Работа с записями Базы данных

Содержание

- Работаем с записями базы данных
- Метод rawQuery()
- Вставка данных для проверки
- Изменение данных
- Удаление данных

Работаем с записями базы данных

Чтобы проверить работоспособность базы данных, в главной активности поместим вспомогательный метод **displayDatabaseInfo()** для отображения информации.

```
private void displayDatabaseInfo() {
 // Создадим и откроем для чтения базу данных
 SQLiteDatabase db = mDbHelper.getReadableDatabase();
// Зададим условие для выборки - список столбцов
String[] projection = { GuestEntry._ID, GuestEntry.COLUMN_NAME,
     GuestEntry.COLUMN_CITY, GuestEntry.COLUMN_GENDER, GuestEntry.COLUMN_AGE };
// Делаем запрос
Cursor cursor = db.query( GuestEntry.TABLE_NAME, // таблица
 projection, // столбцы
 null, // столбцы для условия WHERE
 null, // значения для условия WHERE
 null, // Don't group the rows
 null, // Don't filter by row groups
 null); // порядок сортировки
```

// **Массив projection - это список столбцов, которые нас интересуют. В SQL-запросе мы их указываем в операторе SELECT:**

Работаем с записями базы данных

```
TextView displayTextView = (TextView) findViewById(R.id.text_view_info);
 try { displayTextView.setText("Таблица содержит " + cursor.getCount() + "
     гостей.\n\n");
 displayTextView.append(GuestEntry._ID + " - " + GuestEntry.COLUMN_NAME +
     " - " + GuestEntry.COLUMN_CITY + " - " + GuestEntry.COLUMN_GENDER +
     " - " + GuestEntry.COLUMN_AGE + "\n");
// Узнаем индекс каждого столбца
 int idColumnIndex = cursor.getColumnIndex(GuestEntry._ID);
int nameColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_NAME);
 int cityColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_CITY);
int genderColumnIndex =
     cursor.getColumnIndex(GuestEntry.COLUMN_GENDER);
 int ageColumnIndex = cursor.getColumnIndex(GuestEntry.COLUMN_AGE);
```

Работаем с записями базы данных

```
// Проходим через все ряды
while (cursor.moveToNext()) {
// Используем индекс для получения строки или числа
int currentID = cursor.getInt(idColumnIndex);
String currentName = cursor.getString(nameColumnIndex);
 String currentCity = cursor.getString(cityColumnIndex);
 int currentGender = cursor.getInt(genderColumnIndex);
 int currentAge = cursor.getInt(ageColumnIndex);
 // Выводим значения каждого столбца
    displayTextView.append(("\n" + currentID + " - " + currentName + "
    - " + currentCity + " - " + currentGender + " - " + currentAge)); } }
    finally {
// Всегда закрываем курсор после чтения
 cursor.close(); } } }
```

Комментарий

Для каждой строки можно прочитать значение столбца, вызвав один из методов Get Курсора, таких как getString() или getLong().

Для каждого из методов get необходимо передать позицию индекса нужного столбца, которую можно получить, вызвав getColumnIndex () или getColumnIndexOrThrow ().

После завершения итерации по результатам, вызовите close () на курсоре, чтобы освободить его ресурсы.

Например, ниже показано, как получить все идентификаторы элементов, хранящиеся в курсоре, и добавить их в список:

```
List itemIds = new ArrayList<>();
    while(cursor.moveToNext()) {
      long itemId = cursor.getLong(
          cursor.getColumnIndexOrThrow(FeedEntry._ID));
      itemIds.add(itemId);
    }
    cursor.close();
```

Запрос и код в классе

SELECT * FROM guests WHERE _id = 1;

В коде такое выражение выглядело бы так.

```
String selection = GuestEntry._ID + "=?";
 String[] selectionArgs = {"1"};
```

Как видим, в знак вопроса подставляется нужное значение.

Запрос и код в классе

SELECT name FROM guests WHERE _id > 1 BY age DESC;
// Зададим условие для выборки - список столбцов
String[] projection = { GuestEntry.COLUMN_NAME };
String selection = GuestEntry._ID + ">?";
String[] selectionArgs = {"1"};
Cursor cursor = db.query(GuestEntry.TABLE_NAME, // таблица
projection, // столбцы
selection, // столбцы для условия WHERE
selectionArgs, // значения для условия WHERE
null, // Don't group the rows
null, // Don't filter by row groups
GuestEntry.COLUMN_AGE + " DESC"); // порядок сортировки

Комментарий

Третий и четвертый аргументы (selection и selectionArgs) объединяются для создания предложения WHERE.

Поскольку аргументы предоставляются отдельно от запроса выбора, они экранируются перед объединением.

