи в групи і впорядковує (сортує) групи за алфавітом (точніше, за ASCII-кодами символів).

2.6. Оператор HAVING

Оператор HAVING зазвичай використовується разом з оператором групування GROUP BY і задає фільтр записів в групах. Оператор HAVING

діє точно так само, як і оператор WHERE, із тієї лише різницею, що дія оператора HAVING поширюється на результуючі стовпчики, створені пропозицією GROUP BY, а не на стовпчики фізичної таблиці.

Нехай, ви хочете одержати список назв усіх книг, проданих за рік у кількості більш 100 примірників.

SELECT Title, SUM(Units) AS UnitsSold FROM BookSales, Titles Where Titles.ISBN=BookSales.ISBN GROUP BY Titles.ISBN HAVING SUM(Units)>100;

Стовпчики, використані в пропозиції HAVING, не обов'язково повинні бути присутнім у списку полів вибірки оператора SELECT. Структура пропозиції HAVING підпорядковується тим же правилам, що і структура пропозиції WHERE. У ньому ви можете використовувати логічні оператори AND, OR і NOT. Такий запит SQL повертає суми продажів по всіх книгах, проданих у кількості більш 100 примірників, найменування яких містять у другій позиції букву "a".

SELECT Title, SUM(Sales) AS SalesAmt FROM BookSales, Titles Where Titles.ISBN=BookSales.ISBN GROUP BY Titles.ISBN HAVING SUM(Units)>100 AND Title Like '_a%';

Якщо в SQL-виразі оператора GROUP BY немає, то оператор HAVING застосовується до всіх записів, що повертається оператором WHERE. Якщо ж відсутній і WHERE, ТО HAVING діє на всі записи таблиці.

2.7. Оператор ORDER BY

Оператор SELECT_FROM повертає записи в результуючий набір у тому порядку, у якому вони зустрічаються у фізичній таблиці даних. Але якщо ви захочете показати їх у відсортованому виді, то для цього ви можете скористатися оператором ORDER BY.

Цей оператор сортує рядки всієї таблиці або окремих її груп (у випадку використання оператора GROUP BY). Якщо у виразі запиту оператора GROUP BY немає, то оператор ORDER BY розглядає всі записи таблиці як одну групу.

Додавання ключових слів ASC або DESC після імені поля в пропозиції ORDER BY вказує на зростаючий або спадаючий порядок сортування відповідного стовпчика. Якщо порядок не заданий, припускається сортування по зростанню. Тобто значення ASC прийнято за замовчуванням, тому якщо необхідне сортування, наприклад, в алфавітному порядку, то спеціально вказувати порядок не вимагається;

У такому прикладі показано, як відобразити записи таблиці Authors, відсортовані по полю Name в спадаючому порядку.

SELECT * FROM Authors ORDER BY Name DESC;

У пропозиції ORDER BY може бути зазначено більше одного поля. SQL створить результуючий набір, що відбиває сукупний порядок сортування, зазначений у пропозиції ORDER BY.

Наприклад:

SELECT Region, City FROM Publishers ORDER BY Region DESC, City ASC;

В даному запиті ми сполучили в результуючому наборі можливість зміни порядку рядків із можливістю зміни порядку стовпчиків.

2.8. Пропозиція DISTINCT

Відразу за оператором SELECT до списку стовпців можна використовувати ключові слова ALL (всі) і DISTINCT (відрізняються), які вказують, які записи подавати в результуючій таблиці. Якщо ці ключові слова не використовуються, будуть вибратися всі записи, що також відповідає використанню ключового слова ALL. У випадку використання DISTINCT в результуючій таблиці подаються тільки унікальні записи. При цьому якщо у вихідній таблиці знаходяться декілька ідентичних записів, то з них вибирається тільки перший.

Наприклад, ви хочете одержати список усіх покупців, що зробили хоча б одне замовлення. Проблема полягає в тому, що деякі клієнти зробили декілька замовлень, а ви не хочете бачити ті самі імена в результуючому наборі. Щоб виключити дублікати, можна використовувати ключове слово DISTINCT.

SELECT DISTINCT Name FROM Buyers, BookSales WHERE Buyers. BuyerID – BookSalesID. BuyerID;

Якщо список полів в операторі SELECT містить у собі більше одного стовпчика, то усі вони використовуються для оцінки унікальності рядка.

SELECT DISTINCT Surname, Name, Lastname FROM Buyers, BookSales WHERE Buyers. BuyerID – BookSalesID. BuyerID;

Зверніть увагу, що перший запит для кожного значення поля Name таблиці даних повертає по одному запису. Другий запит повертає для кожного значення поля Name декілька записів, оскільки усі вони відрізняються один від одного значеннями полів Surname та Lastname.

У Microsoft Access крім ключових слів ALL I DISTINCT після SELECT можна використовувати ключове слово TOP з додатковими параметрами, яке використовуються для обмеження числа записів у результуючому наборі. Припустимо, що ви хочете одержати список п'ятьох кращих бестселерів із таблиці продажів. У цьому випадку ви можете скористатися пропозицією TOP n, що повертає перші n записів. Якщо у вас є два записи з однаковими значеннями, то SQL поверне обидва записи. Якби в нашому прикладі з книгами п'ятий і шостий записи були однакові, то в результуючий набір потрапить шість записів, а не п'ять.

Застосовуючи пропозицію TOP n, ви зобов'язані використовувати пропозицію ORDER BY, гарантуючи тим самим, що результуючий набір буде відсортований. У противному випадку ви одержите довільний набір, тому

що SQL спочатку виконує пропозицію ORDER BY, а потім уже вибирає запитані вами перші n записів. Зрозуміло, що без пропозиції ORDER BY ви не одержите очікуваного результату. Якщо в запиті є присутнім і оператор WHERE, то SQL виконує ці оператори в такому порядку: WHERE, ORDER BY, TOP n. Як бачите, відсутність у цьому випадку пропозиції ORDER BY по всій можливості призведе до появи сміття в результуючому наборі.

*SELECT TOP 5 * FROM BookSales ORDER BY Sales DESC;*

Результуючий набір містить перші n записів вибірки, відсортованої по спаданню або по зростанню в залежності від наявності ключових слів DESC або ASC.

Пропозиція TOP n PERCENT повертає не n, а n відсотків перших записів.

*SELECT TOP 5 PERCENT * FROM BookSales ORDER BY Sales;*

3. СКЛАДНІ ЗАПИТИ В SQL

У SQL складні запити є комбінацією простих SQL-запитів. Кожен простий запит в якості відповіді повертає набір записів (таблицю), а комбінація простих запитів повертає результат тих чи інших операцій над відповідями на прості запити.

У SQL складні запити отримуються з інших запитів наступними способами:

- застосуванням до SQL-запитів операторів об’єднання і з’єднання наборів записів, що повертаються запитами. Ці оператори називають теоретико-множинними або реляційними.
- вкладенням SQL-виразу запиту в SQL-вираз іншого запиту. Перший з них називають підзапитом, а другий – зовнішнім або основним запитом;

3.1. Теоретико-множинні операції

Над наборами записів, що містяться в таблицях бази даних і/або повертаються запитами, можна здійснювати теоретико-множинні операції, такі як декартовий добуток, об’єднання, перетин і віднімання.

Декартовий добуток наборів записів.

SELECT *списокСтовпців* FROM Tl, T2, ... Tn;

повертає набір записів, отриманий в результаті декартового добутку наборів записів з таблиць Tl, T2, ... Tn. Таблиці, зазначені в операторі FROM, можуть бути як таблицями бази даних, так і віртуальними таблицями.

Іноді потрібно отримати декартовий добуток таблиці самої на себе. У цьому випадку необхідно застосувати різні псевдоніми для цієї таблиці, наприклад:

SELECT *списокСтовпців* FROM Mytab Tl, Mytab T2;

Спроба виконати запит:

SELECT *списокСтовпців* FROM Mytab, Mytab;
призведе до помилки.

У списку стовпців слід використовувати повні імена стовпців, використовуючи псевдоніми таблиць, або символ (*), якщо потрібно отримати всі стовпці.

Наприклад, потрібно знайти прізвища авторів, що мають однакові дні народження. При з'єднанні таблиці з її ж копією вводяться псевдоніми (аліаси) таблиці. Запит для пошуку прізвищ авторів, що мають однакові дні народження, виглядає таким чином:

SELECT FIRST. Name, SECOND.Name
FROM Authors FIRST, Authors SECOND
WHERE FIRST.Date_of_birth = SECOND. Date_of_birth

У цьому запиті введені два псевдоніма для однієї таблиці Authors, що дозволяє коректно задати вираз, що зв'язує дві копії таблиці. Щоб виключити повторення рядків у виведеному результаті запиту через повторне порівняння однієї і тієї ж пари авторів, необхідно задати порядок проходження для двох значень так, щоб одне значення було менше, від іншого, що робить предикат асиметричним.

SELECT FIRST. Name, SECOND.Name
FROM Authors FIRST, Authors SECOND
WHERE FIRST.Date_of_birth = SECOND. Date_of_birth
AND FIRST. Name < SECOND.Name

Для декартового добутку в SQL також допустимий синтаксис з ключовими словами CROSS JOIN (перехресне з'єднання):

SELECT *списокСтовпців* FROM Mytab Tl CROSS JOIN Mytab T2;

Розглянуті вирази працюють в повнофункціональних базах даних. У Microsoft Access для отримання декартового добутку можливе використання виразу (SELECT списокСтовпців FROM Tl, T2, ... Tn), тільки якщо всі таблиці в списку мають різні імена. Якщо потрібно декартовий добуток таблиці самої на себе, то у виразі (SELECT списокСтовпців FROM Mytab Tl, Mytab T2) Access автоматично додасть ключове слово AS перед кожним псевдонімом. Спроба використання ключових слів CROSS JOIN В Access призведе до помилки.

Об'єднання наборів записів (UNION). Нерідко потрібно об'єднати записи двох або більше таблиць зі схожими структурами в одну таблицю. Інакше кажучи, до набору записів, що повертається одним запитом, потрібно додати записи, що повертаються іншим запитом. Для цього є оператор UNION (об'єднання):

Запит1 UNION Запит2;

При цьому в результатній таблиці залишаються тільки записи, які відрізняються. Щоб зберегти в ній всі записи, після оператора UNION слід написати ключове слово ALL.

Оператор UNION можна застосовувати тільки до таблиць, які задовільнять наступним умовам сумісності:

- кількості стовпців поєднуваних таблиць повинні бути рівні;
- дані у відповідних стовпцях поєднуваних таблиць повинні мати сумісні типи.

При цьому, імена відповідних стовпців і їх розміри можуть бути різними. Важливо, щоб кількості стовпців були рівні, а їх типи були сумісні. Щоб об'єднати набори записів з несумісними за типом даних стовпцями, слід застосувати функцію перетворення типу даних CAST.

Наприклад, якщо у вас є таблиця видавців і таблиця покупців, то можна сформувати список усіх видавців і покупців, що мешкають уВолинськівй області. Можна спочатку написати оператор SQL для вибірки рядків із таблиці видавців, а потім за допомогою другого оператора SQL вибрати рядки з таблиці покупців. Після чого об'єднати обидва цих оператори в одну фразу мови SQL, скориставшись ключовим словом UNION, – і ви одержите єдиний результуючий набір, що містить результати обох запитів.

Такий опрератор SQL створює результуючий набір, що містить усіх видавців і покупців, що мешкають у Волинській області.

```
SELECT Name, City, Region, Zip FROM Publishers WHERE Region='Vol'
UNION
SELECT Name, City, Region, Zip FROM Buyers WHERE Region=' Vol';
```

При формуванні результуючого набору оператор UNION використовує імена стовпчиків першого запиту і при необхідності перетворить тип даних, обраних у відповідності з другим запитом.

Оператор UNION можна також використовувати для об'єднання запитів до однієї і тої ж таблиці даних. Такий оператор SQL повертає в одному результуючому наборі лідерів і аутсайдерів продажів:

```
SELECT SUM(Sales) AS TotSales, Title FROM BookSales
GROUP BY Title HAVING SUM(Sales)>4000
UNION
SELECT SUM(Sales) AS TotSales, Title FROM BookSales
GROUP BY Title HAVING SUM(Sales)<1000
ORDER BY TotSales;
```

Коли потрібно об'єднати записи двох таблиць, що мають одноіменні стовпці з сумісними типами, можна виконати оператор UNION CORRESPONDING (об'єднання відповідних):

```
SELECT * FROM Таблица1
UNION CORRESPONDING (списокСтовпців)
SELECT * FROM Таблиця 2;
```

Після ключового слова CORRESPONDING МОЖНА вказати стовпці, наявні одночасно в обох таблицях. Якщо цей список стовпців не зазначений, то передбачається список всіх імен.

Переріз наборів записів (INTERSECT). Переріз двох наборів записів здійснюється за допомогою оператора INTERSECT (переріз), що повертає таблицю, записи в якій містяться одночасно в двох наборах:

Запит1 INTERSECT Запит2;

При цьому в результатній таблиці залишаються тільки запису, які відрізняються.

Щоб зберегти у ній повторювані записи, після оператора INTERSECT слід написати ключове слово ALL.

Як і в операторі UNION, В INTERSECT можна використовувати ключове слово CORRESPONDING. У цьому випадку вихідні набори даних не обов'язково повинні бути сумісними для об'єднання, але відповідні стовпці повинні мати однакові тип і довжину.

Якщо в операторі INTERSECT використовується ключове слово CORRESPONDING, то після нього в круглих дужках можна вказати імена стовпців, значення яких повинні перевірятися на рівність при виконанні операції перетину.

Віднімання наборів записів (EXCEPT). Для отримання записів, що містяться в одному наборі і відсутні в іншому, служить оператор EXCEPT (за винятком):

Запит1 EXEPT Запит2;

За допомогою цього оператора з першого набору видаляються записи, що входять у другий набір. Так само, як і в операторах UNION і INTERSECT, в операторі EXCEPT можна використовувати ключове слово CORRESPONDING.

3.2. Операції з'єднання

Операції з'єднання наборів записів повертають таблиці, записи в яких виходять шляхом деякої комбінації записів з'єднуваних таблиць. Для цього використовується оператор JOIN (поєднати).

Існують кілька різновидів з'єднання, яким відповідають певні ключові слова, що додаються до слова JOIN. Так, наприклад, декартовий добуток є операцією перехресного з'єднання. У SQL-виразі для позначення цієї операції використовується оператор CROSS JOIN. Втім, декартовий добуток можна отримати і без використання цих ключових слів. В основі будь-якого з'єднання наборів записів лежить операція їх декартового добутку.

Природне з'єднання (NATURAL JOIN). Розглянемо суть операції природного з'єднання на типовому прикладі. Нехай нам потрібно створити результуючий набір, що містить назву книги й ім'я її видавця. Зрозуміло, що в отриманому декартовому добутку нас можуть цікавити не всі записи, а тільки ті, в яких ідентичні стовпці мають однакову значення. Така таблиця і буде природним з'єднанням таблиць.

Вона отримається за допомогою наступного запиту

SELECT Titles.Title, Publishers.Name FROM Publishers, Titles WHERE Publishers.PubID = Titles.PubID;

Даний запит можна переписати, використовуючи псевдоніми:

SELECT T1.Title, T2.Name FROM Titles T1, Publishers T2 WHERE T1.PubID = T2.PubID;

Еквівалентний запит з оператором NATURAL JOIN виглядає наступним чином:

SELECT T1.Title, T2.Name FROM Titles T1 NATURAL JOIN Publishers T2;

У Microsoft Access оператор NATURAL JOIN не підтримується. Замість нього використовується INNER JOIN (внутрішнє з'єднання) і ключове слово ON (при), за яким слідує умова відбору записів. SELECT T1.Title, T2.Name FROM Titles T1 INNER JOIN Publishers T2 ON T1.PubID = T2.PubID; INNER JOIN також можна застосовувати і в повнофункціональних базах даних.

При природному з'єднанні, коли використовується оператор NATURAL JOIN, перевіряється рівність всіх однойменних стовпців з'єднуваних таблиць.

Умовне з'єднання (JOIN. .. ON). Умовне з'єднання схоже на з'єднання з умовою рівності. Відмінність полягає в тому, що в якості умови може виступати будь який вираз, що записується після ключового слова ON (при), а не WHERE. Якщо умова виконується для поточного запису декартового добутку, то він входить у результатну таблицю.

Наприклад, створити результуючий набір, що містить назву книги й ім'я її видавця, який проживає в Lutsk.

SELECT T1.Title, T2.Name FROM Titles T1 JOIN Publishers T2 ON (T1. PubID = T2. PubID) AND (City='Lutsk');

У Microsoft Access використовується оператор INNER JOIN ... ON. У повнофункціональних базах даних також допустиме ключове слово INNER

З'єднання по іменах стовпців (JOIN. .. USING). З'єднання по іменах стовпців схоже на природне з'єднання. Відмінність полягає в тому, що можна вказати, які одноіменні стовпці повинні перевірятися. Нагадаємо, що в природньому з'єднанні перевіряються всі однойменні стовпці.

Припустимо, є дві таблиці Publishers та Buyers, які мають однойменні поля. Нехай нам потрібно вивести всі можливі комбінації прізвищ покупців і видавців, які проживають в одній і тій же області і в тому самому місті.

