Лабораторная работа № 5

SQLite для хранения данных

Общие сведения.

SQLite - реляционная СУБД, которая входит в состав стандартной библиотеки ОС Android и используемая для хранения данных в приложениях.

Библиотека android.database.sqlite предоставляет низкоуровневый API для работы с реляционной БД.

Для работы с БД Google рекомендует реализовать класс контракта, который описывает организацию базы данных.

Например, для БД, хранящей данные пользователей такой класс может выглядеть следующим образом:

**package** com.example.hw;  
**import** android.provider.BaseColumns;  
  
**public final class** DBContract {  
  
 **private** DBContract() {}  
   
 **public static class** UserEntry **implements** BaseColumns {  
 **public static final** String ***TABLE\_NAME*** = **"users"**;  
 **public static final** String ***COLUMN\_NAME\_KEY\_ID*** = **"id"**;  
 **public static final** String ***COLUMN\_NAME\_LOGIN*** = **"login"**;  
 **public static final** String ***COLUMN\_NAME\_PASS*** = **"pass"**;  
 }  
}

class UserEntry в данном контракте содержит константы, задающие имена таблицы для хранения данных пользователей и названия ее столбцов. Эти константы будут использоваться при построении SQL-запросов.

Наличие контракта позволяет иметь одну точку редактирования описания схемы данных, т.е. использовать одни и те же константы во всех классах приложения.

Для доступа к БД используется класс SQLiteOpenHelper, который содержит методы для управления базой данных. Экземпляр БД можно получить, вызвав метод getWritableDatabase() или getReadableDatabase() у класса-наследника SQLiteOpenHelper.

Таким образом, в приложении необходимо создать подкласс SQLiteOpenHelper, который переопределяет методы onCreate() и onUpgrade(). Также можно реализовать методы onDowngrade() или onOpen().

**public class** DatabaseHandler **extends** SQLiteOpenHelper {  
  
 **private static final int *DATABASE\_VERSION*** = 1;  
 **private static final** String ***DATABASE\_NAME*** = **"Users.db"**;  
  
 **public** DatabaseHandler(Context context) {  
 **super**(context, ***DATABASE\_NAME***, **null**, ***DATABASE\_VERSION***);  
 }  
  
 @Override  
 **public void** onCreate(SQLiteDatabase db) {  
 String CREATE\_USERS\_TABLE = **"CREATE TABLE "** + DBContract.UserEntry.***TABLE\_NAME*** + **"("** + DBContract.UserEntry.***COLUMN\_NAME\_KEY\_ID*** + **" INTEGER PRIMARY KEY,"** +  
 DBContract.UserEntry.***COLUMN\_NAME\_LOGIN*** + **" TEXT,"** + DBContract.UserEntry.***COLUMN\_NAME\_PASS*** + **" TEXT"** + **")"**;  
  
 db.execSQL(CREATE\_USERS\_TABLE);  
 }  
  
 @Override  
 **public void** onUpgrade(SQLiteDatabase db, **int** oldVersion, **int** newVersion) {  
 db.execSQL(**"DROP TABLE IF EXISTS "** + DBContract.UserEntry.***TABLE\_NAME***);  
 onCreate(db);  
 }  
  
 **public void** addUser(User user) {  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 ContentValues values = **new** ContentValues();  
 values.put(DBContract.UserEntry.***COLUMN\_NAME\_LOGIN***, user.getLogin());  
 values.put(DBContract.UserEntry.***COLUMN\_NAME\_PASS***, user.getPass());  
  
 db.insert(DBContract.UserEntry.***TABLE\_NAME***, **null**, values);  
 db.close();  
 }

}

Помимо переопределенных методов onCreate() и onUpgrade() созданный класс реализует методы добавления и выборки данных из БД, например, метод добавления нового пользователя addUser.

Реализация данного метода использует getWritableDatabase для получения экземпляра класса, работающего с БД. Затем создается класс ContentValues, в котором формируются данные в форме ключ-значение, пригодной для добавления в БД. Например, метод put используется для добавления данных в столбец таблицы. При этом ключом является название столбца таблицы, а значение – это добавляемые данные.

Метод insert осуществляет непосредственное добавление данных в таблицу.

Аналогично реализуются методы выборки данных, в которых используется метод query. Этот метод возвращает данные в виде объекта Cursor, для навигации по которому используются методы:

getCount() - получает количество извлеченных из базы данных объектов;

moveToFirst() и moveToNext() позволяют переходить к первому и к следующему

элементам выборки;

isAfterLast() позволяет проверить, достигнут ли конец выборки.

