МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«УЖГОРОДСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ»
МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА КІБЕРНЕТИКИ І ПРИКЛАДНОЇ МАТЕМАТИКИ

ОСНОВИ МОВИ ЗАПИТІВ SQL

Ужгород – 2015

Мулеса О.Ю. Основи мови запитів SQL. – Ужгород, 2015. – 48 с.

Рекомендовано до друку кафедрою кібернетики і прикладної математики ДВНЗ "Ужгородський національний університет", протокол № 10 від 12 червня 2015 р.

Рекомендовано до друку методичною комісією математичного факультету ДВНЗ "Ужгородський національний університет", протокол № 7 від 31 серпня 2015 р.

Рецензенти: **Млавець Ю.Ю.** кандидат фізико-математичних наук, доцент кафедри кібернетики і прикладної математики (ДВНЗ "Ужгородський національний університет")

Антосяк П.П., кандидат фізико-математичних наук, доцент кафедри системного аналізу і теорії оптимізації (ДВНЗ "Ужгородський національний університет")

ЗМІСТ

ОСНОВИ SQL

ІНСТРУКЦІЇ SQL

В SQL існує близько 40 інструкцій. Кожна з них звертається до СУБД за виконанням конкретної дії. Відповідно до призначення, інструкції поділяються на види:

– інструкції обробки даних;

– інструкції визначення даних;

– інструкції управління доступом;

– інструкції управління трансакціями;

– програмні інструкції.

Основні інструкції з обробки та представлення даних наведені в табл.1.:

Таблиця 1.

Інструкції з обробки та представлення даних SQL

Інструкція	Опис
SELECT	Отримує дані з однієї або декількох таблиць
INSERT	Додає нові рядки в таблицю
DELETE	Видаляє рядки з таблиці
UPDATE	Обновляє дані, які вже існують в таблиці
CREATE TABLE	Додає нову таблицю в БД
DROP TABLE	Видаляє таблицю з БД
ALTER TABLE	Змінює структуру існуючої таблиці

Кожна інструкція SQL починається з команди, тобто ключового слова, яке описує дію, що виконується інструкцією. Після команди йде одне або декілька речень. Речення описують дані, з якими працює інструкція, або містять інформацію про дію, яку виконує інструкція. Кожне речення також

починається з ключового слові, наприклад WHERE (де), FROM (звідки), INTO (куди). Деякі речення в інструкціях є обов’язковими, інші – ні. Велика частина речень містить імена таблиць чи стовпців, деякі з них можуть містити додаткові ключові слова, константи, вирази.

У кожного об’єкта в БД є унікальне ім'я: імена таблиць, імена стовпців тощо.

ТИПИ ДАНИХ

Найбільш поширені типи даних, які використовуються в SQL:

– ***цілі числа;***

– десяткові числа;

– числа з плаваючою крапкою;

– рядки символів сталої довжини;

– рядки символів змінної довжини;

–грошові величини;

– дата та час;

– булеві величини;

– довгий текст;

– неструктуровані потоки байтів;

– нелатинські символи.

Типи даних стандарту ANSI/ISO наведені в табл.2.

Таблиця 2

Типи даних в SQL

Тип даних	Опис
CHAR (довжина)	Рядки символів сталої довжини
CHARACTER (довжина)	
VARCHAR (довжина)	Рядки символів змінної довжини
CHAR VARYING (довжина)	
CHARACTER VARYING (довжина)	
NCHAR (довжина)	Рядки локалізованих символів сталої довжини
NATIONAL CHAR (довжина)	
NATIONAL CHARACTER (довжина)	
NCHAR VARYING (довжина)	Рядки локалізованих символів змінної довжини
NATIONAL CHAR VARYING (довжина)	

NATIONAL CHARACTER VARYING (довжина)	
INTEGER	Цілі числа
INT	
SMALLINT	Малі цілі числа
BIT (довжина)	Рядки бітів сталої довжини
BIT VARYING (довжина)	Рядки бітів змінної довжини
NUMERIC (точність, степінь)	Числа з плаваючою крапкою
DECIMAL (точність, степінь)	
DEC (точність, степінь)	
FLOAT (точність)	
REAL	Числа з плаваючою крапкою низької точності
DOUBLE PRECISION	Числа з плаваючою крапкою високої точності
DATE	Дата
TIME (точність)	Час
TIMESTAMP (точність)	Дата і час
INTERVAL	Часовий інтервал

ВБУДОВАНІ ФУНКЦІЇ

Найбільш корисні функції, які підтримуються в різних СУБД перераховані в табл. 3:

Таблиця 3

Деякі вбудовані функції стандарту SQL2

Функція	Значення
BIT_LENGTH(рядок)	Кількість біт в рядку
CAST (значення AS тип даних)	Перетворює у вказаний тип даних
CHAR_LENGTH(рядок)	Довжина рядку символів
CONVERT(рядок USING функція)	Рядок, перетворений у відповідності до вказаної функції
CURRENT_DATE	Поточна дата
CURRENT_TIME(точність)	Поточний час із вказаною точністю
CURRENT_TIMESTAMP(точність)	Поточні дата і час із вказаною точністю
DAY(дата)	День
EXTRACT (частина FROM значення)	Вказана частина (DAY, HOUR,…) із значення DATETIME
LOWER(рядок)	Рядок, переведений в нижній регістр
MONTH(дата)	Місяць
OCTET_LENGTH(рядок)	Кількість байт в рядку символів
POSITION(підрядок IN рядок)	Позиція, з якої починається входження підрядка в рядок
SUBSTRING(рядок FROM n FOR довжина)	Частина рядка, починаючи з n-го символу, вказаної довжини
TO_CHAR(дата, специфікація)	Перетворює дату відповідно до заданої специфікації
TRANSLATE(рядок USING функція)	Рядок, трансльований за допомогою

	вказаної функції
TRIM(BOTH символ FROM рядок)	Рядок, з якого видалені перші і останні вказані символи
TRIM(LEADING символ FROM рядок)	Рядок, з якого видалені перші вказані символи
TRIM(TRAILING символ FROM рядок)	Рядок, з якого видалені останні вказані символи
UPPER(рядок)	Рядок, перетворений у верхній регістр
YEAR(дата)	Рік

КОНСТАНТИ ДАТИ І ЧАСУ

В реляційних СУБД значення дати, часу та інтервалів часу представлені у виді рядкових констант. Формати цих констант в різних СУБД відрізняються один від одного. Крім того, способи запису дати і часу змінюються в залежності від країни. Деякі формати дати і часу наведені в табл.4.

Таблиця 4

Формати дати і часу в деяких реляційних СУБД

Формат	Формат значень типу DATE	Приклад значення типу DATE	Формат значень типу TIME	Приклад значення типу TIME
Американський	mm/dd/yyyy	5/19/1960	hh:mm am/pm	2:18 PM
Європейський	dd.mm.yyyy	19.5.1960	hh:mm.ss	14:18.08
Японський	yyyy-mm-dd	1960-5-19	hh:mm:ss	14:18:08
ISO	yyyy-mm-dd	1960-5-19	hh:mm:ss	14:18:08
TIMESTAMP	yyyy-mm-dd-hh.mm.ss.nnnnnn			

СТВОРЕННЯ ТАБЛИЦЬ

ІНСТРУКЦІЯ CREATE TABLE

Інструкція CREATE TABLE визначає нову таблицю та готує її до запису даних. Різні блоки інструкції задають елементи визначення таблиці. Синтаксична структура інструкції є такою:

CREATE TABLE *ім'я таблиці (визначення стовпця або визначення обмежень таблиці,….)*

Після виконання інструкції створюється нова таблиця. Створена таблиця є порожньою; додавати до неї записи можна за допомогою інструкції INSERT.

