

Модуль 4. Введение в СУБД

Тема 4.10. SQL и СУБД SQLite

4 часа

Оглавление

4.10. Введение в SQL. СУБД SQLite	2
4.10.1. СУБД SQLite.....	2
4.10.2. Введение в SQL	2
Типы команд SQL	3
Запись SQL-операторов	3
Комментарии в SQL	4
4.10.3. Создание таблиц - оператор CREATE TABLE	4
Упражнение 4.10.1.....	6
4.10.4. Добавление записей в таблицу - INSERT	6
4.10.5. Выборка данных - инструкция SELECT	7
Инструкция SELECT.....	7
Предложение FROM	8
Предикат DISTINCT	8
Предложение WHERE	9
Предложение ORDER BY.....	11
4.10.6. Изменение таблицы - UPDATE.....	12
4.10.7. Удаление записей - DELETE	13
4.10.8. Агрегированные запросы.....	14
4.10.9. Инструкция INSERT с параметром SELECT.....	15
4.10.10. Работа с базой данных SQLite на Android-устройстве	16
Минипроект 4.1. Продолжение	18
Благодарности	29

4.10. Введение в SQL. СУБД SQLite

4.10.1. СУБД SQLite

SQLite – компактная локальная реляционная СУБД. Исходный код библиотеки передан в общественное достояние. В 2005 году проект получил награду Google-O'Reilly Open Source Awards. Слово «встраиваемый» (embedded) означает, что SQLite не использует парадигму клиент-сервер, то есть движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а предоставляет библиотеку, с которой программа компонуется и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

Благодаря архитектуре движка возможно использовать SQLite как на встраиваемых системах, так и на выделенных машинах с гигабайтными массивами данных.

По ряду оценок, функциональность SQLite находится где-то между MySQL и PostgreSQL. Однако, на практике, SQLite не редко оказывается в 2-3 раза (и даже больше) быстрее. Такое возможно благодаря высокоупорядоченной внутренней архитектуре и устранению необходимости в соединениях типа «сервер-клиент» и «клиент-сервер».



На практике приложения под Android не используют СУБД SQLite для хранения данных большого объема и со сложной структурой.

Это связано с тем, что хотя SQLite позволяет создавать БД объемом до 32 терабайт (с версии 3), запросы к такой БД будут загружать вычислительные ресурсы мобильного устройства, а следовательно существенно тратить заряд батареи. Для таких случаев наиболее распространено применение клиент-серверных СУБД.

Далее мы приступим к изучению SQL – языка запросов для реляционных БД. И по ходу изложения будем отмечать особенности, характерные для СУБД SQLite.

4.10.2. Введение в SQL

Одним из языков, появившихся в результате разработки реляционной модели данных, является язык SQL (Structured Query Language), который в настоящее время получил очень широкое

распространение и фактически превратился в стандартный язык реляционных баз данных. Стандарт на язык SQL был выпущен Американским национальным институтом стандартов (ANSI) в 1986 г., а в 1987 г. Международная организация стандартов (ISO) приняла его в качестве международного. Нынешний стандарт SQL известен под названием SQL/92.

В настоящее время язык SQL поддерживается многими десятками СУБД различных типов, разработанных для самых разнообразных вычислительных платформ, начиная от персональных компьютеров и заканчивая мейнфреймами.

Типы команд SQL

Реализация в SQL концепции операций, ориентированных на табличное представление данных, позволила создать компактный язык с небольшим набором предложений. Язык SQL может использоваться как для выполнения запросов к данным, так и для построения прикладных программ.

Основные категории команд языка SQL предназначены для выполнения различных функций, включая построение объектов базы данных и манипулирование ими, начальную загрузку данных в таблицы, обновление и удаление существующей информации, выполнение запросов к базе данных, управление доступом к ней и ее общее администрирование.

Основные категории команд языка SQL:

1. Data Definition Language (DDL) – язык определения данных;
2. Data Manipulation Language (DML) – язык манипулирования данными;
3. Data Query Language (DQL) – язык для выборки данных;
4. Data Control Language (DCL) – язык управления данными;
5. команды администрирования данных;
6. команды управления транзакциями

Определение структур базы данных (DDL)

Язык определения данных (Data Definition Language, DDL) позволяет создавать и изменять структуру объектов базы данных, например, создавать и удалять таблицы. Основными командами языка DDL являются следующие: CREATE TABLE, ALTER TABLE, DROP TABLE, CREATE INDEX, ALTER INDEX, DROP INDEX.

Манипулирование данными (DML)

Язык манипулирования данными (Data Manipulation Language, DML) используется для манипулирования информацией внутри объектов реляционной базы данных посредством трех основных команд: INSERT, UPDATE, DELETE.

Выборка данных (DQL)

Язык запросов DQL наиболее известен пользователям реляционной базы данных, несмотря на то, что он включает всего одну команду SELECT. Эта команда вместе со своими многочисленными опциями и предложениями используется для формирования запросов к реляционной базе данных.

Оставшиеся три категории мы не будем изучать в данном курсе подробно.

Запись SQL-операторов

Для успешного изучения языка SQL необходимо привести краткое описание структуры SQL-операторов и нотации, которые используются для определения формата различных конструкций языка.

Идентификаторы языка SQL предназначены для обозначения объектов в базе данных: имен таблиц, представлений(view), столбцов и других объектов базы данных. В соответствии со стандартом SQL идентификатор:

- по умолчанию может содержать строчные и прописные буквы латинского алфавита (A-Z, a-z), цифры (0-9), символ подчеркивания (_);
- может иметь длину до 128 символов;
- должен начинаться с буквы;
- не может содержать пробелы.

Большинство компонентов языка не чувствительны к регистру. Поскольку у языка SQL свободный формат, отдельные SQL-операторы и их последовательности будут иметь более читаемый вид при использовании отступов и выравнивания.

Синтаксические определения языков часто задают с помощью специальной металингвистической символики, называемой формулами Бэкуса-Науэра(БНФ). Прописные буквы используются для записи зарезервированных слов и должны указываться в операторах точно так, как это будет показано. Строчные буквы употребляются для записи слов, определяемых пользователем. Применяемые в нотации БНФ символы и их обозначения показаны в таблице.