Для цього слід певним чином з'єднати таблиці Publishers і Buyers. Природне з'єднання в цьому випадку не підійде, оскільки при ньому перевіряються всі однойменні стовпці. У випадку, якщо жоден видавець не є покупцем, і ми використаємо операцію природного з'єднання, то в результаті буде отримана порожня таблиця. Тому слід використовувати з'єднання по іменах стовпців. Це з'єднання може бути представлено так:

SELECT Publishers.Lastname Buyers.Lastname FROM Buyers JOIN Publishers USING (Region, City);

Очевидно, даний запит можна сформулювати інакше:

SELECT Publishers.Lastname Buyers.Lastname FROM Buyers, Publishers WHERE Buyers.Region= Publishers. Region AND Buyers. City = Publishers.City;

У Microsoft Access оператор JOIN ... USING не підтримується. Замість нього можна використовувати INNER JOIN ... ON. Розглянений запит в Access можна сформулювати так:

SELECT Publishers.Lastname Buyers.Lastname FROM Buyers INNER JOIN Publishers ON Buyers.Region= Publishers. Region AND Buyers. City = Publishers.City;

Зовнішні з'єднання. Всі з'єднання таблиць, розглянуті дотепер, є внутрішніми. У всіх прикладах замість ключового слова JOIN можна писати INNER JOIN (внутрішнє з'єднання). З таблиці, одержаної при внутрішньому з'єднанні, відкидаються всі записи, для яких немає відповідних записів одночасно в обох таблицях. При зовнішньому з'єднанні такі невідповідні записи зберігаються.

За допомогою спеціальних ключових слів LEFT OUTER, RIGHT OUTER, FULL та UNION, написаних перед JOIN, можна виконати відповідно ліве, праве, повне поєднання і об'єднання-з'єднання. У SQL-виразі запиту таблиця, зазначена ліворуч від оператора JOIN, називається лівою, а зазначена праворуч від нього – правою.

Ліве з'єднання (LEFT OUTER JOIN). При лівому зовнішньому з'єднанні невідповідні записи, наявні в лівій таблиці, зберігаються в результатній таблиці, а наявні в правій – видаляються. Порожні поля мають значення NULL.

У Microsoft Access ключове слово OUTER не допускається. Пишуть LEFT JOIN.

Еквівалентний запит, який не використовує ключові слова LEFT OUTER JOIN, виглядає досить громіздко!

Праве з'єднання (RIGHT OUTER JOIN). При правому зовнішньому з'єднанні невідповідні записи, наявні в правій таблиці, зберігаються в результатній таблиці, а наявні в лівій – видаляються. Ключовими словами в такому запиті є RIGHT OUTER JOIN.

У Microsoft Access RIGHT JOIN.

Повне з'єднання (FULL JOIN). Повне з'єднання виконує одночасно і ліве, і праве зовнішні з'єднання. Ключовими словами в запиті є FULL JOIN.

У Microsoft Access оператор FULL JOIN не підтримується.

Об'єднання-з'єднання (UNION JOIN). Операцію об'єднання-з'єднання ще називають об'єднання-злиття. Ключовими словами в запиті є UNION JOIN. При об'єднанні-з'єднанні створюється віртуальна таблиця, що містить всі стовпці обох вихідних таблиць. При цьому стовпці з лівої вихідної таблиці містять всі записи лівої таблиці, а в тих же записах в

стовпцях з правої вихідної таблиці містяться значення NULL. Аналогічно, стовпці з правої вихідної таблиці містять всі записи правої таблиці, а ці ж записи в стовпцях з лівої таблиці містять NULL. Загальна кількість записів у віртуальній таблиці дорівнює сумі кількості записів, наявних в обох вихідних таблицях.

3.3. Підзапити

Підзапит – це SQL-вираз, що починається з оператора SELECT, який міститься в умові оператора WHERE АБО HAVING для іншого запиту. Таким чином, підзапит – це запит на вибірку даних, вкладений в інший запит. Зовнішній запит, що містить підзапит, якщо тільки він сам не є підзапитом, не обов'язково повинен починатися з оператора SELECT. У свою чергу, підзапит може містити інший підзапит і т. д. При цьому спочатку виконується підзапит, що має найглибший рівень вкладення, і т. д. Часто, але не завжди, зовнішній запит звертається до однієї таблиці, а підзапит – до іншої. На практиці саме цей випадок найбільш цікавий.

Прості підзапити. Прості підзапити характеризуються тим, що вони формально ніяк не пов'язані з їх зовнішніми запитами. Це дозволяє спочатку виконати підзапит, результат якого потім використовується для виконання зовнішнього запиту.

Розглядаючи прості підзапити, слід виділити три часткових випадки:

- підзапити, що повертають єдине значення;
- підзапити, що повертають список значень з одного стовпця таблиці;
- підзапити, що повертають набір записів.

Розглянемо ці окремі випадки більш детально.

Робота з єдиним значенням. Припустимо, з таблиці продажей BookSales потрібно вибрати дані про книги для яких кількість проданих більше середнього значення. Це можна зробити за допомогою наступного запиту:

SELECT * FROM BookSales WHERE Units> (SELECT AVG (Units) FROM BookSales);

У даному запиті спочатку виконується підзапит (SELECT AVG (Units) FROM BookSales). Він повертає єдине значення (а не набір записів) – середнє значення стовпця кількість проданих Units. Якщо сказати точніше, то даний підзапит повертає єдину запис, що містить єдине поле. Далі виконується зовнішній запит, який виводить всі стовпці таблиці клієнти і записи, в яких значення стовпця Units більше значення, отриманого за допомогою підзапиту. Таким чином, спочатку виконується підзапит, а потім зовнішній запит, що використовує результат підзапиту.

Спроба виконати наступний простий запит призведе до помилки:

SELECT * FROM BookSales WHERE Units> AVG (Units);

Вираз підзапиту обов'язково має бути укладений в круглі дужки.

Робота зі списком значень з одного стовпця. Підзапит, взагалі кажучи, може повертати декілька записів.Наприклад, нам потрібно вивести назви ких, які вже продавалися, тобто є відомості про продажі даних книг в таблиці BookSales.

SELECT Title FROM Titles WHERE ISBN IN (SELECT ISBN FROM BookSales);

Щоб в умові зовнішнього оператора WHERE можна було використовувати оператори порівняння, що вимагають одного значення, використовуються квантори, такі як ALL (наприклад …> ALL (підзапит) означає, що наше значення повинне бути більше всіх значень вибраних підзапитом) і SOME (наприклад …> SOME (підзапит) означає, що наше значення повинне бути більше хоча б одного значення з значень вибраних підзапитом).

Зрозуміло, вкладені підзапити можуть містити умови, зумовлені оператором WHERE.

Зрозуміло, це один з можливих варіантів запитів, який повертає необхідні дані. Робота з набором записів У повнофункціональних базах даних підзапити можна використовувати не тільки в операторах WHERE I HAVING, але і в операторі FROM:

SELECT T.стовпець1, T.стовпець2, ... , T.стовпець n FROM (SELECT ...) T WHERE ... ;

Тут таблиці, яка повертається підзапитом в операторі FROM, присвоюється псевдонім T, а зовнішній запит виділяє стовпці цієї таблиці і, можливо, записи відповідно до деякої умови, яка зазначена в операторі WHERE.

3.4. Пов’язані підзапити

Пов’язані (корельовані) підзапити з практичної точки зору найбільш цікаві. Однак для їх застосування важливо розуміти, що саме буде робити СКБД для надання відповіді. Для початку достатньо усвідомити, що при виконанні запитів, що містять пов’язані підзапити, немає такого чіткого поділу в часі виконання між підзапитом і запитом, як у випадку простих підзапитів. У випадку простих підзапитів спочатку виконується підзапит, а потім запит що його містить. У випадку пов’язаних підзапитів порядок виконання запиту зовсім інший, і його бажано розуміти, щоб уникнути непорозумінь. Основна ознака пов’язаного підзапиту полягає в тому, що він не може бути виконаний самостійно, поза всяким зв’язком з основним запитом. Формально ця ознака виявляється в виразі складного запиту наступним чином: підзапит посилається на таблицю, яка згадується в основному запиті. Отже, підзапит повинен бути виконаний в якомусь контексті з поточним станом виконання основного запиту.

Розглянемо деякий абстрактний і, в той же час, типовий запит, який містить по’язаний підзапит:

```
SELECT A FROM T1
WHERE B=
      (SELECT B FROM T2 WHERE C=T1.C)
```

Даний запит на вибірку даних (оскільки він починається з оператора SELECT) містить підзапит, сформульований в виразі, розміщеному в основному запиті після ключового слова WHERE.

Даний запит використовує дві таблиці: T1 і T2, в яких є стовпці з однаковими іменами B і C і однаковими типами. Підзапит, розташований у виразі після ключового слова WHERE основного запиту (SELECT B FROM T2 WHERE C=T1.C), звертається до цих же таблиць. Оскільки одна з таблиць (T1) зустрічається як в підзапиті, так і в зовнішньому запиті, то підзапит не можна виконати самостійно, без зв’язку із зовнішнім запитом. Тому виконання запиту в цілому відбувається наступним чином:

1. Спочатку виділяється перший запис з таблиці T1, зазначеної в операторі FROM зовнішнього запиту (весь запис таблиці T1, а не тільки значення стовпця A). Цей запис називається поточним. Значення стовпців для цього запису доступні і можуть бути використані в підзапиті.

2. Потім виконується підзапит, який повертає список значень стовпця B таблиці T2 для яких значення стовпця C дорівнює значенню стовпця C таблиці T1.

3. Запит, сформульований в розглянутому прикладі, передбачає, що його підзапит повертає єдине значення (список з єдиним елементом), оскільки в операторі WHERE використовується оператор порівняння (=). Ми будемо вважати, що в цьому прикладі підзапит повертає єдине значення. Тепер виконується оператор WHERE основного запиту. Якщо значення стовпця B поточному (виділеному) записі таблиці T1 дорівнює значенню, отриманому підзапитом, то цей запис виділяється зовнішнім запитом.

4. Оператор SELECT зовнішнього запиту виконує перевірку умови свого оператора WHERE. А саме він перевіряє, чи рівне поточне значення стовпця в таблиці T1 значенню, одержаному підзапитом. Якщо так, то значення стовпця A поточного запису таблиці T1 поміщається в результатну таблицю, в іншому випадку запис ігнорується. Потім відбувається перехід до наступного запису таблиці T1. Тепер для неї виконується підзапит. Аналогічним чином все описане відбувається для кожного запису таблиці T1.

4. СТВОРЕННЯ І МОДИФІКАЦІЯ ТАБЛИЦЬ ЗАСОБАМИ SQL

4.1. Створення таблиць

Створення таблиці виконується за допомогою оператора CREATE TABLE (створити таблицю) із заданням необхідних параметрів. При цьому створюється постійна таблиця. Для створення тимчасової таблиці використовується оператор CREATE TEMPORARY TABLE (створити тимчасову таблицю). Тимчасова таблиця, на відміну від постійної, існує лише протягом сеансу роботи з базою даних, в якому вона була створена. Однак тимчасова таблиця може бути доступна іншим користувачам, як і постійна таблиця. Зазвичай тимчасові таблиці створюються для представлення в них поточних підсумкових (звітних) даних, доступних декільком користувачам бази даних. Далі наведено синтаксис оператора CREATE:

```
CREATE [TEMPORARY] TABLE Ім'яТаблиці (
{Стовпець1 тип [(розмір)] [обмеження_стопця] [, ...]}
{[, CONSTRAINT обмеження_таблиці] [,. . . ]}
);
```

Тут квадратні та фігурні дужки, на відміну від круглих, не є елементами синтаксису. У квадратних дужках задані необов'язкові елементи, а в фігурних – елементи, які можуть повторюватися.

Для створення таблиці необхідно вказати її ім'я та визначити стовпці. Визначення стовпця включає його ім'я і тип. Якщо вказується довжина стовпця, то вона береться в круглі дужки після типу. Крім того, можна вказати обмеження для стовпця. Всі перераховані елементи визначення стовпця задаються один за одним через пробіл. Якщо створювана таблиця містить кілька стовпців, то їх визначення розділяються комами. Обмеження може бути визначено і для всієї таблиці, а не тільки для її стовпців. У цьому випадку використовується ключове слово CONSTRAINT (обмеження), після якого вказується саме обмеження.

Далі наведено запит на створення простої таблиці без обмежень:

```
CREATE TABLE Titles (
ISBN CHAR(20),
Title VARCHAR,
PubID INTEGER,
YearPub DATE,
Description VARCHAR,
Notes VARCHAR,
Subject VARCHAR,
Comments VARCHAR
);
```

Даний запит створює таблицю Titles, що містить вісім стовпців.

4.2. Обмеження для стовпців

При створенні таблиці, ми можемо встановлювати обмеження на дані, які вводяться в таблицю. Наприклад, на числовий стовпець можна накласти обмеження, яке полягає в тому, що число повинне знаходитися в деякому діапазоні. Інший приклад обмеження: значення стовпця не повинно бути невизначеним.

У табл. 8 наведені основні обмеження для стовпців. Однак існують і більш складні обмеження.

Таблиця 8. Обмеження для стовпців

Визначення	**Опис**
NOT NULL	Стовпець не може містити значення NULL, тобто значення цього стовпця повинні бути визначеними.
UNIQUE	Значення, введені в стовпці, мають відрізнятися від всіх інших значень у цьому стовпчику, тобто бути унікальними.
PRIMARY KEY	Стовпець є первинним ключем. У кожній таблиці тільки один стовпець може бути первинним ключем. Це означає, що він не може містити значення NULL, а введенні значення мають відрізнятися від всіх інших значень у цьому стовпчику. Таким чином, PRIMARY KEY є комбінацією NULL і UNIQUE.
DEFAULT(*значення)*	Встановлює значення за замовчуванням. Так, при додаванні нового запису стовпець з таким обмеженням автоматично отримає вказане значення.
CHECK*(умова)*	Дозволяє проводити перевірку умови при введенні даних. Значення буде збережено, якщо умова виконується, у протилежному випадку – ні.

Microsoft Access не підтримує ключові слова DEFAULT і CHECK у визначеннях обмежень.

Розглянемо, як приклад, створення таблиці Titles (ISBN, Title, PubID, YearPub, Description, Notes, Subject, Comments). ISBN (стрічкове значення) повинен однозначно ідентифікувати запис про студента, тобто мати визначені і унікальні значення. Таким чином, даний стовпець повинен бути первинним ключем. Від стовпців Title, PubID та YearPub вимагатимемо, щоб у них не було невизначених значень. Поле Comments по замовчуванню повинне бути “Без коментарів”. Тоді запит на створення такої таблиці буде мати вигляд:

```
CREATE TABLE Titles (
ISBN CHAR(20) PRIMARY KEY,
Title VARCHAR NOT NULL,
PubID INTEGER NOT NULL,
YearPub DATE,
```

```
Description VARCHAR,
Notes VARCHAR,
Subject VARCHAR,
Comments VARCHAR DEFAULT(‘Без коментарів’)
);
```

Припустимо, що даний запит виконаний. Таблиця буде створена. Тоді, запит на додавання порожнього запису в таблицю Titles викличе повідомлення про помилку і не буде виконаний. Це відбудеться тому, що в прожньому записі, всі стовпці містять значення NULL, а, отже, не виконуються обмеження для перших чотирьох стовпців. При спробі добавити перший запис в таблицю Titles із значеннями (‘978-5-699-79339-6’, ‘Sherlock Holmes: ’, 4), буде виконаний і не викличе проблем з боку СКБД.

Якщо слідом за даним запитом ми спробуємо додати ще один запис зі значеннями (978-5-699-79339-6, ‘Unknown adventures of Sherlock Holmes’, 6). Даний запит не буде виконаний через порушення обмеження, накладеного на перший стовпець: його значення повинні бути не тільки певними (відмінними від NULL), але й унікальними. Запис можна додати, якщо стовпцю ISBN привласнити інше значення.

В наступному прикладі подано створення таблиці Authors, в якої поле AUID цілочисельне і є первинним ключем, а значення в полі Contracted не повинні перевищувати значення 1000.

```
CREATE TABLE Authors (
AUID INTEGER PRIMARY KEY,
Surname VARCHAR NOT NULL,
Name VARCHAR,
Lastname VARCHAR,
Address VARCHAR,
Contracted Numeric CHECK (Contracted<=1000)
);
```

4.3. Обмеження для таблиць

Обмеження на дані, що вводяться можна призначити не тільки для окремих стовпців, але і для таблиці в цілому. Це зручно в тих випадках, коли необхідно для декількох стовпців призначити однакові обмеження. Крім того, якщо первинний ключ складений (тобто складається з декількох стовпців), то вказати його можна тільки як обмеження для таблиці, а не для стовпця. Всі обмеження для таблиці вказуються після визначень стовпців і складаються з ключового слова CONSTRAINT, за яким слідує вираз, що визначає саме обмеження. У табл. 9 наведені основні обмеження для таблиці.

Таблиця 9. Обмеження для таблиці

Визначення	**Опис**
UNIQUE(*Список Стовпців)*	Значення в стовпцях, зазначених у списку повинні бути унікальними.
PRIMARY KEY*(Список Стовпців)*	У кожній таблиці повинен бути тільки один первинний ключ, який визначений або як обмеження для стовпця, або як обмеження для таблиці. Якщо первинний ключ складний, то він повинен бути визначеним як обмеження для таблиці. Праворуч від ключових слів PRIMARY KEY в круглих дужках вказується список стовпців, що визначають складений первинний ключ.
CHECK*(умова)*	Дозволяє проводити перевірку умови при введенні даних. Значення буде збережено, якщо умова виконується, у протилежному випадку – ні. На відміну від обмеження для стовпця, тут можна використовувати умови, що оперують значеннями різних стовпців таблиці.
FOREIGN KEY … REEERENCES…	Обмеження типу “зовнішній ключ”

У наступному прикладі створюється Таблиця Authors (Surname, Name, Lastname, Address, Contracted). Передбачається, що комбінація значень перших трьох стовпців повинна однозначно ідентифікувати запис у таблиці, тобто бути первинним ключем.

```
CREATE TABLE Authors (
Surname VARCHAR NOT NULL,
Name VARCHAR, NOT NULL
Lastname VARCHAR, NOT NULL
Address VARCHAR,
Contracted Numeric,
CONSTRAINT PRIMARY KEY (Surname ,Name, Lastname)
);
```

При добавленні наступних значень в таблицю:

(‘Tretyak’, ‘Alina’, ‘Petrivna’) та (‘Tretyak’, ‘Alina’, ‘Sergiivna’)

дані будуть добавлені, оскільки комбінації значень стовпців, що визначають первинний ключ, не містять NULL і відрізняються один від одного.

