**Лабораторная работа 6. Система управления данными SQLite**

**Часть 2. Операции с записями и таблицами**

Цель: освоить операции с записями и таблицами, т.е. научиться изменять и удалять записи и сами таблицы с помощью SQL/DDL для СУБД SQLite.

Задачи:

* освоить изменение метаданных таблиц (переименование, добавление и удаление полей, изменение типа данных, добавление и удаление ограничений);
* научиться на языке SQL обновлять, добавлять и заменять с учётом ограничений целостности данных и ссылок;
* уметь удалять записи из таблицы по условию и удалять целиком таблицу БД.

Задание для выполнения части 2 лабораторной работы:

1. Выполните и проконтролируйте операции **ALTER** **TABLE** для изменения метаданных таблиц(добавление, переименование и удаление полей, изменение типа данных, пере­­именование таблиц, добавление ограничений).
2. С помощью инструкций SQL **UPDATE**, **INSERT** и в выбранном инструменте выполните операции обновления, добавления и замены записей в 3-х ранее созданных связанных таблицах. Выберите данные и операции, приводящие к нарушению целостности данных и ссылок, проверьте адекватность реакций на нарушения ограничений.
3. Выполните и проконтролируйте операции **INSERT** для добавления записей со значениями по умолчанию или NULL.
4. Выполните и проконтролируйте операции **DELETE** для удаления ранее добавленных записей.
5. С помощью инструкций **DROP TABLE** удалите хотя бы одну таблицу.
6. Оформите ход выполнения и результаты работы в виде отчёта.

**Изменение таблиц и столбцов**

Если таблица уже была ранее создана, и необходимо изменить её описание, то для этого применяется команда **ALTER TABLE**. Она поддерживает различные опции и возможности. Рассмотрим лишь основные сценарии, с которыми мы можем столкнуться.

***Добавление нового столбца***

Добавим в таблицу Users новый столбец email:

|  |
| --- |
| **ALTER TABLE** users  **ADD COLUMN** email TEXT NOT NULL; |

В данном случае столбец email имеет тип TEXT и для него определено ограничение NOT NULL.

***Переименование столбца***

Переименуем столбец email в login:

|  |  |
| --- | --- |
|  | **ALTER TABLE** Users  **RENAME COLUMN** email TO login; |

***Удаление столбца***

Удалим столбец login из таблицы Users:

|  |
| --- |
| **ALTER TABLE** Users  **DROP COLUMN** login; |

***Изменение описания столбца***

Оператор **ALTER TABLE** также можно использовать для изменения описания существующего столбца в таблице: *изменения типа данных или размера столбца*.

Например, эта инструкция изменяет тип данных столбца age на INTEGER в таблице Users:

**ALTER TABLE** Users   
**MODIFY COLUMN** age INTEGER;

Также можно изменить размер столбца. Например, эта инструкция увеличивает размер столбца email до 255 символов:

**ALTER TABLE** Users   
**MODIFY COLUMN** VARCHAR(255);

***Переименование таблицы***

Для переименования таблицы применяется операторы**RENAME TO,** после которого указывается новое имя таблицы:

|  |  |
| --- | --- |
|  | **ALTER TABLE** Users  **RENAME TO** People; |

Здесь таблица Users переименовывается в People.

***Добавление ограничений***

Начиная с версии 3.25.2, SQLite не поддерживает добавление ограничения в существующую таблицу. В ранних версиях можно было использовать команду **ALTER TABLE**.

Вот пример добавления именованного ограничения уровня таблицы:

**ALTER TABLE** Employees   
**ADD CONSTRAINT** chk\_Personal **CHECK** (age >=25 OR salary <=10000);

В поздних версиях можно добавить ограничения, только создавая дубль таблицы той же структуры, но с объявленными ограничениями, и того же наполнения. Это можно сделать в рамках транзакции:

**BEGIN**;

-- Создать новую таблицу

**CREATE TABLE** new\_table (

[...],

**CHECK** ([...])

);

-- Копировать данные из старой таблицы в новую

**INSERT INTO** new\_table **SELECT** \* FROM old\_table;

-- Удалить старую таблицу

**DROP TABLE** old\_table;

-- Переименовать новую таблицу по имени старой

**ALTER TABLE** new\_table **RENAME TO** old\_table;

-- Зафиксировать изменения

**COMMIT**;

Другой способ, если ещё не вносили изменения в БД, то следует отредактировать сценарий создания и наполнения определенных таблиц, добавив нужные ограничения и выполнить сценарий.

**Обновление, добавление и замена записей**

***Обновление записей***

Для обновления данных в SQLite применяется команда **UPDATE**, которая имеет следующий синтаксис:

|  |  |
| --- | --- |
|  | **UPDATE** *имя\_таблицы*  **SET** *столбец1* = *значение1*, *столбец2* = *значение2*, ... *столбецN* = *значениеN*  **WHERE** *условие\_обновления* |

Например, имеем таблицу Products, созданную и наполненную по следующему сценарию:

|  |
| --- |
| **CREATE TABLE** Products  (   id INTEGER PRIMARY KEY AUTOINCREMENT,      name TEXT NOT NULL,      company TEXT NOT NULL,      product\_count INTEGER DEFAULT 0,      price INTEGER  );  **INSERT INTO** Products (name, company, product\_count, price)  **VALUES**  ('iPhone 13', 'Apple', 3, 76000),  ('iPhone 12', 'Apple', 2, 51000),  ('Galaxy S21', 'Samsung', 2, 56000),  ('Galaxy S20', 'Samsung', 1, 41000),  ('P40 Pro', 'Huawei', 5, 36000)  ; |