Символ	Обозначение
::=	Равно по определению
	Необходимость выбора одного из нескольких приведенных значений
<...>	Описанная с помощью метаязыка структура языка
{...}	Обязательный выбор некоторой конструкции из списка
[...]	Необязательный выбор некоторой конструкции из списка
[...n]	Необязательная возможность повторения конструкции от нуля до нескольких раз

Комментарии в SQL

Комментарии в SQLite обозначаются двумя последовательными минусами (--), которые комментируют остаток строки. Многострочные комментарии, как в языке Java, обозначаются парами символов (/* */). Можно заметить, что без существенных причин не стоит использовать многострочные комментарии

4.10.3. Создание таблиц - оператор CREATE TABLE

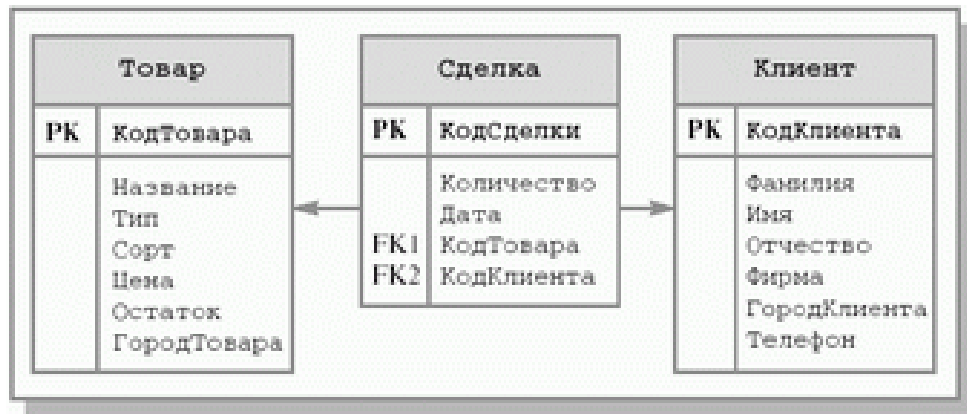
Таблицы базы данных создаются с помощью оператора CREATE TABLE. Эта команда создает пустую таблицу, т.е. таблицу, не имеющую строк. Значения в эту таблицу вводятся с помощью оператора INSERT. Оператор CREATE TABLE определяет имя таблицы и множество поименованных столбцов в указанном порядке. Для каждого столбца могут быть определены тип и размер. Каждая создаваемая таблица должна иметь по крайней мере один столбец.

Базовый синтаксис оператора создания таблицы имеет следующий вид:

```
CREATE TABLE имя_таблицы
(имя_столбца тип_данных
 [NULL | NOT NULL ] [,...n])
```

Конструкция NOT NULL обозначает, что поле должно быть обязательно заполнено.

В качестве примера будет использоваться небольшая база данных, отражающая процесс поставки или продажи некоторого товара постоянным клиентам.



Пример 1.

Создать таблицу для хранения данных о товарах, поступающих в продажу в некоторой торговой фирме. Необходимо учесть такие сведения, как название и тип товара, его цена, сорт и город, где товар производится.

```

CREATE TABLE Товар
(КодТовара INTEGER PRIMARY KEY AUTOINCREMENT,
Название TEXT NOT NULL UNIQUE,
Цена REAL NOT NULL,
Тип TEXT NOT NULL,
Сорт TEXT NOT NULL,
Остаток INTEGER)
  
```

Создать таблицу для сохранения сведений о постоянных клиентах с указанием названий города и фирмы, фамилии, имени и отчества клиента, номера его телефона.

```

CREATE TABLE Клиент
(
КодКлиента INTEGER PRIMARY KEY AUTOINCREMENT,
Фирма TEXT NOT NULL,
ФИО TEXT NOT NULL,
ГородКлиента TEXT,
Телефон TEXT)
  
```

Создать таблицу для сохранения сведений о сделках с указанием количества и даты сделки.

```

CREATE TABLE Сделка
(КодСделки INTEGER PRIMARY KEY AUTOINCREMENT,
КодТовара INTEGER NOT NULL,
КодКлиента INTEGER NOT NULL,
Количество INTEGER NOT NULL DEFAULT 0,
Дата NUMERIC NOT NULL,
CONSTRAINT fk_Товар
FOREIGN KEY(КодТовара) REFERENCES Товар,
CONSTRAINT fk_Клиент
FOREIGN KEY(КодКлиента) REFERENCES Клиент)
  
```

Первый столбец обозначен, как primary key (первичный ключ), это понятие мы уже разбирали в предыдущей теме. Слово autoincrement указывает, что база данных будет автоматически увеличивать значение ключа при добавлении каждой записи, что и обеспечивает его уникальность (поле - счетчик).

Создание таблицы в SQLite

В SQLite существует договоренность первый столбец всегда называть `_id`, это не жесткое требование SQLite, однако может понадобиться при использовании контент-провайдера в Android.

Стоит также иметь в виду, что в SQLite, в отличие от многих других СУБД, типы данных столбцов являются лишь подсказкой, т. е. не вызовет никаких нареканий попытка записать строку в столбец, предназначенный для хранения целых чисел или наоборот. Этот факт можно рассматривать, как особенность базы данных, а не как ошибку, на это обращают внимание авторы SQLite.

Возможные типы полей:

- TEXT - строки или символы в кодировке UTF-8, UTF-16BE или UTF-16LE;
- INTEGER - целое число;
- REAL - дробное число;
- BLOB - бинарные данные;
- TIMESTAMP - метка времени;
- NULL – пустое значение.

Как видите, отсутствует тип, работающий с датами. Можно использовать строковые значения, например, как 2015-02-16 (16 февраля 2015 года). Для даты со временем рекомендуется использовать формат 2015-02-16T07:58. В таких случаях можно использовать некоторые функции SQLite для добавления дней, установки начала месяца и т.д. Учтите, что SQLite не поддерживает часовые пояса. Также не поддерживается тип boolean. Используйте числа 0 для false и 1 для true. Не используйте тип BLOB для хранения данных (картинки) в Android. Лучше хранить в базе путь к изображениям, а сами изображения хранить в файловой системе.



Как было отмечено выше, SQLite поддерживает типы TEXT (аналог String в Java), INTEGER (аналог long в Java) и REAL (аналог double в Java). Остальные типы следует конвертировать, прежде чем сохранять в базе данных. SQLite сама по себе не проверяет типы данных, поэтому вы можете записать целое число в колонку, предназначенную для строк и наоборот.

Упражнение 4.10.1

Выполним Пример 1 в SQLite через консольную утилиту `sqlite`. Все дальнейшие примеры также будем запускать через консольную утилиту.

4.10.4. Добавление записей в таблицу - INSERT

Оператор INSERT применяется для добавления записей в таблицу. Формат оператора:

```
INSERT INTO <имя_таблицы>
```

```
[ (имя_столбца [, ...n]) ]  
{VALUES (значение[, ...n]) } |  
<SELECT_оператор>}
```

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Первая форма оператора INSERT с параметром VALUES предназначена для вставки единственной строки в указанную таблицу. Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список может быть опущен, тогда подразумеваются все столбцы таблицы (кроме объявленных как счетчик), причем в определенном порядке, установленном при создании таблицы. Если в операторе INSERT указывается конкретный список имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значение NULL, за исключением тех случаев, когда при описании столбца использовался параметр DEFAULT. Список значений должен следующим образом соответствовать списку столбцов:

- количество элементов в обоих списках должно быть одинаковым;
- должно существовать прямое соответствие между позицией одного и того же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.
- типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

Пример 2

Добавить в таблицу ТОВАР новую запись.

```
INSERT INTO Товар (Название, Тип, Цена)  
VALUES (" Славянский ", " шоколад ", 12)
```

Если столбцы таблицы ТОВАР указаны в полном составе и в том порядке, в котором они перечислены при создании таблицы ТОВАР, оператор можно упростить.

```
INSERT INTO Товар VALUES (" Славянский ",  
" шоколад ", 12)
```

4.10.5. Выборка данных - инструкция SELECT

Инструкция SELECT

Оператор SELECT – один из наиболее важных и самых распространенных операторов SQL. Он позволяет производить выборки данных из таблиц и преобразовывать к нужному виду полученные результаты.