Визначення стовпців

Визначення стовпців являє собою розміщений в дужках список, елементи якого відділенні один від одного комами. Порядок слідування визначень стовпців в списку відповідає порядку стовпців в таблиці. Кожне визначення містить наступну інформацію:

– імя стовпця;

– тип даних стовпця;

– вказівка на те, чи обовязково стовпець має містити дані: якщо вказано обмеження NOT NULL, то значення NULL не може міститися в стовпці;

– значення за замовчуванням, яке заноситься в таблицю у тому випадку, якщо інструкція INSERT не містить значення даного стовпця.

Значення за замовчуванням та відсутні значення

У визначенні кожного стовпця вказується, чи допускається збереження в ньому значень NULL. Для задання значень елементів стовпців за замовчуванням, у їх визначенні використовується ключова інструкція DEFAULT таким чином:

ім'я стовпця *тип* DEFAULT *значення*

або

ім’я стовпця *тип* NOT NULL DEFAULT *значення*

Визначення первиного та зовнішнього ключів

В інструкції CREATE TABLE вказуєтсья також інформація про первинний ключ та її звязках з іншими таблицямя бази даних. Ця інформація міститься в частині PRIMARI KEY та FOREIGN KEY.

В частині PRIMARY KEY задається стовпець чи стовпці, які утворюють первинний ключ таблиці. Цей стопець чи стовпці є унікальними ідентифікаторами рядків таблиці. СУБД автоматично слідкує за тим, щоб первинний ключ кожного рядка таблиці містив унікальне значення. Крім того, у визначенні рядків первинного ключа має бути вказано, що вони не можуть містити значення NULL.

В частині FOREIGN KEY задається зовнішній ключ таблиці і визначається зв'язок, який задається. В ньому вказуються:

– стовпець чи стовпці створюваної таблиці, які утворюють зовнішній ключ;
– таблиця, зв'язок з якою створюється.

ПРОСТІ ЗАПИТИ

ІНСТРУКЦІЯ SELECT

Інструкція SELECT отримує інформацію з бази даних та повертає її у виді таблиці результатів запиту.

Синтаксична діаграма інструкції SELECT представлена НИЖЧЕ

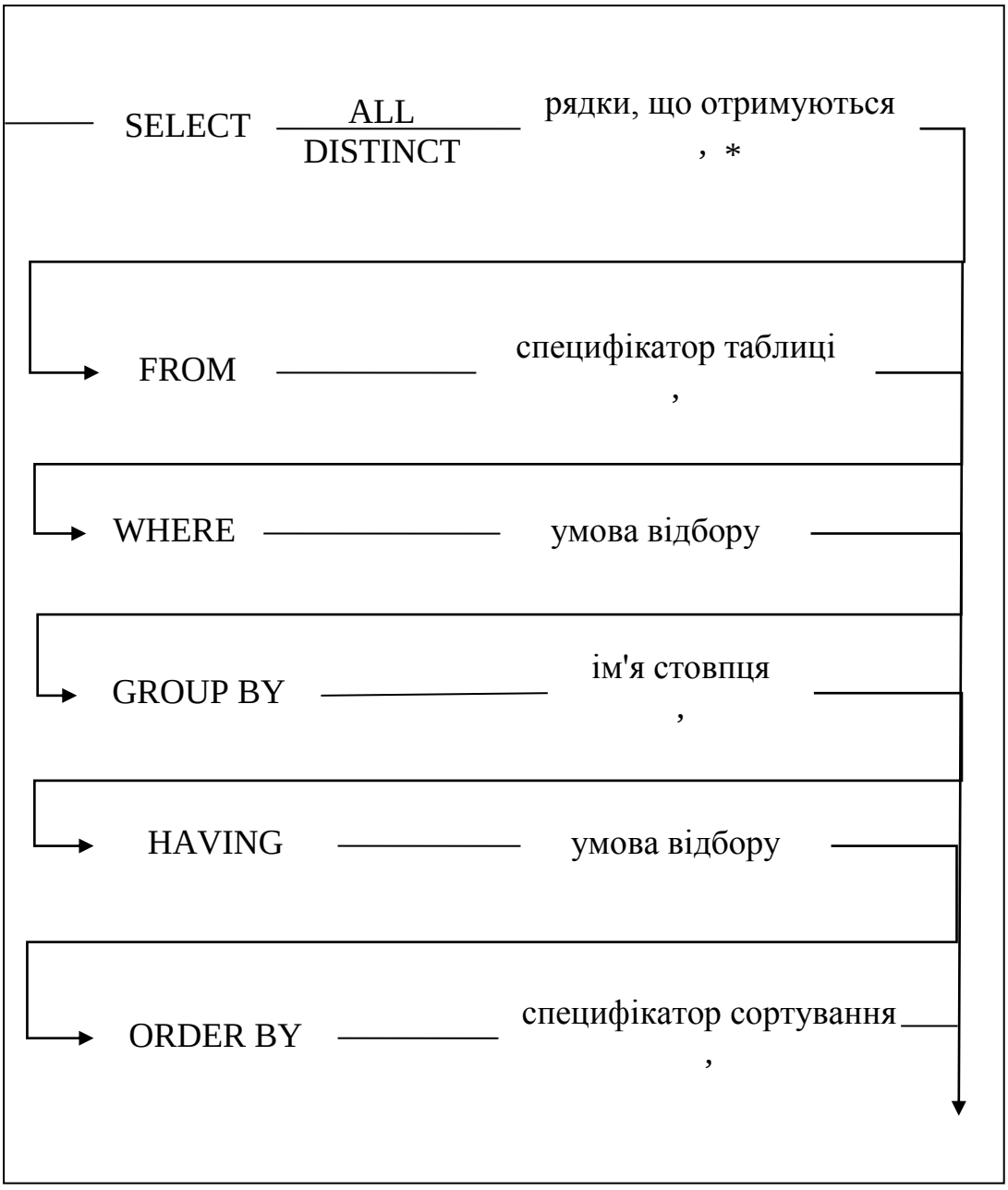

Рис.2. Синтаксична діаграма інструкції SELECT

В частині SELECT вказується список стовбців, які мають бути отримані у результаті виконання запиту. Стовпці можуть містити значення, отримані з стовпців таблиць бази даних, або можуть обчислюватися під час виконання запиту.

В частині FROM вказується список таблиць, які місять елементи даних, до яких звертається запит.

В частині WHERE міститься умова для відбору рядків, які будуть включені у результат запиту.

Блок GROUP BY дозволяє створити підсумковий запит. Звичайний запит включає в результати запиту по одному запису для кожного рядка із таблиці. Підсумковий запит, в свою чергу, спочатку групує рядки бази даних за визначеною ознакою, а потім включає в результат запиту один підсумковий рядок для кожної групи.

Блок HAVING показує, що в результаті запиту, необхідно включити тільки деякі групи, створені за допомогою GROUP BY. В цій частині для відбору груп використовується умова відбору.

