Лабораторна робота № 6

Тема: Об'єднання таблиць.

Мета: Навчитись працювати з об'єднанням таблиць.

Що таке об'єднання

Однією з ключових особливостей SQL є можливість на льоту об'єднувати таблиці під час запитів, пов'язаних із вилученням даних. Об'єднання — це найпотужніші операції, які можна виконати за допомогою інструкції SELECT, тому розуміння об'єднань та їхнього синтаксису є важливою частиною процесу вивчення SQL.

Перш ніж ви зможете ефективно застосовувати об'єднання, слід розібратися, що таке реляційні таблиці які проектуються реляційні бази даних.

Що таке реляційні таблиці

Зрозуміти, що є реляційні таблиці, допоможе приклад із реального життя.

Припустимо, певна таблиця бази даних містить каталог товарів, у якому кожного елемента виділено один рядок. Інформація, що зберігається про кожен товар, повинна включати опис товару та його ціну, а також відомості про компанію, яка випустила цей товар.

Тепер припустимо, що у каталозі є ціла група товарів від одного постачальника. Де слід зберігати інформацію про постачальника (таку, як назва компанії, адреса та контактна інформація)? Ці відомості не рекомендується зберігати разом із даними про товари з кількох причин.

- Інформація про постачальника та сама для всіх його товарів. Повторення цієї інформації для кожного товару призведе до марної втрати часу та місця на диску.
- Якщо інформація про постачальника змінюється (наприклад, якщо він переїжджає або змінюється його поштовий код), вам доведеться оновити всі записи про його товари.
- Коли дані повторюються (а таке відбувається, коли інформація про постачальника вказується для кожного товару), висока ймовірність того, що дані будуть введені з помилкою. Несумісні дані дуже важко використовувати під час створення звітів.

Звідси можна дійти невтішного висновку, що зберігати безліч екземплярів тих самих даних вкрай небажано. Саме цей принцип лежить в основі реляційних баз даних. Реляційні таблиці розробляються в такий спосіб, що вся інформація розподіляється по безлічі таблиць, причому даних кожного типу створюється окрема таблиця. Ці таблиці співвідносяться (зв'язуються) між собою через загальні поля.

У нашому прикладі можна створити дві таблиці: одну – для зберігання інформації про постачальника, іншу – про його товари. Таблиця vendors містить інформацію про постачальників, по одному рядку для кожного постачальника, з обов'язковою вказівкою його унікального ідентифікатора. Це значення називається первинним ключем.

У таблиці products зберігається лише інформація про товари, але немає жодної конкретної інформації про постачальників, за винятком їх ідентифікаторів (первинного ключа таблиці vendors). Цей ключ пов'язує таблицю vendors із таблицею products. За допомогою ідентифікатора постачальника можна використовувати таблицю vendors для пошуку інформації про відповідного постачальника.

Що це дає? Нижче наведено ключові переваги.

- Інформація про постачальника ніколи не повторюється, заощаджуючи час, потрібний для заповнення бази даних, а також місце на диску.
- Якщо інформація про постачальника змінюється, достатньо оновити лише один запис про нього, єдиний у таблиці vendors. Дані у пов'язаних з нею таблицях змінювати не потрібно.
- Оскільки жодні дані не повторюються, вони, очевидно, виявляються несуперечливими, завдяки чому складання звітів значно спрощується.

Таким чином, дані в реляційних таблицях зберігаються досить ефективно і ними легко маніпулювати. Ось чому реляційні бази даних масштабуються значно краще ніж бази даних інших типів.

Масштабування

Можливість справлятися з зростаючим навантаженням без збоїв. Про добре спроектовані бази даних або програми говорять, що вони добре масштабуються.

Навіщо потрібні об'єднання

Розподіл даних за багатьма таблицям забезпечує їх ефективніше зберігання, спрощує обробку даних, і підвищує масштабованість бази

даних загалом. Однак ці переваги не досягаються задарма — за все доводиться платити.

