Залікова Робота з Бази даних та інформаційних систем

Захаров Дмитро

28 листопада, 2024

Варіант 5

Зміст

1 Завдання 1: Мережева Модель

Умова Задачі 1.1. Мережна модель даних. Відмінність простої і складної мережних структур. Переваги та недоліки мережних структур

Відповідь. Мережева модель даних була дуже поширеною на певному етапі розвитку баз даних і досі використовується в деяких системах. Вона подібна до ієрархічної моделі, оскільки також складається з набору записів, які можуть бути власниками або членами групи, але головна відмінність полягає в тому, що один запис у мережевій моделі може бути частиною кількох груп одночасно. Кожна група має свою назву, а типи групових відносин відрізняються від їхніх конкретних екземплярів. Тип визначає спільні характеристики для всіх екземплярів, а сам екземпляр є записом-власником із підлеглими елементами. При цьому обмеження таке: один запис не може одночасно бути частиною двох груп одного типу.

У цій моделі будь-які елементи можуть бути пов'язані один з одним, а підлеглий запис може мати більше одного власника. Це дозволяє підтримувати зв'язки типу "багато-до-багатьох". Щоб спростити мережеву структуру, можна додати надмірні дані, що дозволяє створити дерево через дублювання деяких елементів.

Іноді така надмірність є прийнятною, але в деяких випадках вона може бути значною. Методи, що добре працюють для деревовидних структур, часто не підходять для мережевих моделей, через що програми для обробки дерев не можуть працювати з мережею.

Ситуацію ускладнює те, що методи, які застосовуються до одного виду мережевої структури, можуть бути неефективними для іншого виду. Суттєвим мінусом мережевої моделі є необхідність суворого дотримання фізичної структури даних, що ускладнює організацію запитів. Однак її ключова перевага – можливість створювати складні моделі, які краще відповідають реальним процесам. Щоб зменшити складність ієрархічної та мережевої моделей, було розроблено реляційну модель.

Переваги та недоліки. Сильні сторони мережевої моделі — це її ефективність у використанні пам'яті та швидкість обробки. У порівнянні з ієрархічною, мережева модель точніше відображає предметну область завдяки додатковим зв'язкам між елементами. Проте серед її мінусів — складність у використанні через жорстку схему. Контроль цілісності даних у мережевій моделі послаблений через можливість довільних зв'язків, і при зміні даних часто потрібно змінювати програмне забезпечення.

Відмінність простої і складної мережних структур.

Проста мережева структура	Складна мережева структура
Відображає зв'язки "один-до-багатьох"	Відображає зв'язки "багато-до-багатьох".
Має чітку ієрархію між вузлами	Один вузол пов'язаний із декількома іншими
Легша у розробці та підтримці	Вимагає більш складного управління.
Менш гнучка у вираженні складних відносин	Гнучкіша для моделювання складних систем

2 Завдання 2: JOIN операції

Умова Задачі 2.1. INNER JOIN та FULL JOIN в SQL.

Відповідь.

INNER JOIN. INNER JOIN повертає тільки ті записи, які мають відповідності в обох таблицях. Якщо в одному з таблиць немає відповідності для певного запису, то цей запис виключається з результатів.

Синтаксис цієї команди наведений нижче:

```
SELECT columns
FROM table1
INNER JOIN table2
ON table1.column = table2.column;
```

Приклад. Нехай в нас є дві таблиці. Перша — це таблиця EMPLOYEES.

ID	NAME	DEPARTMENT_ID
1	Ivan	101
2	Olena	102
3	Pavlo	NULL
4	Svitlana	104

Друга таблиця — DEPARTMENTS:

ID	DEPARTMENT_NAME
101	IT
102	HR
103	Sales

Тоді запит може виглядати так:

```
SELECT EMPLOYEES.NAME, DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES
INNER JOIN DEPARTMENTS
ON EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.ID;
```

Результатом буде наступна таблиця:

NAME	DEPARTMENT_NAME
Ivan	IT
Olena	HR

INNER JOIN повернув тільки записи, які мають відповідності між DEPARTMENT_ID у таблиці EMPLOYEES та ID у таблиці DEPARTMENTS. Запис із DEPARTMENT_ID = NULL та запис у таблиці DEPARTMENTS з ID = 103 були виключені.