Оператор SELECT имеет следующий формат:

```
SELECT [ALL | DISTINCT ] { * | [имя_столбца  
[AS новое_имя]] } [, ...n]  
FROM имя_таблицы [[AS] псевдоним] [, ...n]  
[WHERE <условие_поиска>]  
[GROUP BY имя_столбца [, ...n]]  
[HAVING <критерии_выбора_групп>]  
[ORDER BY имя_столбца [, ...n]]
```

Оператор SELECT определяет поля (столбцы), которые будут входить в результат выполнения запроса. В списке они разделяются запятыми и приводятся в такой очередности, в какой должны быть представлены в результате запроса. Если используется имя поля, содержащее пробелы или разделители, его следует заключить в квадратные скобки. Символом * можно выбрать все поля, а вместо имени поля применить выражение из нескольких имен.

Если обрабатывается ряд таблиц, то (при наличии одноименных полей в разных таблицах) в списке полей используется полная спецификация поля, т.е. Имя_таблицы.Имя_поля.

Предложение FROM

Предложение FROM задает имена таблиц и представлений, которые содержат поля, перечисленные в операторе SELECT. Необязательный параметр псевдонима – это сокращение, устанавливаемое для имени таблицы.

Обработка элементов оператора SELECT выполняется в следующей последовательности:

FROM – определяются имена используемых таблиц;

WHERE – выполняется фильтрация строк объекта в соответствии с заданными условиями;

GROUP BY – образуются группы строк, имеющих одно и то же значение в указанном столбце;

HAVING – фильтруются группы строк объекта в соответствии с указанным условием;

SELECT – устанавливается, какие столбцы должны присутствовать в выходных данных;

ORDER BY – определяется упорядоченность результатов выполнения операторов.

Порядок предложений и фраз в операторе SELECT не может быть изменен. Только два предложения SELECT и FROM являются обязательными, все остальные могут быть опущены. SELECT – закрытая операция: результат запроса к таблице представляет собой другую таблицу. Существует множество вариантов записи данного оператора, что иллюстрируется приведенными ниже примерами.

Пример 7. Составить список сведений о всех клиентах.

```
SELECT * FROM Клиент
```

Параметр WHERE определяет критерий отбора записей из входного набора. Но в таблице могут присутствовать повторяющиеся записи (дубликаты). Предикат ALL задает включение в выходной набор всех дубликатов, отобранных по критерию WHERE. Нет необходимости указывать ALL явно, поскольку это значение действует по умолчанию.

Пример 8. Составить список всех фирм.

```
SELECT ALL Клиент.Фирма FROM Клиент  
или (эквивалентно)
```

```
SELECT Клиент.Фирма FROM Клиент
```

Результат выполнения запроса может содержать дублирующие значения, поскольку в отличие от операций реляционной алгебры оператор SELECT не исключает повторяющихся значений при выполнении выборки данных.

Предикат DISTINCT

Предикат DISTINCT следует применять в тех случаях, когда требуется отбросить блоки данных, содержащие дублирующие записи в выбранных полях. Значения для каждого из приведенных в

инструкции SELECT полей должны быть уникальными, чтобы содержащая их запись смогла войти в выходной набор.

Причиной ограничения в применении DISTINCT является то обстоятельство, что его использование может резко замедлить выполнение запросов.

Откорректированный пример выглядит следующим образом:

```
SELECT DISTINCT Клиент.Фирма  
FROM Клиент
```

Предложение WHERE

С помощью WHERE-параметра пользователь определяет, какие блоки данных из приведенных в списке FROM таблиц появятся в результате запроса. За ключевым словом WHERE следует перечень условий поиска, определяющих те строки, которые должны быть выбраны при выполнении запроса. Существует пять основных типов условий поиска (или предикатов):

1. Сравнение: сравниваются результаты вычисления одного выражения с результатами вычисления другого.
2. Диапазон: проверяется, попадает ли результат вычисления выражения в заданный диапазон значений.
3. Принадлежность множеству: проверяется, принадлежит ли результат вычислений выражения заданному множеству значений.
4. Соответствие шаблону: проверяется, отвечает ли некоторое строковое значение заданному шаблону.
5. Значение NULL: проверяется, содержит ли данный столбец определитель NULL (неизвестное значение).

Сравнение

В языке SQL можно использовать следующие операторы сравнения: = – равенство; < – меньше; > – больше; <= – меньше или равно; >= – больше или равно; <> – не равно.

Пример 9. Показать все операции отпуска товаров объемом больше 20.

```
SELECT * FROM Сделка  
WHERE Количество>20
```

Более сложные предикаты могут быть построены с помощью логических операторов AND, OR или NOT, а также скобок, используемых для определения порядка вычисления выражения. Вычисление выражения в условиях выполняется по следующим правилам:

- Выражение вычисляется слева направо.
- Первыми вычисляются подвыражения в скобках.
- Операторы NOT выполняются до выполнения операторов AND и OR.
- Операторы AND выполняются до выполнения операторов OR.

Для устранения любой возможной неоднозначности рекомендуется использовать скобки.

Пример 10. Вывести список товаров, цена которых больше или равна 100 и меньше или равна 150.

```
SELECT Название, Цена  
FROM Товар  
WHERE Цена>=100 And Цена<=150
```

Пример 11. Вывести список клиентов из Москвы или из Самары.

```
SELECT Фамилия, ГородКлиента
```

```
FROM Клиент
WHERE ГородКлиента="Москва" Or
      ГородКлиента="Самара"
```

Диапазон

Оператор BETWEEN используется для поиска значения внутри некоторого интервала, определяемого своими минимальным и максимальным значениями. При этом указанные значения включаются в условие поиска.

Пример 12. Вывести список товаров, цена которых лежит в диапазоне от 100 до 150

```
SELECT Название, Цена
FROM Товар
WHERE Цена BETWEEN 100 And 150
```

При использовании отрицания NOT BETWEEN требуется, чтобы проверяемое значение лежало вне границ заданного диапазона.

Пример 13. Вывести список товаров, цена которых НЕ лежит в диапазоне от 100 до 150.

```
SELECT Товар.Название, Товар.Цена
FROM Товар
WHERE Товар.Цена NOT BETWEEN 100 And 150
```

Или (что эквивалентно)

```
SELECT Товар.Название, Товар.Цена
FROM Товар
WHERE (Товар.Цена<100) OR (Товар.Цена>150)
```

Принадлежность множеству

Оператор IN используется для сравнения некоторого значения со списком заданных значений, при этом проверяется, соответствует ли результат вычисления выражения одному из значений в предоставленном списке. При помощи оператора IN может быть достигнут тот же результат, что и в случае применения оператора OR, однако оператор IN выполняется быстрее.

