Лабораторна робота № 5

Тема: Угруповання даних. Підзапити.

Мета: Навчитись працювати з угрупованням даних. Здобути навички з написання підзапитів.

Принципи угруповання даних

На попередньому лабораторному занятті ви дізналися, що підсумкові функції SQL можна використовувати для отримання статистичних показників. Це дозволяє підраховувати число рядків, обчислювати суми та середні значення, а також визначати найбільше та найменше значення, не вдаючись до вилучення всіх даних.

Насамперед усі підсумкові обчислення виконувались над усіма даними таблиці чи над даними, які відповідали умові WHERE. Як нагадування наведемо приклад, у якому повертається кількість товарів, запропонованих постачальником DLL01.

Запит

```
SELECT COUNT (*) AS num_products
FROM products
WHERE vendor_id = 'DLL01';
```

Результат

num_products
4

Але що якщо ви хочете дізнатися кількість товарів, що пропонуються кожним постачальником? Або з'ясувати, які постачальники редагують лише один товар чи, навпаки, кілька товарів?

Саме в таких випадках слід використовувати групи. Угруповання дає можливість розділити всі дані на логічні набори, завдяки чому стає можливим виконання статистичних обчислень окремо з кожної групи.

Створення груп

Групи створюються за допомогою пропозиції GROUP BY інструкції SELECT.

Найкраще це можна продемонструвати на конкретному прикладі.

Запит

```
SELECT vendor_id, COUNT(*) AS num_products
FROM products
GROUP BY vendor_id;
```

Результат

vendor_id	num_products
BRS01	3
DLL01	4
FNG01	2

Аналіз

Ця інструкція SELECT виводить два стовпці: vendor_id, що містить ідентифікатор постачальника товару, і num_products, що містить поля, що обчислюються (він створюється за допомогою функції COUNT (*)). Пропозиція GROUP BY змушує СУБД відсортувати дані та згрупувати їх по стовпцю vendor_id. В результаті значення num_products буде обчислюватися один раз для кожної групи записів vendor_id, а не один раз для всієї таблиці products. Як бачите, у результатах вказується, що постачальник BRS01 пропонує три товари, постачальник DLL01 – чотири, а постачальник FNG01 – два.

Оскільки було використано пропозицію GROUP BY, не довелося вказувати кожну групу, для якої мають бути виконані обчислення. Це було зроблено автоматично. Пропозиція GROUP BY змушує СУБД спочатку групувати дані, а потім виконувати обчислення по кожній групі, а не по всьому набору результатів

Перш ніж застосовувати пропозицію GROUP BY, ознайомтеся з важливими правилами, якими слід керуватися.

- У пропозиціях GROUP BY можна вказувати довільне кількість стовпців. Це дозволяє вкладати групи одна в іншу, завдяки чому забезпечується ретельний контроль над тим, які дані підлягають угрупованню.
- Якщо у пропозиції GROUP BY використовуються вкладені групи, дані підсумовуються для останньої вказаної групи. Іншими словами, якщо задане угруповання, обчислення здійснюються для всіх зазначених стовпців (ви не можете отримати дані для кожного окремого стовпця).
- Кожен стовпець, зазначений у пропозиції GROUP BY, повинен бути вилученим стовпцем або виразом (але не підсумковий функцією).

Якщо в інструкції SELECT використовується якийсь вираз, то саме вираз має бути вказано у пропозиції GROUP BY. Псевдоніми застосовувати не можна.

- У більшості SQL реалізацій не можна вказувати в пропозиції GROUP BY стовпці, в яких містяться дані змінної довжини (наприклад, текстові поля чи поля коментарів).
- За винятком інструкцій, пов'язаних із підсумковими обчисленнями, кожен стовпець, згаданий в інструкції SELECT, має бути представлений у пропозиції GROUP BY.
- Якщо стовпець, за яким виконується угруповання, містить рядок із значенням NULL, він буде трактуватися як окрема група. Якщо є кілька рядків зі значеннями NULL, вони будуть згруповані разом.
- Пропозиція GROUP BY повинна стояти після пропозиції WHERE та перед пропозицією ORDER BY.