Увеличим в этой таблице цену всех товаров цену на 3000:

|  |
| --- |
| **UPDATE** Products  **SET** price = price + 3000  ; |

Выполнено безусловное обновление всех строк таблицы. Теперь конкретизируем обновляемые строки с помощью выражения **WHERE**. Например, изменим название производителя с 'Samsung' на 'Samsung Inc.':

|  |
| --- |
| **UPDATE** Products  **SET** company = 'Samsung Inc.'  **WHERE** company = 'Samsung'  ; |

Также можно обновлять сразу несколько столбцов:

|  |
| --- |
| **UPDATE** Products  **SET** company = 'Samsung',      product\_count = product\_count + 3000  **WHERE** company = 'Samsung Inc.'  ; |

***Частные случаи добавления данных***

В части 1 этой лабораторной работы мы в целом изучили инструкцию **INSERT** и применили её для наполнения таблиц БД. Здесь рассмотрим частные случаи.

Добавление NULL

При использовании INSERT можно опускать столбцы, которые поддерживают значение **NULL** (которые не имеют ограничения **NOT NULL**):

|  |  |
| --- | --- |
|  | **INSERT INTO** Users (name) **VALUES** ('Sam'); |

В данном случае для столбца age не указано значение, и поскольку данный столбец поддерживает значение NULL, то для него будет установлено значение NULL.

Также подобным столбцам, которые поддерживают NULL, можно явным образом передатьNULL:

|  |
| --- |
| **INSERT INTO** users (name, age) **VALUES** (NULL, NULL); |

Значения по умолчанию

Если для столбца задано ограничение**DEFAULT**, то есть значение по умолчанию, то для него тоже можно не передавать значение. Например, возьмем следующую таблицу:

|  |
| --- |
| **CREATE TABLE** Users  (  id INTEGER PRIMARY KEY AUTOINCREMENT,      name TEXT **DEFAULT** 'Undefined',      age INTEGER **DEFAULT** 18  ); |

Теперь столбцы name и age имеют значения по умолчанию. При добавлении данных из можно опустить:

|  |
| --- |
| **INSERT INTO** Users(name) **VALUES** ('Tom');  **INSERT INTO** Users(age) **VALUES** (22); |

Если все столбцы поддерживают значения по умолчанию или автогенерацию или значение NULL, то с помощью ключевого слова DEFAULT можно явно указать, что в качестве значения будут использоваться значения по умолчанию:

|  |  |
| --- | --- |
|  | **INSERT INTO** Users **DEFAULT VALUES**; |

В этом случае столбцы, для которых определено значение по умолчанию, получат это значение. Остальные столбцы получать значение NULL.

Замена данных

Команда **REPLACE** по сути является сокращением от **INSERT OR REPLACE**. Ее идея состоит в следующем. Пи вставке данных может нарушаются ограничения **UNIQUE** или **PRIMARY KEY**, например, когда мы пытаемся добавить для столбца, который должен иметь уникальные значения, данные, которые уже есть в таблице. Этот конфликт ограничений призвана разрешить команда **REPLACE**.

Эта команда сначала удаляет строку, которая вызвала конфликт на уникальность данных, и затем вместо нее вставляет новую строку. То есть фактически все выглядит как замена строки.

Если происходит конфликт с ограничением **NOT NULL** (в столбец, для которого задано ограничение **NOT NULL**, вставляется значение NULL), команда **REPLACE** заменяется вставляемое значение NULL значением по умолчанию, которое принято для этого столбца. Если для столбца не установлено значение по умолчанию, то выполнение запроса отменяется.

Если конфликтов с ограничениями не происходит, то команда **REPLACE,** по сути, действует аналогично команде **INSERT**.

Команда **REPLACE** имеет следующий формальный синтаксис:

|  |
| --- |
| **INSERT OR REPLACE INTO** *таблица*(*столбец1, столбец2, ... столбецN*)  **VALUES**(*значение1, значение2, ... значениеN*); |

Вместо **INSERT OR REPLACE** можно писать просто **REPLACE**.

**Удаление записей из таблицы**

Команда **DELETE** удаляет данные из БД. Она имеет следующий формальный синтаксис:

|  |  |
| --- | --- |
|  | **DELETE FROM** *имя\_таблицы*  **WHERE** *условие\_удаления;* |

Например, из таблицы Products удалим все товары, производителем которых является Apple и которые имеют цену меньше 60000:

|  |
| --- |
| **DELETE FROM** Products  **WHERE** company='Apple' AND price < 60000; |

Будьте внимательны, *при отсутствии условия* из таблицы *будут удалены все записи*!

**Удаление таблицы**

Инструкция **DROP TABLE** используется для удаления определения таблицы и всех связанных данных, индексов, триггеров, ограничений и спецификаций разрешений для этой таблицы.

Ниже приведен основной синтаксис оператора DROP TABLE. Можно указать имя базы данных вместе с именем таблицы следующим образом:

**DROP TABLE** *имя\_БД.имя\_таблицы*;

Нужно быть предельно осторожным при использовании этой команды, потому что, как только таблица будет удалена*, вся информация* *будет потеряна навсегда*.

Можно применять условное удаление таблицы при её наличии:

**DROP TABLE IF EXISTS** *имя\_таблицы*;