Пример 14. Вывести список клиентов из Москвы или из Самары

```
SELECT Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента IN ("Москва", "Самара")
```

NOT IN используется для отбора любых значений, кроме тех, которые указаны в представленном списке.

Пример 15. Вывести список клиентов, проживающих не в Москве и не в Самаре.

```
SELECT Фамилия, ГородКлиента
FROM Клиент
WHERE ГородКлиента
      NOT IN ("Москва", "Самара")
```

Соответствие шаблону

С помощью оператора LIKE можно выполнять сравнение выражения с заданным шаблоном, в котором допускается использование символов-заменителей:

- Символ % – вместо этого символа может быть подставлено любое количество произвольных символов.

- Символ `_` заменяет один символ строки.
- `[]` – вместо символа строки будет подставлен один из возможных символов, указанный в этих ограничителях.
- `[^]` – вместо соответствующего символа строки будут подставлены все символы, кроме указанных в ограничителях.

Пример 16. Найти клиентов, у которых в номере телефона вторая цифра – 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
FROM Клиент
WHERE Клиент.Телефон LIKE '_4%'
```

Пример 17. Найти клиентов, у которых в номере телефона вторая цифра – 2 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
FROM Клиент
WHERE Клиент.Телефон LIKE '_[2,4]%'
```

Пример 18. Найти клиентов, у которых в номере телефона вторая цифра 2, 3 или 4.

```
SELECT Клиент.Фамилия, Клиент.Телефон
FROM Клиент
WHERE Клиент.Телефон LIKE '_[2-4]%'
```

Пример 19. Найти клиентов, у которых в фамилии встречается слог "ро".

```
SELECT Клиент.Фамилия
FROM Клиент
WHERE Клиент.Фамилия LIKE "%ро%"
```

Значение NULL

Оператор `IS NULL` используется для сравнения текущего значения со значением `NULL` – специальным значением, указывающим на отсутствие любого значения. `NULL` – это не то же самое, что знак пробела (пробел – допустимый символ) или ноль (0 – допустимое число). `NULL` отличается и от строки нулевой длины (пустой строки).

Пример 20. Найти сотрудников, у которых нет телефона (поле Телефон не содержит никакого значения).

```
SELECT Фамилия, Телефон
FROM Клиент
WHERE Телефон IS NULL
```

`IS NOT NULL` используется для проверки присутствия значения в поле.

Пример 21. Выборка клиентов, у которых есть телефон (поле Телефон содержит какое-либо значение).

```
SELECT Клиент.Фамилия, Клиент.Телефон
FROM Клиент
WHERE Клиент.Телефон Is Not Null
```

Предложение ORDER BY

В общем случае строки в результирующей таблице SQL-запроса никак не упорядочены. Однако их можно требуемым образом отсортировать, для чего в оператор `SELECT` помещается фраза `ORDER BY`, которая сортирует данные выходного набора в заданной последовательности. Сортировка может выполняться по нескольким полям, в этом случае они перечисляются за ключевым словом `ORDER BY` через запятую. Способ сортировки задается ключевым словом, указываемым в рамках параметра `ORDER BY` следом за названием поля, по которому выполняется сортировка. По

умолчанию реализуется сортировка по возрастанию. Явно она задается ключевым словом ASC. Для выполнения сортировки в обратной последовательности необходимо после имени поля, по которому она выполняется, указать ключевое слово DESC. Фраза ORDER BY позволяет упорядочить выбранные записи в порядке возрастания или убывания значений любого столбца или комбинации столбцов, независимо от того, присутствуют эти столбцы в таблице результата или нет. Фраза ORDER BY всегда должна быть последним элементом в операторе SELECT.

Пример 22. Вывести список клиентов в алфавитном порядке.

```
SELECT Клиент.Фамилия, Клиент.Фирма
FROM Клиент
ORDER BY Клиент.Фамилия
```

Во фразе ORDER BY может быть указано и больше одного элемента. Главный (первый) ключ сортировки определяет общую упорядоченность строк результирующей таблицы. Если во всех строках результирующей таблицы значения главного ключа сортировки являются уникальными, нет необходимости использовать дополнительные ключи сортировки. Однако, если значения главного ключа не уникальны, в результирующей таблице будет присутствовать несколько строк с одним и тем же значением старшего ключа сортировки. В этом случае, возможно, придется упорядочить строки с одним и тем же значением главного ключа по какому-либо дополнительному ключу сортировки.

Пример 23. Вывести список фирм и клиентов. Названия фирм упорядочить в алфавитном порядке, имена клиентов в каждой фирме отсортировать в обратном порядке.

```
SELECT Клиент.Фирма, Клиент.Фамилия
FROM Клиент
ORDER BY Клиент.Фирма, Клиент.Фамилия DESC
```

Язык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.

4.10.6. Изменение таблицы - UPDATE

Оператор UPDATE применяется для изменения значений в группе записей или в одной записи указанной таблицы.

Формат оператора:

```
UPDATE имя_таблицы SET имя_столбца=
    <выражение>[, ...n]
[WHERE <условие_отбора>]
```

Параметр имя_таблицы — это либо имя таблицы базы данных, либо имя обновляемого представления. В предложении SET указываются имена одного и более столбцов, данные в которых необходимо изменить. Предложение WHERE является необязательным. Если оно опущено, значения указанных столбцов будут изменены во всех строках таблицы. Если предложение WHERE присутствует, то обновлены будут только те строки, которые удовлетворяют условию отбора. Выражение представляет собой новое значение соответствующего столбца и должно быть совместимо с ним по типу данных.

Пример 24. Для товаров первого сорта установить цену в значение 140 и остаток – в значение 20 единиц.

```
UPDATE Товар SET Товар.Цена=140, Товар.Остаток=20
WHERE Товар.Сорт=" Первый "
```

Пример 25. Увеличить цену товаров первого сорта на 25%.

```
UPDATE Товар SET Товар.Цена=Товар.Цена*1.25
WHERE Товар.Сорт=" Первый "
```

Пример 26. В сделке с максимальным количеством товара увеличить число товаров на 10%.

```
UPDATE Сделка SET Сделка.Количество=
    Сделка.Количество*1.1
WHERE Сделка.Количество=
    (SELECT Max(Сделка.Количество) FROM Сделка)
```

4.10.7. Удаление записей - DELETE

Оператор DELETE предназначен для удаления группы записей из таблицы.

Формат оператора:

```
DELETE
FROM <имя_таблицы> [WHERE <условие_отбора>]
```

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя обновляемого представления.

Если предложение WHERE присутствует, удаляются записи из таблицы, удовлетворяющие условию отбора. Если опустить предложение WHERE, из таблицы будут удалены все записи, однако сама таблица сохранится.

Пример 27. Удалить все прошлогодние сделки.

```
DELETE
FROM Сделка
WHERE Year(Сделка.Дата)=Year(GETDATE())-1
```

В приведенном примере условие отбора формируется с учетом года (функция Year) от текущей даты (функция GETDATE()).