FULL JOIN. FULL JOIN повертає всі записи з обох таблиць. Якщо запис у одній таблиці не має відповідності в іншій, відповідні колонки заповнюються значенням NULL. Синтаксис виглядає наступним чином:

```sql
SELECT columns
FROM table1
FULL JOIN table2
ON table1.column = table2.column;
```

Як приклад для таблиць вище, нехай маємо наступний запит:

```sql
SELECT EMPLOYEES.NAME, DEPARTMENTS.DEPARTMENT_NAME
FROM EMPLOYEES
FULL JOIN DEPARTMENTS
ON EMPLOYEES.DEPARTMENT_ID = DEPARTMENTS.ID;
```

Результатом буде наступна таблиця:

NAME	DEPARTMENT_NAME
Ivan	IT
Olena	HR
Pavlo	NULL
Svitlana	NULL
NULL	Sales

Таким чином, `FULL JOIN` повернув усі записи з обох таблиць:

- Записи з таблиці `EMPLOYEES`, які не мали відповідностей у `DEPARTMENTS` (наприклад, Pavlo та Svitlana).
- Записи з таблиці `DEPARTMENTS`, які не мали відповідностей у `EMPLOYEES` (наприклад, Sales).

Таким чином, має наступну порівняльну таблицю:

INNER JOIN	FULL JOIN
Повертає тільки записи з відповідністю	Повертає всі записи з обох таблиць
Виключає записи без відповідностей	Заповнює `NULL` для записів без відповідностей.
Результат зазвичай менше	Результат зазвичай більше

Також, візуалізувати процес поєднання множин можна за допомогою картинки нижче:

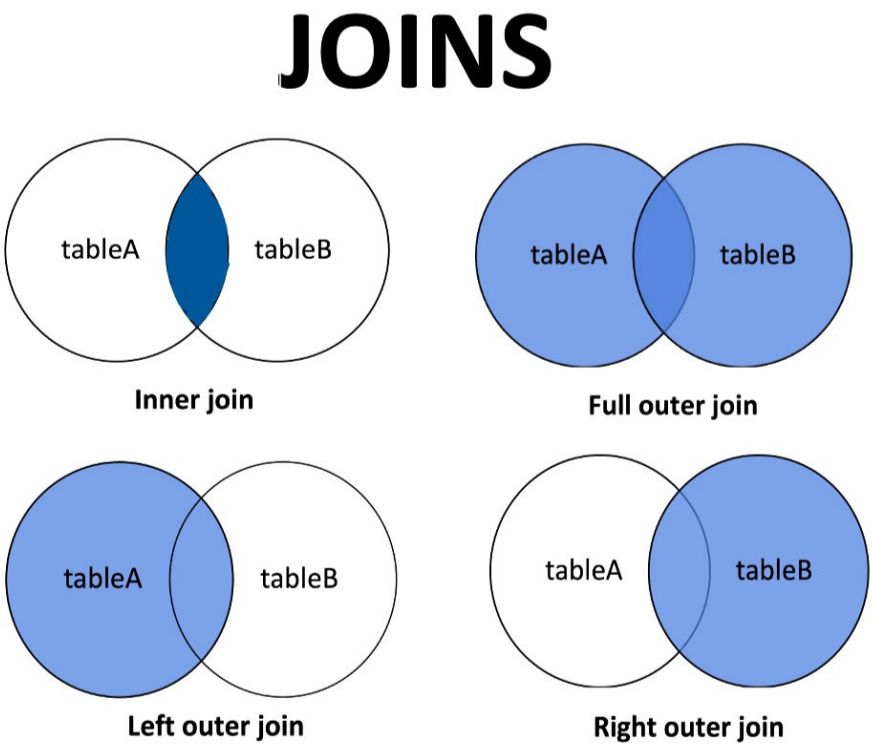

Рис. 1: Порівняння `INNER JOIN` та `FULL JOIN`

3 Завдання 3: Regex

Умова Задачі 3.1. Пошук рядка, що не починається з цифри

Відповідь. Створимо дуже просту таблицю та заповнимо її даними:

```sql
-- Creating the test table
CREATE TABLE TEST (
    ID INT PRIMARY KEY,
    NAME VARCHAR(100)
);

-- Inserting some random data
INSERT INTO TEST (ID, NAME)
VALUES
(1, 'Ivan Kovaly'),
(2, 'Olena Sidorenko'),
(3, '123Dmytro Tkach'),
(4, 'Pavlo Shevchenko'),
(5, '30ksana Dovgan'),
(6, 'Roman Kovtun');
```