Блок ORDER BY впорядковує результати запиту на основі даних, що містяться в одному чи декількох стовпцях.

Блок SELECT

В частині SELECT необхідно вказати елементи даних, які будуть отримані в результаті виконання запиту. Ці елементи задаються у вигляді списку стовпців, розділених комами. Для кожного елемента із цього списку в таблиці результатів буде створений стовпець. Стовпець результуючої таблиці може являти собою:

– ім'я стовпця, яке відповідає стовпцю однієї з таблиць, які перераховані в частині FROM;

– константу, яка показує, що в кожному рядку результату запиту має бути одне і те ж значення;

– вираз, який показує, що СУБД має обчислити значення за формулою, визначеною у виразі.

Вирази для обчислення значень певних стовпців можуть містити операції додавання, віднімання, множення та ділення. Тут також можна використовувати дужки.

Для того, щоб отримати всі стовпці таблиці, замість списку стовпців можна використовувати символ зірочки (*).

Якщо із таблиці-результату запиту необхідно прибрати рядки, які містять однакові значення, то в частині SELECT перед списком стовпців необхідно вказати предикат DISTINCT, що забезпечить уникнення повторів при виводі результату.

Блок FROM

Блок FROM містить список специфікаторів таблиць, розділених комами. Кожен специфіктор таблиці ідентифікує таблицю, що містить дані, які отримує запит.

Блок WHERE

Для того, щоб вказати які саме рядки необхідно відібрати при виконанні запиту, використоувється Блок WHERE. У ній записують умову відбору рядків. Дял кожного з рядяків умова відбору може мати одне з трьох значень:

– якщо умова має значення TRUE, то рядок включається в результат відбору;

– якщо умова приймає значення FALSE, то рядок виключається з результатів запиту;

– якщо умова має значення NULL, то рядок виключається із результатів відбору.

Існує багато умов відбору, які дозволяють ефективно створювати різні типи запитів. Основними умовами відбору є:

1. Порівняння. Значення одного виразу порівнюється із значенням іншого виразу для кожного рядка даних. Існує шість різних способів порівняння виразів:

= , < > , < , <= , > , >=.

Результататом виконання СУБД порівняння двох виразів може бути:

– якщо порівняння істинне, то результат перевірки має значення TRUE;

– якщо порівняння хибне, то результат перевірки має значення FALSE;

– яякщ хоча б один з двох виразів має значення NULL, то результатом перевірки буде NULL.

2. Перевірка на належність діапазону значень. Перевіряється чи потрапляє вказане значення в визначений діапазон. Схематично таку форму умови відбору можна зобразити так:

вираз, що перевіряється BETWEEN *нижня межа* AND *верхня межа*

або

вираз, що перевіряється NOT BETWEEN *нижня межа* AND *верхня межа*

При такій перевірці верхня та нижня межі вважаються частиною діапазону.

В деяких СУБД визначені такі правила обробки значення NULL в перевірці BETWEEN:

– якщо вираз, що перевіряється має значення NULL або якщо обидва виразів, які визначають діапазон, рівні NULL, то і перевірка BETWEEN повертає NULL;

– якщо вираз, що визначає нижню межу діапазону, має значення NULL, то перевірка BETWEEN повертає FALSE, коли значення, що перевіряється більше, ніж верхня межа діапазону, і NULL в протилежному випадку;

– якщо вираз, що визначає верхню межу діапазону, має значення NULL, то перевірка BETWEEN повертає FALSE, коли значення, що

перевіряється менше, ніж нижня межа діапазону, і NULL в протилежному випадку.

3. Перевірка на входження до множини. Перевіряється, чи співпадає значення виразу з одним із значень заданої множини. Схематично таку форму умови відбору можна зобразити так:

вираз, що перевіряється IN (*список констант через кому*)

або

вираз, що перевіряється NOT IN (*список констант через кому*)

4. Перевірка на відповідність шаблону. Перевіряється чи відповідає рядкове значення, яке міститься в стовпці певному шаблону. Схематично таку форму умови відбору можна зобразити так:

ім'я стовпця LIKE *шаблон*

або

ім'я стовпця NOT LIKE *шаблон*

Шаблон являє собою рядок, в який може входити один або більше підстановочних знаків. В SQL використовуються такі підстановочні знаки:

1) % – співпадає з будь-якою послідовністю з нуля чи більше символів;

2) _ (символ підкреслення) – співпадає з будь-яким окремим символом.

У випадку, коли підстановочний знак може виявитися елементом рядка, для побудови шаблону використовуються символи пропуску. Коли, в шаблоні зустрічається такий символ, символ, який слідує за ним, вважається не підстановочним. Структура умови в такому випадку є наступною:

ім'я стовпця LIKE *шаблон* ESCAPE *символ пропуску*

або

ім'я стовпця NOT LIKE *шаблон* ESCAPE *символ пропуску*

Наприклад, якщо шаблон містить символ %, то умова буде такою:

```
WHERE name LIKE 'A$%BC' ESCAPE '$'
```

У цьому випадку '$' є символом пропуску і символ %, який слідує після нього є простим елементом рядка.

5. Перевірка на рівність значенню NULL. Значення NULL дозволяє застосовувати тризначну логіку в умовах відбору. У випадку, коли необхідно явно перевірити значення стовпців на рівність NULL використовується така структура умови:

ім'я стовпця IS NULL

або

ім'я стовпця IS NOT NULL

Дана перевірка завжди повертає значення TRUE або FALSE.

Перераховані прості умови відбору, після застосування до деякого рядка повертають значення TRUE, FALSE або NULL. За допомогою правил логіки ці прості умови можна об’єднувати в більш складні, використовуючи при цьому логічні операції AND, OR, NOT. Їх таблиці істинності наведені нижче:

Таблиця 5

Таблиця істинності оператора AND

AND	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

Таблиця 6

Таблиця істинності оператора OR

OR	**TRUE**	**FALSE**	**NULL**
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

Таблиця 7

Таблиця істинності оператора NOT

NOT	**TRUE**	**FALSE**	**NULL**
	FALSE	TRUE	NULL

Оператор NOT володіє найвищим пріоритетом, наступний пріоритет має оператор AND, найнижчий – OR.

Блок ORDER BY

Для впорядкування результатів запиту використовується блок ORDER BY. Структура блоку є такою:

ORDER BY *ім'я чи порядковий номер стовпця* ASC/DESC

При впорядкуванні можна обирати зростаючий чи спадний порядок. За замовчуванням дані сортуються по зростанню. Щоб сортування відбувалося по спаданню, необхідно в речення включити ключове слово DESC. Відповідно, ключове слово ASC задає впорядкування по зростанню.

ПІДСУМКОВІ ЗАПИТИ НА ВИБІРКУ

СТАТИСТИЧНІ ФУНКЦІЇ

Для проведення підсумків по інформації, яка міститься в базі даних, застосовуються статистичні (агрегатні) функції. Статистична функція приймає в якості аргументу будь-який стовпець даних, а повертає одне значення. Існують такі статистичні функції:

Таблиця 8.

Статистичні функції

Функція	Значення
SUM( )	обчислює суму всіх значень стовпця
AVG( )	обчислює середнє всіх значень стовпця
MIN( )	знаходить найменше серед всіх значень стовпця
MAX( )	знаходить найбільше серед всіх значень стовпця
COUNT( )	підраховує кількість значень, що містить стовпець
COUNT(*)	підраховує кількість рядків в таблиці результату запиту

Аргументом статистичної функції може бути ім'я стовпця або арифметичний вираз.

