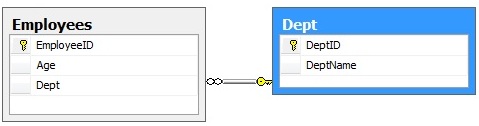
**Программирование SQL базы данных в Android**

По умолчанию в Android используется база данных SQLite - это довольно популярная, легкая и быстрая реляционная база данных. Если для вас SQL – это пустой звук, то, скорее всего, вы мало что поймете из этой статьи. Предполагается, что читатель знаком с SQL, знает что такое запросы выборки, триггеры и представления. Если это не так - почитайте какое-нибудь введение в SQL, а потом возвращайтесь сюда.  
  
База данных SQLite  имеет некоторые особенности, о которых нужно помнить:

* Не поддерживается проверка типов данных. Это значит, что Вы случайно сможете записать данные не того типа, например поместить строку в колонку, предназначенную для целых чисел.
* Не поддерживается ссылочная целостность:  нет поддержки FOREIGN KEY и конструкций JOIN.
* Полная поддержка Unicode отключена по умолчанию.

Давайте создадим простую базу данных, содержащую информацию о работниках. База содержит две таблицы:  Employees - для хранения данных о сотрудниках и Dept - о департаментах. И представление ViewEmps, связывающее работников с департаментами.



**Создание SQLite базы данных в Android проекте**

По умолчанию, Android не содержит удобных СУБД для создания и работы с базами данных (есть sqlite3.exe - консоль для работы с базой данных), поэтому давайте создадим базу и все необходимые таблицы непосредственно из кода.  
  
Во-первых, мы напишем класс, который будет управлять всеми операциями с базой данных: созданием базы, созданием таблиц, вставкой и удалением записей. Давайте создадим класс потомок от**SQLiteOpenHelper**. Нам нужно переопределить два метода:

* onCreate(SQLiteDatabase db) - вызывается при создании базы данных. Здесь мы можем создать таблицы, определить в них колонки, создать виды и триггеры.
* onUpgrade(SQLiteDatabse db, int oldVersion, int newVersion) - вызывается, при внесении изменений в базу, например при создании и удалении таблиц

Внутри нашего класса определим переменные

**publicclass** DatabaseHelper**extends** SQLiteOpenHelper{

static final String dbName="demoDB";

static final String employeeTable="Employees";

static final String colID="EmployeeID";

static final String colName="EmployeeName";

static final String colAge="Age";

static final String colDept="Dept";

static final String deptTable="Dept";

static final String colDeptID="DeptID";

static final String colDeptName="DeptName";

static final String viewEmps="ViewEmps";

и создадим конструктор

**public** DatabaseHelper(Context context){

super(context, dbName,**null**,33);

}

Как видите, здесь мы просто вызываем конструктор суперкласса, который имеет четыре параметра:

* **Context**- контекст, связанный с базой данных.
* **dataBaseName**- имя базы данных.
* **CursorFactory**- можно использовать класс, созданный на основе класса Cursor. Это позволяет проводить некоторые дополнительные проверки и операции с результатами запросов. Если сюда подставить ссылку на наш класс, он будет использоваться вместо стандартного курсора. Мы не будем работать с **CursorFactory**, поэтому передадим значение null.
* **Version**- версия базы данных. Смысл этого параметра будет раскрыт ниже

**Создание базы данных**

Давайте переопределим метод **onCreate**

**public** void onCreate(SQLiteDatabase db){

*// TODO Auto-generated method stub*

db.execSQL("CREATE TABLE "+deptTable+" ("+

colDeptID+" INTEGER PRIMARY KEY , "+

colDeptName+" TEXT)");

db.execSQL("CREATE TABLE "+employeeTable+

" ("+colID+" INTEGER PRIMARY KEY AUTOINCREMENT, "+

colName+" TEXT, "+colAge+" Integer, "+colDept+

" INTEGER NOT NULL ,FOREIGN KEY ("+colDept+")

REFERENCES "+deptTable+" ("+colDeptID+"));");

db.execSQL("CREATE TRIGGER fk\_empdept\_deptid "+

" BEFORE INSERT "+

" ON "+employeeTable+

" FOR EACH ROW BEGIN"+

" SELECT CASE WHEN ((SELECT "+colDeptID+" FROM "+

deptTable+" WHERE "+colDeptID+"=new."+colDept+" ) IS NULL)"+

" THEN RAISE (ABORT,'Foreign Key Violation') END;"+

" END;");

db.execSQL("CREATE VIEW "+viewEmps+

" AS SELECT "+employeeTable+"."+colID+" AS \_id,"+

" "+employeeTable+"."+colName+","+

" "+employeeTable+"."+colAge+","+

" "+deptTable+"."+colDeptName+""+

" FROM "+employeeTable+" JOIN "+deptTable+

" ON "+employeeTable+"."+colDept+" ="+deptTable+"."+colDeptID

);

*//Inserts pre-defined departments*

InsertDepts(db);

}

Внутри метода мы с помощью SQL запросов создаем таблицы, триггеры и представления. Запросы вызываются для базы db с помощью метода **execSQL**.  
   