4.4. Зовнішні ключі

Одна з найважливіших різновидів обмежень пов’язана з визначенням зовнішніх ключів. Зовнішній ключ – це стовпець або група стовпців,

відповідних первинному ключу іншої таблиці. Щоб зрозуміти синтаксис виразу, що визначає зовнішній ключ, розглянемо приклад.

Нехай в базі даних є дві таблиці (для прикладу візьмемо спрощені таблиці бази даних “Книжковий магазин”):

Titles (ISBN, Title, YearPub) – містить список назв книг;

BookSales (BookSalesID, Sales, ISBN, Units) відомості про продажі книг.

У таблиці Titles стовпець ISBN є первинним ключем, тобто його значення відмінні від NULL і унікальні. У таблиці BookSales стовпець ISBN не зобов’язаний мати унікальні значення, оскільки одна і та ж книга може продаватися в різний час і по різній ціні. Разом з тим будь-якому значенню стовпця BookSales.ISBN відповідає єдине значення стовпця Titles.ISBN. При описаних умовах стовпець ISBN таблиці BookSales є зовнішнім ключем, який посилається на первинний ключ ISBN таблиці Titles.

Зовнішній ключ визначається як обмеження для таблиці в виразі з ключовими словами CONSTRAIN FOREIGN KEY (обмеження “зовнішній ключ”):

CONSTRAINT FOREIGN KEY ЗовнішнійКлюч REFERENCES зовнішня Таблиця (ПервинніКлюч)

Тут ЗовнішнійКлюч – ім’я стовпця або список стовпців, розділених комами, які визначають зовнішній ключ, за ключовим словом REFERENCES (посилання) вказується зовнішня таблиця і її первинний ключ, на який посилається зовнішній ключ.

Для розглянутого раніше прикладу таблицю BookSales можна визначити наступним чином:

```
CREATE TABLE BookSales (
BookSalesID INTEGER PRIMARY KEY,
Sales NUMERIC,
ISBN CHAR(20),
Units NUMERIC,
CONSTRAINT FOREIGN KEY ISBN REFERENCES Titles (ISBN)
);
```

Використання зовнішніх ключів забезпечує збереження посилкової цілісності бази даних при зміні і видаленні записів. Якби таблиці Titles і BookSales не були пов’язані, то при видаленні записів з таблиці Titles у таблиці BookSales могли залишитися посилання на книгу, про яку вже немає відомостей. Цей факт зазвичай розцінюється як аномалія видалення. У випадку визначення в таблиці BookSales зовнішнього ключа ISBN, з таблиці Titles не вдасться видалити книгу, якщо ця книга продається в магазині. Якщо потрібно видалити з бази даних все, що стосується певної книги, то спочатку видаляються записи в таблиці BookSales, а потім – з таблиці Titles.

Аналогічна ситуація може статися і при оновленні даних. Наприклад, у таблиці Titles ви змінили ідентифікатор книги, яка має відомості в таблиці

BookSales і, таким чином, зовнішньому ключу тепер немає на що посилатися. Це аномалія зміни. В даному випадку необхідно спочатку додати новий запис в таблицю Titles, вказавши в ній необхідне значення ISBN, потім змінити в таблиці BookSales всі старі значення ISBN на ті, які ви тільки що ввели в новому записі таблиці Titles, а потім видалити з таблиці Titles запис зі старим ISBN.

Щоб в таблицях, пов'язаних зовнішнім ключем, не робити модифікацію даних в декілька етапів, у виразі CONSTRAIN FOREIGN KEY можна використовувати додаткові ключові слова:

- ON DELETE CASCADE | SET NULL (при видаленні каскадувати | встановити NULL);
- ON UPDATE CASCADE | SET NULL (при оновленні каскадувати | встановити NULL).

Тут вертикальна риска не є елементом синтаксису, а лише розділяє можливі варіанти ключових слів.

Так, при використанні ON DELETE CASCADE у випадку видалення запису зі значенням первинного ключа, яке є у зовнішньому ключі іншої таблиці, відповідні записи видаляються автоматично з двох таблиць.

Наприклад, при видаленні з таблиці Titles записи про книгу, що має дані про продажі, в таблиці BookSales також будуть видалені всі записи, що посилаються на дану книгу. Щоб дана стратегія виконувалася, таблиця BookSales повинна бути визначена таким чином:

```
CREATE TABLE BookSales (
BookSalesID INTEGER PRIMARY KEY,
Sales NUMERIC,
ISBN CHAR(20),
Units NUMERIC,
CONSTRAINT FOREIGN KEY ISBN REFERENCES Titles (ISBN) ON DELETE CASCADE);
```

Однак на практиці зазвичай стараються спочатку переконатися, що для книги немає відомостей про її продажі, і лише потім викреслити її з списку наявних книг.

Варіант SET NULL зазвичай використовується при оновленні даних. Наприклад:

```
CREATE TABLE BookSales (
BookSalesID INTEGER PRIMARY KEY,
Sales NUMERIC,
ISBN INTEGER,
Units NUMERIC,
CONSTRAINT FOREIGN KEY ISBN REFERENCES Titles (ISBN) ON UPDATE SET NULL);
```

В даному випадку при зміні (у тому числі і при видаленні) в таблиці Titles запису, на який посилається зовнішній ключ таблиці BookSales,

значення зовнішнього ключа встановлюються в NULL. Проте цей варіант не спрацює, якщо на стовпець замовлення, ISBN накладено обмеження NOT NULL. Зазвичай так і буває, оскільки при продажі книги, книга повинна бути в списку наявних книг в магазині. Тому, на всяк випадок, краще використовувати ключові слова ON UPDATE CASCADE.

Деякі СКБД (наприклад, PostgreSQL) допускають комбінування додаткових ключових слів. Наприклад:

ON UPDATE SET NULL ON DELETE CASCADE

4.1.Видалення таблиць

Видалити таблицю з бази даних можна наступним чином:

DROP TABLE Ім'яТаблиці;

Зрозуміло, що при видаленні таблиці втрачаються і всі дані що містяться в ній. Під час роботи з базою даних нерідко створюється таблиця для тимчасового зберігання даних, отриманих на якомусь проміжному етапі. Рано чи пізно такі таблиці підлягають видаленню. Однак можна забути це зробити. Крім того, проміжні таблиці, що створюються додатками, можуть залишитися в базі даних через збої. Тому для створення тимчасових таблиць краще використовувати оператор CREATE TEMPORARY TABLE, а не CREATE TABLE. Тимчасова таблиця, створена за допомогою CREATE TEMPORARY TABLE, автоматично знищується по закінченні сеансу роботи з базою даних.

4.2.Модифікація таблиць

При розробці бази даних рідко коли вдається відразу оптимально визначити структуру таблиць, що входять до неї. Крім того, вже готова база даних з часом може поповнюватися новими таблицями, які зв'язуються з уже наявними. А установка нових зв'язків вимагає корекції параметрів в старих таблицях. Модифікація таблиці – це зміна її структури, тобто додавання, видалення і перейменування стовпців, а також зміна їх типів і розмірів.

Мова SQL володіє спеціальними засобами модифікації таблиць.

Значно простіше змінити структуру таблиці за допомогою оператора ALTER TABLE (змінити таблицю). За допомогою додаткових ключових слів можна виконати наступні операції:

- ADD COLUMN – додати стовпець;
- DROP COLUMN – видалити стовпець;
- ALTER COLUMN – змінити тип, розмір і обмеження стовпця;
- RENAME COLUMN – перейменувати стовпець;
- RENAME TO – перейменувати таблицю.

Типовою задачею зміни структури таблиці є додавання стовпця. Це можна зробити за допомогою SQL-виразу з таким синтаксисом:

ALTER TABLE Ім'яТаблиці ADD COLUMN ім'яСтовпця тип (розмір)

Наприклад:

ALTER TABLE Authors ADD COLUMN Telephone CHAR (25);

Доданий стовпець виявляється останнім у таблиці, тобто займає крайню праву позицію. Іноді це незручно. Нехай, наприклад, стовпець Name в таблиці займає першу позицію, а раніше забутий і пізніше доданий стовпець Surname – сьому позицію. Щоб вибірка даних з цієї таблиці вивелася звичним чином, доводиться спеціально вказувати необхідне розташування стовпців в операторі SELECT, наприклад, SELECT Name, Surname FROM. . .

Вираз ADD COLUMN може використовуватися у виразі ALTER TABLE кілька разів – для кожного окремого стовпця. При визначенні параметрів доданого стовпця можна вказати обмеження для нього. Наступний SQL-вираз додає в таблицю Authors стовпець AUID і оголошує його первинним ключем (якщо при створенні первинний ключ не був вказаний):

ALTER TABLE Authors ADD COLUMN AUID INTEGER PRIMARY KEY;

Не слід забувати, що первинний ключ у таблиці може бути тільки один. Якщо в таблиці вже є первинний ключ, то додавання стовпця як первинного ключа не буде виконано. Якщо вам потрібно перепризначити первинний ключ, то спочатку необхідно скоригувати відповідним чином параметри вже наявного стовпця, оголошеного як первинний ключ, а потім додати новий стовпець з параметром PRIMARY KEY. Корекцію стовпця можна виконати так: або спочатку видалити його, а потім додати новий стовпець з необхідними параметрами або змінити параметри існуючого стовпця за допомогою оператора ALTER COLUMN.

Щоб видалити стовпець з таблиці, досить виконати SQL-вираз з наступним синтаксисом:

ALTER TABLE Ім’яТаблиці DROP COLUMN ім’яСтовпця;

При видаленні стовпця видаляються всі дані, що містяться в ньому. Якщо видаляється стовпець який є первинним ключем, на який посилається зовнішній ключ з іншої таблиці, то видалення не буде виконано. У цьому випадку вам доведеться спочатку скоригувати дані в іншій таблиці.

Наприклад, наступний вираз видаляє з таблиці Authors стовпець Contracted.

ALTER TABLE Authors DROP COLUMN Contracted;

Для зміни параметрів існуючого стовпця, таких як тип, розмір і обмеження, застосовуються ключові слова ALTER COLUMN:

ALTER TABLE Ім’яТаблиці ALTER COLUMN ім’яСтовпця тип (розмір) [Обмеження];

Тут квадратні дужки вказують, що укладений в них вираз не є обов’язковим.

У наступному прикладі в таблиці Authors вже існуючий стовпець Name набуває нові параметри. А саме, будучи символьним, він отримує збільшення довжини до 50 символів і стає первинним ключем:

ALTER TABLE Authors ALTER COLUMN Name CHAR (50) PRIMARY KEY;

Зрозуміло, якщо в таблиці вже є первинний ключ (наприклад, стовпець AUID), то зробити первинним ключем ще один стовпець не вдасться. Крім того, не слід забувати, що при перетворенні типів можуть бути втрачені дані. Так, якщо ви перетворите стовпець символьного типу в числовий тип, то всі дані будуть втрачені. При зменшенні розміру символьного стовпця його значення можуть виявитися обрізаними праворуч. Таким чином, слід дуже уважно змінювати параметри стовпців, що містять деякі дані.

Перейменувати стовпці і таблицю можна за допомогою оператора ALTER TABLE З ключовими словами RENAME COLUMN ... TO (перейменувати стовпець ... в).

Для перейменування стовпця використовується наступний синтаксис:

ALTER TABLE Ім'яТаблиці RENAME COLUMN ім'яСтовпця TO нове_ім'яСтовпця;

При перейменуванні стовпця його тип, розмір і обмеження зберігаються.

Для перейменування таблиці використовується схожий синтаксис:

ALTER TABLE Ім'яТаблиці RENAME TO нове_Ім'яТаблиці;

Зміна структури таблиці є небезпечною операцією з точки зору можливості втрати даних. Якщо ви не цілком упевнені, що отримаєте потрібний результат, то краще використовувати резервне копіювання даних, наступним чином:

1. Створіть тимчасову таблицю за допомогою оператора CREATE TEMPORARY TABLE.
2. Скопіюйте в тимчасову таблицю дані з таблиці, структуру якої ви збираєтеся модифікувати.
3. Далі можна зробити двома способами:
   - змінити структуру вихідної таблиці так, як вам треба. Цей спосіб краще застосовувати при корекції структури, не пов'язаної зі змінами існуючих обмежень на стовпці і / або на таблицю в цілому;
   - видалити вихідну таблицю і створити нову під тим же ім'ям і з необхідною структурою.
4. Скопіюйте дані з тимчасової таблиці в таблицю з новою структурою, використовуючи оператор INSERT INTO. Більше тимчасова таблиця не потрібна, вона буде автоматично видалена після закінчення сеансу роботи з базою даних.

5. МАНІПУЛЮВАННЯ ДАНИМИ В SQL

У SQL для виконання операцій введення даних в таблицю, їх зміни і видалення призначені три команди мови маніпулювання даними (DML). Це команди – INSERT (вставити), UPDATE (оновити), DELETE (видалити).

5.1. Добавлення даних в таблицю бази даних

Команда INSERT здійснює вставку в таблицю нового рядка. У найпростішому випадку вона має наступний вигляд:

INSERT INTO <ім'я таблиці> VALUES (<значення>, <значення>, ...);

При такому записі зазначені в дужках після ключового слова VALUES значення вводяться в поля доданого в таблицю нового рядка в тому порядку, в якому відповідні стовпці вказані при створенні таблиці.

Наприклад, введення нового рядка в таблицю Authors може бути здійснений таким чином

INSERT INTO Authors VALUES (101, 'Semenyuk', 'Oleksandr', 'Oleksandrovich', 'Kyiv', 15);

Щоб така команда могла бути виконана, таблиця з вказаними в ній ім'ям (Authors) повинна бути попередньо визначена (створена) командою CREATE TABLE. Якщо в будь-поле необхідно вставити NULL-значення, то воно вводиться як звичайне значення:

INSERT INTO Authors VALUES (101, 'Semenyuk', 'Oleksandr', NULL, 'Kyiv', 15);

У випадках, коли необхідно ввести значення полів в порядку, відмінному від порядку стовпців, заданого командою CREATE TABLE, або якщо потрібно ввести значення не у всі стовпці, то слід використовувати наступну форму команди INSERT:

INSERT INTO Authors (AUID, Address, SURNAME, NAME) VALUES (101, 'Kyiv', 'Semenyuk', 'Oleksandr');

Стовпцям, найменування яких не зазначені в наведеному в дужках списку, автоматично присвоюється значення за замовчуванням, якщо воно призначене при описі таблиці (команда CREATE TABLE), або значення NULL.

За допомогою команди INSERT можна витягти значення з однієї таблиці і розмістити його в іншій. Наприклад:

INSERT INTO Authors1 SELECT * FROM Authors WHERE Address = 'Kyiv';

При цьому таблиця Authors1 повинна бути попередньо створена командою CREATE TABLE і мати структуру, ідентичну таблиці Authors.

5.2. Видалення даних з таблиці БД

Команда DELETE видаляє рядки з таблиці.

Наступний вираз видаляє всі рядки таблиці Authors1.

DELETE FROM Authors1;

В результаті таблиця стає порожньою.

Для видалення з таблиці відразу декількох рядків, що задовільняють деякій умові можна скористатися пропозицією WHERE, наприклад,

DELETE FROM Authors1 WHERE AUID = 103;

Можна видалити групу рядків

DELETE FROM Authors1 WHERE Address = ‘Kyiv’;

5.3. Оновлення даних в таблиці БД

Команда UPDATE дозволяє змінювати, тобто оновлювати, значення деяких або всіх полів в існуючому рядку або рядках таблиці. Наприклад, щоб для всіх продажей, відомості про яких містяться в таблиці BookSales, змінити значення поля Sales на 200, можна використовувати конструкцію:

UPDATE BookSales SET Sales = 200;

Для задання конкретних рядків таблиці, значення полів яких повинні бути змінені, в команді UPDATE можна використовувати предикат, що вказується в пропозиції WHERE.

UPDATE BookSales SET Sales = 200 WHERE ISBN =’978-5-699-79339-6’;

В результаті виконання цього запиту буде змінено поле Sales тільки для книг з ISBN =’978-5-699-79339-6’.

Команда UPDATE дозволяє змінювати не тільки один, але й декілька стовпців:

UPDATE BookSales SET Sales = 200, Units=1000 WHERE ISBN = ‘978-5-699-79339-6’;

У пропозиції SET команди UPDATE можна використовувати скалярні вирази, що вказують спосіб зміни значень поля, в які можуть входити значення змінюваного та інших полів.

Наприклад, для збільшення в таблиці BookSales значення поля Sales в два рази для книг з ISBN = ‘978-5-699-79339-6’ можна використовувати запит:

UPDATE BookSales SET Sales = Sales*2 WHERE ISBN=’978-5-699-79339-6’;

Пропозиція SET не є предикатом, тому в ньому можна вказати значення NULL наступним чином:

UPDATE BookSales SET Sales = NULL WHERE ISBN=’978-5-699-79339-6’;

6. ПРЕДСТАВЛЕННЯ в SQL

6.1. Представлення – іменовані запити

База даних складається з таблиць – об'єктів, що містять дані й існують в довготривалій пам'яті комп'ютера, наприклад, на жорсткому диску. Це можуть бути окремі файли або частини одного файлу (як, наприклад в Microsoft Access). За допомогою запитів на вибірку (SQL-виразів, що починаються з ключового слова SELECT) створюються віртуальні таблиці, які є тимчасовими і доступними тільки тому, хто виконав цей запит.

Представлення – це теж віртуальні таблиці, але вони можуть бути доступні багатьом користувачам і існують в базі даних до тих пір, поки не будуть примусово видалені. Вони у всьому схожі на звичайні таблиці бази даних, за винятком того, що не є фізичними об'єктами зберігання даних.