Структура статистичних функцій наведена нижче:

– SUM (*вираз*) або SUM (DISTINCT *ім'я стовпця*)

– AVG (*вираз*) або AVG (DISTINCT *ім'я стовпця*)

– MIN (*вираз*)

– MAX (*вираз*)

– COUNT (*вираз*) або COUNT (DISTINCT *ім'я стовпця*)

– COUNT (*)

Для видалення рядків, що повторюється використовується предикат DISTINCT. Проте, в одному запиті цей предикат можна використовувати не більше одного разу.

Запити з групуванням (блок GROUP BY)

Запит, який включає в себе блок GROUP BY, називається запитом з групуванням, оскільки він об’єднує рядки початкових таблиць в групи і для кожної групи рядків генерує один рядок в таблиці результатів запиту. Стовпці, вказані в цьому блоці, називаються стовпцями групування.

На запити, в яких використовується групування, накладаються додаткові обмеження. Стовпці з групуванням мають бути реальними стовпцями таблиць, перерахованими в блоці FROM. Не можна групувати рядки на основі виразу, значення якого обчислюється.

Умови відбору груп (блок HAVING)

Так само як блок WHERE використовується для відбору окремих рядків, які беруть участь у запиті, блок HAVING можна використовувати для відбору груп рядків. Його формат відповідає формату блоку WHERE.

Блок HAVING майже завжди використовується з блоком GROUP BY, проте синтаксис запиту SELECT цього не вимагає. Якщо блок HAVING використовується без блоку GROUP BY, то СУБД розглядає результати запиту як одну групу.

ВНЕСЕННЯ ЗМІН В БАЗУ ДАНИХ

ДОДАВАННЯ НОВИХ ДАНИХ (ІНСТРУКЦІЯ INSERT)

Існує декілька способів додавання нових рядків в БД, серед них:

– однорядкова інструкція INSERT, яка дозволяє додати в таблицю один новий рядок;

– багаторядкова інструкція INSERT, забезпечує отримання рядків із одної частини БД та додавання їх в іншу частину.

Інструкція INSERT додає в таблицю новий рядок або групу рядкців. При цьому, значення стовпців можуть являти собою константи, а можуть бути результатом виконання підзапиту. У першому випадку для вставки кожного рядка виконується окремий оператор, а в другому – буде додано стільки рядків, скільки повертає підзапит.

Синтаксична структура інструкції INSERT наведена нижче:

INSERT INTO *ім'я таблиці* (*ім'я стовпця1*,…)
VALUES (*значення 1*,…)

або

INSERT INTO *ім'я таблиці* (*ім'я стовпця1*,…)
підзапит

або

INSERT INTO *ім'я таблиці* (*ім'я стовпця1*,…)
DEFAULT VALUES

Варто зазначити, у тому випадку, коли список стовпців запису співпадає з структурою таблиці, тобто запис забезпечує значення для всіх стовпців, зберігаючи їх порядок, – перераховувати назви стовпців необов’язково. Тоді структура інструкції матиме вид:

INSERT INTO *ім'я таблиці* VALUES (*значення 1*,…)

або

INSERT INTO *ім'я таблиці* *підзапит*

або

INSERT INTO *ім'я таблиці* DEFAULT VALUES

У випадку багаторядкової інструкції INSERT, на вкладений запит, як правло, накладаються деякі обмеження:

– в запит не модна включати частину ORDER BY;

– таблиця результатів запиту має містити таку ж кількість стовпців, як і список стовпців в інструкції INSERT.

ВИДАЛЕННЯ ІСНУЮЧХ ДАНИХ (ІНСТРУКЦІЯ DELETE)

Інструкція DELETE видаляє вибрані рядки з одної таблиці. Синтаксична структура інструкції є такою:

DELETE *ім'я таблиці* WHERE *умова відбору*

У блоці FROM вказується таблиця, що містить рядки, які потрібно видалити. У блоці WHERE вказують критерії відбору рядків, які будуть видалені. У випадку, коли блок WHERE відсутній – інструкція видаляє всі рядки таблиці.

Також, допустимим є використання вкладеного запиту для задання умови відбору.

ОБНОВЛЕННЯ ІСНУЮЧИХ ДАНИХ (ІНСТРУКЦІЯ UPDATE)

Інструкція UPDATE обновляє значення одного чи декількох стовпців у вибраних рядках однієї таблиці. Синтаксична структура інструкції є такою:

UPDATE *ім'я таблиці* SET *ім'я стовпця=вираз,....*
WHERE *умова відбору*

В інструкції вказується цільова таблиця, яка має бути модифікована. В блоці SET вказується, які стовпці мають бути модифікованими. Блок WHERE відбирає рядки таблиці для обновлення. Якщо цей блок відсутній, то обновляться всі рядки таблиці.

Також, допустимим є використання вкладених запитів для задання умов відбору рядків.

ВИДАЛЕННЯ ТАБЛИЦІ (ІНСТРУКЦІЯ DROP TABLE)

Видалення таблиці виконується за допомогою інструкції DROP TABLE, синтаксична структура якої є наступною:

DROP TABLE *ім'я таблиці*

Інструкція містить ім'я таблиці, яка видаляється. Після виконання інструкції, визначення бази даних та весь її вміст втрачаються і не можуть бути відновленими автоматично.

ЗМІНА ВИЗНАЧЕННЯ ТАБЛИЦІ (ІНСТРУКЦІЯ ALTER TABLE)

За допомогою інструкції ALTER TABLE можна виконати такі дії:

- додати в таблицю стовпці;
- видалити стовпці з таблиці;
- змінити значення за замовчуванням для стовпця;
- додати чи видалити первинний ключ таблиці;
- додати чи видалии зовнішній ключ таблиці;
- додати чи видалити умову унікальності;
- додати чи видалити умову на значення.

Синтаксична структура інструкції є такою:

ALTER TABLE *ім'я таблиці змістовний блок інструкції*

Додавання стовпця

Для додавання стовпця до існуючої таблиці до інструкції ALTER TABLE дописують блок ADD та частину з визначенням стовпця в такому ж форматі, як в інструкції CREATE TABLE. Новий стовпець додається в кінець таблиці. Синтаксична структура інструкції в цьому випадку має вид:

ALTER TABLE *ім'я таблиці* ADD *визначення стовпця*

Видалення стовпця

За допомогою інструкції ALTER TABLE для видалення стовпця необхідно використати блок DROP. Синтаксична структура інструкції є такою:

ALTER TABLE *ім'я таблиці* DROP *ім'я стовпця*

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. А.Н.Наумов, А.М.Вендров і ін., "Системи керування базами даних і знань", М.:Фінанси і статистика, 1991р.
2. Аткинсон, Леон MySQL. Библиотека профессионала; М.: Вильямс, 2008. - 624 с.
3. Бекаревич Ю.Б., Пушкина Н.В. Microsoft Access 2000.-СПб.: БХВ – Санкт- Петербург, 1999.- 480 с., ил.
4. В.В. Кириллов, Г.Ю. Громов. Учебное пособие по SQL: Структурированный язык запросов (SQL). http://www.citforum.ru/database/sql_kg/index.shtml
5. Грофф Дж. Энциклопедия SQL // Дж. Грофф, П. Вайнберг. – СПб.: Питер, 2003. – 896 с.
6.. Грофф, Джеймс; Вайнберг, Пол SQL: полное руководство; Киев: BHV, 2005. - 608 с.
7. Дейт К., "Введення в системи баз даних", М.: Наука, 1980 р.
8. Дж.Ульман, "Основи систем баз даних", М.:Фінанси і статистика,1983р.
9. Джен Л. Харрингтон. Проектирование реляционных баз даннях. − М.: Издательство «Лори», 2006. − 230 с.
10. К. Дж. Кейт. Введення в системи баз даних/ пер. с англ.− 8-е изд. − М.: Издательский дом «Вильямс», 2006.– 1328 С.
11. Киммел Пол. Освой самостоятельно программирование для Microsoft Access 2002 за 24 часа.: пер. с англ.- М.:Издательский дом «Вильямс», 2003.- 480 с.: илл.-Парал. тит. англ.
12. Кириллов В.В. Основы проектирования реляционных баз даннях: учебное пособие. - СПб.: ИТМО, 1994. – 90 с.
13. Кузнецов С. Д. Основы баз данных. - М.: БИНОМ. Лаборатория знаний, 2007 г.
14. Кумскова И. Базы данных. - Кнорус, 2010 г.
15. Марков А. С. Лисовский К.Ю. Базы данных. Введение в теорию и

методологию. - Финансы и статистика, 2006 г.

16. Мартин Грубер. Понимание SQL. /пер. В.Н. Лебедева.− М., 1993.– 291 с.
17. Нанда, А. и др. Oracle PL/SQL для администраторов баз данных; Символ, 2008. - 496 с.
18. Пасічник В.В.Організація баз даних та знань: підручник для ВНЗ/ В.В. Пасічник, В.А. Резніченко.-К.: Видавнича група BHV,2006.-384с.
19. Пушников А.Ю. Введение в системы управления базами данных. Часть 1. Реляционная модель данных: учебное пособие/ Изд-е Башкирского ун-та. - Уфа, 1999. - 108 с.
20. Пушников А.Ю. Введение в системы управления базами данных. Часть 2. Нормальные формы отношений и транзакции: учебное пособие/ Изд-е Башкирского ун-та. - Уфа, 1999. - 138 с.
21. С.М.Диго "Проектирование и использования баз данных". Москва: Финансы и статистика 1995.
22. Стоунз, Ричард; Мэттью, Нейл PostgreSQL. Основы; СПб: Символ-Плюс, 2007. - 640 с.
23. Томас Коннолли, Каролин Бегг. Базы данных. Проектирование, реализация и сопровождение. Теория и практика.– 3-е изд.− М.: Издательский дом «Вильямс», 2003. − 1436 с.
24. Фейерштейн, С.; Прибыл, Б. Oracle PL/SQL для профессионалов; СПб: Питер, 2005. - 941 с.
25. Шнайдер, Роберт Microsoft SQL Server 6.5. Проектирование высокопроизводительных баз данных; М.: Лори, 2010. - 361 с.
26. Яргер, Р.Дж.; Риз, Дж.; Кинг, Т. MySQL и mSQL: Базы данных для небольших предприятий и Интернета; СПб: Символ-Плюс, 2013. - 560 с.

ДОДАТКИ

БАЗОВІ ПОНЯТТЯ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

Основними поняттями реляційних баз даних є: тип даних, домен, атрибут, кортеж, первинний ключ і відношення.

Поняття *тип даних* в реляційній моделі даних повністю адекватно поняттю типу даних в мовах програмування. Зазвичай всі сучасні реляційні БД підтримують наступні типи даних:

- числові;
- символьні;
- великі двійкові об'єкти (малюнки та медіа-файли);
- бітові рядки;
- спеціалізовані числові дані (такі як «гроші»);
- спеціальні «темпоральні дані» (дата, час та часовий інтервал).

Домен – це семантичне поняття. Домен можна розглядати як підмножину значень деякого типу даних, які мають певний сенс. Домен характеризується наступними властивостями:

- домен має унікальну назву (в межах бази даних);
- домен визначений на деякому простому типі даних або на іншому домені;
- домен може мати деяку логічну умову, що дозволяє описати підмножину даних, допустимих для даного домену;
- домен несе певне смислове навантаження.

Кортеж, що відповідає даній схемі відношення, - це множина пар {назва атрибута, значення}, яке містить одне входження кожної назви атрибута, що належить схемі відношення. «Значення» є припустимим значенням домену цього атрибута (або типу даних, якщо поняття домену не підтримується). Попросту кажучи, кортеж - це набір іменованих значень заданого типу.

Відношення - це множина кортежів, які відповідають одній схемі відношення. Насправді, поняття схеми відношення найближче до поняття структурного типу даних в мовах програмування.

Відношення зазвичай записується у вигляді:

$R(<A_1: D_1>, <A_2: D_2>, ..., <A_n: D_n>)$,

або

$R(A_1, A_2, ..., A_n)$.

Число атрибутів у відношенні називають *степенем* (або -*арністю*) відношення.

Потужність множини кортежів відношення називають *потужністю відношення*.

Тоді, *реляційною базою даних* називається набір відношень.

Звичайним для користувача поданням відношення є таблиця, заголовком якої є схема відношення, а рядками - кортежі відношення-екземляру; в цьому випадку назви атрибутів іменують стовпці цієї таблиці. Тому іноді кажуть «стовпець таблиці», маючи на увазі «атрибут відношення». Такої термінології дотримуються в більшості комерційних реляційних СКБД.

Схема відношення - це іменована множина пар {назва атрибута, назва домену (або типу, якщо поняття домену не підтримується)}. Степінь або «арність» схеми відношень - потужність цієї множини. Схема БД (в структурному сенсі) - це набір іменованих схем відношень.

Схема БД - це пойменована сукупність схем відношень, які входять до неї.

Фундаментальні властивості відношень:

– *відсутність кортежів-дублікатів* - відношення не містить кортежів-дублікатів, випливає з визначення відношення як множини кортежів. У класичній теорії множин по визначенню кожна множина складається з різних елементів. З цієї властивості випливає наявність у кожного відношення так званого первинного ключа - набору атрибутів,

значення яких однозначно визначають кортеж відношення. Поняття *первинного ключа* є виключно важливим у зв'язку з поняттям цілісності баз даних. Зауважимо, що в багатьох практичних реалізаціях РСУБД допускається порушення властивості унікальності кортежів для проміжних відношень, породжуваних неявно при виконанні запитів. Такі відношення є *не множинами, а мультимножинами*, що в ряді випадків дозволяє добитися певних переваг, але іноді призводить до серйозних проблем;

– *відсутність упорядкованості кортежів* також є наслідком визначення відношення-екземпляра як множини кортежів. Відсутність вимоги до підтримання порядку на множині кортежів відношення дає додаткову гнучкість СУБД при зберіганні баз даних у зовнішній пам'яті та при виконанні запитів до бази даних. Це не суперечить тому, що при формулюванні запиту до БД, наприклад, на мові SQL можна вказати сортування результуючої таблиці у відповідності зі значеннями деяких стовпців. Такий результат, взагалі кажучи, не відношення, а деякий впорядкований список кортежів;