Фильтрация по группам

SQL дозволяє як групувати дані з допомогою пропозиції GROUP BY, а й здійснювати їх фільтрацію, тобто вказувати, які групи мають бути включені до результатів запиту, а які - виключені з них. Наприклад, вам може знадобитися список клієнтів, які зробили хоча б два замовлення. Щоб отримати такі дані, необхідний фільтр, який належить до цілої групи, а чи не до окремих рядків.

Ви вже знаєте, як працює пропозиція WHERE

Однак у цьому випадку його не можна використовувати, оскільки умови WHERE стосуються рядків, а не груп. Власне, пропозиція WHERE “не знає”, що таке групи.

Але що тоді слід застосувати замість пропозиції WHERE?

У SQL передбачена інша пропозиція, що підходить для цих цілей: HAVING. Воно дуже нагадує пропозицію WHERE. Всі типи виразів у пропозиції WHERE, з якими ви вже знайомі, допустимі і в пропозиції HAVING. Єдина різниця полягає в тому, що WHERE фільтрує рядки, а HAVING - групи.

Як здійснюється фільтрація по групам? Розглянемо наступний приклад.

Запит

```
SELECT cust_id, COUNT(*) AS orders
FROM orders
GROUP BY cust_id
HAVING COUNT(*) >= 2;
```

Результат

cust id	orders
1000000001	2

Аналіз

Перші три рядки цього запиту нагадують інструкцію SELECT, розглянуту раніше. Однак в останньому рядку з'являється пропозиція HAVING, яка фільтрує групи за допомогою вирази COUNT(*) >= 2 — два або більше замовлень.

Як бачите, пропозиція WHERE тут не працює, оскільки фільтрація полягає в підсумковому значенні групи, а чи не на значеннях відібраних рядків.

А тепер подумаємо: чи виникає необхідність використання як пропозиції WHERE, так і пропозиції HAVING в одній інструкції? Звісно, виникає. Припустимо, ви хочете вдосконалити фільтр попередньої інструкції таким чином, щоб поверталися імена всіх клієнтів, які зробили два або більше замовлень за останні 12 місяців. Щоб досягти цього, можна додати пропозицію WHERE, яка враховує лише замовлення, зроблені за останні 12 місяців. Потім ви додаєте пропозицію HAVING, щоб відфільтрувати лише ті групи, в яких є щонайменше два рядки.

Щоб краще розібратися в цьому, розглянемо наступний приклад, в якому перераховуються всі постачальники, які не пропонують менше двох товарів за ціною 4 долари та більше за одиницю

Запит

```
SELECT vend_id, COUNT(*) AS num_prods
FROM products
WHERE prod_price >= 4
GROUP BY vendor_id
HAVING COUNT(*) >= 2;
```

Результат

vendor_id	num_prods
BRS01	3
FNG01	2

Аналіз

Цей приклад потребує пояснення. Перший рядок є основною інструкцією SELECT *, яка використовує підсумкову функцію — так само, як і в попередніх прикладах.

Пропозиція WHERE фільтрує всі рядки зі значеннями в стовпці prod ptice не менше 4. Потім дані групуються по стовпцю vend id, після чого пропозиція HAVING фільтрує лише групи, що містять не менше двох членів. За відсутності пропозиції WHERE було б отримано зайвий рядок (постачальник, що пропонує чотири товари, кожен з яких дешевше 4 доларів), як показано нижче.

Запит

```
SELECT vend_id, COUNT(*) AS num_prods
FROM Products
WHERE prod_price >= 4
GROUP BY vend
HAVING COUNT(*) >= 2;
```

Результат

vend_id	num_prods
BRS01	3
FNG01	2

Аналіз

Цей приклад потребує пояснення. Перший рядок є основною інструкцією SELECT *, яка використовує підсумкову функцію — так само, як і в попередніх прикладах.