Таблиці-представлення не містять ніяких власних даних. Фактично представлення – це іменована таблиця, вміст якої є результатом запиту, заданого при описі представлення. Причому даний запит виконується всякий раз, коли таблиця-представлення стає об'єктом команди SQL. Ці запити або параметри представлення зберігаються в розділі метаданих бази. Працювати з представленням можна як із звичайною таблицею. Однак будь-який запит до представлення в дійсності ініціює прихований запит, який комбінується з користувацьким.

Представлення дозволяють:

- обмежувати число стовпців, з яких користувач вибирає або в які вводить дані;
- обмежувати число рядків, з яких користувач вибирає або в які вводить дані;
- виводити додаткові стовпці, перетворені з інших стовпців базової таблиці;
- виводити групи рядків таблиці.

Завдяки цьому представлення дають можливість гнучкого налагодження виведеної з таблиць інформації відповідно до вимог конкретних користувачів, дозволяють забезпечувати захист інформації на рівні рядків і стовпців, спрощують формування складних звітів і вихідних форм.

Представлення визначається за допомогою команди CREATE VIEW (СТВОРИТИ ПРЕДСТАВЛЕННЯ). Наприклад:

CREATE VIEW KYIV_Buyers AS SELECT * FROM Buyers WHERE CITY = 'Kyiv';

Дані з базової таблиці, пропоновані користувачеві у представленні, залежать від умови (предиката), описаного в SELECT-запиті при визначенні представлення.

У створену в результаті наведеного вище запиту таблицю-представлення KYIV_Buyers передаються дані з базової таблиці Buyers, але

не всі, а тільки записи про замовників, для яких значення поля CITY рівне ‘Kyiv’. До таблиці KYIV_Buyers можна тепер звертатися за допомогою запитів так само, як і до будь-якої іншої таблиці бази даних. Наприклад, запит для перегляду представлення KYIV_Buyers має вигляд:

SELECT * FROM KYIV_Buyers;

6.2. Модифікація представлень

Дані, що подаються користувачеві через представлення, можуть змінюватися за допомогою команд модифікації DML, але при цьому фактична модифікація даних буде здійснюватися не в самій віртуальній таблиці-представлення, а буде перенаправлена до відповідної базової таблиці. Наприклад, запит на оновлення представлення NEW_Buyers

UPDATE NEW_Buyers SET CITY = ‘Kyiv’ WHERE BuyerID = 1004;

еквівалентний виконанню команди UPDATE над базовою таблицею Buyers.

Використання команд модифікації мови SQL – INSERT (ВСТАВИТИ), UPDATE (ОНОВИТИ), і DELETE (ВИДАЛИТИ) для представлень має ряд особливостей. Не всі представлення можуть модифікуватися.

Якщо команди модифікації можуть виконуватися в представленні, то представлення є оновлюваним (модифікується), в іншому випадку воно призначене тільки для читання при запиті. Яким чином можна визначити, чи представлення модифікується? Критерії оновлюваності представлення можна сформулювати наступним чином:

- представлення будується на основі однієї і тільки однієї базової таблиці;
- представлення має містити первинний ключ базової таблиці;
- представлення не повинно мати жодних полів, які являють собою агрегатні функції;
- представлення не повинно містити DISTINCT в своєму визначенні;
- представлення не повинно використовувати GROUP BY або HAVING у своєму визначенні;
- представлення не повинно використовувати підзапити;
- представлення може бути використано в іншому представленні, але це представлення повинне бути таким, що модифікується;
- представлення не повинно використовувати в якості полів виводу константи або вирази значень.

6.3. Маскуючі представлення

Представлення, що маскують стовпці. Даний вид представлень обмежує число стовпців базової таблиці, до яких можливий доступ. Наприклад:

CREATE VIEW NEW_Buyers AS SELECT BuyerID, NAME, CITY FROM Buyers; дає доступ користувачеві до полів BuyerID, NAME, CITY базової таблиці Buyers, повністю приховуючи від нього як вміст, так і сам

факт наявності в базовій таблиці полів CompanyName, Address, City, Region, Zip Phone Fax BookSalesID.

Оператор INSERT, застосований до такого представлення, фактично здійснює вставку рядка у відповідну базову таблицю, причому у всі стовпці цієї таблиці незалежно від того, видно їх користувачеві через представлення чи вони приховані від нього. У зв'язку з цим, у стовпцях, не включених до представлення, встановлюється NULL-значення або значення за замовчуванням. Якщо не включений до представлення стовпець має обмеження NOT NULL, то генерується повідомлення про помилку.

Будь-яке застосування оператора DELETE видаляє рядки базової таблиці

Представлення, що маскують рядки. Представлення можуть також обмежувати доступ до рядків. Охоплювані представленням рядки базової таблиці задаються умовою (предикатом) в конструкції WHERE при описі представлення. Доступ через представлення можливий тільки до рядків, що задовольняють умові.

Наприклад:

CREATE VIEW KYIV_Buyers AS SELECT * FROM Buyers WHERE CITY = 'Kyiv';

показує користувачеві тільки ті рядки таблиці Buyers, для яких значення поля CITY рівне 'Kyiv'.

Кожен включений в представлення рядок доступний для виведення, оновлення і видалення. Будь-який допустимий для основної таблиці рядок вставляється в базову таблицю незалежно від її включення в представлення. При цьому може виникнути проблема, яка полягає в тому, що значення, введені користувачем в базову таблицю через представлення, будуть відсутні в представленні, залишаючись при цьому в базовій таблиці. Розглянемо такий випадок:

CREATE VIEW KYIV_Buyers AS SELECT * FROM Buyers WHERE CITY = 'Kyiv';

Це представлення є оновлюваним. Воно просто обмежує доступ користувача до певних рядків у таблиці Buyers. Припустимо, необхідно вставити за допомогою команди INSERT наступний рядок:

INSERT INTO KYIV_Buyers VALUES (180, 'Sobchuk' 'Lesya Ukrainka Eastern European National University', 'Potapova/9, 'Lutsk', 'Vol', 43000, 0332777777, 0332777777, 23);

Команда INSERT допустима в цьому поданні. За допомогою представлення KYIV_Buyers рядок буде добавлений в базову таблицю Buyers. Однак, після появи цього рядка в базовій таблиці, з самого представлення він зникне, оскільки значення поля CITY не дорівнює 'Kyiv', і, отже, цей рядок не задовольняє умові пропозиції WHERE для відбору рядків в представлення. Для користувача таке зникнення тільки що введеного рядка є несподіваним. Дійсно, не зрозуміло, чому після введення рядка в

таблицю її не можна побачити і, наприклад, тут же видалити. Тим більше, що користувач взагалі може не знати – працює він в даний момент з базовою таблицею чи з таблицею-представленням.

Аналогічна ситуація виникне, якщо в якомусь записі представлення KYIV_Buyers змінити значення поля CITY на значення, відмінне від ‘Kyiv’.

Подібні проблеми можна усунути шляхом включення у визначення представлення опції WITH CHECK OPTION. Ця опція поширює умову WHERE для запиту на операції оновлення і вставки в опис представлення. Наприклад:

CREATE VIEW KYIV_Buyers AS SELECT * FROM Buyers WHERE CITY = ‘Kyiv’ WITH CHECK OPTION;

У цьому випадку вищезазначені операції вставки рядка або корекції поля CITY буде відхилено. Опція WITH CHECK OPTION поміщається у визначення представлення, а не в команду DML, так що всі команди модифікації в представленні будуть перевірятися. Рекомендується використовувати цю опцію у всіх випадках, коли немає причини дозволяти представленням поміщати в таблицю значення, які в ньому самому не можуть бути видимі.

Операції модифікації в представленнях, що маскують рядки і стовпці. Розглянута вище проблема виникає і при вставці рядків у представлення з предикатом, що використовує поля базової таблиці, не присутні в самому представленні. Наприклад, розглянемо представлення

CREATE VIEW KYIV_Buyers AS SELECT BuyerID, Name FROM Buyers WHERE CITY = ‘Kyiv’;

Видно, що в дане представлення не включено поле CITY таблиці Buyers.

Що буде відбуватися при спробі вставки рядка в це представлення? Так як ми не можемо вказати значення CITY в представленні як значення за замовчуванням (зважаючи на відсутність в ньому цього поля), то цим значенням буде NULL, і воно буде введено в поле CITY базової таблиці Buyers (вважаємо, що для цього поля опція NOT NULL НЕ використовується). Так як в цьому випадку значення поля CITY базової таблиці Buyers не буде дорівнювати значенню ‘Kyiv’, то рядок що вставляється буде виключений з самого представлення і, тому, не буде видимий користувачеві. Причому так буде відбуватися для будь-яких рядків, що вводяться в представлення KYIV_Buyers. Іншими словами, користувач взагалі не зможе бачити рядки, що вводяться ним у це представлення. Дана проблема не вирішується і тоді, якщо в визначення представлення буде додана опція WITH CHECK OPTION:

CREATE VIEW KYIV_Buyers AS SELECT BuyerID, Name FROM Buyers WHERE CITY = ‘Kyiv’ WITH CHECK OPTION;

Таким чином, в даному представленні, можна модифікувати значення полів або видаляти рядки, але не можна вставляти рядки.

6.4. Агреговані представлення

Створення представлень з використанням агрегованих функцій і пропозиції GROUP BY є зручним інструментом для безперервної обробки та інтерпретації інформації. Припустимо, необхідно стежити за середньою вартістю проданих книг та їх загальною кількістю. Для цього можна сформувати таке представлення

CREATE VIEW TOTALSales AS

SELECT BookSalesID, ISBN, AVG (Sales) AS Sales_AVG, SUM (Units) AS Units_SUM FROM BookSales GROUP BY ISBN;

Тепер необхідну інформацію можна побачити за допомогою простого запиту до представлення:

SELECT * FROM TOTALSales;

6.5. Представлення, засновані на кількох таблицях

Рредставлення часто використовуються для об’єднання декількох таблиць (базових і / або інших представлень) в одну велику віртуальну таблицю. Таке рішення має ряд переваг:

- представлення, що поєднує кілька таблиць, може використовуватися як проміжний макет при формуванні складних звітів, що приховує деталі об’єднання великої кількості вихідних таблиць;
- попередньо об’єднані пошукові та базові таблиці забезпечують найкращі умови для транзакцій, дозволяють використовувати компактні схеми кодів, усуваючи необхідність написання для кожного звіту довгих об’єднуючих процедур;
- дозволяє використовувати при формуванні звітів більш надійний модульний підхід;
- попередньо об’єднані і перевірені представлення зменшують імовірність помилок, пов’язаних з неповним виконанням умов об’єднання.

6.6. Обмеження застосування оператора SELECT для створення представлень

Є деякі види запитів, не допустимі у визначеннях представлень. Одиночне представлення повинне грунтуватися на одиночному запиті, тому UNION та UNION ALL в представленнях не дозволяються. Пропозиція ORDER BY також ніколи не використовується у визначенні представлень. Представлення є реляційною таблицею-відношенням, тому його рядки за визначенням є неупорядкованими.

6.7. Видалення представлень

Синтаксис видалення представлення з бази даних подібний до синтаксису видалення базових таблиць:

DROP VIEW <ім’я представлення>

7. ТРАНЗАКЦІЇ В SQL

7.1. Поняття транзакції

При роботі з базою даних завжди слід мати на увазі, що існує хоча і мала, але відмінна від нуля вірогідність пошкодити або втратити дані. Ця ймовірність тим більша, чим більша база даних і складніші запити до неї. В кожній СКБД є спеціальні засоби захисту даних, проте в деяких випадках можна використовувати додаткові заходи мови SQL.

Припустимо, що потрібно перевести гроші з одного рахунку на інший. Для цього необхідно виконати декілька операторів SQL. Якщо в ході їх виконання станеться який-небудь апаратний або програмний збій, через який не виконається один або кілька операторів, то може статися так, що гроші будуть зняті з одного рахунку, але не надійдуть на інший. Щоб цього не сталося, кілька SQL-операторів визнаються як єдиний блок, який називається транзакцією (від англ. Transaction – справа, угода). У транзакції або всі оператори виконуються успішно, або жоден з них не виконується, а база даних повертається в початковий стан, в якому вона перебувала до початку виконання операторів транзакції. Іншими словами, якщо якийсь оператор в транзакції з якоїсь причини не виконався, то відміняється дія всіх вже виконаних операторів в цій транзакції – відбувається так званий відкат.

Неприємності можуть виникнути і при роботі з базою даних в багатокористувацькому режимі, навіть в умовах абсолютної надійності обладнання, СКБД і програм додатка. Якщо кілька користувачів намагаються одночасно використовувати одну й ту ж таблицю, то може виникнути так звана колізія одночасного доступу. Для того, щоб виключити небажану взаємодію між користувачами, при виконанні запитів до бази даних робляться спеціальні заходи. Запити до бази даних, що складаються з декількох SQL-операторів, беруться у транзакцію, а для різних транзакцій визначаються так звані рівні ізоляції.

Транзакція складається з будь-яких операторів SQL, які можуть змінити базу даних. У SQL: 2003 не передбачений спеціальний оператор початку транзакції. Однак деякі реалізації SQL вимагають наявності такого оператора. Він може виглядати по-різному, наприклад, BEGIN WORK, BEGIN TRAN АБО BEGIN. У системах, сумісних з SQL: 2003, якщо ніяка транзакція ще не розпочата, а на виконання відправляється оператор SQL, то автоматично створюється нова транзакція. Наприклад, оператори CREATE TABLE (створити таблицю), UPDATE (оновити) I SELECT (Вибрати) виконуються тільки в транзакції.

Отже, оператора початку транзакції може і не бути. Однак для її завершення передбачені два оператори:

- COMMIT – завершити виконання. Виконання всіх операторів транзакції відбувається послідовно в єдиному блоці;

- ROLLBACK – відкат. Відбувається скасування дії всіх операторів транзакції, і база даних повертається в початковий стан, в якому вона перебувала до початку транзакції.

Оператор COMMIT застосовується тоді, коли всі SQL-оператори транзакції імовірно виконані (сприйняті СКБД і не викликали помилок) і потрібно підтвердити зміни, внесені в базу даних. Якщо при виконанні оператора COMMIT станеться системний збій або помилка, можна виконати відкат транзакції, а потім спробувати виконати її знову.

Оператор ROLLBACK застосовується тоді, коли необхідно відмінити зміни, внесені транзакцією, і відновити базу даних в попередній стан. Якщо при виконанні оператора ROLLBACK станеться системний збій, то після перезавантаження оператор ROLLBACK можна виконати знову, і він повинен відновити базу даних.

Додаток може складатися з декількох транзакцій. Якщо не зазначений явний оператор початку транзакції (наприклад, BEGIN WORK), то перша транзакція починається на самому початку додатка. Остання транзакція закінчується в кінці додатка. Для вказівки кінця транзакції використовується оператор COMMIT або ROLLBACK.

7.2. Визначення параметрів транзакції

Для транзакції можна встановити параметри – режим і рівень ізоляції. З цією метою використовується оператор:

SET TRANSACTION Режим, ISOLATION LEVEL Рівень_Ізоляції;

Режим може приймати наступні два значення:

- READ WRITE (читання-запис) – значення за замовчуванням;
- READ ONLY (тільки читання).

Рівень ізоляції транзакції може приймати чотири значення:

- SERIALIZABLE (послідовне виконання) – значення за замовчуванням;
- REPEATABLE READ (повторне читання);
- READ COMMITED (підтверджене читання);
- READ UNCOMMITED (непідтверджене читання).

Транзакція має параметри за замовчуванням: READ WRITE і SERIALIZABLE. Режим READ WRITE дозволяє вносити зміни в базу даних, а рівень ізоляції SERIALIZABLE є найбільш безпечним. Якщо ж необхідно задати інші значення, то слід використовувати оператор SET TRANSACTION. При цьому він записується або на початку додатка, визначаючи першу транзакцію, або після оператора COMMIT або ROLLBACK, визначаючи наступну транзакцію. Якщо ж після COMMIT або ROLLBACK не викликати оператор SET TRANSACTION, то наступна транзакція буде мати параметри, прийняті за замовчуванням.

Наприклад:

SET TRANSACTION READ ONLY, ISOLATION LEVEL READ COMMITED;

7.3. Рівні ізоляції транзакцій

Рівень ізоляції транзакції в багатокористувацькій системі визначається через відсутність певної аномалії доступу до бази даних, яка може в кінцевому результаті загрожувати цілісності даних. Існують класичні аномалії при паралельному виконанні транзакцій.

Втрачені зміни. Транзакція T1 читає дані. Транзакція T2 читає ті самі дані. T1 на основі прочитаного значення обчислює нове значення даних записує його в базу даних і завершується. T2 на основі прочитаного значення обчислює нове значення даних записує його в базу даних і завершується. В результаті значення, записане транзакцією T2, "перезапише" значення записане транзакцією T1.

Брудне читання. Транзакція T1 змінює деякі дані, але ще не завершується. Транзакція T2 читає ті ж дані (з змінами внесеними T1) і приймає на їх основі якесь рішення. Транзакція T1 виконує відкат. В результаті рішення, прийняте транзакцією T2 основане на неправильних даних.

Неповторне читання. Транзакція T1 під час свого виконання декілька раз читає ті самі дані. Транзакція T2 в проміжках між читаннями транзакцією T1 змінює ці дані і завершується. В результаті, виявляється, що читання одних і тих же даних у T1 дає різні результати.

Фіктивне читання. Транзакція T1 під час виконання кілька разів вибирає множину рядків за одними і тими ж критеріями. Транзакція T2 в інтервалах між вибірками транзакції T1 додає або видаляє рядки або змінює стовпці деяких рядків, які використовуються в критерії відбору і фіксується. У результаті виходить, що одні і ті ж вибірки транзакції T1 вибирають різну кількість рядків.