Якщо дані зберігаються в багатьох таблицях, то як їх отримати допомогою однієї інструкції SELECT?

Відповідь така: за допомогою об'єднань. Об'єднання є механізмом злиття таблиць в інструкції SELECT. Використовуючи особливий синтаксис, можна об'єднати кілька вихідних таблиць в одну загальну, яка пов'язуватиме на льоту потрібні рядки з кожної таблиці.

ПРИМІТКА: використання інтерактивних інструментів СУБД

Важливо розуміти, що об'єднання не є фізичною таблицею - іншими словами, воно не існує як реальна таблиці в базі даних. Об'єднання створюється СУБД у міру необхідності та зберігається тільки на час виконання запиту. Багато СУБД пропонують графічний інтерфейс, який можна використовувати для інтерактивного визначення зв'язків таблиці. Ці інструменти можуть виявитися надзвичайно корисними для підтримки цілісності посилання. При використанні реляційних таблиць важливо, щоб у зв'язані стовпці заносили лише коректні дані. Повернімося до нашого прикладу: якщо в таблиці products зберігається недостовірний ідентифікатор постачальника, відповідні товари виявляться недоступними, оскільки вони не належать до жодного постачальника. Щоб уникнути цього, база даних повинна дозволяти користувачеві вводити лише достовірні значення (тобто такі, які представлені в таблиці vendors) у стовпці ідентифікаторів постачальників у таблиці products. Цілісність посилань означає, що СУБД змушує користувача дотримуватися правил, що забезпечують несуперечність даних. І контроль цих правил часто забезпечується завдяки інтерфейсам СУБД.

Створення об'єднання

Створення об'єднання дуже проста процедура. Потрібно зазначити всі таблиці, які мають бути включені до об'єднання, а також підказати СУБД, як вони мають бути пов'язані між собою.

Розглянемо наступний приклад

Запит

```
SELECT vend_name, prod_name, prod_price
FROM vendors, products
WHERE vendors.id = products.vendor_id;
```

Результат

vend_name	prod_name	prod_price
Doll House Inc.	Fish bean bag toy	3.49
Doll House Inc.	Bird bean bag toy	3.49
Doll House Inc.	Rabbit bean bag toy	3.49
Bears R Us	8 inch teddy bear	5.99
Bears R Us	12 inch teddy bear	8.99
Bears R Us	18 inch teddy bear	11.99
Doll House Inc.	Raggedy Ann	4.99
Fun and Games	King doll	9.49
Fun and Games	Queen doll	9.49

Аналіз

Інструкція SELECT починається так само, як і всі інструкції, які ми досі розглядали, — із вказівки стовпців, які мають бути вилучені. Ключова різниця полягає в тому, що два із зазначених стовпців (prod_name і prod_price) знаходяться в одній таблиці, а третій (vend name) - в іншій.

Погляньте на пропозицію FROM. На відміну від попередніх інструкцій SELECT воно містить дві таблиці: vendors і products. Це імена двох таблиць, які мають бути об'єднані в даному запиті. Таблиці коректно поєднуються у пропозиції WHERE, яка змушує СУБД зв'язати ідентифікатор постачальника vendor_id з таблиці vendors з полем vendor_id таблиці products.

Ці стовпці вказані як vendors.vend_id та products.vend_id. Цілком певні імена необхідні тут тому, що, якщо ви вкажете тільки vend_id, СУБД не зможе зрозуміти, на які стовпці vend_id ви посилаєтеся (їх два, по одному в кожній таблиці). Як видно з представлених результатів, одна інструкція SELECT зуміла отримати дані з двох різних таблиць

ПОПЕРЕДЖЕННЯ: повністю визначені імена стовпців

Використовуйте цілком певні імена стовпців (у яких назви таблиць і стовпців поділяються точкою) щоразу, коли може виникнути неоднозначність щодо того, який стовпець ви посилаєтеся. У більшості СУБД буде видано повідомлення про помилку, якщо введете неоднозначне ім'я стовпця, не визначивши його повністю шляхом вказівки імені таблиці.

