Национальный исследовательский ядерный университет «МИФИ» (Московский Инженерно–Физический Институт)
Кафедра №42 «Криптология и кибербезопасность»

Лабораторная работа №1-3: «Сложные запросы на выборку. Соединения»

Цель работы

Как правило, при составлении запросов к базе данных требуется провести частичную денормализацию и использовать данные из двух и более таблиц одновременно. Для проведения денормализации используется оператор соединения (JOIN). Данная лабораторная работа посвящена использованию соединений и другим сложным аспектам запросов к базе данных.

Ход работы

1. Подготовить «легенду»: на какие вопросы требуется знать ответы сотрудникам организации в выбранной предметной области?
2. Разработать и проверить SQL-запросы, реализующие эти ответы. Необходимо разработать не менее 8 различных запросов с применением различных возможностей, предоставляемых стандартом SQL и SQLite3. Необходимо уделить особое внимание различным случаям соединения таблиц.
3. Оформить отчёт.

Основные возможности сложных запросов на выборку

1. Подзапросы: вместо переменной или литерала в запросе можно использовать вложенный подзапрос. Этот подзапрос должен возвращать одно значение, либо один столбец значений, если оператор работает с диапазонами. Например, этот запрос вернёт список событий, запланированных в крупных городах:

SELECT * FROM events WHERE city_id IN (SELECT city_id FROM cities WHERE pop > 1000000);

Этот запрос вернёт стоимость изделий в долларах:

SELECT name, cost_rub / (SELECT exchange_rate FROM stock WHERE currency = 'USD') FROM goods;

2. Соединение. Такой запрос сопоставляет друг другу ряды из двух таблиц. Без дополнительной условной фильтрации, каждая строка из одной таблиц соединяется с каждой строкой из другой. Операция соединения применяется для денормализации данных. Например, этот запрос позволяет извлечь список мероприятий с указанием города, где они проводятся:

SELECT cities.name, events.name FROM events, cities WHERE events.city_id = cities.city_id;

Таблицу можно соединять саму с собой. Например, этот запрос выводит список потенциальных пар по списку людей:

SELECT p1.name, p2.name FROM people p1, people p2 WHERE p1.gender > p2.gender;

3. Теоретико-множественные операции (UNION, UNION ALL, INTERSECT, MINUS). Позволяют построить объединение, пересечение, разность множеств рядов из двух таблиц. Ценность этих инструкций состоит в возможности объединить две различных таблицы в одну. Оператор UNION устраняет возникающие при объединении дубликаты, UNION ALL этого не делает, поэтому работает существенно быстрее.

4. Обычные и рекурсивные табличные выражения (TABLE EXPRESSIONS). Табличные выражения позволяют выделить отдельные элементы сложного запроса в именованные «вре́менные» таблицы. Это позволяет упростить понимание текста запроса. Используется синтаксис WITH [название выражения] AS (SELECT [текст подзапроса]). Например, следующий запрос выведет информацию обо всех сотрудниках организации, имеющих автомобили.

```
WITH car_owners AS (SELECT DISTINCT owner_id FROM cars)
SELECT * FROM employees WHERE id IN car_owners;
```

Табличные выражения могут быть рекурсивными. Рекурсивные табличные выражения применяются для формирования и обхода графов на табличных данных. Они имеют следующий вид:

WITH RECURSIVE recursive_table(x) AS ([начальное значение] UNION ALL SELECT [...] FROM recursive_table(x[...])

Например, данный запрос генерирует все числа от 0 до 100:

WITH RECURSIVE cnt(x) AS (VALUES(0) UNION ALL SELECT (x+1) FROM cnt WHERE x<100) SELECT * FROM cnt;

В этом примере необходимо отметить, что табличные выражения могут, при необходимости, иметь параметры. Наличие нескольких уровней рекурсии индуцирует на данных таблицы определённую иерархию. Поэтому такие запросы также называются иерархическими запросами (HIERARCHICAL QUERIES). Реализация иерархических запросов существенно отличается в различных СУБД.

5. Аналитические функции (OVER). Предназначены, для анализа того, как значения полей меняются с течением времени (или любого другого параметра). Позволяют извлечь значение заданного поля не для текущей записи, а для предшествующей; вычислить скользящее среднее, и т.д. Так как аналитические функции оперируют понятиями «предыдущий», «следующий», «первый (лучший)», — их применение осмысленно только при наличии сортировки. Поэтому, параметр и порядок сортировки при использовании аналитической функции указывать обязательно. При этом, результаты запроса можно отсортировать по-другому. Пример: этот запрос вычисляет изменение дохода за месяц:

```
SELECT
        income,
        (income - lag(income,1,NULL) OVER (ORDER BY month)) as change
FROM income;
```

Этот запрос вычисляет баланс организации в каждом месяце по доходам и расходам:

```
SELECT
        month,
        sum(income) OVER
            (ORDER BY month
            ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
        - sum(expense) OVER
            (ORDER BY month
            ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
        AS balance
FROM income;
```

Оформление отчёта

1. Титульный лист: название института, название лабораторной работы, имя, фамилия, номер группы, год,...
2. Список выполненных запросов SQL с описаниями их назначения и смысла;
3. Приложение: Листинг использованных инструкций SQL. В тексте отчёта должна быть ссылка на приложение;
4. Заключение: краткое описание проделанной работы.

Справочные материалы

1. The WITH Clause. https://www.sqlite.org/lang_with.html Официальная документация SQLite3 по обычным и рекурсивным табличным выражениям, с рельсовыми диаграммами и примерами иерархических запросов;

2. Window Functions. https://www.sqlite.org/windowfunctions.html Официальная документация по аналитическим функциям (в SQLite3: Window Functions). Рельсовые диаграммы, описание окна, список доступных функций;

3. Join-Operator. https://www.sqlite.org/syntax/join-operator.html Различные формы оператора JOIN (заметьте, что JOIN — это просто бинарный оператор, отображающий два элемента из множества таблиц в множество таблиц (если не брать во внимание условие соединения).