Пропозиція WHERE фільтрує всі рядки зі значеннями в стовпці prod ptice не менше 4. Потім дані групуються по стовпцю vend id, після чого пропозиція HAVING фільтрує лише групи, що містять не менше двох членів. За відсутності пропозиції WHERE було б отримано зайвий рядок (постачальник, що пропонує чотири товари, кожен з яких дешевше 4 доларів), як показано нижче.

Запит

```
SELECT vendor_id, COUNT(*) AS num_prods
FROM products
GROUP BY vend_id
HAVING COUNT(*) >= 2;
```

Результат

vendor_id	num_prods
BRS01	3
DLL01	4
FNG01	2

ПРИМІТКА: різниця між пропозиціями HAVING та WHERE

Пропозиція HAVING настільки нагадує пропозицію WHERE, що в більшості СУБД воно трактується так само, якщо тільки не вказано пропозицію GROUP BY. Проте слід знати, що з-поміж них існує різниця. Використовуйте пропозиція HAVING тільки разом із пропозицією GROUP BY, а пропозиція HAVING - для стандартної фільтрації на рівні рядків.

Порядок пропозицій в інструкції SELECT

Пропозиції інструкції SELECT повинні вказуватись у визначеному порядку. У таблиці перелічені і усі пропозиції, які ми вивчили досі, у порядку, в якому вони повинні слідувати.

Пропозиція	**Опис**	**Необхідність**
SELECT	Стовпці або вирази, які мають бути отримані	Так
FROM	Таблиця для отримання даних	Тільки якщо вилучаються дані з таблиці
WHERE	Фільтрування на рівні рядків	Ні
GROUP BY	Визначення групи	Тільки якщо виконуються підсумкові обчислення за групами
HAVING	Фільтрування на рівні груп	Ні
ORDER BY	Порядок сортування результатів	Ні

Підзапити

Інструкції SELECT – це запити SQL. Усі інструкції, з якими ми мали справу досі, були простими запитами: за допомогою окремих інструкцій витягувалися дані з певних таблиць.

У SQL можна також створювати підзапити: запити, які вкладені в інші запити. Для чого це може знадобитися.

Фільтрування за допомогою підзапитів

Таблиці баз даних, що використовуються у всіх прикладах книги, є реляційними. Замовлення зберігаються у двох таблицях. Таблиця orders містить один рядок для кожного замовлення; у ній зазначаються номер замовлення, ідентифікатор клієнта та дата замовлення. Окремі елементи замовлень зберігаються у таблиці order_items. Таблиця orders не містить інформації про клієнтів. Вона зберігає лише ідентифікатор клієнта. Інформація про клієнтів міститься в таблиці customers.

Тепер припустимо, що ви бажаєте отримати список всіх клієнтів, які замовили товар RGAN01. Для цього необхідно виконати таке:

1. отримати номери всіх замовлень, що містять товар RGAN01;
2. отримати ідентифікатори всіх клієнтів, які зробили замовлення, перелічені на попередньому етапі;
3. отримати інформацію про всіх клієнтів, ідентифікатори яких були отримані на попередньому кроці.

Кожен із цих пунктів можна виконати у вигляді окремого запиту. Здійснюючи так, ви використовуєте результати, що повертаються однією інструкцією SELECT, щоб заповнити пропозицію WHERE для наступної інструкції SELECT.

Але можна також скористатися підзапитами для того, щоб об'єднати всі три запити в одну-єдину інструкцію.