Важливість пропозиції WHERE

Використання пропозиції WHERE для встановлення зв'язку між таблицями може здатися дивним, але вагома причина. Згадайте: коли таблиці поєднуються в інструкції SELECT, ставлення створюється на льоту. У визначеннях таблиць бази даних нічого не йдеться про те, як СУБД має об'єднувати їх.

Ви маєте вказати це самі. Коли ви об'єднуєте дві таблиці, то насправді створюєте пари, що складаються з кожного рядка першої таблиці та кожного рядка другого таблиці. Пропозиція WHERE діє як фільтр, що дозволяє включати до результату лише рядки, які відповідають зазначеній умові фільтрації – у даному випадку умові об'єднання. Без пропозиції WHERE кожен рядок у першій таблиці утворюватиме пару з кожним рядком другої таблиці незалежно від того, чи є логіка в їх об'єднанні чи ні

Декартовий добуток

Результати, що повертаються під час злиття таблиць без зазначення умови об'єднання. Кількість отриманих рядків дорівнюватиме числу рядків у першій таблиці, помноженому на число рядків у другій таблиці.

Для того, щоб розібратися в цьому, розглянемо наступну інструкцію SELECT та результат її виконання.

Запит

```
SELECT vend_name, prod_name, prod_price
 FROM vendors, products;
```

Результат

vend name	prod_name	prod_price
Bears R Us	8 inch teddy bear	5.99
Bears R Us	12 inch teddy bear	8.99
Bears R Us	18 inch teddy bear	11.99
Bears R Us	Fish bean bag toy	3.49
Bears R Us	Bird bean bag toy	3.49
Bears R Us	Rabbit bean bag toy 3	3.49
Bears R Us	Raggedy Ann	4.99
Bears R Us	King doll	9.49
Bears R Us	Queen doll	9.49
Bear Emporium	8 inch teddy bear	5.99
Bear Emporium	12 inch teddy bear	8.99
Bear Emporium	18 inch teddy bear	11.99
Bear Emporium	Fish bean bag toy	3.49
Bear Emporium	Bird bean bag toy	3.49
Bear Emporium	Rabbit bean bag toy 3	3.49
Bear Emporium	Raggedy Ann	4.99
Bear Emporium	King doll	9.49
Bear Emporium	Queen doll	9.49
Doll House Inc.	8 inch teddy bear	5.99
Doll House Inc.	12 inch teddy bear	8.99
Doll House Inc.	18 inch teddy bear	11.99
Doll House Inc.	Fish bean bag toy	3.49

Doll House Inc.	Bird bean bag toy	3.49
Doll House Inc.	Rabbit bean bag toy 3	3.49
Doll House Inc.	Raggedy Ann	4.99
Doll House Inc.	King doll	9.49
Doll House Inc.	Queen doll	9.49
Furball Inc.	8 inch teddy bear	5.99
Furball Inc.	12 inch teddy bear	8.99
Furball Inc.	18 inch teddy bear	11.99
Furball Inc.	Fish bean bag toy	3.49
Furball Inc.	Bird bean bag toy	3.49
Furball Inc.	Rabbit bean bag toy 3	3.49
Furball Inc.	Raggedy Ann	4.99
Furball Inc.	King doll	9.49
Furball Inc.	Queen doll	9.49
Fun and Games	8 inch teddy bear	5.99
Fun and Games	12 inch teddy bear	8.99
Fun and Games	18 inch teddy bear	11.99
Fun and Games	Fish bean bag toy	3.49
Fun and Games	Bird bean bag toy	3.49
Fun and Games	Rabbit bean bag toy 3	3.49
Fun and Games	Raggedy Ann	4.99
Fun and Games	King doll	9.49
Fun and Games	Queen doll	9.49

Аналіз

Як видно з представлених результатів, декартове твір ви, швидше за все, використовуватимете дуже рідко. Дані, отримані в такий спосіб, ставлять у відповідність кожному товару кожного постачальника, включаючи товари із зазначенням "не того" постачальника (і навіть постачальників, які взагалі не пропонують жодних товарів).