Стандартний SQL/92 визначає рівень ізоляції транзакції в багатокористувацькій системі установка яких запобігає певним конфліктним ситуаціям (табл. 10.). Введені наступні чотири рівні ізоляції.

Непідтверджене читання. Найнижчий рівень ізоляції – READ UNCOMMITED (непідтверджене читання або читання непідтверджених даних). При даному рівні зміни, внесені одним користувачем, можуть бути прочитані іншим, навіть якщо перший користувач ще не підтвердив їх за допомогою оператора COMMIT. Проблема виникне у випадку, якщо перший користувач виконає відкат своєї транзакції. Тоді всі наступні дії другого користувача виявляються заснованими на неправильних даних.

Таким чином, рівень READ UNCOMMITED не слід використовувати, коли потрібні точні результати. Він більш-менш підходить в тому випадку, коли достатньо отримати приблизні результати, наприклад, при статистичній обробці мало змінних даних.

Очевидно, що при рівні ізоляції READ UNCOMMITED обидві транзакції можуть виконуватися одночасно.

Підтверджене читання. Наступним по силі є рівень ізоляції READ COMMITED (підтверджене читання або читання підтверджених даних). У цьому випадку зміни, зроблені іншою транзакцією, залишаються невидимими у вашій транзакції до тих пір, поки інший користувач не завершить її оператором COMMIT. Хоча до завершення іншої транзакції внесені нею зміни вам не видно, обидві транзакції можуть виконуватися одночасно.

У порівнянні з READ UNCOMMITED даний рівень ізоляції дає кращі результати, але не звільняє від неприємностей так званого неповторного читання. Розглянемо наступну ситуацію.

Припустимо, перший користувач збирається спочатку прочитати дані, а потім використовувати їх при внесенні змін в іншу таблицю. Наприклад, він хоче спочатку дізнатися, скільки є деякого товару на складі, а потім оформити замовлення на цей товар. Може статися так, що цей користувач прочитає застарілі дані, які вже змінені другим користувачем, але ще не підтверджені ним за допомогою оператора COMMIT. Наприклад, другий користувач вже "забрав" весь запас даного товару, а перший користувач, не знаючи про це, оформив замовлення на відсутній товар. Таким чином, проблема полягає в тому, що результати читання даних першим користувачем до і після виконання другим користувачем оператора COMMIT можуть відрізнятися. Це так звана проблема неповторного читання.

Повторюване читання. Рівень ізоляції REPATABLE READ. Транзакція з цим рівнем ізоляції може читати тільки ті записи, які були змінені і вже зафіксовані іншою транзакцією, і ніяка інша транзакція не може змінити записи, які були прочитані в межах цієї транзакції. Цей рівень ізоляції виключає проблему неповторного читання. Тим не менш, цей рівень не усуває іншу неприємність, так зване фіктивне читання. Вона виникає тоді, коли користувач читає дані, що змінюються в результаті виконання іншої транзакції. При цьому зміни даних відбуваються під час їх читання. Розглянемо як приклад наступну ситуацію.

Припустимо, перший користувач виконує оператор вибірки даних (SELECT) З деякою умовою (WHERE або HAVING). В результаті він вибирає якусь множину записів. Другий користувач відразу ж за цим змінює дані в одному або декількох записах, які вибрав перший користувач. Більш того, другий користувач завершує свою транзакцію оператором COMMIT. Таким чином, виходить, що записи, які раніше задовільняли умові вибірки, тепер перестали їй відповідати. Не виключено, що з'явилися нові записи, які раніше не задовільняли умові, а тепер відповідні їй. У цей час перший користувач, ще не завершивши свою транзакцію, не знає про зміни, що відбулися і відправляє на виконання ще один SQL-оператор з такою ж умовою вибірки, що і на початку. Однак цей оператор буде працювати не з тими самими записами, що попередній оператор. Дана проблема і є проблемою фіктивного читання.

Послідовне виконання. Уникнути неприємностей, пов'язаних з раніше розглянутими рівнями ізоляції, дозволяє рівень SERIALIZABLE (послідовне виконання). Транзакції з рівнем ізоляції SERIALIZABLE завжди виконуються не паралельно, а послідовно. Якщо одна з них вже розпочата, то інша буде чекати її закінчення. Апаратні та програмні відмови можуть призвести до невиконання транзакції, однак при даному рівні ізоляції можна бути впевненим, що результати роботи з базою даних будуть коректними.

Надаючи найбільшу надійність, рівень ізоляції SERIALIZABLE максимально знижує загальну продуктивність системи.

Таблиця. 10. Визначення рівнів ізоляції

Рівні ізоляції SQL/92	**АНОМАЛІЇ**				**Oracle**
	Втрачені зміни	Грязне читання	Неповторне читання	фіктивне читання	
READ UNCOMMITTED	ні	так	так	так	–
READ COMMITTED	ні	так	так	так	READ COMMITTED
REPEATABLE READ	ні	ні	ні	так	–
SERIALIZABLE	ні	ні	ні	ні	SERIALIZABLE

Наведемо приклади сценаріїв перевірки небажаних ситуацій на прикладі таблиці EXAMPLE, структура і вміст якої наведені в табл. 11. В табл. 12. наведено сценарії самих перевірок.

Таблиця. 11. Таблиця EXAMPLE

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Таблиця 12. Сценарії перевірок

Крок	**Транзакція T1**	**Транзакція T2**
1. Втрачені зміни		
1	UPDATE example SET dat=101 WHERE id=1	
2		UPDATE example SET dat=102 WHERE id=1
4		COMMIT
3	COMMIT	
Якщо втрачені зміни допускаються, то сценарій виконається без помилок і блокувань. В базі даних збережуться зміни, зроблені на кроці 1.		

2. Брудне читання		
1	SELECT * FROM example WHERE id=2	
2		UPDATE example SET dat=101 WHERE id=2
3	SELECT * FROM example WHERE id=2	
4		ROLLBACK
5	SELECT * FROM example WHERE id=2	
Якщо допускається незавершене читання, то сценарій виконається без помилок і блокувань. На кроці 1 будуть вибрані значення (2,110). На кроці 2 - (2,101). На кроці 3 - (2,110).		
3. Неповторне читання		
1	SELECT * FROM example WHERE id=3	
2	[COMMIT]	UPDATE example SET dat=101 WHERE id=3
3		COMMIT
4	SELECT * FROM example WHERE id=3	
5	COMMIT	
Если допускається неповторне читання, то сценарій виконається без помилок і блокувань. Операцію COMMIT на кроці 2 виконувати не прийдеться. На кроці 1 будуть вибрані значення (1,120). На кроці 2 – (1,101).		
4. Фіктивне читання		
1	SELECT * FROM example WHERE dat>180	
2	[COMMIT]	INSERT INTO example VALUES(12,210)
3		COMMIT
4	SELECT * FROM example WHERE dat>180	
5	COMMIT	
Ящо допускається фіктивне читання,то сценарій виконається без помилок і блокувань. Операцію COMMIT на кроці 2 виконувати не прийдеться. На кроці 1 будуть вибрані значення (10,190), (11,200). На кроці 2 – (10,190), (11,200), (12,210).		
5. Безвихідь		
1	UPDATE example SET dat=101 WHERE id=1	

2		UPDATE example SET dat=112 WHERE id=2
3	UPDATE example SET dat=111 WHERE id=2	
4		UPDATE example SET dat=102 WHERE id=1
Якщо система не знаходить і не усуває безвихідних ситуацій, то після виконання кроку 4 транзакції повинні заблокуватися.		

7.4. Субтранзакції

Починаючи з SQL: 1999 транзакції можуть складатися з декількох субтранзакцій. Субтранзакції відокремлюються одна від одної так званими точками відкату, які задаються за допомогою оператора SAVEPOINT (точка збереження, відкату). Він може використовуватися разом з оператором ROLLBACK. ЯКЩО раніше ROOLLBACK застосовувався для скасування всієї транзакції, то тепер його можна використовувати для скасування тільки тієї частини транзакції, яка розташована між ROLLBACK і точкою відкату.

Субтранзакції використовуються не для того, щоб частково відновити базу даних у випадку виникнення помилки (в цьому випадку необхідно виконати відкат всій транзакції). Бувають ситуації, коли виконується складна послідовність дій з багатьма варіантами вибору. На якомусь етапі за отриманими результатами ви можете вирішити, що слід піти по іншому шляху, а для цього доведеться повернутися на декілька кроків назад. Якби не субтранзакції, то довелося б відміняти всю транзакцію цілком.

Точка відкату позначається за допомогою наступного оператора:

SAVEPOINT ім’яТочкиВідкату;

Відкат транзакції до цієї точки виконується за допомогою оператора:

ROLLBACK TO SAVEPOINT ім’яТочкиВідкату;

7.5. Обмеження в транзакціях

Обмеження, що накладаються на стовпці, таблиці та зв’язки між ними, впливають на виконання транзакцій. Наприклад, таблиця має стовпець, на який накладено обмеження NOT NULL (значення стовпця не можуть бути невизначеними), і потрібно додати нові дані. Як правило, в цьому випадку спочатку додають новий порожній запис, а потім заповнюють його значеннями. Однак даний спосіб не можна здійснити, навіть якщо спробувати його виконати до оператора завершення транзакції COMMIT. Справа в тому, що обмеження NOT NULL не дозволить створити рядок з невизначеними значеннями. У SQL: 2003 для вирішення даної проблеми можна надати обмеженням додаткові властивості:

- DEFERABLE (допускає затримку). Обмеження з даною властивістю можуть бути задані як IMMEDIATE (негайні) або як DEFERED (затримані);
- NOT DEFERABLE (не допускає затримку). Обмеження з цією властивістю виконуються негайно.

Якщо обмеження типу DEFERABLE задано як IMMEDIATE, то воно діє негайно, як і обмеження типу NOT DEFERABLE. Якщо обмеження типу DEFERABLE задано як DEFERED, то його дія може бути затримана.

Для додавання порожніх записів і виконання інших операцій, які можуть порушити обмеження типу DEFERABLE, використовується наступний оператор:

SET CONSTRAINS ALL DEFERED;

(Встановити всі обмеження як затримані).

Цей оператор визначає всі обмеження типу DEFERABLE як DEFERED, а на обмеження типу NOT DEFERABLE він не діє. Після виконання всіх операцій, які можуть порушити обмеження, і досягнення таблицею стану, в якому їх порушити вже не можна, всі обмеження можна відновити:

SET CONSTRAINS ALL IMMEDIATE;

(Встановити всі обмеження як негайні).

Якщо обмеження, раніше задані як DEFERED, ви не задаєте явно як IMMEDIATE, то при спробі завершити свою транзакцію оператором COMMIT будуть активовані всі затримані обмеження. Якщо до цього моменту виконуються не всі обмеження, транзакція буде скасована з повідомленням про помилку.

Отже, обмеження захищають базу даних від неправильного введення даних або від їх відсутності. Однак у вас є можливість усередині транзакції тимчасово відмінити обмеження, які заважають вам.

Розглянемо наступний приклад:

Створимо таблицю test.

```
create table test (
x int constraint check_x check (x> 0) deferrable initially immediate,
y int constraint check_y check (y> 0) deferrable initially deferred);
```

Попробуємо добавити дані, які задовольняють обмеженням.

```
insert into test values (1,1); commit;
```

Отже, коли обидва обмеження задовольняються, рядки вставляються без помилки. Однак якщо ми спробуємо вставити рядок, який порушує обмеження CHECK_X, спочатку негайне обмеження, воно негайно перевіриться, і при виконанні наступного оператора

```
insert into test values (-1,1);
```

виникне повідомлення про помилку. Обмеження CHECK_X – відкладене, але спочатку негайне, тому рядок відразу ж перевіряється. Обмеження CHECK_Y, веде себе по-іншому. Воно не тільки відкладене, але і

спочатку відкладене, а це означає, що обмеження не буде перевірятися до виконання оператора COMMIT або зміни стану обмеження на негайне.

При виконанні

insert into test values (1, -1);

Вставка виконається успішно (у всякому випадку, зараз), так як ми відклали перевірку обмеження до завершення транзакції:

Але після виконання оператора commit знову ж таки виникне повідомлення про помилку. У цей момент сервер бази даних виконає відкат транзакції, так як стався збій оператора COMMIT через порушення обмеження. Отже, ця послідовність операторів демонструє відмінності між спочатку негайними і спочатку відкладеними обмеженнями. Компонент “initially” в стані обмеження вказує, коли буде за замовчуванням перевірятися обмеження: або в кінці виконання оператора (immediate), або в кінці виконання транзакції (deferred).

Тепер виконаємо оператор, який робить всі обмеження відкладеними (всі, які можуть бути відкладеними). Зауважимо, ви можете також виконувати цей оператор для окремих обмежень, якщо це потрібно; ви не повинні обов’язково відкладати всі обмеження, які можуть бути відкладеними:

set constraints all deferred;

Після цього знову спробуємо вставити дані

insert into test values (-1,1);

Коли спочатку негайне обмеження переводиться в відкладений режим, здається, що операція вставки виконується успішно; але при спробі фіксації транзакції:

commit відбувається збій транзакції і виконується її відкат, оскільки перевірочне обмеження CHECK_X перевіряється під час виконання оператора COMMIT. І навпаки, можна змусити спочатку відкладені обмеження діяти як “негайні” обмеження:

set constraints all immediate;

insert into test values (1, -1);

Тепер оператор, який працював до моменту фіксації транзакції, збивається відразу ж. Ми вручну змінили режим перевірки обмежень за замовчуванням.

Для чого потрібні відкладені обмеження? Прикладів їх використання можна привести багато. Одна з причин використання відкладених обмежень – забезпечення каскадного оновлення, коли виникає необхідність оновлення первинного ключа в зв’язку “головний-підлеглий”. Якщо ви робите зовнішній ключ відкладеним, але спочатку негайним, ви можете:

- перевести всі обмеження в відкладений режим;
- оновити первинний ключ – на даний момент дочірні обмеження цілісності перевірятися не будуть;
- оновити дочірні зовнішні ключі;

- виконати оператор COMMIT (він буде виконаний успішно, якщо всі дочірні записи, порушені оновленням, вказують на існуючий батьківський запис).

Без відкладених обмежень процес оновлення буде надмірно важким. Крім того, відкладені обмеження ви можете використовувати в різних багатооператорних транзакціях, в процесі виконання яких вимагається тимчасово порушувати цілісність, але в кінці транзакцій цілісність відновлюється.

8. ПОСЛІДОВНОСТІ (SEQUENCE)

В SQL:2003 появилась можливість визначення нового виду об’єктів бази даних – *генераторів послідовностей (sequence generators)*. Послідовність – це об’єкт бази даних, який генерує цілі числа у відповідності з правилами, встановленими під час його створення. Для послідовності можна вказувати як позитивні, так і негативні цілі числа. У системах баз даних послідовності застосовують для самих різних цілей, але в основному для автоматичної генерації первинних ключів. Тим не менше до первинного ключа таблиці послідовність ніяк не прив’язана, так що в деякому розумінні вона є ще й об’єктом спільного користування. Якщо первинний ключ потрібен лише для забезпечення унікальності, а не для того, щоб нести певний зміст, послідовність є відмінним засобом.

8.1.Створення послідовності

Послідовність створюється командою CREATE SEQUENCE.

Для створення генератора послідовності в SQL:2003 введено оператор CREATE SEQUENCE.

```
CREATE SEQUENCE sequence_name
[START WITH <constant>]
[INCREMENT BY <constant>]
[MINVALUE <constant> | NO MINVALUE]
[MAXVALUE <constant> | NO MAXVALUE]
[CYCLE | NO CYCLE]
```

Основні ключові слова та параметри CREATE SEQUENCE:

- **sequence_name** – ім’я послідовності;
- **start with** – дозволяє вказати перше значення, що генерується послідовністю. Після створення послідовність генерує зазначення в start with при першому посиланні на її віртуальний стовпець NEXTVAL;
- **increment by *n*** – визначає приріст послідовності при кожному посиланні на віртуальний стовпець NEXVAL. Якщо значення не вказане явно, за

умовчанням встановлюється 1. Для зростаючих послідовностей встановлюється позитивне *n*, для спадаючих, або послідовностей із зворотним відліком – негативне;

- **minvalue** – визначає мінімальне значення, створюване послідовністю. Якщо воно не зазначено, то використовується значення за замовчуванням NOMINVALUE;
- **nominvalue** – вказує, що мінімальне значення дорівнює 1, якщо послідовність зростає, або -1026, якщо послідовність спадає;
- **maxvalue** – визначає максимальне значення, створюване послідовністю. Якщо воно не зазначено, то використовується значення за замовчуванням NOMAXVALUE;
- **nomaxvalue** – вказує, що максимальне значення дорівнює 1027, якщо послідовність зростає, або -1, якщо послідовність спадає. За замовчуванням приймається NOMAXVALUE;
- **cycle** - дозволяє послідовності повторно використовувати створені значення при досягненні MAXVALUE або MINVALUE. Тобто послідовність буде продовжувати генерувати значення після досягнення свого максимуму або мінімуму. Зростаюча послідовність після досягнення свого максимуму генерує свій мінімум. Спадаюча послідовність після досягнення свого мінімуму генерує свій максимум. Якщо циклічний режим небажаний або не встановлений явним чином, то використовується значення за замовчуванням – NOCYCLE. Вказувати CYCLE разом з NOMAXVALUE або NOMINVALUE не можна. Якщо потрібна циклічна послідовність, необхідно вказати MAXVALUE для зростаючої послідовності або MINVALUE – для спадаючої;
- **nocycle** – вказує, що послідовність не може продовжувати генерувати значення після досягнення свого максимуму або мінімуму;
- **cache *n*** – вказує, скільки значень послідовності розподіляється заздалегідь і підтримується в пам’яті для швидкого доступу. Мінімальне значення цього параметра дорівнює 2. Для циклічних послідовностей це значення повинно бути менше, ніж кількість значень у циклі. Якщо кешування небажане або не встановлено явним чином, то використовується значення за замовчуванням.