– *відсутність упорядкованості атрибутів* - атрибути відношень не впорядковані, оскільки за визначенням схема відношення є множиною пар {назва атрибута, назва домену}. Для посилання на значення атрибута в кортежі відношення завжди використовується назва атрибута. Ця властивість теоретично дозволяє, наприклад, модифікувати схеми існуючих відношень не тільки шляхом додавання нових атрибутів, але і шляхом видалення існуючих атрибутів. Однак у більшості існуючих систем така можливість не допускається, і хоча впорядкованість набору атрибутів відношення явно не потрібно, часто в ролі неявного порядку атрибутів використовується їх порядок у лінійній формі визначення схеми відношення;

атомарність значень атрибутів - значення всіх атрибутів є атомарними. Це випливає з визначення домену як потенційної множини значень простого типу даних, тобто серед значень домену не можуть

міститися множини значень (відношення). Прийнято говорити, що в реляційних базах даних допускаються тільки *нормалізовані відношення* або відношення, які представлені в *першій нормальній формі*. *Нормалізовані відношення* становлять основу класичного реляційного підходу до організації баз даних. Вони мають деякі обмеження (не будь-яку інформацію зручно представляти у вигляді плоских таблиць), але істотно спрощують маніпулювання даними.

РЕЛЯЦІЙНА МОДЕЛЬ ДАНИХ

Реляційна модель даних — логічна модель даних. Вперше була запропонована британським ученим співробітником компанії IBM Едгаром Франком Коддом (E. F. Codd) в 1970 році в статті «A Relational Model of Data for Large Shared Data Banks». В даний час ця модель є фактичним стандартом, на який орієнтуються практично всі сучасні комерційні системи управління базами даних (СУБД).

У реляційній моделі досягається більш високий рівень абстракції даних, ніж в ієрархічній або мережевій. У згаданій статті Е. Ф. Кодда стверджується, що «реляційна модель надає засоби опису даних на основі тільки їх природної структури, тобто без потреби введення якоїсь додаткової структури для цілей машинного представлення». Іншими словами, подання даних не залежить від способу їх фізичної організації. Це забезпечується за рахунок використання математичного поняття відношення.

До складу реляційної моделі даних зазвичай включають теорію нормалізації. Крістофер Дейт визначив три складові частини реляційної моделі даних:

- структурна
- маніпуляційна
- цілісна

Структурна частина моделі визначає, що єдиною структурою даних є нормалізоване n-арне відношення. Відношення зручно представляти у формі таблиць, де кожен рядок є кортеж, а кожен стовпець — атрибут, визначений на деякому домені. Даний неформальний підхід до поняття відношення дає більш звичну для розробників і користувачів форму представлення, де реляційна база даних являє собою кінцевий набір таблиць.

Маніпуляційна частина моделі визначає два фундаментальних механізми маніпулювання даними — реляційну алгебру і реляційне числення. Основною функцією маніпуляційної частини реляційної моделі є

забезпечення заходів реляційності будь-якої конкретної мови реляційних БД: мова називається реляційною, якщо вона має не меншу виразність і потужність, ніж реляційна алгебра або реляційне числення.

Цілісна частина моделі визначає вимоги цілісності сутностей і цілісності посилань. Перша вимога полягає в тому, що будь-який кортеж будь-якого відношення відмінний від будь-якого іншого кортежу цього відношення, тобто іншими словами, будь-яке відношення має володіти первинним ключем. Вимога цілісності щодо посилань, або вимога зовнішнього ключа полягає в тому, що для кожного значення зовнішнього ключа, що з'являється у відношенні, на яке веде посилання, повинен знайтися кортеж з таким же значенням первинного ключа, або значення зовнішнього ключа повинно бути невизначеним (тобто ні на що не вказувати).

Можна провести аналогію між елементами реляційної моделі даних і елементами моделі «сутність-зв'язок». Реляційні відносини відповідають наборам сутностей, а кортежі — сутностям. Тому, як і в моделі «сутність-зв'язок», стовпці в таблиці, що представляє реляційне відношення, називають атрибутами.

Кожен атрибут визначений на домені, тому домен можна розглядати як множина допустимих значень даного атрибуту. Кілька атрибутів одних відношень і навіть атрибути різних відношень можуть бути визначені на одному і тому ж домені.

Іменована множина пар «ім'я атрибута — ім'я домену» називається схемою відношення. Потужність цієї множини — називають ступенем чи «арністю» відносини. Набір іменованих схем відносин являє собою схему бази даних.

Атрибут, значення якого однозначно ідентифікує кортежі, називається ключовим (або просто ключем). Якщо кортежі ідентифікуються тільки зчепленням значень декількох атрибутів, то говорять, що відношення має складовий ключ. Ставлення може містити кілька ключів. Завжди один із

ключів оголошується первинним, його значення не можуть оновлюватися. Всі інші ключі відносини називаються можливими ключами.

На відміну від ієрархічної і мережної моделей даних в реляційної відсутнє поняття групових відношень. Для відображення асоціацій між кортежами різних відносин використовується дублювання їх ключів.

Переваги реляційної моделі:

– простота і доступність для розуміння користувачем. Єдиною використовуваною інформаційною конструкцією є «таблиця»;

– суворі правила проектування, які базуються на математичному апараті;

– повна незалежність даних. Зміни в прикладній програмі при зміні реляційної БД мінімальні;

– для організації запитів і написання прикладного ПЗ немає необхідності знати конкретну організацію БД у зовнішній пам'яті.

Недоліки реляційної моделі:

– далеко не завжди предметна область може бути представлена у вигляді «таблиць»;

в результаті логічного проектування з'являється множина «таблиць». Це призводить до труднощів розуміння структури даних;

– БД займає відносно багато зовнішньої пам'яті;

– відносно низька швидкість доступу до даних.

Реляційна база даних — база даних, основана на реляційній моделі даних. Для роботи з реляційними БД застосовуютьреляційні СУБД. Інакше кажучи, реляційна база даних — це база даних, яка сприймається користувачем як набір нормалізованих відношень різного ступеню.

Використання реляційних БД було запропоноване Едгаром Коддом в 1970 році.

ПРАВИЛА КОДДА

12 правил Кодда — набір 13 правил (пронумерованих від нуля до дванадцяти) запропонованих Едгаром Коддом, піонером реляційної моделі для баз даних, спроектовані для визначення того чи є СУБД реляційною.

Правила настільки суворі, що всі популярні так звані «реляційні» СУБД не відповідають багатьом критеріям. Особливо складні 6, 9, 10, 11 і 12 правила.

0. **Фундаментальне правило** (Foundation Rule)

Реляційна СУБД має бути здатною повністю керувати базою даних, використовуючи зв'язки між даними

1. **Інформаційне правило** (Information Rule)

Інформація має бути представлена у вигляді даних, що зберігаються в осередках. Дані, що зберігаються у комірках, мають бути атомарними. Порядок рядків у реляційній таблиці не повинен впливати на зміст даних

2. **Правило гарантованого доступу** (Guaranteed Access Rule)

Доступ до даних має бути вільним від двозначності. До кожного елементу даних має бути гарантований доступ за допомогою комбінації імені таблиці, первинного ключа рядку й імені стовпця.

3. **Систематична обробка Null-значень** (Systematic Treatment of Null Values)

Невідомі значення NULL, відмінні від будь-якого відомого значення, мають підтримуватись для всіх типів даних при виконанні будь-яких операцій. Наприклад, для числових даних невідомі значення не повинні розглядатись як нулі, а для символьних даних — як порожні рядки.