ПОПЕРЕДЖЕННЯ: не забудьте вказати пропозицію WHERE

Перевірте, чи включили ви запит пропозицію WHERE , інакше СУБД поверне набагато більше даних, ніж вам потрібно. Крім того, переконайтеся, що пропозиція WHERE сформульована правильно. Некоректна умова фільтрації призведе до того, що СУБД видасть неправильні дані.

Перехресне об'єднання
Іноді об'єднання, яке повертає декартове твір, називають перехресним об'єднанням.

Внутрішні об'єднання

Об'єднання, яке ми досі використовували, називається об'єднанням по рівності - воно засноване на перевірці рівності записів двох таблиць. Об'єднання такого типу називають також внутрішнім об'єднанням. Для таких об'єднань можна використовувати дещо інший синтаксис, який явно вказує на тип об'єднання. Наступна інструкція SELECT повертає ті самі дані, що й у попередньому прикладі.

Запит

```
SELECT vend_name, prod_name, prod_price
FROM vendors
INNER JOIN products ON vendors.id = products.vend_id;
```

Аналіз

Пропозиція SELECT тут така сама, як і в попередньому у випадку, а ось пропозиція FROM інша. У цьому запиті відношення між двома таблицями визначається пропозиції FROM, що містить специфікацію INNER JOIN. При використанні такого синтаксису умова об'єднання визначається за допомогою спеціальної пропозиції ON, а не WHERE. Фактична умова, що вказується в пропозиції ON, та сама, яка задавалася б у пропозицію WHERE.

Зверніться до документації СУБД, щоб дізнатися, який синтаксис краще використовувати.

Об'єднання кількох таблиць

SQL не обмежує кількість таблиць, які можуть бути об'єднані за допомогою інструкції SELECT. Основні правила створення об'єднання залишаються тими самими. Спочатку перераховуються всі таблиці, та був визначаються відносини з-поміж них. Розглянемо приклад.

Запит

```
SELECT prod_name, vend_name, prod_price, quantity
FROM order_items, products, vendors
WHERE products.vend_id = vendors.id AND order_items.prod_id =
products.id AND order_num = 20007;
```

Результат

vend name	vend name	prod_price	quantity
18 inch teddy bear	Bears R Us	11.99	50
Fish bean bag toy	Doll House Inc.	3.49	100
Bird bean bag toy	Doll House Inc.	3.49	100
Rabbit bean bag toy	Doll House Inc.	3.49	100
Raggedy Ann	Doll House Inc.	4.99	50

Аналіз

У цьому прикладі виводяться елементи замовлення номер 20007. Усі вони перебувають у таблиці order_items. Кожен елемент зберігається відповідно до ідентифікатора, який посилається на товар у таблиці Products. Ці товари пов'язані з відповідними постачальниками у таблиці Vendors за ідентифікатором постачальника, який зберігається разом із кожним записом про товар. У пропозиції FROM цього запиту перераховуються три таблиці, а пропозиція WHERE визначає обидві

названі умови об'єднання. Додаткова умова служить для фільтрації лише елементів замовлення 20007

ПОПЕРЕДЖЕННЯ: до питання про продуктивність

Усі СУБД обробляють об'єднання динамічно, витрачаючи час для обробки кожної зазначеної таблиці. Цей процес може виявитися дуже ресурсомістким, тому не слід використовувати об'єднання таблиць без особливої потреби. Чим більше таблиць ви об'єднуєте, тим нижча продуктивність.

ПОПЕРЕДЖЕННЯ: максимальна кількість таблиць в об'єднанні

Незважаючи на те, що SQL не накладає будь-яких обмежень на число таблиць в об'єднанні, багато СУБД насправді мають такі обмеження. Зверніться до документації своєї СУБД, щоб дізнатися, які обмеження такого роду вона накладає (якщо вони є).

Тепер саме час повернутися до прикладу з попередньої лабораторної роботи, в якому інструкція SELECT повертала список клієнтів, які замовили RGAN01.