Наприклад створимо послідовність sequence_1. Перше звернення до цієї послідовності повинно повернути 1. Друге звернення 11. Кожне наступне звернення повинно повернути значення, на 10 більше попереднього:

CREATE SEQUENCE sequence_1 INCREMENT BY 10;

Створимо послідовність sequence_2. Послідовність повинна бути спадною, циклічною і при досягненні нуля послідовність знову повинна звертатися до старшого числа. Такою послідовністю зручно користуватися в тих програмах, де до настання деякої події повинен бути виконаний зворотний відлік:

```
CREATE SEQUENCE sequence_2
START WITH 20
INCREMENT BY -1
MAXVALUE 20
MINVALUE 0
CYCLE
ORDER
CACHE 2;
```

Після створення послідовності до неї можна звертатися через псевдоствбці CURRVAL (повертає поточне значення послідовності) і NEXTVAL (виконує приріст послідовності і повертає її наступне значення). Поточне і наступне значення послідовності користувачі бази даних отримують, виконуючи команду SELECT. Послідовності – не таблиці, а прості об’єкти, що генерують цілі числа за допомогою віртуальних стовпців, тому потрібна загальнодоступна таблиця словника даних DUAL, з якої будуть вилучатись дані віртуальних стовпців.

Перше звернення до NEXTVAL повертає початкове значення послідовності. Подальші звернення до NEXTVAL змінюють значення послідовності на приріст, який був визначений, і повертають нове значення. Будь-яке звернення до CURRVAL завжди повертає поточне значення послідовності, а саме, те значення, яке було повернуто останнім зверненням до NEXTVAL. Перш ніж звертатися до CURRVAL в поточному сеансі роботи, необхідно хоча б один раз виконати звернення до NEXTVAL.

В одному запиті SQL приріст послідовності може бути виконано тільки один раз. Якщо пропозиція містить кілька звернень до NEXTVAL для однієї і тієї ж послідовності, то нарощується послідовність один раз, і повертає одне і те ж значення для всіх входжень NEXTVAL. Якщо пропозиція містить звернення як до CURRVAL, так і до NEXTVAL, то нарощується послідовність і повертає одне і те ж значення як для CURRVAL, так і для NEXTVAL, незалежно від того, в якому порядку вони зустрічаються в запиті.

До однієї і тієї ж послідовності можуть звертатися одночасно кілька користувачів, без будь-якого очікування або блокування:

<Ім’я послідовності>. CURRVAL

<Ім’я послідовності>. NEXTVAL

Значення CURRVAL і NEXTVAL використовуються у наступних місцях:

- в списку SELECT пропозиції SELECT;
- в фразі VALUES пропозиції INSERT;
- в фразі SET пропозиції UPDATE.

Не можна використовувати значення CURRVAL і NEXTVAL в наступних місцях:

- в підзапитах;

- в пропозиції SELECT з оператором DISTINCT;
- в пропозиції SELECT з оператором GROUP BY або ORDER BY;
- в пропозиції SELECT, об’єднаному з іншим пропозицією SELECT оператором множин UNION;
- в фразі WHERE пропозиції SELECT;
- для значення стовпця за замовчуванням (DEFAULT) в пропозиції CREATE TABLE або ALTER TABLE;
- в умові обмеження CHECK.

Наприклад дія циклічної послідовності sequence_2 при досягненні нею значення MINVALUE:

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат
NEXTVAL

20

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат
NEXTVAL

19
…

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат
NEXTVAL

1

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат
NEXTVAL

0

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат

NEXTVAL

20

У наступному прикладі SEQUENCE після посилання на стовпець NEXVAL значення CURRVAL оновлюється так, щоб відповідати значенню NEXVAL, а попереднє значення CURRVAL втрачається:

Запит
SELECT sequence_2.CURRVAL FROM dual;
Результат
CURRVAL

20

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат
NEXTVAL

19

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат
NEXTVAL

18

Запит
SELECT sequence_2.NEXTVAL FROM dual;
Результат
NEXTVAL

17

Запит
SELECT sequence_2.CURRVAL FROM dual;
Результат
CURRVAL

17

Посилання на послідовності при зміні даних:

INSERT INTO Authors (AUID, Address, SURNAME, NAME) VALUES (sequence_1.nextval,, ‘Kyiv’, ‘Semenyuk’, ‘Oleksandr’);

UPDATE Authors SET AUID = sequence_1.currval WHERE Name = ‘Jones’

8.2. Зміна параметрів послідовності

Будь-який параметр послідовності можна змінити командою ALTER SEQUENCE. Нове значення набуває чинності негайно. Всі параметри послідовності, не зазначені в команді ALTER SEQUENCE, залишаються без змін:

ALTER SEQUENCE sequence_2
INCREMENT BY -4;

8.3. Видалення послідовності

Коли послідовність більше не потрібна, її можна видалити. Для цього адміністратор бази даних або власник послідовності повинен виконати команду DROP SEQUENCE. В результаті віртуальні стовпці послідовності NEXVAL і CURRVAL – переводяться в розряд невикористовуваних. Але, якщо послідовність застосовувалася для генерації значень первинних ключів, створені нею значення залишаться в базі даних. Каскадного видалення значень, згенерованих послідовністю, при її видаленні не відбувається.

Видалення послідовності SEQUENCE:

DROP SEQUENCE sequence_2;

9. ВИКОРИСТАННЯ ІНДЕКСАЦІЇ ДЛЯ ШВИДКОГО ДОСТУПУ ДО ДАНИХ

Індекс – це впорядкований (буквений або числовий) список стовпців або груп стовпців в таблиці. Таблиці можуть мати велику кількість рядків, а, так як рядки не знаходяться в якомусь певному порядку, на їх пошук за вказаним значенням може бути затрачено значний час.

Стандарт ANSI в даний час не підтримує індекси, хоча вони дуже корисні та широко застосовуються. Коли ви створюєте індекс у полі, ваша база даних запам’ятовує відповідний порядок всіх значень цього поля в області пам’яті. Припустимо що наша таблиця Buyers має тисячі записів, а ви хочете знайти покупця з номером BuyerID = 2999. Так як рядки не впорядковані, ваша програма буде переглядати всю таблицю, рядок за рядком, перевіряючи кожного разу значення поля BuyerID на рівність значенню 2999. Однак, якщо є індекс у полі BuyerID, то програма могла б вийти на номер 2999 прямо по індексу і дати інформацію про те як знайти

правильну рядок в таблиці. У той час як індекс значно покращує ефективність запитів, використання індексу дещо уповільнює операції модифікації DML (такі як INSERT і DELETE), а сам індекс займає об'єм пам'яті. Отже, кожен раз коли ви створюєте таблицю потрібно прийняти рішення, індексувати її чи ні. Індекси можуть складатися з кількох полів. Якщо більше ніж одне поле вказується для одного індексу, друге упорядковується всередині першого, третє всередині другого, і так далі. Якщо ви мали перше і останнє ім'я в двох різних полях таблиці, ви могли б створити індекс який би впорядкував попереднє поле всередині наступного. Це може бути виконано незалежно від способу упорядкування стовпців в таблиці. Синтаксис для створення індексу – зазвичай наступний (пам'ятайте, що це не ANSI стандарт):

CREATE INDEX <ім'я індексу> ON <ім'я таблиці> (<ім'я стовпця> [, <ім'я стовпця>]);

При цьому таблиця повинна бути вже створена і містити стовпці, імена яких вказані в команді створення індексу. Ім'я індексу, визначене в команді, має бути унікальним в базі даних. Будучи один раз створеним, індекс є невидимим для користувача, всі операції з ним здійснює СКБД.

Наприклад, якщо таблиця BookSales часто використовується для пошуку відомостей про продажі конкретної книги за значенням поля ISBN, то слід створити індекс по цьому полю.

CREATE INDEX ISBN_Sales ON BookSales (ISBN);

Для видалення індексу (при цьому обов'язково потрібно знати його ім'я) використовується команда DROP INDEX, що має наступний синтаксис:

DROP INDEX <ім'я індексу>;

Видалення індексу не змінює вмісту поля або полів, індекс яких видаляється.

10. ТЕСТОВІ ЗАВДАННЯ

Вибрати 1 правильну відповідь

ПОСЛІДОВНОСТІ, ІНДЕКСИ

1. Послідовність (sequence) –…

a. це об’єкт бази даних, який генерує цілі числа у відповідності з правилами, встановленими під час його створення;
b. це об’єкт бази даних, який генерує первинний ключ при вставці нового запису в таблицю;
c. це об’єкт бази даних, який генерує зовнішній ключ при вставці нового запису в таблицю;
d. це об’єкт бази даних, який генерує потенційний ключ при вставці нового запису в таблицю;
e. це об’єкт бази даних, який генерує унікальні значення поля при вставці запису в таблицю.

2. Нехай дано послідовність

```
CREATE SEQUENCE seq
START WITH 20
INCREMENT BY -1
MAXVALUE 20
MINVALUE 0
CYCLE;
```

При виконанні запиту SELECT seq.NEXTVAL FROM dual; в результаті отримано значення NEXTVAL, яке рівне 17. Яке значення CURRVAL ми отримаєм при наступному запиті SELECT seq.CURRVAL FROM dual;

a. 16.
b. 17.
c. 18.
d. 19
e. 20

3. Створити послідовність seq. Перше звернення до цієї послідовності поверне 1. Друге звернення поверне 11. Кожне наступне звернення поверне значення, на 10 більше попереднього. Вибрати правильну відповідь.

a. CREATE SEQUENCE seq START WITH 0 INCREMENT BY 11;
b. CREATE SEQUENCE seq INCREMENT BY 11;
c. CREATE SEQUENCE seq START WITH 0 INCREMENT BY 10;

d. CREATE SEQUENCE seq INCREMENT BY 10;.
e. CREATE SEQUENCE seq INCREMENT BY -10;

4. Створити послідовність seq. Перше звернення до цієї послідовності поверне 10. Кожне наступне звернення поверне значення, на 1 менше попереднього. Вибрати правильну відповідь.

a. CREATE SEQUENCE seq START WITH 10 INCREMENT BY 1;
b. CREATE SEQUENCE seq START WITH 10 INCREMENT BY -1;
c. CREATE SEQUENCE seq START WITH 1 INCREMENT BY 10;
d. CREATE SEQUENCE seq INCREMENT BY 1;
e. CREATE SEQUENCE seq INCREMENT BY 10;.

5. Розмістіть елементи оператора CREATE INDEX в порядку їх написання:

1. **ім'я індекса**
2. **CREATE INDEX**
3. **(список полів)**
4. **ім'я таблиці**
5. **ON**

Виберіть правильну відповідь:

a. 2 1 3 4 5
b. 2 1 5 4 3
c. 1 3 2 4 5
d. 5 2 3 1 4
e. 3 4 2 1 5

6. Індекси використовуються для…

a. оновлення таблиць;
b. швидкого пошуку потрібних записів;
c. створення таблиць;
d. керування введенням даних у записі.

7. Для видаления індекса використовують оператор…

a. REMOVE INDEX
b. DROP INDEX
c. DELETE INDEX
d. ERASE INDEX

8. Для створення індексу використовують оператор…

a. CREATE INDEX
b. INSERT INDEX
c. ADD INDEX
d. EXEC INDEX
e. UPDATE INDEX

ЗАПИТИ НА ВИБІРКУ

1. Підмножина операторів мови SQL для вибірки, добавлення і видалення даних в базі даних…

a. Data Definition Language
b. Data Manipulation Language
c. Data Control Language

2. Мова SQL призначеа для роботи з ..... базами даних.

a. ієрархічними
b. мережевими
c. реляційними
d. об'єктно-орієнтованими
e. багатомірними

3. Для вибору всіх полів з таблиці використовується конструкція

a. SELECT ALL
b. SELECT *
c. SELECT %
d. SELECT COUNT(*)

4. Дано базу даних, яка складається з 2 таблиць:

STUDENT (Студент) з полями STUDENT_ID – дентифікатор студента, SURNAME – прізвище, NAME – ім'я, STIPEND – стипендія, KURS – курс, CITY – місто, в якому живе студент.

EXAM_MARKS (Екзаменаційні оцінки) з полями EXAM_ID – ідентифікатор іспиту, STUDENT_ID – ідентифікатор студента, SUBJ_ID – ідентифікатор предмета навчання, MARK – екзаменаційна оцінка, EXAM_DATE – дата іспиту.

Вивести середній бал, ідентифікатор студента та імена студентів в яких середній бал більший 80. Який із SQL - запитів правильний?

a. **SELECT** Student.Student_id, Student.Name, AVG(Exam_marks.Mark) **FROM** Student **INNER JOIN** Exam_marks **GROUP BY** Student.Name, Student_id **HAVING** AVG(Exam_marks.Mark)>80;
b. **SELECT** Student.Student_id, Student.Name, AVG(Exam_marks.Mark) **FROM** Student **INNER JOIN** Exam_marks **ON** Student.Student_id = Exam_marks.Student_id **GROUP BY** Student.Name, Student_id **HAVING** AVG(Exam_marks.Mark)>80;
c. **SELECT** Student.Student_id, Student.Name, AVG(Exam_marks.Mark) **FROM** Student, Exam_marks **WHERE** Student.Student_id = Exam_marks.Student_id **AND** AVG(Exam_marks.Mark)>80 **GROUP BY** Student.Name, Student_id;

d. **SELECT** Student.Student_id, Student.Name, AVG(Exam_marks.Mark) **FROM** Student, Exam_marks **WHERE** Student.Student_id = Exam_marks.Student_id **AND** AVG(Exam_marks.Mark)>80;
e. **SELECT** Student.Student_id, Student.Name, AVG(Exam_marks.Mark)>80 **FROM** Student, Exam_marks **WHERE** Student.Student_id = Exam_marks.Student_id;

5. Для вибору прізвища самого молодого студента з таблиці Студенти використовується оператор

a. SELECT Вік FROM Студенти;
b. SELECT MAX(Вік) FROM Студенти;
c. SELECT Вік FROM Студенти WHERE Вік = (SELECT MAX(Вік) FROM Студенти);
d. SELECT Прізвище FROM Студенти WHERE Вік = (SELECT MAX(Вік) FROM Студенти);
e. SELECT Прізвище FROM Студенти WHERE Вік = MAX(Вік);
f. SELECT Прізвище FROM Студенти ORDER BY Вік ASC
g. SELECT Прізвище FROM Студенти ORDER BY Вік DESC

6. Умова Ціна BETWEEN 100 AND 200 еквівалентна умові…

a. (Ціна > 100) AND (Ціна < 200)
b. (Ціна >= 100) AND (Ціна <= 200)
c. (Ціна >= 100) OR (Ціна <= 200)
d. (Ціна > 100) OR (Ціна < 200)

7. Відсутність конструкції WHERE в операторі SELECT вказує на необхідність вибору

a. тільки першого запису таблиці
b. тільки останнього запису
c. всіх записів таблиці
d. тільки першого поля таблиці
e. тільки останнього поля таблиці
f. всіх полів таблиці

8. Умова Місто IN (‘Київ’,’Париж’) еквівалентна умові

a. Місто = (‘Київ’,’Париж’)
b. (Місто = ‘Київ’) OR (Місто = ‘Париж’)
c. (Місто = ‘Київ’) AND (Місто = ‘Париж’)
d. Місто BETWEEN ‘Київ’ AND ‘Париж’

9. Нехай ми маємо таблицю Students бази даних.

Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Петро	Сергійович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Скільки записів поверне запит:
SELECT DISTINCT Group FROM Students?

a. 1 запис
b. 2 записи
c. 3 записи
d. 4 записи
e. 5 записи
f. 6 записів

10.Нехай ми маємо таблицю Students бази даних.

Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Петро	Сергійович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Скільки записів поверне запит: SELECT DISTINCT Name FROM Students?

a. 3 записи
b. 4 записи
c. 5 записів
d. 6 записів
e. 7 записів

11. Для вибору різних прізвищ студентів з таблиці Студенти використовують оператор

a. SELECT Прізвище FROM Студенти
b. SELECT DISTINCT Прізвище FROM Студенти
c. SELECT SOME Прізвище FROM Студенти ORDER BY Прізвище
d. SELECT Прізвище FROM Студенти ORDER BY Прізвище

12. Дано БД, яка містить таблицю EXAM_MARKS (Екзаменаційні оцінки) з полями EXAM_ID – ідентифікатор іспиту, STUDENT_ID – ідентифікатор студента, SUBJ_ID – ідентифікатор предмета навчання, MARK – екзаменаційна оцінка, EXAM_DATE – дата іспиту. Вивести ідентифікатори студентів (не дублюючи їх), які здавали хоча б один екзамен. Який із SQL -запитів правильний?

a. SELECT Student_id FROM Exam_marks;
b. SELECT DISTINCT Student_id FROM Exam_marks;
c. SELECT Student_id FROM Exam_marks WHERE mark NOT Null;
d. SELECT Top 1 Student_id FROM Exam_marks ORDER BY.

13. Нехай ми маємо таблицю Students бази даних.

Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Петро	Іванович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Скільки записів поверне запит SELECT DISTINCT Name, Lastname FROM Students WHERE Group=615?

a. 1 запис
b. 2 записи
c. 3 записи
d. 4 записи
e. 5 записи
f. 6 записів

14. Нехай ми маємо таблицю Students бази даних.

Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Петро	Сергійович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Скільки записів поверне запит SELECT DISTINCT *Surname, Name* FROM Students ?

a. 0 записів
b. 1 запис
c. 2 записи

d. 3 записи
e. 4 записи
f. 5 записів
g. 6 записів

15. Нехай ми маємо таблицю Students бази даних.

Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Петро	Іванович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Скільки записів поверне запит SELECT DISTINCT Name, Lastname FROM Students WHERE Group<615?

a. 0 записів
b. 1 запис
c. 2 записи
d. 3 записи
e. 4 записи
f. 5 записи
g. 6 записів

16. Конструкція DISTINCT в операторі SELECT використовується для…

a. упорядкування записів
b. фільтрації записів
c. групування записів
d. виключення записів, що дублюються
e. вибору записів

17. Пропозиція HAVING мови SQL діє точно так само, як і пропозиція WHERE, із тією лише різницею, що…

a. HAVING використовується з пропозицією GROUP BY.
b. З WHERE не можна **використовувати** агрегатні функції.
c. З HAVING не можна **використовувати** агрегатні функції.
d. Пропозиція WHERE є обов'язковою в запиті на вибірку даних, а HAVING не обов'язковою.
e. Пропозиція HAVING є обов'язковою в запиті на вибірку даних, а WHERE не обов'язковою.

18. Дано дві таблиці БД.

Students			
Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Степан	Сергійович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Teachers			
Surname	***Name***	***Lastname***	***Cathedra***
Михайлюк	Петро	Іванович	IC
Богданюк	Віктор	Степанович	MA
Семенюк	Сергій	Петрович	IT
Давидюк	Степан	Степанович	IT
Ткачук	Олена	Олексіївна	IC
Філозоф	Ольга	Іванівна	MA

Скільки записів поверне запит
SELECT Surname FROM Students WHERE Group=615
UNION
SELECT Surname FROM Teachers WHERE Cathedra =‘IC’

a. 2 записи
b. 4 записи
c. 5 записів
d. 6 записів
e. 12 записів

19. Для об’єднання результуючих наборів двох або більше запитів SELECT в один результуючих набір використовується оператор…

a. JOIN
b. UNION
c. INTERSECT
d. GROUP BY
e. EXEPT

20. Переріз двох наборів записів запитів SELECT здійснюється за допомогою оператора …

a. JOIN
b. UNION
c. INTERSECT

d. GROUP BY
e. EXEPT

21. Для отримання записів, що містяться в одному наборі запиту SELECT і відсутні в іншому, служить оператор …

a. JOIN
b. UNION
c. INTERSECT
d. GROUP BY
e. EXEPT

КОМАНДИ МАНІПУЛЮВАННЯ ДАНИМИ

1. Нехай ми маємо таблицю Students бази даних.

Surname	Name	Lastname	Group
Степанюк	Петро	Іванович	234
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Степан	Сергійович	234
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	564

Скільки записів буде видалено при виконанні запиту:
DELETE Students WHERE Surname LIKE 'C%a%' ?

a. 1 запис
b. 2 записи
c. 3 записи
d. 4 записи
e. 6 записів

2. Дано база даних "Успішність студентів", яка складається з 5 таблиць

Students	Groups	Progress	Subjects	Lectors
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

Видалити з таблиці Subjects всі записи, у яких у поле Name_subject немає даних або в полі міститься порожнє значення. Який із SQL - запитів правильний?

a. **DELETE FROM** Subjects **WHERE** Name_subject=Null**;**
b. **DELETE FROM** Subjects **WHERE** Name_subject=’ ‘;
c. **DELETE FROM** Subjects **WHERE** Name_subject=Null **OR** Name_subject=’ ‘ ;
d. **DELETE** Subjects **WHERE** Name_subject=Null;
e. **DELETE** Subjects **WHERE** Name_subject=’ ‘;
f. **DELETE** Subjects **WHERE** Name_subject=Null **OR** Name_subject=’ ‘;

3. Якщо в операторі DELETE конструкція WHERE не задана…

a. не видаляється ні один запис
b. видаляються всі записи
c. видаляється перший запис
d. видаляється останній запис

4. Нехай ми маємо таблицю Students бази даних.

Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Степан	Сергійович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Скільки записів буде видалено при виконанні запиту:
DELETE Students WHERE Group IN (615,616)?

a. 1 запис
b. 2 записи
c. 3 записи
d. 6 записів

5. Нехай ми маємо таблицю Students бази даних.

Surname	***Name***	***Lastname***	***Group***
Степанюк	Петро	Іванович	615
Сашков	Антон	Степанович	345
Семенюк	Сергій	Петрович	455
Сашков	Петро	Сергійович	615
Іванова	Олена	Олександрівна	234
Семенова	Ольга	Іванівна	615

Скільки записів буде видалено при виконанні запиту:
DELETE Students WHERE Group =615 AND Group =455?

a. 0 записів
b. 1 запис
c. 2 записи
d. 3 записи
e. 4 записи
f. 5 записів
g. 6 записів

6. Яка необов'язкова частина оператора INSERT:
INSERT INTO таблиця (список полів) VALUES (список значень)

a. INSERT INTO
b. таблиця
c. список полів
d. VALUES
e. список значень

7. Для добавлення запису в таблицю Товари з полями
"Назва" (varchar(50), NOT NULL),
"Ціна" (float, NOT NULL),
"Кількість" (int)
можна використати запит:

a. INSERT INTO Товари VALUES (хліб,11.5,50)
b. INSERT INTO Товари VALUES ('хліб',12,11.5)
c. INSERT INTO Товари VALUES (1,'хліб',50,11.5)
d. INSERT INTO Товари VALUES ('хліб',11.5,50)

8. Нехай створено таблицю за допомогою запиту
GREATE TABLE Student (ID_Student Integer Primary key, Surname Char(20) NOT NULL, Spesiality Integer); Що відбудеться при виконанні наступної послідовності запитів.

1. **INSERT INTO Student;**
2. **INSERT INTO Student (ID_Student, Surname, Spesiality) VALUES (1, 'Малюс', 5);**
3. **INSERT INTO Student (ID_Student, Surname, Spesiality) VALUES (2, 'Боярчук', 5);**
4. **INSERT INTO Student (ID_Student, Surname, Spesiality) VALUES (1, 'Волошин', 8);**

a. 1 запит виконається, 2 виконається, 3 виконається, 4 не виконається.
b. 1 запит виконається, 2 виконається, 3 не виконається, 4 не виконається.

c. 1 запит не виконається, 2 виконається, 3 виконається, 4 не виконається.
d. 1 запит виконається, 2 виконається, 3 не виконається, 4 виконається.
e. 1 запит не виконається, 2 виконається, 3 не виконається, 4 виконається.
f. 1 запит не виконається, 2 не виконається, 3 виконається, 4 виконається.

9. Розмістіть елементи оператора UPDATE в порядку його написання. Виберіть правильну відповідь.

1. таблиця
2. WHERE
3. поле=значення
4. SET
5. UPDATE
6. Умова

Відповідь

a. 243156
b. 542316
c. 524316
b. 514326
c. 125436

10. Якщо в операторі UPDATE конструкція WHERE не задана…

a. не змінюється ні один запис
b. змінюються всі записи
c. змінюється перший запис
d. змінюється останній запис

СТВОРЕННЯ ТАБЛИЦЬ

1.Тип даних символьних стрічок змінної довжини, максимальна довжина n символів

a. char(n)
b. varchar(n)
c. string
d. string(n)

2. У записі таблиці реляційної бази даних (БД) можуть міститися…

a. тільки логічні величини;
b. тільки текстова інформація;
c. винятково однорідна інформація (дані тільки одного типу);

d. винятково числова інформація;
e. неоднорідна інформація (дані різних типів).

3. Для заборони можливості введення в поле порожніх значень використовується конструкція…

a. NOT NIL
b. NOT NOTHING
c. NOT NULL
d. NOT EMPTY
e. DEFAULT

4. Зовнішній ключ визначається як обмеження для таблиці реляційної бази даних в SQL-виразі з ключовими словами…

a. PRIMARY KEY
b. CONSTRAINT FOREIGN KEY
c. UNIQUE
d. CHEK
e. DEFAULT

5. Щоб вказати значення по замовчуванню для поля використовується конструкція…

a. NOT NULL
b. DEFAULT
c. FOREIGN KEY
d. ON DEFAULT
e. UNIQUE
f. CHECK

6. Представлення (view) в базі даних зберігає…

a. тимчасову таблицю з даними
b. результат виконання запиту SELECT
c. текст запиту SELECT
d. текст запиту SELECT і результат його виконання

7. Для створення представлення (view) в базі даних використовується оператор…

a. CREATE TABLE
b. CREATE SELECT
c. CREATE QUERY
d. CREATE VIEW
e. CREATE TEMPORARY TABLE

8. В таблиці допускається тільки одне обмеження…

a. UNIQUE
b. FOREIGN KEY
c. PRIMARY KEY
d. CHECK
e. NOT NULL
f. NULL

9. Який ключ використовується для порядку відображення даних та визначає унікальний запис в таблиці реляційної бази даних?

a. Зовнішній ключ
b. Супер ключ
c. Первинний ключ
d. Вторинний ключ
e. Потенційний ключ

10. Виберіть оператор SQL для зміни таблиці.

a. Create Table
b. Delete Table
c. Drop Table
d. Alter Table
e. Rename Table

11. Виберіть оператор SQL для видалення таблиці.

a. Create Table
b. Delete Table
c. Drop Table
d. Alter Table

12. Для заборони можливості введення в поле порожніх значень використовується конструкція…

a. NOT NIL
b. NOT NOTHING
c. NOT NULL
d. NOT EMPTY
e. DEFAULT
f. NULL

13. Вибрати тип даних символьних стрічок змінної довжини, максимальна довжина n символові.

a. char(n)
b. varchar(n)
c. string
d. string(n)

14. Який ключ використовується для зв'язку полів із полями інших таблиць реляційної бази даних?

a. Первинний ключ
b. Вторинний ключ
c. Зовнішній ключ
d. Супер ключ
e. Потенційний ключ

ТРАНЗАКЦІЇ

1. Дано таблиця

Таблиця EXAMPLE	**id INTEGER**	1	2	3	4	5	6	7	8	9	10	11
	dat INTEGER	100	110	120	130	140	150	160	170	180	190	200

Нехай паралельно виконуються транзакції T1 і T2, в порядку поданому в таблиці:

рок	**Транзакція T1**	**Транзакція T2**
	UPDATE example SET dat=101 WHERE id=1	
		UPDATE example SET dat=102 WHERE id=1
		COMMIT
	COMMIT	

Якщо аномалія втрачені зміни допускається, то сценарій:

a. виконається без помилок і блокувань. В базі даних збережуться зміни, зроблені на кроці 2;
b. виконається без помилок і блокувань. В базі даних збережуться зміни, зроблені на кроці 1.
c. після виконання кроку 4 транзакції повинні заблокуватися;
d. після виконання кроку 3 транзакції повинні заблокуватися;
e. виконається без помилок і блокувань. В базі даних зміни виконані транзакціями T1 і T2 не збережуться.

2. Якою командою SQL відбувається відкат транзакції до попереднього стану бази даних?

a. ROLLBACK
b. COMMIT
c. SET TRANSACTION
d. BEGIN TRAN
e. BEGIN WORK

3. Дано дві таблиці:

create table t (x int constraint check_x check (x> 0) deferrable initially immediate,

y int constraint check_y check (y> 0) deferrable initially deferred);

Що відбудетьсяпри спробі добавити дані: insert into t values (1,-1); Вибрати правильну відповідь.

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.

4. Дано дві таблиці:

create table t (x int constraint check_x check (x> 0) deferrable initially immediate,

y int constraint check_y check (y> 0) deferrable initially deferred);

Що відбудетьсяпри спробі добавити дані: insert into t values (-1,1); Вибрати правильну відповідь.

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.

5. Дано дві таблиці:

create table t (x int constraint check_x check (x> 0) deferrable initially immediate,

y int constraint check_y check (y> 0) deferrable initially deferred);

Що відбудетьсяпри спробі добавити дані: insert into t values (-1,-1); Вибрати правильну відповідь.

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.

6. Дано дві таблиці:

create table t (x int constraint check_x check (x> 0) deferrable initially immediate,

y int constraint check_y check (y> 0) deferrable initially deferred);

set constraints all deferred;

Що відбудетьсяпри спробі добавити дані: insert into t values (-1,1); Вибрати правильну відповідь.

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.

7. Дано дві таблиці:

create table t (x int constraint check_x check (x> 0) deferrable initially immediate,

y int constraint check_y check (y> 0) deferrable initially deferred);

set constraints all immediate;

Що відбудетьсяпри спробі добавити дані: insert into t values (1,-1); Вибрати правильну відповідь.

a. Вставка виконається успішно і після виконання оператора commit виконається успішне підтвердження транзакції.
b. Вставка виконається успішно, але після виконання оператора commit виникне повідомлення про помилку і сервер бази даних виконає відкат транзакції.
c. Виникне повідомлення про помилку, ще до виконання оператора commit.
d. В таку таблицю ми не зможемо взагалі добавити дані.

Вибрати кілька правильних відповідей.

1. Дано база даних, яка складається з 5 таблиць

Students	**Groups**	**Progress**	**Subjects**	**Lectors**
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

За допомогою запиту SQL вивести відомості про поставку (поля Date_order (дата замовлення), Type_purchase (тип покупки), Cost (ціна), Amount (кількість)), а також назву книги (поле Title_book) з максимальною загальною вартістю замовлення (використовувати поля Cost (ціна) і Amount (кількість)). Вибрати правильні запити.

a. **SELECT** Purchases.Date_order, Purchases.Type_purchase, Purchases.Cost, Amount, Books. Title_book **FROM** Purchases **INNER JOIN** Books **ON** Purchases. Code_book=Books. Code_book **WHERE** Purchases.Cost* Purchases.Amount=(**SELECT Max(**Purchases.Cost* Purchases.Amount**) FROM** Purchases);
b. **SELECT** Purchases.Date_order, Purchases.Type_purchase, Purchases.Cost, Amount, Books. Title_book, **Max(**Purchases.Cost* Purchases.Amount**) FROM** Purchases **INNER JOIN** Books **ON** Purchases. Code_book=Books. Code_book **WHERE** Purchases.Cost* Purchases.Amount=**Max(**Purchases.Cost* Purchases.Amount**);**
c. **SELECT** Purchases.Date_order, Purchases.Type_purchase, Purchases.Cost, Amount, Books. Title_book **FROM** Purchases, Books **WHERE** Purchases. Code_book=Books. Code_book **AND** Purchases.Cost* Purchases.Amount=(**SELECT Max(**Purchases.Cost* Purchases.Amount**) FROM** Purchases);
d. **SELECT** Purchases.Date_order, Purchases.Type_purchase, Purchases.Cost, Amount, Books. Title_book, **Max(**Purchases.Cost* Purchases.Amount **FROM** Purchases, Books **WHERE** Purchases. Code_book=Books. Code_book **AND** Purchases.Cost* Purchases.Amount=**Max(**Purchases.Cost* Purchases.Amount);
e. **SELECT** Purchases.Date_order, Purchases.Type_purchase, Purchases.Cost, Amount, Books. Title_book, **Max(**Purchases.Cost* Purchases.Amount) **FROM** Purchases, Books **WHERE** Purchases. Code_book=Books. Code_book;

2. Дано база даних “Успішність студентів”, яка складається з 5 таблиць

Students	**Groups**	**Progress**	**Subjects**	**Lectors**
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

За допомогою запитів SQL вивести всі відомості про здачі іспитів (всі поля з таблиці Progress) і список студентів (поля Surname, Name з таблиці Students), які по таблиці Progress здавали іспити самими останніми (дата здачі іспитів Date_exam максимальна).

a. **SELECT** Progress.*, Surname, Name **FROM** Progress, Students **WHERE** Progress. Code_stud=Students**.** Code_stud **AND** Date_exam=(**SELECT max(**Date_exam**) FROM** Progress);
b. **SELECT** Progress.Code_stud, Progress.Code_subje, Progress.Code_lector, Progress.Date_exam, Progress.Estimate, Progress.Code_progress, Students**.**Surname, Students**.**Name **FROM** Progress **INNER JOIN** Students **WHERE** Progress. Code_stud=Students**.** Code_stud **OR** Progress.Date_exam=(**SELECT max(**Date_exam**) FROM** Progress);
c. **SELECT** Progress.Code_stud, Progress.Code_subje, Progress.Code_lector, Progress.Date_exam, Progress.Estimate, Progress.Code_progress, Students**.**Surname, Students**.**Name **FROM** Progress, Students **WHERE** Date_exam=(**SELECT max(**Date_exam**) FROM** Progress);
d. **SELECT** Progress.Code_stud, Progress.Code_subje, Progress.Code_lector, Progress.Date_exam, Progress.Estimate, Progress.Code_progress, Students**.**Surname, Students**.**Name **FROM** Progress, Students **WHERE** Progress. Code_stud=Students**.** Code_stud **AND** Date_exam=(**SELECT max(**Date_exam**) FROM** Progress);
e. **SELECT** Progress.*, Surname, Name **FROM** Progress **INNER JOIN** Students **ON** Progress. Code_stud=Students**.** Code_stud **WHERE** Date_exam=(**SELECT max(**Date_exam**) FROM** Progress);
f. **SELECT** Progress.Code_stud, Progress.Code_subje, Progress.Code_lector, Progress.Date_exam, Progress.Estimate, Progress.Code_progress, Students**.**Surname, Students**.**Name **FROM** Progress **INNER JOIN** Students **WHERE** Progress.Date_exam=(**SELECT max(**Date_exam**) FROM** Progress);
g. **SELECT** Progress.Code_stud, Progress.Code_subje, Progress.Code_lector, Progress.Date_exam, Progress.Estimate,

Progress.Code_progress, Students**.**Surname, Students**.**Name **FROM** Progress **INNER JOIN** Students **ON** Progress. Code_stud=Students**.** Code_stud **WHERE** Progress.Date_exam=(**SELECT max(**Date_exam**) FROM** Progress);

3. Дано база даних "Успішність студентів", яка складається з 5 таблиць

Students	**Groups**	**Progress**	**Subjects**	**Lectors**
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