Метод**onCreate**вызывается при создании базы данных, как только база создана, мы добавляем туда таблицы с необходимым набором полей. Вызов метода происходит, когда база данных не существует на диске, то есть для данного конкретного устройства этот метод будет вызван всего один раз, сколько бы мы потом не запускали нашу программу.

**Изменение базы данных**

Иногда возникает необходимость внести изменения в базу данных: поменять схему, добавить новые таблицы или поменять типы столбцов таблиц. Все это можно сделать внутри метода**onUpdate(SQLiteDatabase db,int old Version,int newVerison)** .

**Public** void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion){

*// TODO Auto-generated method stub*

db.execSQL("DROP TABLE IF EXISTS "+employeeTable);

db.execSQL("DROP TABLE IF EXISTS "+deptTable);

db.execSQL("DROP TRIGGER IF EXISTS dept\_id\_trigger");

db.execSQL("DROP TRIGGER IF EXISTS dept\_id\_trigger22");

db.execSQL("DROP TRIGGER IF EXISTS fk\_empdept\_deptid");

db.execSQL("DROP VIEW IF EXISTS "+viewEmps);

onCreate(db);

}

Этот метод вызывается, когда изменяется номер, указанный в конструкторе класса. Помните то загадочное число 33, внутри конструктора**DatabaseHelper**? Если Вам вдруг понадобится внести изменение в свою базу, просто поменяйте номер в конструкторе. То есть, если вы измените конструктор

**public** DatabaseHelper(Context context){

super(context, dbName,**null**,34);

}

приложение поймет, что вы хотите изменить свою базу и вызовет метод onUpgrade. Обычно внутри этого метода удаляются таблицы и создаются заново с другим набором полей.

**Управление внешними ключами**

Давайте создадим триггер, который будет следить при добавлении нового сотрудника за тем, чтобы указанный в описании сотрудника департамент присутствовал в таблице Dept.  
   
 SQL команда, создающая такой триггер, выглядит следующим образом:

CREATE TRIGGER fk\_empdept\_deptid Before INSERT ON Employees

FOREACH ROW BEGIN

SELECTCASE WHEN((SELECT DeptID FROM Dept WHERE DeptID=**new**.Dept) IS**NULL**)

THEN RAISE(ABORT,'Foreign Key Violation')END;

END

Добавим в метод onCreate вызов метода execSQL с соответствующим запросом:

db.execSQL("CREATE TRIGGER fk\_empdept\_deptid "+

" BEFORE INSERT "+

" ON "+employeeTable+

" FOR EACH ROW BEGIN"+

" SELECT CASE WHEN ((SELECT "+colDeptID+" FROM "+deptTable+

" WHERE "+colDeptID+"=new."+colDept+" ) IS NULL)"+

" THEN RAISE (ABORT,'Foreign Key Violation') END;"+

" END;");

**Выполнение SQL команд в Android**

 Как уже было отмечено выше, Вы можете выполнить любую SQL команду с помощью мtтода**db.execSQL(String statement)**.

**Добавление записи в таблицу**

 Можно добавить новую запись в таблицу с помощью SQL команды**INSERT**, а можно воспользоваться методом**db.insert**:

SQLiteDatabase db=this.getWritableDatabase();

ContentValues cv=**new** ContentValues();

cv.put(colDeptID,1);

cv.put(colDeptName,"Sales");

db.insert(deptTable, colDeptID, cv);

cv.put(colDeptID,2);

cv.put(colDeptName,"IT");

db.insert(deptTable, colDeptID, cv);

db.close();

Обратите внимание, для соединения с базой данных мы вызвали метод**this.getWritableDatabase()**. Класс**ContentValues**позволяет создать карту с параметрами ключ-значение. Эта карта затем используется методом**insert**, который фактически добавляет запись в таблицу. Метод**ContentValues.put** создает пару ключ-значение. Еще одно замечание, хорошим тоном считается закрывать базу данных вызовом метода **close()** после завершения выполнения группы команд.

**Изменение значений в таблице**

Изменить значение в какой-либо записи можно с помощью SQL команды**UPDATE**или с помощью метода **db.update**:

**public** int UpdateEmp(Employee emp)

{

SQLiteDatabase db=this.getWritableDatabase();

ContentValues cv=**new** ContentValues();

cv.put(colName, emp.getName());

cv.put(colAge, emp.getAge());

cv.put(colDept, emp.getDept());

return db.update(employeeTable, cv, colID+"=?",**new** String[]{String.valueOf(emp.getID())});

}

Метод update имеет следующие параметры:

* **String Table**: название таблицы в которой мы хотим что-то изменить.
* **ContentValues cv**: пара ключ-значение с новой информацией.
* **String where**: строка-шаблон, определяющая условие  WHERE, которое проверяется для каждой строки таблицы. Ксли условие истинно, то в этой строке обновляется информация в соответствии с параметром cv.
* **String[] args**: массив с аргументами, которые подставляются в строку-шаблон where.