4. **Правило доступу до системного каталогу на основі реляційної моделі** (Dynamic On-line Catalog Based on the Relational Model)

Словник даних має зберігатись у формі реляційних таблиць, і СУБД повинна підтримувати доступ до нього за допомогою стандартних мовних

засобів, тих самих, що використовуються для роботи з реляційними таблицями, які містять дані користувача.

5. **Правило повноти підмови маніпулювання даними** (Comprehensive Data Sublanguage Rule)

Система управління реляційними базами даних має підтримувати хоча б одну реляційну мову, яка

а) має лінійний синтаксис,

б) може використовуватись інтерактивно і в прикладних програмах,

в) підтримує операції визначення даних, визначення уявлень, маніпулювання даними (інтерактивні та програмні), обмежувачі цілісності, управління доступом та операції управління транзакціями (begin, commit і rollback).

6. **Правило модифікації поглядів** (View Updating Rule)

Кожне подання має підтримувати усі операції маніпулювання даними, які підтримують реляційні таблиці: операції вибірки, вставки, модифікації і видалення даних.

7. **Правило високорівневих операцій модифікації даних** (High-level Insert, Update, and Delete)

Операції вставки, модифікації і видалення даних мають підтримуватись не тільки щодо одного рядку реляційної таблиці, але й щодо будь-якої безлічі рядків.

8. **Правило фізичної незалежності даних** (Physical Data Independence)

Додатки не повинні залежати від використовуваних способів зберігання даних на носіях, від апаратного забезпечення комп'ютерів, на яких знаходиться реляційна база даних.

9. **Правило логічної незалежності даних** (Logical Data Independence)

Представлення даних в додатку не повинно залежати від структури реляційних таблиць. Якщо в процесі нормалізації одна реляційна таблиця розділяється на дві, подання має забезпечити об'єднання цих даних, щоб зміна структури реляційних таблиць не позначалась на роботі додатків.

10. **Правило незалежності контролю цілісності** (Integrity Independence)

Вся інформація, необхідна для підтримки цілісності, має бути у словнику даних. Мова для роботи з даними має виконувати перевірку вхідних даних і автоматично підтримувати цілісність даних.

11. **Правило незалежності від розміщення** (Distribution Independence)

База даних може бути розподіленою, може перебувати на кількох комп'ютерах, і це не повинно впливати на додатки. Перенесення бази даних на інший комп'ютер не повинне впливати на додатки.

12. **Правило узгодженості мовних рівнів** (The Nonsubversion Rule)

Якщо використовується низькорівнева мова доступу до даних, вона не повинна ігнорувати правила безпеки і правила цілісності, які підтримуються мовою більш високого рівня.

МОДЕЛЬ "СУТНІСТЬ-ЗВ'ЯЗОК"

Модель "сутність-зв'язок" (ER-модель) (англ. *Entity-relationship model або entity-relationship diagram*) – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків. ER-модель – це мета-модель даних, тобто засіб опису моделей даних.

ER-модель зручна при проектуванні інформаційних систем, баз даних, архітектур комп'ютерних застосунків та інших систем (моделей). За допомогою такої моделі виділяють найсуттєвіші елементи (вузли, блоки) моделі і встановлюють зв'язки між ними.

Існує ряд моделей для представлення знань. Одним з найзручніших інструментів уніфікованого представлення даних, незалежного від реалізовуючого його програмного забезпечення, є модель "сутність-зв'язок" (*entity - relationship model, ER - model*).

Модель "сутність-зв'язок" ґрунтується на якійсь важливій семантичній інформації про реальний світ і призначена для логічного представлення даних. Вона визначає значення даних в контексті їх взаємозв'язку з іншими даними. Важливим для нас є той факт, що з моделі "сутність-зв'язок" можуть бути породжені всі існуючі моделі даних (ієрархічна, мережева, реляційна, об'єктна), тому вона є найбільш загальною. Будь-який фрагмент наочної області може бути представлений як безліч сутностей, між якими існує деяка безліч зв'язків.

ER-модель – це одна з найпростіших візуальних моделей. Вона дозволяє осягнути структуру об'єкта «крупними мазками», в загальних рисах. Такий загальний опис структури називається ER-діаграмою або онтологією вибраної предметної області (area of interest).

Сутність (*entity*) – це об'єкт, який може бути ідентифікований якимись способом, що відрізняє його від інших об'єктів. Приклади: конкретна людина, підприємство, подія і т.д.

Набір сутностей (*entity set*) – множини сутностей одного типу (що володіють однаковими властивостями). Сутність фактично є множиною атрибутів, які описують властивості всіх членів даного набору сутності

Тип сутності визначає множину однорідних об'єктів, а «екземпляр сутності» - конкретний об'єкт з множини об'єктів.

Атрибут – поіменована характеристика сутності. За допомогою атрибутів моделюються властивості сутностей. Основна роль атрибутів – опис властивостей сутності. Інша роль – ідентифікація екземпляра сутності. Тобто кожен екземпляр сутності повинен мати унікальну назву. Як назва виступає один або декілька атрибутів.

Зв'язок – засіб подання відношень між сутностями в моделі пердметної області. Тип зв'язку розглядається між типами сутностей.

Властивості зв'язків у моделі сутність-зв'язок:

- підтримуються лише бінарні зв'язки. У загальному випадку в предметній області можуть бути N-арні зв'язки між сутностями;
- наявність ключа;
- відсутність агрегатів (щодо атрибутів);
- відсутність повторюваних груп (щодо атрибутів).

Коли задаються атрибути, ми в змозі визначити тип даних. Нові типи даних називаються *доменами (Domain)*.

ТИПИ ЗВ'ЯЗКІВ

Зв'язок між таблицями встановлює стосунки між співпадаючими значеннями в ключових полях.

У більшості випадків із ключовим полем однієї таблиці (головної таблиці), що є унікальним ідентифікатором кожного запису, зв'язується зовнішній ключ іншої таблиці (підлеглої таблиці).

Зовнішній ключ – одне (або декілька) полів у таблиці, що містять посилання на поле (або поля) первинного ключа в іншій таблиці.

Поле зовнішнього ключа визначає спосіб зв'язування таблиць. Вміст поля зовнішнього ключа повинен збігатися з умістом ключового поля, хоча імена полів можуть при цьому не збігатися.

Міжтабличний зв'язок – це відношення, встановлені між полями (стовпцями) двох таблиць.

В моделі «Сутність-зв'язок» використовуються бінарні зв'язки, яких може бути три види:

– один до одного 1:1;

– один до багатьох 1:M;

– багато до багатьох M:N.

При *відношенні «один-до-багатьох»* кожному запису в таблиці **A** можуть відповідати кілька записів у таблиці **B**, але запис у таблиці **B** не може мати більш одного відповідного йому запису в таблиці **A**.

При *відношенні «один-до-одного»* запис у таблиці **A** може мати не більш одного зв'язаного запису в таблиці **B** і навпаки.

При *відношенні «багато-до-багатьох»* одному запису в таблиці **A** можуть відповідати кілька записів у таблиці **B**, а одному запису в таблиці **B** кілька записів у таблиці **A**.