Запит

```
SELECT cust_name, cust_contact
FROM customers
WHERE cust_id IN ( SELECT cust_id
                   FROM orders
                   WHERE order_num IN ( SELECT order_num
                                        FROM order_items
                                        WHERE prod_id = ’RGANOIV) ) ;
```

Аналіз

Підзапити не завжди є найефективнішим способом виконання складних інструкцій SELECT, тому той самий запит можна переписати з використанням синтаксису об'єднань.

Запит

```
SELECT cust_name, cust_contact
FROM customers, orders, order_items
WHERE customers.cust_id = orders.cust_id AND order_iterns.order_num
= orders.order_num AND prod_id = ’RGAN01';
```

Результат

cust_name	cust_contact
Fun4All	Denise L. Stephens
The Toy Store	Kim Howard

Аналіз

Отримання необхідних цього запиту даних вимагає звернення до трьох таблиць. Однак замість підзапитів тут було застосовано два об'єднання таблиць, а також вказано три умови WHERE. Перші два зв'язують таблиці в об'єднання, а останнє фільтрує дані товару RGAN01.

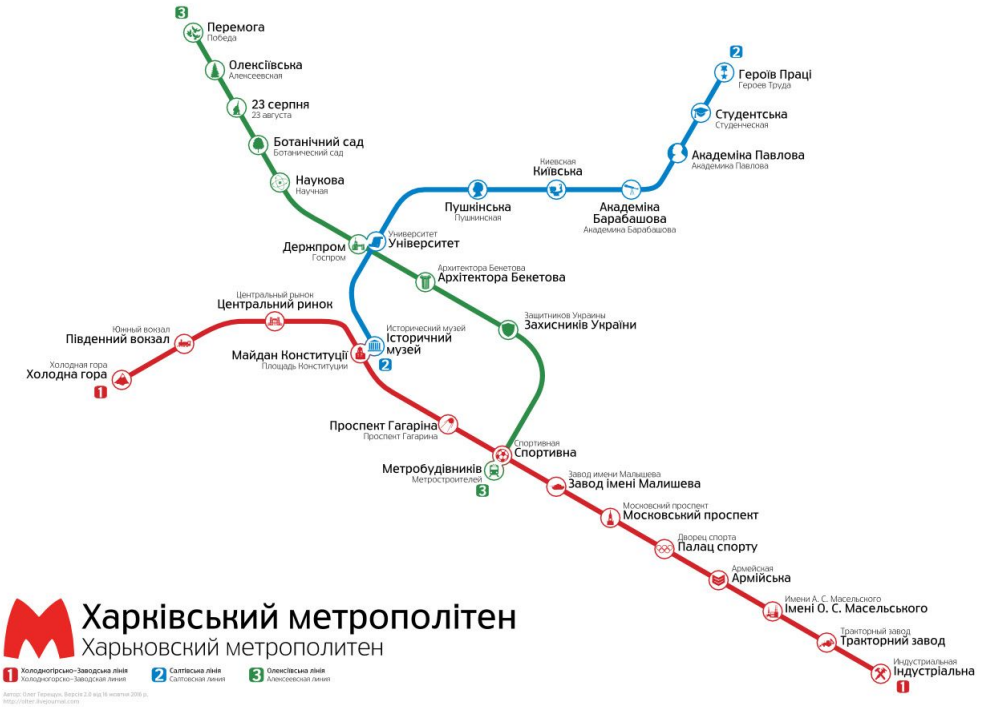
ЗАВДАННЯ:

Для виконання роботи необхідно імпортувати таблицю cities.

Необхідно написати наступні запити:

1. Отримати назву міста і назву регіону в якому знаходиться місто. У результат мають потрапити міста з населенням більше ніж 350 000.
2. За допомогою поєднань двох таблиць отримати міста яки знаходяться у регіоні з назвою Nord.
3. Створити структуру таблиць (таблиці та поля в кожній за таблиць) яка б дозволяла зберігати та однозначно відтворювати мапу будь якого метро світу.