**Удаление строк**

Удалить строку в таблице можно с помощью SQL команды**DELETE**, а также с помощью метода**db.delete**. Этот метод имеет аналогичный **update**набор параметров (за исключением cv).

**public** void DeleteEmp(Employee emp)

{

SQLiteDatabase db=this.getWritableDatabase();

db.delete(employeeTable,colID+"=?",**new** String[]{String.valueOf(emp.getID())});

db.close();

}

**Выполнение SQL запросов**

 Для выполнения запросов в Android используются два метода:**db.rawQuery** и**db.query**. Давайте сделаем запрос к базе, с целью получения списка всех департаментов:

Cursor getAllDepts()

{

SQLiteDatabase db=this.getReadableDatabase();

Cursor cur=db.rawQuery("SELECT "+colDeptID+" as \_id, "+

colDeptName+" from "+deptTable,**new** String[]{});

return cur;

}

  Метод**rawQuery**имеет два параметра:

* **String query**: Строка с SQL запросом SELECT.
* **String[] selection args**: массив аргументов where, если он используется в SELECT запросе.

**Замечания!** В качестве результата этого запроса возвращается объект**Cursor**. Если в качестве ключевого поля в таблице используется колонка с именем, отличным от**\_id**, Вам необходимо использовать alias в форме "SELECT [Column Name] as \_id". Дело в том, что объект**Cursor**всегда считает, что ключевое поле имеет имя**\_id**, и если это не так, генерируется исключение.  
Другим способом выполнения запроса к базе является вызов метода**db.query**. Давайте напишем метод, который выдает всех сотрудников, работающих в каком-то конкретном департаменте:

**public** Cursor getEmpByDept(String Dept)

{

SQLiteDatabase db=this.getReadableDatabase();

String[] columns=**new** String[]{"\_id",colName,colAge,colDeptName};

Cursor c=db.query(viewEmps, columns, colDeptName+"=?",**new** String[]{Dept},**null**,**null**,**null**);

return c;

}

 Метод**db.query** имеет следующий набор параметров:

* **String Table Name**: Имя таблицы, из которой осуществляется выборка.
* **String [ ] columns**: список столбцов, которые войдут в результат.
* **String WHERE clause**: шаблон where-условия или null.
* **String [ ] selection args**: массив с аргументами where-условия.
* **String Group by**: условие группировки.
* **String Having**: условие HAVING.
* **String Order By by**: порядок сортировки.

**Управление объектом Cursor**

В результате выполнения запросов нам возвращается объект**Cursor**, содержащий таблицу с результатами запроса.**Cursor**предполагает последовательную работу со строками результата. В каждый момент времени активна одна строка, на которую ссылается указатель. Ниже перечислены методы, которые используются для работы с этим объектом:

* **boolean moveToNext()**- перемещает указатель на одну запись вперед. Возвращает false, если достигнут конец таблицы результата.
* **boolean moveToFirst()**- перемещает указатель на первую строку. Возвращает false, если таблица-результат не содержит строк.
* **boolean moveToPosition(int position)** - перемещает указатель на конкретную строку в таблице результатов. Если строки с таким номером нет - возвращает false.
* **boolean moveToPrevious()**- перемещает указатель на одну запись назад. Возвращает false, если достигнуто начало таблицы результата.
* **boolean moveToLast()**- перемещает указатель на последнюю строку. Возвращает false, если таблица-результат не содержит строк.
* **int  getColumnIndex(String column)**- возвращает ссылку на колонку column в текущей строке. Для получения значения по ссылке используются методы**Cursor.getInt(int ColumnIndex), getShort,getString,getDouble, getBlob**.

Для определение текущей позиции указателя можно воспользоваться методами:**boolean isAfterLast(), isBeforeFirst, isFirst, isLast и isNull(columnIndex)**. Их назначение без труда можно понять из названий.  
  
Если в результате выполнения запроса вы получили только одну строку, то извлечь из нее значения столбцов можно следующим образом:

**public** int GetDeptID(String Dept)

{

SQLiteDatabase db=this.getReadableDatabase();

Cursor c=db.query(deptTable,**new** String[]{colDeptID+" as \_id",colDeptName},colDeptName+"=?",

**new** String[]{Dept},**null**,**null**,**null**);

c.moveToFirst();

return c.getInt(c.getColumnIndex("\_id"));

}

 В результате выполнения этого метода, мы получим значение колонки \_id текущей строки.  После использования курсора его хорошо бы закрыть с помощью метода**close()**.  
   
 Вот собственно и все. Теперь вы вполне сможете использовать базы данных в своих Android проектах.