У SQL можна використовувати оператор `NOT LIKE` для перевірки, що рядок не починається з цифри. Для цього використовуємо регулярні вирази (для MySQL використовуємо

REGEXP):

```
SELECT * FROM TEST
WHERE NAME NOT REGEXP '^[0-9]';
```

У Postgres можна використати ще більш короткий запит:

```
SELECT * FROM TEST
WHERE NAME !~ '^[0-9]';
```

Так чи інакше, задана команда обирає усі рядки, в яких поле NAME не починається з цифри. Результат виглядатиме наступним чином:

ID	NAME
1	Ivan Kovaly
2	Olena Sidorenko
4	Pavlo Shevchenko
6	Roman Kovtun

Якщо ж ми хочемо зробити regex саме під умову “не починається з цифри”, то регулярний вираз стає дещо складнішим:

```
^(?!^[0-9].*$).*
```

В такому разі, наступний запит дасть той самий результат:

```
SELECT * FROM TEST
WHERE NAME REGEXP '^(?!^[0-9].*$).*';
```

4 Завдання 4: Групування та середні

Умова Задачі 4.1. Дана таблиця SALES з стовбчиками:

- ID — id агента,
- AGENT — агент (прізвище та ім'я),
- ORDERNUMBER — номер замовлення,
- ORDERDATE — його дата,
- ORDERSUM — його сума.

Написати SQL запит, що виводить середні суми замовлень за кожний день по агентам (8 балів).

Відповідь. Спочатку, пропонується створити таблицю і накидати в неї деякі дані:

```sql
-- Creating the table itself
CREATE TABLE SALES (
    ID INT PRIMARY KEY,
    AGENT VARCHAR(100),
    ORDERNUMBER INT,
    ORDERDATE DATE,
    ORDERSUM DECIMAL(10, 2)
);

-- Inserting some data
INSERT INTO SALES (ID, AGENT, ORDERNUMBER, ORDERDATE, ORDERSUM)
VALUES
(1, 'Dimon Davis', 8805, '2023-07-18', 5000),
(2, 'Dimon Davis', 8806, '2023-07-18', 7000),
(3, 'Dimon Davis', 8807, '2023-07-19', 5000),
(4, 'Dimon Davis', 8809, '2023-07-19', 5100),
(5, 'Kate Kate', 8810, '2023-07-19', 5000),
(6, 'Kate Kate', 8816, '2023-07-19', 20000);
```

Тепер, як і просить завдання, напишемо SQL запит, що виводить середні суми замовлень за кожний день по агентам. Для цього скористаємося командою GROUP BY разом з командою AVG:

```sql
SELECT AGENT, ORDERDATE, AVG(ORDERSUM) AS AVG_ORDER_SUM
FROM SALES
GROUP BY AGENT, ORDERDATE
ORDER BY ORDERDATE, AGENT;
```

Як результат для наданих даних отримаємо наступне:

AGENT	ORDERDATE	AVG_ORDER_SUM
Dimon Davis	2023-07-18	6000.00
Dimon Davis	2023-07-19	5050.00
Kate Kate	2023-07-19	12500.00

Якщо в задані просто малось на увазі вивести середню суму замовлень за кожний день, то можна використати наступний більш простий запит:

```
SELECT ORDERDATE, AVG(ORDERSUM) AS AVG_ORDER_SUM
FROM SALES
GROUP BY ORDERDATE
ORDER BY ORDERDATE;
```

В такому разі, результат буде наступним:

ORDERDATE	AVG_ORDER_SUM
2023-07-18	6000.00
2023-07-19	8775.00