Это делает ваши операторы выбора невосприимчивыми к SQL-инъекции.

Чтобы просмотреть строку в курсоре, используйте один из методов перемещения курсора, который необходимо всегда вызывать перед началом чтения значений.

Поскольку курсор начинается с позиции -1, вызов moveToNext () помещает "позицию чтения"на первую запись в результатах и возвращает, находится ли курсор уже за последней записью в результирующем наборе.

Метод rawQuery()

Второй способ использует сырой (raw) SQL-запрос.
Сначала формируется строка запроса и отдаётся методу **rawQuery()**.

```
// Абстрактный пример
// Метод 2: Сырой SQL-запрос
String query = "SELECT " + DatabaseHelper.COLUMN_ID + ", " +
    DatabaseHelper.CAT_NAME_COLUMN + " FROM " + DatabaseHelper.TABLE_NAME;
Cursor cursor2 = mDatabase.rawQuery(query, null);
while (cursor2.moveToNext()) {
int id = cursor2.getInt(cursor2.getColumnIndex(DatabaseHelper.COLUMN_ID));
String name =
    cursor2.getString(cursor2.getColumnIndex(DatabaseHelper.CAT_NAME_COLUMN));
Log.i("LOG_TAG", "ROW " + id + " HAS NAME " + name); }
cursor2.close();
```

При работе с базой данных мы обращаемся к файлу.
Если база данных очень большая, то запросы не будут мгновенными.
Операции с файлами являются медленными, поэтому следует использовать **многопоточность.**

Вставка данных для проверки

Рассмотрим, как вставлять новые данные. Добавим в меню главной активности пункт "Вставить данные".

Для вставки данных применяется метод **ContentValues.put()**.

В методе указываются ключ и значение.

В качестве ключа выступает имя столбца таблицы, а его значением будет нужная информация о госте.

Так как идентификатор будет вставляться автоматически, то его не используем.

После того, как вы заполните все столбцы таблицы, вызывайте метод **insert()**, который и разместит данные в базе.

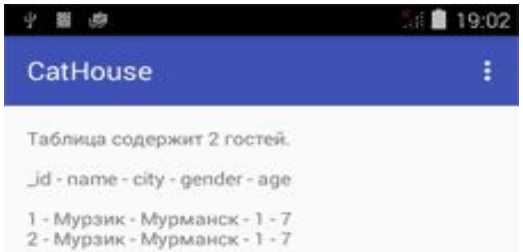
Вставка

```
private void insertGuest() {
 // Gets the database in write mode
 SQLiteDatabase db = mDbHelper.getWritableDatabase();
// Создаем объект ContentValues, где имена столбцов ключи,
 // а информация о госте является значениями ключей
 ContentValues values = new ContentValues();
values.put(GuestEntry.COLUMN_NAME, "Мурзик");
values.put(GuestEntry.COLUMN_CITY, "Мурманск");
values.put(GuestEntry.COLUMN_GENDER,
    GuestEntry.GENDER_MALE);
values.put(GuestEntry.COLUMN_AGE, 7);
 long newRowId = db.insert(GuestEntry.TABLE_NAME, null, values); }
```

Вызовем метод в обработчике нажатия пункта меню

```
case R.id.action_insert_new_data:
 insertGuest();
displayDatabaseInfo();
 return true;
```

Сразу после вставки вызываем метод **displayDatabaseInfo()**, чтобы увидеть результат



Первый способ вставки - ContentValues

Для вставки сначала подготавливаются данные с помощью класса **ContentValues**.

Вы указываете имя колонки таблицы и значение для неё, т.е. работает по принципу "ключ-значение".

Когда подготовите все данные во все столбцы, то вызывайте метод **insert()**, который сразу раскидает данные по столбцам.

Способ очень удобен, требует мало кода и легко читаем.

У метода **insert()** три аргумента.

В первом указывается имя таблицы, в которую будут добавляться записи.

В третьем указывается объект **ContentValues**, созданный ранее. Второй аргумент используется для указания колонки.

SQL не позволяет вставлять пустую запись, и если будет использоваться пустой **ContentValue**, то укажите во втором аргументе **null** во избежание ошибки.

Второй способ вставки - SQL-запрос

Существует также другой способ вставки через метод **execSQL()**, когда подготавливается нужная строка и запускается скрипт.

В этом варианте используется традиционный SQL-запрос **INSERT INTO...**.

Основное неудобство при этом способе - не запутаться в кавычках.