Цілісність даних визначає систему правил, які використовуються для підтримки зв'язків між записами у зв'язаних таблицях, а також забезпечують захист від випадкового видалення або зміни зв'язаних даних.

Для відносин, у яких перевіряється цілісність даних, користувач має можливість указати, чи варто автоматично виконувати для зв'язаних записів операції каскадного відновлення і каскадного видалення. Тоді зміни у даних головної таблиці автоматично відобразяться у всіх підлеглих.

ЦІЛІСНІСТЬ ДАНИХ

Термін цілісність даних відноситься до повноти інформації, яка міститься в базі даних. При зміні вмісту бази даних за допомогою інструкцій INSERT, DELETE, UPDATE може виникнути порушення цілісності даних, що в ній містяться. Для збереження несуперечності та правильності інформації, що зберігається в реляційній СУБД встановлюється одна або декілька умов цілісності даних. Ці умови визначають які значення можуть бути записані в базу даних в результаті додавання чи обновлення даних. Як правило, в реляційних базах даних використовують такі умови цілісності даних:

– *Обов'язкова наявність даних*. Деякі стовпці в базі даних повинні містити значення кожному рядку; рядки в таких стовпцях не можуть включати значення NULL або не містити ніякого значення.

– *Умова на значення*. У кожного стовпця є свій домен, тобто набір значень, які можна зберігати в даному стовпці. Можна вказати СУБД, що запис значень, які не входять в певний діапазон, в такі стовпці є недопустимим.

– *Цілісність таблиці*. Первинний ключ має в кожному рядку мати унікальне значення, яке відрізняється від значень у всіх інших рядках. Повтор значень в таких стовпцях є недопустимим, так як СУБД не зможе відрізняти один запис від іншого.

– *Цілісність за посиланням*. В реляційній базі даних кожен рядок таблиці-нащадка за допомогою первинного ключа пов'язаний з рядком таблиці-предка, яка містить первинний ключ, значення якого рівне значенню зовнішнього ключа.

НОРМАЛІЗАЦІЯ БАЗИ ДАНИХ

В теорії реляційних баз даних звичайно виділяється наступна послідовність нормальних форм:

- перша нормальна форма (1NF);
- друга нормальна форма (2NF);
- третя нормальна форма (3NF);
- нормальна форма Бойса-Кодда (BCNF);
- четверта нормальна форма (4NF);
- п'ята нормальна форма, або нормальна форма проекції-з'єднання (5NF або PJ / NF).

Основні властивості нормальних форм:

– кожна наступна нормальна форма в деякому сенсі краще попередньої;

– при переході до наступної нормальної форми властивості попередніх нормальних форм зберігаються.

В основі процесу проектування лежить *метод нормалізації* – декомпозиція відношення (таблиці), що знаходиться в попередній нормальній формі, в два або більше відношення, що задовольняють вимогам наступної нормальної форми.

Найбільш важливі на практиці нормальні форми відношень грунтуються на фундаментальному в теорії реляційних баз даних понятті функціональної залежності. Для подальшого розгляду нам будуть потрібні кілька визначень.

1) Функціональна залежність. У відношенні *R* атрибут *Y функціонально залежить* від атрибута *X* (*X* і *Y* можуть бути складовими) в тому і тільки в тому випадку, якщо кожному значенню *X* відповідає в точності одне значення *Y: R.X (r) R.Y*.

2) Повна функціональна залежність. Функціональна залежність *R.X (r) R.Y* називається *повною*, якщо атрибут *Y* не залежить функціонально від будь-якої точної підмножини *X*.

3) Транзитивна функціональна залежність. Функціональна залежність *R.X (r) R.Y* називається *транзитивною*, якщо існує такий атрибут *Z*, що є функціональні залежності *R.X (r) R.Z* і *R.Z (r) R.Y* і відсутня функціональна залежність *R.Z (r) R.X* (При відсутності останньої вимоги ми мали б «нецікаві» транзитивні залежності в будь-якому відношенні, яке має декілька ключів).

4) Неключовий атрибут. Неключовим атрибутом називається будь-який атрибут відношення, який не входить до складу ключа (зокрема, первинного).

5) Взаємно незалежні атрибути. Два або більше атрибута *взаємно незалежні*, якщо жоден з цих атрибутів не є функціонально залежним від інших.

Перша нормальна форма

Відношення знаходиться у першій нормальній формі (1NF), якщо:

- кожний рядок таблиці має унікальний ключ;
- рядки не впорядковані;
- атрибути не впорядковані (можна переставляти стовпці);
- відсутні структурні атрибути - всі атрибути «атомарні»;
- нема рядків, які повторюються.

Друга нормальна форма

Відношення R знаходиться у другій нормальній формі (2NF) в тому і тільки в тому випадку, коли перебуває в 1NF, і кожен неключовий атрибут повністю залежить від первинного ключа.

Відношення R знаходиться у *другій нормальній формі (2NF)* в тому і тільки в тому випадку, коли воно знаходиться в 1NF, і кожен неключовий атрибут повністю залежить від кожного ключа R.

Третя нормальна форма

Відношення R знаходиться в *третій нормальній формі (3NF)* в тому і тільки в тому випадку, якщо перебуває в 2NF і кожен з неключових атрибутів нетранзитивно залежить від первинного ключа.

Відношення R знаходиться в *третій нормальній формі (3NF)* в тому і тільки в тому випадку, якщо перебуває в 2NF, і кожен з неключових атрибутів не є транзитивно залежним від будь-якого ключа R.

На практиці третя нормальна форма схем відношень достатня в більшості випадків, і приведенням до третьої нормальної форми процес проектування реляційної бази даних зазвичай закінчується. Однак іноді корисно продовжити процес нормалізації.

Нормальна форма Бойса-Кодда

Детермінант – будь-який атрибут, від якого повністю функціонально залежить деякий інший атрибут.

Відношення *R* знаходиться в *нормальній формі Бойса-Кодда (BCNF)* в тому і тільки в тому випадку, якщо кожен детермінант є можливим ключем.

Четверта нормальна форма

У відношенні *R (A, B, C)* існує *багатозначна залежність R.A (r) (r) R.B* в тому і тільки в тому випадку, якщо множина значень *B*, відповідна парі значень *A* і *C*, залежить тільки від *A* і не залежить від *C*.

Відношення *R* знаходиться в *четвертій нормальній формі (4NF)* в тому і тільки в тому випадку, якщо відношення знаходиться у 3NF та в разі існування багатозначної залежності *A (r) (r) B* всі інші атрибути *R* функціонально залежать від *A*.

П'ята нормальна форма

Залежність з'єднання. Відношення *R (X, Y, ..., Z)* задовольняє *залежності з'єднання (X, Y, ..., Z)* в тому і тільки в тому випадку, коли *R* відновлюється без втрат шляхом з'єднання своїх проекцій на *X, Y, ..., Z.*

П'ята нормальна форма. Відношення *R* знаходиться в *п'ятій нормальній формі (нормальній формі проекції-з'єднання - PJ / NF)* в тому і тільки в тому випадку, коли будь-яка залежність з'єднання в *R* випливає з існування деякого можливого ключа в *R.*

П'ята нормальна форма – це остання нормальна форма, яку можна отримати шляхом декомпозиції. Її умови досить нетривіальні, і на практиці 5NF не використовується. Зауважимо, що залежність з'єднання є узагальненням як багатозначної залежності, так і функціональної залежності.