В документации по SQLite указано, что не поддерживаются конструкции SQL, которые позволяют удалить или изменить существующий столбец в таблице (например, его имя или тип). Но можно прибегнуть к «хитрости»: создать другую таблицу, скопировав в нее нужные данные из старой:

```
-- создаем временную таблицу
CREATE TEMPORARY TABLE Temper_backup(name,temp);
-- копируем данные из таблицы Temper во временную таблицу Temper_backup
INSERT INTO Temper_backup SELECT name,temp FROM Temper;
-- удаляем таблицу Temper
DROP TABLE Temper;
```

```
-- создаем таблицу Temper
CREATE TABLE Temper(name,new);
-- вставляем данные из таблицы Temper_backup в таблицу Temper
INSERT INTO Temper SELECT name,temp FROM Temper_backup;
-- удаляем таблицу Temper_backup
DROP TABLE Temper_backup;
```

4.10.8. Агрегированные запросы

С помощью итоговых (агрегатных) функций в рамках SQL-запроса можно получить ряд обобщающих статистических сведений о множестве отобранных значений выходного набора.

Пользователю доступны следующие основные итоговые функции:

- Count (Выражение) - определяет количество записей в выходном наборе SQL-запроса;
- Min/Max (Выражение) - определяют наименьшее и наибольшее из множества значений в некотором поле запроса;
- Avg (Выражение) - эта функция позволяет рассчитать среднее значение множества значений, хранящихся в определенном поле отобранных запросом записей. Оно является арифметическим средним значением, т.е. суммой значений, деленной на их количество.
- Sum (Выражение) - вычисляет сумму множества значений, содержащихся в определенном поле отобранных запросом записей.

Чаще всего в качестве выражения выступают имена столбцов. Выражение может вычисляться и по значениям нескольких таблиц.

Все эти функции оперируют со значениями в единственном столбце таблицы или с арифметическим выражением и возвращают единственное значение. Функции COUNT, MIN и MAX применимы как к числовым, так и к нечисловым полям, тогда как функции SUM и AVG могут использоваться только в случае числовых полей, за исключением COUNT(*). При вычислении результатов любых функций сначала исключаются все пустые значения, после чего требуемая операция применяется только к оставшимся конкретным значениям столбца. Вариант COUNT(*) - особый случай использования функции COUNT, его назначение состоит в подсчете всех строк в результирующей таблице, независимо от того, содержатся там пустые, дублирующиеся или любые другие значения.

Если до применения обобщающей функции необходимо исключить дублирующиеся значения, следует перед именем столбца в определении функции поместить ключевое слово DISTINCT. Оно не имеет смысла для функций MIN и MAX, однако его использование может повлиять на результаты выполнения функций SUM и AVG, поэтому необходимо заранее обдумать, должно ли оно присутствовать в каждом конкретном случае. Кроме того, ключевое слово DISTINCT может быть указано в любом запросе не более одного раза.

Очень важно отметить, что итоговые функции могут использоваться только в списке предложения SELECT и в составе предложения HAVING. Во всех других случаях это недопустимо. Если список в предложении SELECT содержит итоговые функции, а в тексте запроса отсутствует фраза GROUP BY, обеспечивающая объединение данных в группы, то ни один из элементов списка предложения SELECT не может включать каких-либо ссылок на поля, за исключением ситуации, когда поля выступают в качестве аргументов итоговых функций.

Пример 28. Определить первое по алфавиту название товара.

```
SELECT Min(Товар.Название) AS Min_Название  
FROM Товар
```

Определить количество сделок.

```
SELECT Count(*) AS Количество_сделок  
FROM Сделка
```

Определить суммарное количество проданного товара.

```
SELECT Sum(Сделка.Количество)  
      AS Количество_товара  
FROM Сделка
```

Определить среднюю цену проданного товара.

```
SELECT Avg(Товар.Цена) AS Avg_Цена  
FROM Товар INNER JOIN Сделка  
      ON Товар.КодТовара=Сделка.КодТовара;
```

Подсчитать общую стоимость проданных товаров.

```
SELECT Sum(Товар.Цена*Сделка.Количество)  
      AS Стоимость  
FROM Товар INNER JOIN Сделка  
      ON Товар.КодТовара=Сделка.КодТовара
```

4.10.9. Инструкция INSERT с параметром SELECT

Она позволяет скопировать множество строк из одной таблицы в другую. Предложение SELECT может представлять собой любой допустимый оператор SELECT. Вставляемые в указанную таблицу строки в точности должны соответствовать строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператора INSERT, применимы и в этом случае.

Поскольку оператор SELECT в общем случае возвращает множество записей, то оператор INSERT в такой форме приводит к добавлению в таблицу аналогичного числа новых записей.

Пример 29

Добавить в итоговую таблицу сведения об общей сумме ежемесячных продаж каждого наименования товара.

```
INSERT INTO Итог  
      (Название, Месяц, Стоимость)  
SELECT Товар.Название, Month(Сделка.Дата)  
      AS Месяц, Sum(Товар.Цена*Сделка.Количество)  
      AS Стоимость  
FROM Товар INNER JOIN Сделка  
      ON Товар.КодТовара= Сделка.КодТовара  
GROUP BY Товар.Название, Month(Сделка.Дата)
```

4.10.10. Работа с базой данных SQLite на Android-устройстве

Когда ваше приложение создаёт базу данных, она сохраняется в каталоге **DATA/data/имя_пакета/databases/имя_базы.db**. Например, с помощью контент-провайдера несколько приложений могут использовать одну и ту же базу данных. Приведенный метод возвращает путь к каталогу **DATA**.

```
Environment.getDataDirectory()
```

Основными пакетами для работы с базой данных являются **android.database** и **android.database.sqlite**



*Кроме пользовательских таблиц, которые создаются для работы приложения, Android в той же БД дополнительно создаёт системную таблицу **android_metadata**. Это нужно помнить при ручном создании базы - необходимо самостоятельно создать, как минимум две таблицы: системную и свою рабочую.*

Для создания и обновления базы данных в Android предусмотрен класс **SQLiteOpenHelper**. При разработке приложения, работающего с базами данных, необходимо создать класс-наследник от **SQLiteOpenHelper**, в котором обязательно реализовать два абстрактных метода:

- **onCreate(SQLiteDatabase db)** — Вызывается один раз при создании БД;
- **onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)** — вызывается когда необходимо обновить БД (в данном случае под обновлением имеется в виду не обновление записей, а обновление структуры базы данных).

По желанию можно реализовать метод:

- **onOpen()** - вызывается при открытии базы данных.

```
// Класс для создания БД
private class OpenHelper extends SQLiteOpenHelper {

    OpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION); }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String query = "CREATE TABLE " + TABLE_NAME + " (" +
            COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COLUMN_DATE + " LONG, " +
            COLUMN_TITLE + " TEXT, " +
            COLUMN_ICON + " INTEGER); ";
        db.execSQL(query);
    }

    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion
    ) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```


В реальном приложении изменение структуры базы данных и ее таблиц, конечно, должно происходить без потери пользовательских данных.


В этом же классе принято объявлять открытые строковые константы для названия таблиц и полей создаваемой базы данных, которые клиенты могут использовать для определения столбцов при выполнении запросов к базе данных.