Если что-то не вставляется, то смотрите логи сообщений.

```
// пример
 db = new DatabaseHelper(this);
 SQLiteDatabase sqdb = db.getWritableDatabase();
 String insertQuery = "INSERT INTO " +
    DatabaseHelper.DATABASE_TABLE + " (" +
    DatabaseHelper.CAT_NAME_COLUMN + ") VALUES ('Васька')";
sqdb.execSQL(insertQuery);
```

Наполняем базу данных

Создадим вспомогательный метод для вставки записи в базу данных.

Для этого считываем данные, которые вводятся в текстовые поля, а далее по предыдущему учебному примеру.

```
private void insertGuest() {
 // Считываем данные из текстовых полей
String name = mNameEditText.getText().toString().trim();
String city = mCityEditText.getText().toString().trim();
 String ageString = mAgeEditText.getText().toString().trim();
 int age = Integer.parseInt(ageString);
HotelDbHelper mDbHelper = new HotelDbHelper(this);
 SQLiteDatabase db = mDbHelper.getWritableDatabase();
```

Наполняем базу данных

```
ContentValues values = new ContentValues();
    values.put(GuestEntry.COLUMN_NAME, name);
    values.put(GuestEntry.COLUMN_CITY, city);
    values.put(GuestEntry.COLUMN_GENDER, mGender);
    values.put(GuestEntry.COLUMN_AGE, age);
// Вставляем новый ряд в базу данных и запоминаем его
    идентификатор
 long newRowId = db.insert(GuestEntry.TABLE_NAME, null, values);
// Выводим сообщение в успешном случае или при ошибке
if (newRowId == -1) {
// Если ID -1, значит произошла ошибка
 Toast.makeText(this, "Ошибка при заведении гостя",
    Toast.LENGTH_SHORT).show();
} else { Toast.makeText(this, "Гость заведён под номером: " +
    newRowId, Toast.LENGTH_SHORT).show(); } }
```

Изменение данных

Если запись уже существует, но вам нужно изменить какое-то значение, то вместо **insert()**используйте метод **update()**.

В остальном принцип тот же.

Вызываем метод **put()**, а затем обновляем запись в базе данных.

Cont**entValues values = new ContentValues();**

values.put(GuestEntry.COLUMN_NAME, "Мурзик");

db.update(GuestEntry.TABLE_NAME, values, GuestEntry.COLUMN_NAME + "= ?", new String[]{"Мурка"});

- Первый параметр метода **update()** содержит имя таблицы.
- Второй параметр указывает, какие значения должны использоваться для обновления.
- Третий параметр задает условия отбора обновляемых записей (WHERE).

В приведенном примере **"NAME = ?"** означает, что столбец **NAME**должен быть равен некоторому значению.

Символ **?** обозначает значение столбца, которое определяется содержимым последнего параметра.

Если в двух последних параметрах метода передаётся значение **null**, будут обновлены ВСЕ записи в таблице.

Возможны и сложные условия

db.update(GuestEntry.TABLE_NAME, values, "NAME = ?
OR EMAIL = ?",
new String[] {"Васька", "vaska@cat.com"});

Если столбец не является строкой, то его нужно преобразовать в строку, чтобы использовать в качестве условий.

db.update(GuestEntry.TABLE_NAME.TABLE_NAME, values, "_id = ?", new String[] {Integer.toString(1)});

Будьте осторожны с обновлениями. Если в последних двух параметрах передать значение **null**, то будут обновлены все записи в таблице, так как в запросе нет условий.

db.update(mDatabaseHelper.TABLE_NAME, values, null, null);

Удаление данных

Метод **delete()** класса **SQLiteDatabase** работ ает по тому же принципу, как и метод **update()**.

Он имеет следующую форму:

public int delete (String table, String whereClause, String[] whereArgs)

Удаление данных

Чтобы удалить строки из таблицы, необходимо указать критерии выбора, определяющие строки для метода delete ().

Механизм работает так же, как и аргументы выбора для метода query ().

Он разделяет спецификацию выбора на предложение выбора и аргументы выбора.

Предложение определяет столбцы для просмотра, а также позволяет комбинировать тесты столбцов. Аргументы - это значения для проверки, которые связаны с предложением.

Поскольку результат не обрабатывается так же, как обычный оператор SQL, он невосприимчив к SQL-инъекции.:

```
/ Define 'where' part of query.
    String selection =
    FeedEntry.COLUMN_NAME_TITLE + " LIKE ?";
    // Specify arguments in placeholder order.
    String[] selectionArgs = { "MyTitle" };
    // Issue SQL statement.
    int deletedRows =
    db.delete(FeedEntry.TABLE_NAME, selection,
    selectionArgs);
```

Литература

- http://developer.alexanderklimov.ru/android/sqlite/cathouse2.php
- https://developer.android.com/training/data-storage/sqlite