   Для прикладу наведено мапу харківського метрополітену



Усі запити мають бути збережені у файл в форматі .sql

Таблиця cities

```
SET NAMES utf8;
SET time_zone = '+00:00';
SET foreign_key_checks = 0;
SET sql_mode = 'NO_AUTO_VALUE_ON_ZERO';

DROP TABLE IF EXISTS `cities`;
CREATE TABLE `cities` (
  `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `population` int(10) unsigned DEFAULT NULL,
  `region` varchar(5) COLLATE utf8_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO `cities` (`id`, `name`, `population`, `region`) VALUES
(1,     'Київ',2888470,    'N'),
(2,     'Харків',      1444540,    'E'),
(3,     'Одеса',       1010000,    'S'),
(4,     'Дніпро',      984423,     'C'),
(5,     'Донецьк',  932562,     'E'),
(6,     'Запоріжжя',       758011,     'E'),
(7,     'Львів',       728545,     'W'),
(8,     'Кривий Ріг',      646748,     'S'),
(9,     'Миколаїв',494381,     'S'),
(10,    'Маріуполь',       458533,     'S'),
(11,    'Луганськ', 417990,     'E'),
```

(12, 'Севастополь', 412630, 'S'),
(13, 'Вінниця', 372432, 'W'),
(14, 'Макіївка', 348173, 'E'),
(15, 'Сімферополь', 332608, 'S'),
(16, 'Херсон', 296161, 'S'),
(17, 'Полтава', 294695, 'E'),
(18, 'Чернігів', 294522, 'N'),
(19, 'Черкаси', 284459, 'C'),
(20, 'Суми', 268409, 'E'),
(21, 'Житомир', 268000, 'N'),
(22, 'Хмельницький', 267891, 'W'),
(23, 'Чернівці', 264427, 'W'),
(24, 'Горлівка', 250991, 'E'),
(25, 'Рівне', 249477, 'W'),
(26, 'Кам\'янське', 240477, 'C'),
(27, 'Кропивницький', 232052, 'C'),
(28, 'Івано-Франківськ', 229447, 'W'),
(29, 'Кременчук', 224997, 'C'),
(30, 'Тернопіль', 217950, 'W'),
(31, 'Луцьк', 217082, 'W'),
(32, 'Біла Церква', 211080, 'N'),
(33, 'Краматорськ', 160895, 'E'),
(34, 'Мелітополь', 156719, 'S'),
(35, 'Керч', 147668, 'S'),
(36, 'Сєвєродонецьк', 130000, 'E'),
(37, 'Хрустальний', 124000, 'E'),
(38, 'Нікополь', 119627, 'C'),
(39, 'Бердянськ', 115476, 'S'),

```
(40,  'Слов\'янськ',     115421,     'E'),
(41,  'Ужгород',  115195,     'W'),
(42,  'Алчевськ', 111360,     'E'),
(43,  'Павлоград',       110144,     'E'),
(44,  'Євпаторія',       106115,     'S'),
(45,  'Лисичанськ',      103459,     'E'),
(46,  'Кам\'янець-Подільський',  101590,     'W'),
(47,  'Бровари',  100374,     'N'),
(48,  'Дрогобич',98015,       'W'),
(49,  'Кадіївка',  92132,       'E'),
(50,  'Конотоп',  92000,       'E');
```

Таблиця regions

```
DROP TABLE IF EXISTS `regions`;
CREATE TABLE `regions` (
  `uuid` varchar(1) COLLATE utf8_unicode_ci NOT NULL,
  `name` varchar(255) COLLATE utf8_unicode_ci NOT NULL,
  `area_quantity` int(10) unsigned NOT NULL,
  PRIMARY KEY (`uuid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;

INSERT INTO `regions` (`uuid`, `name`, `area_quantity`) VALUES
('C',  'Center',     5),
('E',  'East',3),
('N',  'Nord',       4),
('S',  'South',      5),
('W',  'West',       8);
```