Перша інструкція SELECT витягує стовпець order_num для всіх елементів замовлень, у яких у стовпці product_id значиться RGAN01. Результат є номерами двох замовлень, що містять даний товар

Запит

```
SELECT order_num
FROM order_items
WHERE product_id = 'RGAN01';
```

Результат

order_num
20007
20008

Наступний крок полягає в отриманні ідентифікаторів клієнтів, пов'язаних із замовленнями 20007 та 20008. Використовуючи пропозицію IN, можна створити наведену нижче інструкцію SELECT.

Запит

```
SELECT customer_id
FROM orders
WHERE num IN (20007,20008);
```

Результат

customer_id
1000000004
1000000005

Теперь объединим эти два запроса путем превращения первого из них (того, который возвращает номера заказов) в подзапрос.

Запит

```
SELECT customer_id
FROM orders
WHERE num IN ( SELECT num
                FROM order_items
                WHERE product_id = 'RGAN01');
```

Результат

customer_id
1000000004
1000000005

Аналіз

Підзапити завжди обробляються, починаючи із самої внутрішньої інструкції SELECT у напрямку “зсередини назовні”. Під час обробки попередньої інструкції СУБД насправді виконує дві операції.

Спочатку вона виконує наступне підзапит:

```
SELECT num FROM order_items WHERE product_id = 'RGAN01’
```

В результаті повертаються два номери замовлення: 20007 та 20008. Ці два значення потім передаються в пропозицію зовнішнього запиту WHERE у форматі з роздільником у вигляді коми, необхідному для оператора IN. Тепер зовнішній запит стає таким:

```
SELECT customer_id FROM orders WHERE num IN  (20007,20008)
```

Як бачите, результат коректний і є таким самим, як і отриманий шляхом жорсткого кодування пропозиції WHERE в попередньому прикладі

ПОРАДА: форматуйте SQL-запити

Інструкції SELECT, що містять підзапити, можуть виявитися важкими для читання та налагодження, особливо якщо їхня складність зростає. Розбиття запитів на кілька рядків і вирівнювання рядків відступами, як показано в прикладах, значно полегшує роботу з підзапитами.

Тепер ми маємо ідентифікатори всіх клієнтів, які замовили товар RGAN01. Наступним кроком є отримання клієнтської інформації для кожного з цих ідентифікаторів. Інструкція SQL, яка здійснює вибірку двох стовпців, виглядає так.

Запит

```
SELECT name, contact
FROM customers
WHERE id IN (1000000004, 1000000005);
```

Але замість жорсткої вказівки ідентифікаторів клієнтів можна перетворити дану пропозицію WHERE на підзапит.

Запит

```
SELECT name, contact
FROM customers
WHERE id IN (     SELECT customer_id
                  FROM orders
                  WHERE num IN (
                                    SELECT order_num
                                    FROM order_items
                                    WHERE product_id = 'RGAN01') ) ;
```

Результат

name	contact
Fun4All	Denise L. Stephens
The Toy Store	Kim Howard

Щоб виконати такий запит, СУБД має по суті опрацювати три інструкції SELECT. Найвнутрішній підзапит повертає перелік номерів замовлень, який потім використовується як пропозиція WHERE для підзапиту, зовнішнього по відношенню до даного. Це підзапит повертає перелік ідентифікаторів клієнтів, які використовуються у пропозиції WHERE запиту найвищого рівня. Запит верхнього рівня повертає дані, що шукаються.

Як бачите, завдяки підзапиту можна створювати дуже потужні та гнучкі інструкції SQL. Немає обмежень на кількість підлеглих запитів, хоча на практиці можна зіткнутися з відчутним зниженням продуктивності, яке підкаже вам, що було використано дуже багато рівнів підзапитів.

ПОПЕРЕДЖЕННЯ: тільки один стовпець

Інструкції SELECT у підзапитах можуть повертати тільки один стовпець. Спроба витягти кілька стовпців призведе до появі повідомлення про помилку.

ПОПЕРЕДЖЕННЯ: підзапити та продуктивність

Подані тут приклади працюють і призводять до досягнення необхідних результатів. Однак підзапити — не завжди найефективніший спосіб отримання таких даних.

