Data Storage and Exchange
Databases & Content Providers

Databases

- Android uses the built-in SQLite:
  - open-source, stand-alone SQL database:
  - widely used by popular applications/systems
  - FireFox uses SQLite to store configuration data
  - iPhone uses SQLite for database storage
- A database is private to the application:
  - but data can be exposed through a content provider

Database Cursor

- The *Cursor* class is the return value for queries:
  - *Cursor* is a pointer to the result set from a query
  - this efficiently manage rows and columns
- A *ContentValues* object stores key-value pairs:
  - *put* inserts keys with values of different data types

Database Example

- Create a helper class to encapsulate particular aspects of accessing the database:
  - becomes transparent to the calling code
  - create, open, close and use the database
- Book database example:

id	isbn	title	publisher
0	158603524X	Using Android	Wiley
1	0201101947	Android Decoded	CRC Press

Database Adapter

```
class DatabaseAdapter(private val dbContext: Context) {
   private val DBHelper: DatabaseHelper
   private lateinit var db: SQLiteDatabase

…

 companion object {
       const val KEY_ISBN = "isbn"
       const val KEY_TITLE = "title"
       const val KEY_PUB = "publisher"
       const val DATABASE_NAME = "books"
       const val DATABASE_TABLE = "titles"
       const val DATABASE_VERSION = 1
   }

   init {
       DBHelper = DatabaseHelper(dbContext)
   }
}
```

Database Helper

```
internal inner class DatabaseHelper(context: Context?) :
    SQLiteOpenHelper(context, DATABASE_NAME, null, DATABASE_VERSION) {

    override fun onCreate(db: SQLiteDatabase) {
        db.execSQL(
            "create table " + DATABASE_TABLE + "(" +
                "id integer primary key autoincrement," +
                "isbn text not null," +
                "title text not null," +
                "publisher text not null);"
        )
    }
```

… also overwrite onUpgrade(), and optionally onOpen()

Database Adapter: Inserting Data

```
@Throws(SQLException::class)
 fun open(): DatabaseAdapter {
     db = DBHelper.writableDatabase
     return this
 }

 fun close() {
     DBHelper.close()
 }

 // insert new book, returning new row identifier
 fun insertBook(isbn: String?, title: String?, publisher: String?): Long {
     val contentValues = ContentValues()
     contentValues.put(KEY_ISBN, isbn)
     contentValues.put(KEY_TITLE, title)
     contentValues.put(KEY_PUB, publisher)
     return db.insert(DATABASE_TABLE, null, contentValues)
 }
```

Database Adapter: Retrieving Data

```
@Throws(SQLException::class)
 fun getBook(rowId: Long): Cursor {

   //parameters  to db.query include columns to return and condition
   //returned Cursor positioned before the first item, if not empty

    val where = "id=$rowId"
    val columns = arrayOf(
       KEY_ISBN, KEY_TITLE, KEY_PUB
    )
    return db.query(DATABASE_TABLE, columns, where, …)
 }
```

Database Adapter: Retrieving Data

SQLiteDatabase::query():

table	The table name to compile the query against.
columns	A list of which columns to return. Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used.
selection	A filter declaring which rows to return, formatted as an SQL WHERE clause (excluding the WHERE itself). Passing null will return all rows for the given table.
selectionArgs	You may include ?s in selection, which will be replaced by the values from selectionArgs, in order that they appear in the selection. The values will be bound as Strings.
groupBy	A filter declaring how to group rows, formatted as an SQL GROUP BY clause (excluding the GROUP BY itself). Passing null will cause the rows to not be grouped.
having	A filter declare which row groups to include in the cursor, if row grouping is being used, formatted as an SQL HAVING clause (excluding the HAVING itself). Passing null will cause all row groups to be included, and is required when row grouping is not being used.
orderBy	How to order the rows, formatted as an SQL ORDER BY clause (excluding the ORDER BY itself). Passing null will use the default sort order, which may be unordered.
limit	Limits the number of rows returned by the query, formatted as LIMIT clause. Passing null denotes no LIMIT clause.

Database Adapter: Deleting/Updating Data

```
// delete a book , use WHERE argument with rowId
 //returning true if it worked
 fun deleteBook(rowId: Long): Boolean {
     val where = "id=$rowId"
     return db.delete(DATABASE_TABLE, where, null) > 0
 }

// update a book, returning true if it worked
fun updateBook(rowId: Long, isbn: String?, title: String?, publisher: String?): Boolean {
     val where = "id=$rowId"
     val contentValues = ContentValues()
     contentValues.put(KEY_ISBN, isbn)
     return db.update(DATABASE_TABLE, contentValues, where, null) > 0
}
```

Using The Database

- Instantiate *DatabaseAdapter* in the *Activity* constructor

```
class MyActivity : AppCompatActivity() {
  lateinit var db: DatabaseAdapter
  public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    db = DatabaseAdapter(this)
    db.open()
    db.insertBook("1234567", "Brilliant Book", “Top Publisher")
    val cursor = db.getBook(1)
    cursor.moveToFirst()
    while (!cursor.isAfterLast) {
        val isbn = cursor.getString(0)
        val title = cursor.getString(1)
        cursor.moveToNext()
     }
  }
}
```

Content Providers

- Files and databases are normally private:
  - unless specifically created otherwise
- Content providers provide data to other apps:
  - retrieve, modify and create data
- Example content providers in Android are *Contacts* and *MediaStore* (audio, images, video)
- Data is mapped to a URI used by clients:
  - *content://contacts/people/* all contact names
  - *content://contacts/people/23* contact with ID 23
  - *Uri.parse("content://contacts/people/23")*
  - *managedQuery(myPerson, …)*

Customised Content Provider

- Extend *android.content.ContentProvider*
- Required methods need overriding:
  - *onCreate()* called when a provider is created
  - *getType(uri)* return MIME type of data
  - *insert(uri, contentValues)*
  - *query(uri, columns, selection, selectionArgs, sortOrder)*
  - *update(uri, contentValues, selection, selectionArgs)*
  - *delete(uri, selection, selectionArgs)*
- To create a CP for the earlier DB example, these methods can call the methods in the DatabaseAdapter class. → lab session