```
// Данные базы данных и таблиц
private static final String DATABASE_NAME = "itschool.db";
private static final int DATABASE_VERSION = 1;
private static final String TABLE_NAME = "MyData";

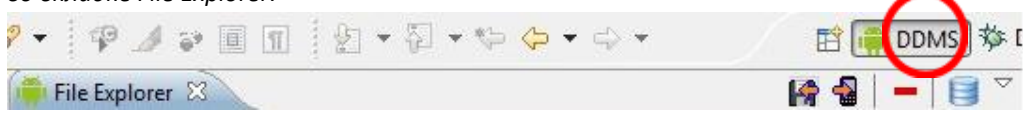
// Название столбцов
private static final String COLUMN_ID = "_id";
private static final String COLUMN_DATE = "Date";
private static final String COLUMN_TITLE = "Title";

// Номера столбцов
private static final int NUM_COLUMN_ID = 0;
private static final int NUM_COLUMN_DATE = 1;
private static final int NUM_COLUMN_TITLE = 2;
```

Для приложения можно создать несколько БД, все они будут доступны из любого класса программы, но не доступны для других программ. Класс, который предоставляет методы для добавления, обновления, удаления и выборки данных из БД носит имя SQLiteDatabase. Он работает с базой данных SQLite напрямую и имеет свои методы для открытия, запроса, обновления данных и закрытия базы, такие как insert(), update(), delete() соответственно.



Существует плагин *SQLiteManager* для *Eclipse*, позволяющий видеть данные в базе. Для работы может потребоваться скачать библиотеку *com.questoid.sqlitemanager_1.0.0.jar* и скопировать её в папку *Eclipse/dropins*, после этого перезагрузите *Eclipse* и откройте перспективу *DDMS*. Плагин будет находиться во вкладке *File Explorer*:



Щелкнув по значку, можно увидеть таблицы. Вкладка *Browse Data* позволяет смотреть непосредственно сами данные.

Основные этапы при работе с базой данных следующие:

1. Создание и открытие базы данных.
2. Создание таблицы.
3. Создание Insert-интерфейса (используется для вставки данных).
4. Создание Query-интерфейса (используется для выполнения запроса, обычно для выборки данных).
5. Закрытие базы данных.

Минипроект 4.1. Продолжение

Закончим разработку нашего приложения “Чемпионат России по футболу/хоккею”, начатого в предыдущей теме.

Нами спроектирована следующая структура таблицы, в которой мы будем хранить результаты матчей (вы можете реализовать свой вариант БД, но тогда это будет необходимо учесть в коде).

Таблица: tableMatches		
Столбец	Тип данных	Ограничения
id	INTEGER	•PRIMARY KEY AUTOINCREMENT
TeamHome	TEXT	
TeamGuest	TEXT	
GoalsHome	INT	
GoalsGuast	INT	

Ниже приведено приложение, которое позволяет:

- вводить данные о матчах: названия команд и счет игры между ними
- видеть и удалять все введенные данные
- удалять и редактировать нужную запись

Файл разметки `/res/layout/activity_main.xml`

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical" >

    <ListView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:id="@+id/list"
    />

</LinearLayout>
```

Файл разметки `/res/layout/activity_add.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <LinearLayout
        android:id="@+id/buttons"
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true" >
        <Button
            android:id="@+id/butSave"
            android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/save"
    />

    <Button
        android:id="@+id/butCancel"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/cancel"
    />

</LinearLayout>

<ScrollView
    android:layout_above="@id/buttons"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:fillViewport="true">

    <LinearLayout
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >
            <LinearLayout
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="vertical" >

                <TextView
                    style="@style/TextStyle"
                    android:layout_width="125dp"
                    android:layout_height="wrap_content"
                    android:text="@string/teamhomefield" />

                <EditText
                    android:id="@+id/TeamHome"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:inputType="text" />

            </LinearLayout>
            <LinearLayout
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="vertical" >

                <TextView
                    style="@style/TextStyle"
                    android:layout_width="120dp"
                    android:layout_height="wrap_content"
                    android:text="@string/teamquestfield" />

                <EditText
```

```
        android:id="@+id/TeamGuest"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="text" >

        <requestFocus />
    </EditText>

</LinearLayout>
</LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal" >
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="vertical" >

            <TextView
                style="@style/TextStyle"
                android:layout_width="123dp"
                android:layout_height="wrap_content"
                android:text="@string/goalshomefield" />

            <EditText
                android:id="@+id/GoalsHome"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:inputType="text" />

        </LinearLayout>
        <LinearLayout
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="vertical" >

            <TextView
                style="@style/TextStyle"
                android:layout_width="120dp"
                android:layout_height="wrap_content"
                android:text="@string/goalsguestfield" />

            <EditText
                android:id="@+id/GoalsGuest"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:inputType="text" />

        </LinearLayout>
    </LinearLayout>
</ScrollView>
</RelativeLayout>
```

Файл `/res/layout/item.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content">

    <LinearLayout
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:orientation="horizontal"
        android:paddingLeft="5dp" >
        <TextView
            android:id="@+id/TeamHome"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:textStyle="bold"
            android:textColor="#677566"
            />
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textSize="20sp"
                android:textStyle="bold"
                android:textColor="#677566"
                android:text="- "
            />
        <TextView
            android:id="@+id/TeamGuest"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:textStyle="bold"
            android:textColor="#677566"
            />
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:textSize="20sp"
                android:textStyle="bold"
                android:textColor="#677566"
                android:text=" "
            />
        <TextView
            android:id="@+id/TeamTotal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:textStyle="bold"
            android:textColor="#677566"
            />
    </LinearLayout>
</LinearLayout>

```

Файл /res/values/strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, SimpleDBActivity!</string>
    <string name="app_name">SimpleDB</string>

```

```

<string name="title">Чемпионат мира по футболу</string>
<string name="teamhomefield">Команда хозяев</string>
<string name="teamguestfield">Команда гостей</string>
<string name="goalshomefield">Голы хозяев</string>
<string name="goalsguestfield">Голы гостей</string>
<string name="save">Сохранить</string>
<string name="edit">Редактировать</string>
<string name="delete">Удалить</string>
<string name="cancel">Отмена</string>
<string name="icon">Icon</string>
<string name="deleteAll">Удалить все</string>
<string name="exit">Выход</string>
<string name="add">Добавить</string>
</resources>

```

Файл `/res/menu/menu_main.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/tools"
      tools:context="com.example.op.simplesdb.MainActivity">
    <item android:id="@+id/add"
          android:title="@string/add"
          android:icon="@android:drawable/ic_menu_add"/>

    <item android:id="@+id/deleteAll"
          android:title="@string/deleteAll"
          android:icon="@android:drawable/ic_menu_delete"/>

    <item android:id="@+id/exit"
          android:title="@string/exit"
          android:icon="@android:drawable/ic_menu_close_clear_cancel"/>
</menu>