Также имеется набор методов для чтения каждого типа данных: getString(), getInt() и getFloat().

Например, так будет выглядеть метод загрузки из БД всех строк таблицы Users:

**public** List<User> getAllUsers() {  
 List<User> usersList = **new** ArrayList<User>();  
 String selectQuery = **"SELECT \* FROM "** + DBContract.UserEntry.***TABLE\_NAME***;  
  
 SQLiteDatabase db = **this**.getWritableDatabase();  
 Cursor cursor = db.rawQuery(selectQuery, **null**);  
  
 **if** (cursor.moveToFirst()) {  
 **do** {  
 User user = **new** User();  
 user.setID(Integer.*parseInt*(cursor.getString(0)));  
 user.setLogin(cursor.getString(1));  
 user.setPass(cursor.getString(2));  
 usersList.add(user);  
 } **while** (cursor.moveToNext());  
 }  
 **return** usersList;  
}

Данные каждой строки загружаются в отдельный экземпляр класса User, который описывает модель (бизнес-логику) данных.

Модель данных User представлена в качестве учебного примера, в реальных приложениях не стоит хранить пароль пользователя в открытом виде в базе данных

**public class** User {  
  
 **int \_id**;  
 String **\_login**;  
 String **\_pass**;  
  
 **public** User(){  
 }  
 **public** User(**int** id, String login, String pass){  
 **this**.**\_id** = id;  
 **this**.**\_login** = login;  
 **this**.**\_pass** = pass;  
 }  
 **public** User(String login, String pass){  
 **this**.**\_login** = login;  
 **this**.**\_pass** = pass;  
 }  
  
 **public int** getID(){  
 **return this**.**\_id**;  
 }  
 **public void** setID(**int** id){  
 **this**.**\_id** = id;  
 }  
  
 **public** String getLogin(){  
 **return this**.**\_login**;  
 }  
 **public void** setLogin(String login){  
 **this**.**\_login** = login;  
 }  
  
 **public** String getPass(){  
 **return this**.**\_pass**;  
 }  
 **public void** setPass(String pass){  
 **this**.**\_pass** = pass;  
 }  
}

Для работы с БД из класса активити (fragment или сервиса) необходимо создать экземпляр класса-наследника SQLiteOpenHelper, после чего можно использовать его методы (например, addUser или getAllUsers) для работы с БД:

**public class** HelloActivity **extends** Activity **implements** OnClickListener {  
 EditText **login\_Text**, **pass\_Text**;  
 Button **btn\_Save**, **btn\_Load**;  
 DatabaseHandler **db** = **new** DatabaseHandler(**this**);  
 @Override  
 **protected void** onCreate(@Nullable Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_helloactivity***);  
  
 **login\_Text** = (EditText) findViewById(R.id.***login\_Text***);  
 **pass\_Text** = (EditText) findViewById(R.id.***pass\_Text***);  
  
 **btn\_Save** = (Button) findViewById(R.id.***btn\_Save***);  
 **btn\_Save**.setOnClickListener(**this**);  
 **btn\_Load** = (Button) findViewById(R.id.***btn\_Load***);  
 **btn\_Load**.setOnClickListener(**this**);  
 }  
  
 @Override  
 **protected void** onPause() {  
 **super**.onPause();  
 **db**.deleteAll();  
 }  
  
 @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
 **db**.close();  
 }  
  
 @Override  
 **public void** onClick(View v) {  
 **switch** (v.getId()) {  
 **case** R.id.***btn\_Save***:  
 **db**.addUser(**new** User(**login\_Text**.getText().toString(), **pass\_Text**.getText().toString()));  
 **break**;  
 **case** R.id.***btn\_Load***:  
 List<User> users = **db**.getAllUsers();  
 **for** (User usr : users) {  
 String log = **"Id: "**+usr.getID()+**" ,Login: "** + usr.getLogin() + **" ,Password: "** + usr.getPass();  
 Log.*v*(**"Loading..."**, log);  
 }  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
}

Ход выполнения проекта.

1 Откройте проект, созданный в лабораторной работе № 3.

2 Измените код в соответствии с заданием.

Задание:

1 В приведенном примере кода загруженные данные о пользователях выводятся в лог. Реализуйте вывод загруженных данных на экран.

2 Реализуйте метод проверки пары Логин – Пароль для авторизации в приложении.

3 Реализуйте метод изменения пароля для авторизованного пользователя.

4 Реализуйте метод удаления пользователя (по логину) из БД.