За допомогою запитів Вивести список назв дисциплін (поле Name_subject з таблиці Subjects) і імен викладачів (поле Name_lector з таблиці Lectors), які приймали по цих дисциплінах іспити. Вибрати правильні запити.

a. **SELECT** Name_subject, Name_lector **FROM** Subjects, Lectors, Progress **WHERE** Subjects.Code_subject = Progress.Code_subject **AND** Lectors.Code_lector = Progress.Code_lector;
b. **SELECT** Name_subject, Name_lector **FROM** Subjects, Lectors, Progress **WHERE** Subjects.Code_subject = Progress.Code_subject **OR** Lectors.Code_lector = Progress.Code_lector;
c. **SELECT** Name_subject, Name_lector **FROM** Subjects INNER JOIN Lectors **ON** Lectors.Code_lector = Progress.Code_lector INNER JOIN Progress **ON** Subjects.Code_subject = Progress.Code_subject
d. **SELECT** Name_subject, Name_lector **FROM** Subjects **INNER JOIN** (Lectors **INNER JOIN** Progress **ON** Lectors.Code_lector = Progress.Code_lector) **ON** Subjects.Code_subject = Progress.Code_subject
e. **SELECT** Name_subject, Name_lector **FROM** Subjects **INNER JOIN** (Lectors **INNER JOIN** Progress **ON** Lectors.Code_lector = Progress.Code_lector) **ON** Subjects.Code_subject = Progress.Code_subject **WHERE** Subjects.Code_subject = Progress.Code_subject **OR** Lectors.Code_lector = Progress.Code_lector;

4. **Дано база даних “Успішність студентів”, яка складається з 5 таблиць**

Students	**Groups**	**Progress**	**Subjects**	**Lectors**
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

За допомогою запитів SQL вивести список студентів (поля Surname, Name, Lastname) з таблиці Students, які здавали екзамени з дисциплін з наступними кодами 5, 8, 12, 25 (умова по полю Code_subject). Вибрати правильні запити.

a. **SELECT** Surname, Name, Lastname **FROM** Students, Progress, Subjects **WHERE** Code_subject **IN** (5, 8, 12, 25);
b. **SELECT** Surname, Name, Lastname **FROM** Students, Progress, Subjects **WHERE** Progress. Code_subject=Subjects. Code_subject **AND** Students. Code_stud=Progress.Code_stud **AND** Code_subject **IN** (5, 8, 12, 25);
c. **SELECT** Surname, Name, Lastname **FROM** Students, Progress, Subjects **WHERE** Progress. Code_subject=Subjects. Code_subject **AND** Students. Code_stud=Progress.Code_stud **OR** Code_subject **IN** (5, 8, 12, 25);
d. **SELECT** Surname, Name, Lastname **FROM** Students, Subjects **WHERE** Progress. Code_subject=Subjects. Code_subject **AND** Students.Code_stud=Progress.Code_stud **AND** Code_subject **IN** (5, 8, 12, 25);
e. **SELECT** Surname, Name, Lastname **FROM** Students, Progress, Subjects **WHERE** Progress. Code_subject=Subjects. Code_subject **AND** Students. Code_stud=Progress. Code_stud **AND** Code_subject = 5 **AND** Code_subject = 8 **AND** Code_subject = 12 AND Code_subject = 25;
f. **SELECT** Surname, Name, Lastname **FROM** Students, Subjects **WHERE** Progress. Code_subject=Subjects. Code_subject **AND** Students. Code_stud=Progress. Code_stud **AND** Code_subject = 5 **AND** Code_subject = 8 **AND** Code_subject = 12 **AND** Code_subject = 25;
g. **SELECT** Surname, Name, Lastname **FROM** Students, Progress, Subjects **WHERE** Progress. Code_subject=Subjects. Code_subject **AND** Students. Code_stud=Progress. Code_stud **AND (**Code_subject = 5 **OR** Code_subject = 8 **OR** Code_subject = 12 **OR** Code_subject = 25);

5. Дано база даних “Успішність студентів”, яка складається з 5 таблиць

Students	Groups	Progress	Subjects	Lectors
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

За допомогою запитів SQL вивести список прізвищ (поле Surname), імен (поле Name), по батькові (поле Lastname) студентів з таблиці Students і назв груп (поле Name_group) з таблиці Groups, у яких вони навчаються і відсортувати результат за спаданням по прізвищу студента (поле Surname). Вибрати правильні запити.

a. SELECT Surname, Name, Lastname, Name_group FROM Students, Groups WHERE Students.Code_group= Groups.Code_group ORDER BY Surname DESC;
b. SELECT Surname, Name, Lastname, Name_group FROM Students, Groups WHERE Students.Code_group=Groups.Code_group ORDER BY Surname ASC;
c. SELECT Surname, Name, Lastname, Name_group FROM Students INNER JOIN Groups ON Students.Code_group= Groups.Code_group ORDER BY Surname DESC;
d. SELECT Surname, Name, Lastname, Name_group FROM Students INNER JOIN Groups ON Students. Code_group= Groups. Code_group ORDER BY Surname ASC;
e. SELECT Surname, Name, Lastname, Name_group FROM Students, Groups ORDER BY Surname DESC;
f. SELECT Students.Surname, Students.Name, Students.Lastname, Groups.Name_group FROM Students, Groups WHERE Students.Code_group= Groups.Code_group ORDER BY Students.Surname DESC;

6. Дано база даних “Успішність студентів”, яка складається з 5 таблиць

Students	**Groups**	**Progress**	**Subjects**	**Lectors**
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

За допомогою запитів SQL вивести список студентів (поля Surname, Name, Lastname) і їхні телефони (поле Phone) з таблиці Students, якщо значення телефонів перебувають у діапазоні від 220000 до 226666 (умова по полю Phone). Вибрати правильні запити.

a. **SELECT** Surname, Name, Lastname, Phone **FROM** Students **WHERE** Phone <= 220000 **AND** >=226666;
b. **SELECT** Surname, Name, Lastname, Phone **FROM** Students **WHERE** Phone <= 220000 **AND** Phone >=226666;
c. **SELECT** Surname, Name, Lastname, Phone **FROM** Students **WHERE** Phone **BETWEEN** 220000 **AND** 226666;
d. **SELECT** Surname, Name, Lastname, Phone **FROM** Students **WHERE** Phone **BETWEEN** (220000, 226666);
e. **SELECT** Students**.**Surname, Students**.**Name, Students**.**Lastname, Students**.**Phone **FROM** Students **WHERE** Students**.**Phone **BETWEEN** 220000 **AND** 226666;

7. Дано база даних “Успішність студентів”, яка складається з 5 таблиць

Students	**Groups**	**Progress**	**Subjects**	**Lectors**
Code_stud	Code_group	Code_stud	Code_subject	Code_lector
Surname	Name_group	Code_subject	Name_subject	Name_lector
Name	Num_course	Code_lector	Count_hours	Science
Lastname	Name_speciality	Date_exam		Post
Birthday		Estimate		Date_
Phone		Code_progress		
Code_group				

За допомогою запитів Видалити з таблиці Students всі записи, код групи яких дорівнює 35, або 15, або 19 (умова по полю Code_group). Вибрати правильні запити.

a. DELETE FROM Students WHERE Code_group=35 OR Code_group=15 OR Code_group=19;
b. DROP TABLE Students WHERE Code_group=35 OR Code_group=15 OR Code_group=19;
c. DELETE FROM Students WHERE Code_group=35 AND Code_group=15 AND Code_group=19;
d. DELETE FROM Students WHERE Code_group IN (35, 15, 19);
e. DELETE Students WHERE Code_group=35 OR Code_group=15 OR Code_group=19;

8. Дано БД, яка містить таблиці:

Student (ID_Student, Name, Surname, Lastname, Phone, Adress, ID_ Univ); University (ID_ Univ, Name_ University, City).

Нехай існує таблиця Student1, визначення стовпців якої повнстю співпадають з визначенням стовпців таблиці Student. За допомогою запиту SQL добавити в цю таблицю відомості про всіх студентів, які вчаться в Києві. Вибрати правильні запити.

a. INSERT INTO Student1 SELECT * FROM Student WHERE ID_ Univ IN (SELECT ID_ Univ FROM University WHERE City=”Київ”)
b. INSERT INTO Student1 SELECT * FROM Student WHERE ID_Student IN (SELECT ID_Student FROM Student, University WHERE Student. ID_ Univ=University. ID_ Univ AND City=”Київ”)
c. INSERT INTO Student1 SELECT * FROM Student WHERE City=”Київ”
d. INSERT INTO Student1 SELECT * FROM Student WHERE ID_Student IN (SELECT ID_Student FROM Student INNER JOIN University ON Student. ID_ Univ=University. ID_ Univ WHERE City=”Київ”)
e. INSERT INTO Student1 SELECT * FROM Student, University WHERE Student. ID_ Univ=University. ID_ Univ AND City=”Київ”
f. INSERT INTO Student1 SELECT * FROM Student INNER JOIN University ON Student. ID_ Univ=University. ID_ Univ WHERE City=”Київ”)

9. За допомогою SQL-запитів створити таблицю Student, яка має поля: ID_Student, Surname, Spesiality, Remark. Ідентифікатор студента (цілочисельний стовпець ID_Student), стовбець Surname (симвотний, кількість символів не більше 20), цілочисельний стовпець Spesiality повинен містити номери спеціальностей які не перевищують значення 12 і стовпець Remark повинен бути типу типу VARCHAR. Вибрати правильні запити.

a. GREATE TABLE Student (ID_Student INTEGER, Surname Char(20), Spesiality INTEGER Chek (Spesiality<12), Remark VARCHAR);
b. GREATE TABLE Student (ID_Student INTEGER, Surname Char(20), Spesiality INTEGER Chek <12, Remark VARCHAR);
c. GREATE TABLE Student (ID_Student INTEGER, Surname Char (20), Spesiality INTEGER (12), Remark VARCHAR);
d. GREATE TABLE Student (ID_Student INTEGER, Surname Char (20), Spesiality INTEGER<12, Remark VARCHAR);
e. CREATE DOMAIN specDomain INTEGER Chek (Spesiality<12);
GREATE TABLE Student (ID_Student INTEGER, Surname Char(20), Spesiality specDomain, Remark VARCHAR);
f. CREATE DOMAIN specDomain INTEGER Chek <12;
GREATE TABLE Student (ID_Student INTEGER, Surname Char(20), Spesiality specDomain, Remark VARCHAR);

10. Якщо в операторі INSERT для поля таблиці не вказане значення, то це поле в добавленому записі отримає значення

a. NULL
b. NIL
c. 0
d. по замовчуванню
e. порожня стрічка

11. Умова перевірки того, що ціле поле Оцінка містить значення від 2 до 5

a. Оцінка BETWEEN 2 AND 5
b. (Оцінка >= 2) AND (Оцінка <= 5)
c. Оцінка >= 2 AND <= 5
d. (Оцінка <= 2) AND (Оцінка >= 5)
e. Оцінка IN (2, 3, 4, 5)

12. Для добавлення запису в таблицю Абітурієнти з полями
"Прізвище" (NOT NULL),
"Імя" (NOT NULL),
"Рік закінчення" (NOT NULL, DEFAULT = 2007)
можна використати запит

a. INSERT INTO Абітурієнти VALUES (Іванов',’Петро’)
b. INSERT INTO Абітурієнти VALUES (‘Петро’,’Іванов’)
c. INSERT INTO Абітурієнти VALUES (‘Іванов’,’Петро’,2006)
d. INSERT INTO Абітурієнти VALUES (75,’Іванов’,’Петро’)

13. Запит SELECT, що визначає представлення,

a. може посилатися не більше як на одну таблицу
b. може посилатися на декілька таблиць

c. може посилатися не більше як на три таблиці
d. не може посилатися на декілька таблиць
e. може посилатися на інше представлення
f. не може посилатися на інше представлення

14. Для зменшення ціни на 10% для товарів кількості меньшим 10 в таблиці Товари з полями "Назва", "Ціна", "Кількість" можна використати запит

a. UPDATE Товари SET Ціна= Ціна - Ціна *10/100 WHERE Кількість < 10
b. UPDATE Ціна = Ціна *0.9 WHERE Кількість < 10
c. UPDATE Ціна = Ціна - Ціна *0.1 WHERE Кількість < 10
d. UPDATE Товари SET Ціна=Ціна*0.9 WHERE Кількість < 10
e. UPDATE Товари SET Ціна=Ціна-Ціна*0.1 WHERE Кількість < 10

16. Тимчасова таблиця, сформована з результуючого набору оператора SELECT

a. представлення
b. запит
c. процедура
d. записи
e. view

16. Для виконання оператора UNION необхідне співпадіння в таблицях

a. кількості полів
b. назв полів
c. типів даних полів
d. обмежень для полів таблиць

17. Щоб добавити запис в таблицю “Абітурієнти” можна використати запит

Абітурієнти			
Прізвище	**Ім’я**	**Побатькові**	**Група**

a. INSERT INTO **Абітурієнти** VALUES (6151,’Степанюк’,’Петро’,’Сергійович’)
b. INSERT INTO **Абітурієнти** VALUES (‘Степанюк’,’Петро’,’Сергійович’,6151)
c. INSERT INTO **Абітурієнти** (Група, Прізвище, Ім’я, Побатькові) VALUES (6151, ‘Степанюк’, ’Петро’, ’Сергійович’)
d. INSERT INTO **Абітурієнти** (Прізвище, Ім’я, Побатькові, Група) VALUES (6151, ‘Степанюк’, ’Петро’, ’Сергійович’)

18. В таблиці допускається більше одного обмеження

a. UNIQUE
b. FOREIGN KEY
c. PRIMARY KEY
d. CHECK
e. NOT NULL
f. NULL

19. Для збільшення кількості жителів на 10000 для міст Рівне і Луцьк в таблиці Міста з полями "Назва", "Країна", "Жителів" можна використати запит

a. UPDATE Міста SET Жителів = Жителів +10000 WHERE Назва IN (‘Рівне’, ‘Луцьк’)
b. UPDATE Жителів = Жителів +10000 WHERE Назва=‘Рівне’ OR Назва=‘Луцьк’
c. UPDATE Міста SET Жителів +=10000 WHERE Назва IN (‘Рівне’, ‘Луцьк’)
d. UPDATE Міста SET Жителів = Жителів +10000 WHERE Назва=‘Рівне’ OR Назва=‘Луцьк’
e. UPDATE Міста SET Жителів = Жителів +10000 WHERE Назва=‘Рівне’ AND Назва=‘Луцьк’

20. Тимчасова таблиця, сформована з результуючого набору оператора SELECT

a. представлення
b. запит
c. процедура
d. записи
e. view
f. procedure

21. Умова перевірки того, що ціле поле Вік містить значення від 16 до 50

a. Вік BETWEEN 16 AND 50
b. Вік IN (16, 50)
c. Вік >= 16 AND <= 50
d. (Вік >= 16) AND (Вік <= 50)
e. (Вік <= 16) AND (Вік >= 50)

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Дунаев В. В. Базы даннях. Язык SQL / В. В. Дунаев. — СПб. : БХВ-Петербург, 2006. — 288 с.
2. Андон Ф. Язык запросов SQL. Учебный курс. / Ф. Андон, В. Резниченко. — СПб. : Питер; Киев: Издательская группа BHV, 2006. — 416 с.
3. Астахова И.Ф. SQL в примерах и задачах; Учеб. пособие / И.Ф. Астахова, А.П. Толстобров, В.М. Мельников. — М. : Новое знание, 2002. — 176 с.
4. Форта, Бен. Освой самостоятельно SQL. 10 минут на урок / Бен Форта ; [пер. с англ. В.С. Гусева]. — [3-е изд]. — М. : Издательский дом "Вильяме", 2005. — 288 с.
5. Грофф Д. Р. SQL: полное руководство / Д. Р. Грофф, Н. П. Вайнберг; [пер с англ. под редакцией В. Р. Гинсбурга]. — К. : BHV, 2001. — 816 с.
6. Реляційні бази даних: табличні алгебри та SQL-подібні мови / В. Н. Редько, Ю. Й. Брона, Д. Б. Буй, С.А. Поляков. — К. : Видавничий дім “Академперіодика”, 2001. — 198 с.
7. Гайджаржи В. І., Дацюк О. А. Основи проектування та використання баз данних : навчальний посібник / В. І. Гайджаржи, О. А. Дацюк. — [2 вид., виправл. і доповн]. — К. : Політехніка, 2004 . – 256 с.
8. Хернандес М. Д. SQL-запросы для простых смертных : практическое руководство по манипулированию данными в SQL / Майкл Дж. Хернандес, Джон Л. Вьескас; [пер. с англ. А. Головко]. — М. : Издательство “Лори”, 2003. — 473 с.
9. Автоинкрементные поля в MySQL [Електронний ресурс] – Доступний з: http://shpargalki.org.ua/144/avtoinkrementnye-polya-v-mysql
10. SQL Increment [Електронний ресурс] – Доступний з: http://dimonchik.com/sql-increment.html
11. Кузнецов С. Наиболее интересные новшества в стандарте SQL:2003 / Сергей Кузнецов. – 2004 г. – [Електронний ресурс] – Доступний з: http://citforum.univ.kyiv.ua/database/sql/sql2003/
12. Хернандес М. Д. SQL-запросы для простых смертных : практическое руководство по манипулированию данными в SQL / Майкл Дж. Хернандес, Джон Л. Вьескас; [пер. с англ. А. Головко]. — М. : Издательство “Лори”, 2003. — 473 с.

ЗМІСТ

УДК 004.655

Навчально-методичне видання
Л. В. Булатецька, В. В. Булатецький
МОВА ЗАПИТІВ SQL
Текст лекцій
нормативної навчальної дисципліни
“Бази даних та розподілені
інформаційно-аналітичні системи”
Гарн. Times. Обсяг 5,34 ум. друк. арк.
Наклад 30