```

Файл `/res/menu/context_menu.xml`

```

<?xml version="1.0" encoding="utf-8"?>
<menu
      xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/edit"
          android:title="@string/edit" />
    <item android:id="@+id/delete"
          android:title="@string/delete" />
</menu>

```

Файл класса `Matches`

```

public class Matches implements Serializable{
    private long id;
    private String teamhouse;
    private String teamguest;
    private int goalshouse;
    private int goalsguest;
}

```

```
public Matches (long id, String teamh, String teamg, int gh,int gg) {
    this.id = id;
    this.teamhouse = teamh;
    this.teamguest = teamg;
    this.goalshouse = gh;
    this.goalsguest=gg;
}

public long getId() {
    return id;
}

public String getTeamhouse() {
    return teamhouse;
}

public String getTeamguest() {
    return teamguest;
}

public int getGoalshouse() {
    return goalshouse;
}

public int getGoalsguest() {
    return goalsguest;
}
}
```

DBMatches.java

```
public class DBMatches {

    private static final String DATABASE_NAME = "simple.db";
    private static final int DATABASE_VERSION = 1;
    private static final String TABLE_NAME = "tableMatches";

    private static final String COLUMN_ID = "id";
    private static final String COLUMN_TEAMHOME = "TeamHome";
    private static final String COLUMN_TEAMGUAUST = "TeamGuest";
    private static final String COLUMN_GOALSHOME = "GoalsHome";
    private static final String COLUMN_GOALSGUAUST = "GoalsGuast";

    private static final int NUM_COLUMN_ID = 0;
    private static final int NUM_COLUMN_TEAMHOME = 1;
    private static final int NUM_COLUMN_TEAMGUAUST = 2;
    private static final int NUM_COLUMN_GOALSHOME = 3;
    private static final int NUM_COLUMN_GOALSGUEST = 4;

    private SQLiteDatabase mDataBase;

    public DBMatches(Context context) {
        OpenHelper mOpenHelper = new OpenHelper(context);
        mDataBase = mOpenHelper.getWritableDatabase();
    }
}
```



```

    public long insert(String teamhouse,String teamguest,int goalshouse,int
    goalsguest) {
        ContentValues cv=new ContentValues();
        cv.put(COLUMN_TEAMHOME, teamhouse);
        cv.put(COLUMN_TEAMGUAST, teamguest);
        cv.put(COLUMN_GOALSHOME, goalshouse);
        cv.put(COLUMN_GOALSGUAST,goalsguest);
        return mDataBase.insert(TABLE_NAME, null, cv);
    }

    public int update(Matches md) {
        ContentValues cv=new ContentValues();
        cv.put(COLUMN_TEAMHOME, md.getTeamhouse());
        cv.put(COLUMN_TEAMGUAST, md.getTeamguest());
        cv.put(COLUMN_GOALSHOME, md.getGoalshouse());
        cv.put(COLUMN_GOALSGUAST,md.getGoalsguest());
        return mDataBase.update(TABLE_NAME, cv, COLUMN_ID + " = ?",new String[] {
String.valueOf(md.getId())});
    }

    public void deleteAll() {
        mDataBase.delete(TABLE_NAME, null, null);
    }

    public void delete(long id) {
        mDataBase.delete(TABLE_NAME, COLUMN_ID + " = ?", new String[] {
String.valueOf(id) });
    }

    public Matches select(long id) {
        Cursor mCursor = mDataBase.query(TABLE_NAME, null, COLUMN_ID + " = ?", new
String[]{String.valueOf(id)}, null, null, null);

        mCursor.moveToFirst();
        String TeamHome = mCursor.getString(NUM_COLUMN_TEAMHOME);
        String TeamGuest = mCursor.getString(NUM_COLUMN_TEAMGUAST);
        int GoalsHome = mCursor.getInt(NUM_COLUMN_GOALSHOME);
        int GoalsGuest=mCursor.getInt(NUM_COLUMN_GOALSGUEST);
        return new Matches(id, TeamHome, TeamGuest, GoalsHome,GoalsGuest);
    }

    public ArrayList<Matches> selectAll() {
        Cursor mCursor = mDataBase.query(TABLE_NAME, null, null, null, null, null,
null);

        ArrayList<Matches> arr = new ArrayList<Matches>();
        mCursor.moveToFirst();
        if (!mCursor.isAfterLast()) {
            do {
                long id = mCursor.getLong(NUM_COLUMN_ID);
                String TeamHome = mCursor.getString(NUM_COLUMN_TEAMHOME);
                String TeamGuest = mCursor.getString(NUM_COLUMN_TEAMGUAST);
                int GoalsHome = mCursor.getInt(NUM_COLUMN_GOALSHOME);
                int GoalsGuest=mCursor.getInt(NUM_COLUMN_GOALSGUEST);
                arr.add(new Matches(id, TeamHome, TeamGuest,
GoalsHome,GoalsGuest));
            } while (mCursor.moveToNext());
        }
        return arr;
    }
}

```



```

private class OpenHelper extends SQLiteOpenHelper {

    OpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
    @Override
    public void onCreate(SQLiteDatabase db) {
        String query = "CREATE TABLE " + TABLE_NAME + " (" +
            COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            COLUMN_TEAMHOME+ " TEXT, " +
            COLUMN_TEAMGUAST + " TEXT, " +
            COLUMN_GOALSHOME + " INT,"+
            COLUMN_GOALSGUAST+" INT);";
        db.execSQL(query);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
}

```

Файл MainActivity.java

```

public class MainActivity extends Activity {
    DBMatches mDBConnector;
    Context mContext;
    ListView mListView;
    SimpleCursorAdapter scAdapter;
    Cursor cursor;
    myListAdapter myAdapter;

    int ADD_ACTIVITY = 0;
    int UPDATE_ACTIVITY = 1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        mContext=this;
        mDBConnector=new DBMatches(this);
        mListView=(ListView)findViewById(R.id.list);
        myAdapter=new myListAdapter(mContext,mDBConnector.selectAll());
        mListView.setAdapter(myAdapter);
        registerForContextMenu(mListView);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu; this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.menu_main, menu);
        return true;
    }
}

```

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.add:
            Intent i = new Intent(mContext, AddActivity.class);
            startActivityForResult (i, ADD_ACTIVITY);
            updateList();
            return true;
        case R.id.deleteAll:
            mDBConnector.deleteAll();
            updateList();
            return true;
        case R.id.exit:
            finish();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    AdapterView.AdapterContextMenuInfo info =
(AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
    switch(item.getItemId()) {
        case R.id.edit:
            Intent i = new Intent(mContext, AddActivity.class);
            Matches md = mDBConnector.select(info.id);
            i.putExtra("Matches", md);
            startActivityForResult(i, UPDATE_ACTIVITY);
            updateList();
            return true;
        case R.id.delete:
            mDBConnector.delete (info.id);
            updateList();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}

private void updateList () {
    myAdapter.setArrayMyData(mDBConnector.selectAll());
    myAdapter.notifyDataSetChanged();
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {

    if (resultCode == Activity.RESULT_OK) {
        Matches md = (Matches) data.getExtras().getSerializable("Matches");
        if (requestCode == UPDATE_ACTIVITY)

```

