将重复使用或者结构化的数据存储在数据库当中是一个不错的选择，比如联系人信息。Sqlite的API位于android.database.sqlite包中。

# 定义结构和契约类

数据库存在的基础就是结构。结构通过创建数据库时的sql语句体现出来。另外创建一个契约类是有用的--以一种系统性、自记录的方式明确指定结构布局。契约类包括实体类。一般实体类实现BaseCloumns接口（包括一个主键字段\_ID)，如下是一个契约类：

public final class FeedReaderContract {  
    // To prevent someone from accidentally instantiating the contract class,  
    // give it an empty constructor.  
    public FeedReaderContract() {}  
  
    /\* Inner class that defines the table contents \*/  
    public static abstract class FeedEntry implements BaseColumns {  
        public static final String TABLE\_NAME = "entry";  
        public static final String COLUMN\_NAME\_ENTRY\_ID = "entryid";  
        public static final String COLUMN\_NAME\_TITLE = "title";  
        public static final String COLUMN\_NAME\_SUBTITLE = "subtitle";  
        ...  
    }  
}

# 使用SQL Helper创建数据库

定义好数据库的结构之后，开始实现创建、维护数据库和表。这里有一些典型的语句用来创建删除表格：

private static final String TEXT\_TYPE = " TEXT";  
private static final String COMMA\_SEP = ",";  
private static final String SQL\_CREATE\_ENTRIES =  
    "CREATE TABLE " + FeedEntry.TABLE\_NAME + " (" +  
    FeedEntry.\_ID + " INTEGER PRIMARY KEY," +  
    FeedEntry.COLUMN\_NAME\_ENTRY\_ID + TEXT\_TYPE + COMMA\_SEP +  
    FeedEntry.COLUMN\_NAME\_TITLE + TEXT\_TYPE + COMMA\_SEP +  
    ... // Any other options for the CREATE command  
    " )";  
  
private static final String SQL\_DELETE\_ENTRIES =  
    "DROP TABLE IF EXISTS " + FeedEntry.TABLE\_NAME;

之后数据库文件存储在设备的内部存储中，默认情况下是不能被其他应用访问的。

SQLiteOpenHelper类中有一组有用的API,当调用 getWritableDatabase() 或 getReadableDatabase()获取db的时候，系统才会去创建更新数据库。确保在后台线程中调用这两个方法。

下面有一个使用SQLiteOpenHelper的实现：

public class FeedReaderDbHelper extends SQLiteOpenHelper {  
    // If you change the database schema, you must increment the database version.  
    public static final int DATABASE\_VERSION = 1;  
    public static final String DATABASE\_NAME = "FeedReader.db";  
  
    public FeedReaderDbHelper(Context context) {  
        super(context, DATABASE\_NAME, null, DATABASE\_VERSION);  
    }  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL(SQL\_CREATE\_ENTRIES);  
    }  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        // This database is only a cache for online data, so its upgrade policy is  
        // to simply to discard the data and start over  
        db.execSQL(SQL\_DELETE\_ENTRIES);  
        onCreate(db);  
    }  
    public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion) {  
        onUpgrade(db, oldVersion, newVersion);  
    }  
}

# 将信息输入到数据库

通过将一个 ContentValues 对象传递至 insert() 方法将数据插入数据库：

// Gets the data repository in write mode  
SQLiteDatabase db = mDbHelper.getWritableDatabase();  
  
// Create a new map of values, where column names are the keys  
ContentValues values = new ContentValues();  
values.put(FeedEntry.COLUMN\_NAME\_ENTRY\_ID, id);  
values.put(FeedEntry.COLUMN\_NAME\_TITLE, title);  
values.put(FeedEntry.COLUMN\_NAME\_CONTENT, content);  
  
// Insert the new row, returning the primary key value of the new row  
long newRowId;  
newRowId = db.insert(  
         FeedEntry.TABLE\_NAME,  
         FeedEntry.COLUMN\_NAME\_NULLABLE,  
         values);

insert() 的第一个参数即为表格名称。第二个参数指定在 ContentValues 为空的情况下框架可在其中插入 NULL 的列的名称。如果将其设置为 "null"， 那么框架将不会在没有值时插入行。

# 从数据库读取信息

要从数据库中读取信息，请使用 query() 方法，查询的结果将在 Cursor 对象中返回。

SQLiteDatabase db = mDbHelper.getReadableDatabase();  
  
// Define a *projection* that specifies which columns from the database  
// you will actually use after this query.  
String[] projection = {  
    FeedEntry.\_ID,  
    FeedEntry.COLUMN\_NAME\_TITLE,  
    FeedEntry.COLUMN\_NAME\_UPDATED,  
    ...  
    };  
  
// How you want the results sorted in the resulting Cursor  
String sortOrder =  
    FeedEntry.COLUMN\_NAME\_UPDATED + " DESC";  
  
Cursor c = db.query(  
    FeedEntry.TABLE\_NAME,  // The table to query  
    projection,                               // The columns to return  
    selection,                                // The columns for the WHERE clause  
    selectionArgs,                            // The values for the WHERE clause  
    null,                                     // don't group the rows  
    null,                                     // don't filter by row groups  
    sortOrder                                 // The sort order  
    );

查看游标中的某一行，从调用moveToFirst()开始，获取所需列的索引位置之后传递给相应的get方法。例如：

cursor.moveToFirst();  
long itemId = cursor.getLong(  
    cursor.getColumnIndexOrThrow(FeedEntry.\_ID)  
);

# 从数据库中删除信息

要从表格中删除行，需要提供删选条件。数据库为了防止SQL注入，将选择划分为选择子句和选择参数。例如：

// Define 'where' part of query.  
String selection = FeedEntry.COLUMN\_NAME\_ENTRY\_ID + " LIKE ?";  
// Specify arguments in placeholder order.  
String[] selectionArgs = { String.valueOf(rowId) };  
// Issue SQL statement.  
db.delete(table\_name, selection, selectionArgs);

# 更新数据库

SQLiteDatabase db = mDbHelper.getReadableDatabase();  
  
// New value for one column  
ContentValues values = new ContentValues();  
values.put(FeedEntry.COLUMN\_NAME\_TITLE, title);  
  
// Which row to update, based on the ID  
String selection = FeedEntry.COLUMN\_NAME\_ENTRY\_ID + " LIKE ?";  
String[] selectionArgs = { String.valueOf(rowId) };  
  
int count = db.update(  
    FeedReaderDbHelper.FeedEntry.TABLE\_NAME,  
    values,  
    selection,  
    selectionArgs);