Використання підзапитів як обчислювані поля

Інший спосіб використання підзапитів полягає у створенні обчислюваних полів. Припустимо, необхідно вивести загальну кількість замовлень, зроблених кожним клієнтом із таблиці customers (клієнти). Замовлення зберігаються у таблиці orders разом із відповідними ідентифікаторами клієнтів.

Щоб виконати цю операцію, необхідно зробити таке:

1. отримати список клієнтів з таблиці customers;
2. для кожного обраного клієнта підрахувати кількість його замовлень у таблиці orders.

Як пояснювалося на попередніх двох уроках, можна виконати

Інструкцію SELECT COUNT (*) для підрахунку рядків у таблиці, а використовуючи пропозицію WHERE для фільтрації ідентифікатора конкретного клієнта, можна підрахувати замовлення цього клієнта.

Наприклад, за допомогою наступного запиту можна підрахувати кількість замовлень, зроблених клієнтом 1000000001.

Запит

```
SELECT COUNT(*) AS orders
FROM orders
WHERE customer_id = '1000000001';
```

Щоб отримати підсумкову інформацію за допомогою функції COUNT(*) для кожного клієнта, використовуйте вираз COUNT(*) як підзапит. Розглянемо наступний приклад.

Запит

```
SELECT    name,
          state,
          (    SELECT COUNT(*)
               FROM orders
               WHERE orders.customer_id = customers.id) AS orders

FROM customers
ORDER BY name;
```

Результат

name	state	orders
Fun4All	IN	1
Fun4All	AZ	1
Kids Place	OH	0
The Toy Store	IL	1
Village Toys	MI	2

Аналіз

Ця інструкція SELECT повертає три стовпці кожного клієнта з таблиці customers: name, state і orders. Поле orders є обчислюваним; воно формується внаслідок виконання підзапиту, який укладено у круглі дужки. Підзапит виконується один раз для кожного обраного клієнта. У

наведеному прикладі підзапит виконується п'ять разів, тому що отримано імена п'яти клієнтів.

Пропозиція WHERE у підзапиті дещо відрізняється від пропозицій WHERE, з якими ми працювали раніше, тому що в ньому використовуються повні імена стовпців. Наступна пропозиція вимагає від СУБД, щоб було проведено порівняння значення customer_id у таблиці orders з тим, яке витягується з таблиці customers.

```
WHERE orders.customer_id = customer.id
```

Подібний синтаксис - ім'я таблиці та ім'я стовпця поділяються точкою - повинен застосовуватися щоразу, коли може виникнути невизначеність в іменах стовпців.

Підзапити надзвичайно корисні під час створення інструкцій SELECT такого типу, проте уважно слідкуйте за тим, щоб правильно вказували неоднозначні імена стовпців.

ПОРАД: підзапити не завжди є оптимальним рішенням

Незважаючи на те, що показаний тут приклад працездатний, найчастіше він виявляється не найефективнішим способом отримання даних такого типу. Ми ще раз розглянемо цей приклад на одному з наступних уроків.

ЗАВДАННЯ:

Для виконання роботи необхідно імпортувати таблицю cities.

Необхідно написати наступні запити:

1. Отримати кількість населення в кожному регіоні.
2. Отримати регіони та кількість населення в цих регіонах де кількість міст в регіоні більше або дорівнює 10.
3. Отримати третю п'ятірку міст (назва та кількість населення) за кількістю населення якщо ці міста знаходяться в регіонах з кількістю областей не меньше 5
4. Отримати назви регіонів та кількістю в них населення, за умови що в підрахунку кількості населення брали участь міста з населенням більше ніж 300 000.
5. Отримати назву так кількість населення міст які знаходяться в регіонах з кількістю областей не більше 5 та кількість населення в цих містах не входить до діапазону 150 000 - 500 000

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);
```