```

        mDBConnector.update(md);
    else
        mDBConnector.insert(md.getTeamhouse(), md.getTeamguest(),
md.getGoalshouse(), md.getGoalsguest());
        updateList();
    }
}

class myListAdapter extends BaseAdapter {
    private LayoutInflater mLayoutInflater;
    private ArrayList<Matches> arrayMyMatches;

    public myListAdapter (Context ctx, ArrayList<Matches> arr) {
        mLayoutInflater = LayoutInflater.from(ctx);
        setArrayMyData(arr);
    }

    public ArrayList<Matches> getArrayMyData() {
        return arrayMyMatches;
    }

    public void setArrayMyData(ArrayList<Matches> arrayMyData) {
        this.arrayMyMatches = arrayMyData;
    }

    public int getCount () {
        return arrayMyMatches.size();
    }

    public Object getItem (int position) {

        return position;
    }

    public long getItemId (int position) {
        Matches md = arrayMyMatches.get(position);
        if (md != null) {
            return md.getId();
        }
        return 0;
    }

    public View getView(int position, View convertView, ViewGroup parent) {

        if (convertView == null)
            convertView = mLayoutInflater.inflate(R.layout.item, null);

        TextView vTeamHome= (TextView)convertView.findViewById(R.id.TeamHome);
        TextView vTeamGuest =
(TextView)convertView.findViewById(R.id.TeamGuest);
        TextView vTotal=(TextView)convertView.findViewById(R.id.TeamTotal);

        Matches md = arrayMyMatches.get(position);
        vTeamHome.setText(md.getTeamhouse());
        vTeamGuest.setText(md.getTeamguest());
        vTotal.setText(md.getGoalshouse()+"."+md.getGoalsguest());

        return convertView;
    }
}

```

```

    }
    } // end myAdapter
}

```

AddActivity.java

```

public class AddActivity extends Activity {
    private Button btSave, btCancel;
    private EditText etTeamHome, etTeamGuest, etGoalsHome, etGoalsGuest;
    private Context context;
    private long MyMatchID;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_add);

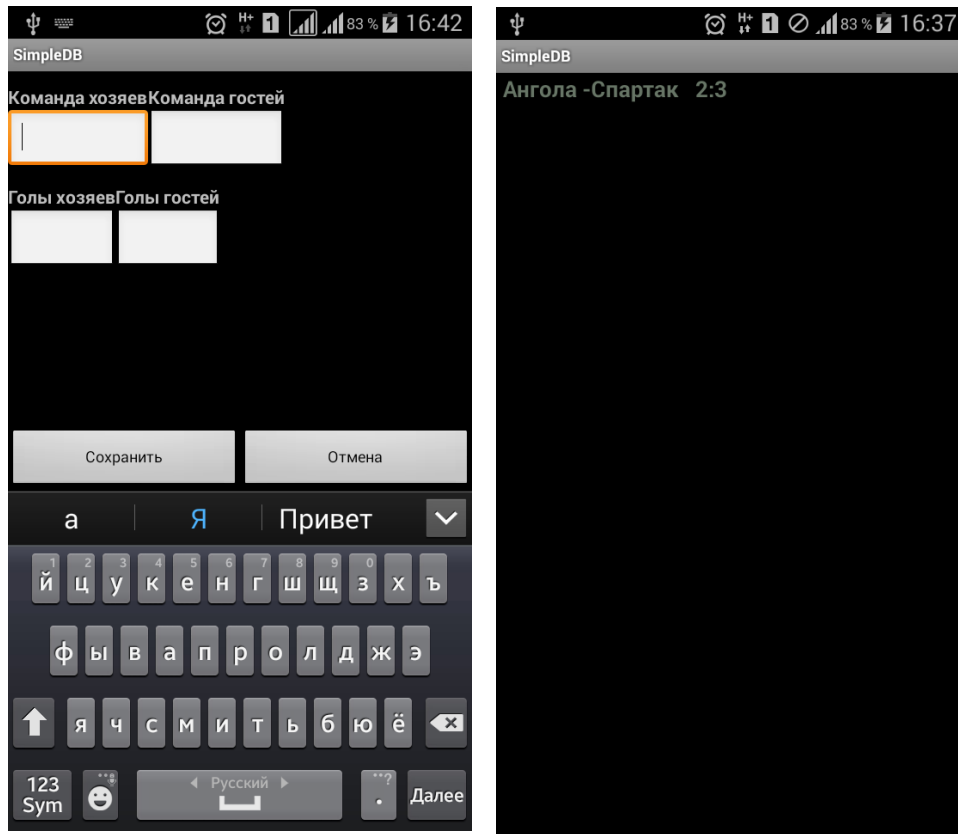
        etTeamHome=(EditText)findViewById(R.id.TeamHome);
        etTeamGuest=(EditText)findViewById(R.id.TeamGuest);
        etGoalsHome=(EditText)findViewById(R.id.GoalsHome);
        etGoalsGuest=(EditText)findViewById(R.id.GoalsGuest);
        btSave=(Button)findViewById(R.id.butSave);
        btCancel=(Button)findViewById(R.id.butCancel);

        if(getIntent().hasExtra("Matches")){
            Matches matches=(Matches)getIntent().getSerializableExtra("Matches");
            etTeamHome.setText(matches.getTeamhouse());
            etTeamGuest.setText(matches.getTeamguest());
            etGoalsHome.setText(Integer.toString(matches.getGoalshouse()));
            etGoalsGuest.setText(Integer.toString(matches.getGoalsguest()));
            MyMatchID=matches.getId();
        }
        else
        {
            MyMatchID=-1;
        }
        btSave.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Matches matches=new
Matches(MyMatchID,etTeamHome.getText().toString(),etTeamGuest.getText().toString(),
Integer.parseInt(etGoalsHome.getText().toString()),
Integer.parseInt(etGoalsGuest.getText().toString()));
                Intent intent=getIntent();
                intent.putExtra("Matches",matches);
                setResult(RESULT_OK,intent);
                finish();
            }
        });

        btCancel.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                finish();
            }
        });
    }
}

```

После запуска приложения нажимаем на кнопку настроек и в меню выбираем Добавить, чтобы внести новую запись в БД. Открывается другая Activity для ввода данных. Затем сохраняем и видим введенную информацию на экране:



При нажатии на строчку с результатами игр появляется меню, с помощью которого мы можем удалить или отредактировать ее.

Благодарности

Компания Samsung Electronics выражает благодарность за участие в подготовке данного материала преподавателю ИТ ШКОЛЫ SAMSUNG Дубинину Андрею Валентиновичу и Плотникову Денису Геннадьевичу.