6. PERSISTENCE DAT

BI-AND

Obsah

- Databáze
- Content Provider
- Soubory

SQLite

- Nejrozšířenější RDBMS (nejenom) pro embedded zařízení
  - Android, iOS, Symbian, BlackBerry, MeeGo, webOS
- Velmi malá (~275kB) a rychlá
- Public domain licence

- Použitá i ve webových prohlížečích a dalších aplikacích
  - Chrome, Firefox, Opera, standard pro HTML5
  - Adobe Systems, Skype, Solaris

Vlastnosti SQLite

- Transakce jsou atomické, konzistentní a trvanlivé (i po pádu systému) – ACID
- Kompletní databáze uložená v 1 souboru
- Podpora většiny standardů SQL92

Některé vlastnosti/kontroly nejsou implicitně zapnuty!

- Výjimky a omezení
  - Chybí RIGHT and FULL OUTER JOIN
  - Částečně ALTER TABLE – pouze RENAME a ADD COLUMN
  - Částečná podpora triggerů – chybí FOR EACH STATEMENT
  - VIEWs jsou read-only
  - Chybí GRANT a REVOKE

http://www.sqlite.org/omitted.html

Datové typy

- **NULL**
- **INTEGER** – hodnota se znaménkem uložená v 1, 2, 3, 4, 6, nebo 8 bytech v závislosti na rozsahu hodnoty
- **REAL** – Hodnota s plovoucí desetinnou čárkou uložená v 8-byte IEEE číslu s plovoucí desetinnou čárkou
- **TEXT** – Řetězec (délka až 2^{31}-1 bytů) uložen v kódování DB (UTF-8, UTF-16BE nebo UTF-16LE).
- **BLOB** – Binární data

SQLite v Androidu

- Plná podpora pro SQLite databáze
- V různých verzích OS se používá různá verze SQLite
- DB je uložena ve vnitřní paměti zařízení
  - Možnost uložit i na paměťovou kartu – řada nevýhod a problémů
- Doporučené praktiky:
  - Normalizovat data
  - Zapouzdření přístupu k DB do pomocných tříd
  - Neukládat do databáze soubory (obrázky, audio…)
- Po ukončení přístupu k DB je třeba ji uzavřít

SQLiteOpenHelper

- Zajišťuje vytvoření (pokud neexistuje), otevření, upgrade, downgrade databáze

```
1.  public class DictionaryOpenHelper extends SQLiteOpenHelper {
2.      private static final int DATABASE_VERSION = 2;
3.      private static final String DATABASE_NAME = "databaze";
4.
5.      DictionaryOpenHelper(Context context) {
6.          super(context, DATABASE_NAME, null, DATABASE_VERSION);
7.      }
8.
9.      @Override
10.     public void onCreate(SQLiteDatabase db) {
11.         String create =  "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
12.                 KEY_WORD + " TEXT, " +
13.                 KEY_DEFINITION + " TEXT);";
14.         db.execSQL(create);
15.     }
16.
17.     @Override
18.     public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
19.         //Upgrade databaze - většinou vytvoření dočasných tabulek a překopírování dat
20.         //nebo smazání DB a vytvoření nové
21.     }
22. }
```

Zavolá se, pokud databáze není vytvořená

Zavolá se, pokud se zvýší *DATABASE_VERSION*

Přístup k databázi

- Pro získání přístupu použijeme náš `DictionaryOpenHelper`
- Samotnou databázi dostaneme pomocí `dbHelper.getWritableDatabase();`

```
1. SQLiteOpenHelper dbHelper = new DictionaryOpenHelper(mCtx);
2. SQLiteDatabase mDb = dbHelper.getWritableDatabase();
```

- Po provedení potřebných úkonů databázi zavřeme

```
1. mDb.close();
```

Vkládání dat do databáze

- Po získání odkazu na databázi, provedeme vložení dat příkazem `insert(table,nullColumnHack,values);`
- Metoda `insert` vrací ID právě vloženého řádku nebo -1, pokud nastala chyba

Vytvořit si třídu s konstantami – klíče DB

- `ContentValues`
  - Dvojice klíč/hodnota
  - Používá se při vkládání nebo updatování tabulek
  - Klíč odpovídá jménu sloupce tabulky

```
1. ContentValues initialValues = new ContentValues();
2. initialValues.put(KEY_WORD, "word");
3. initialValues.put(KEY_DEFINITION, "definice slova word -...");
4. mDb.insert(DICTIONARY_TABLE_NAME, null, initialValues);
```

Vyhledávání v DB

Návrhový vzor – Query Object

- K prohledávání databáze slouží metoda `query`
- Vrací `Cursor`
  - Prostředek pro procházení výsledky dotazu, čtení řádků a sloupců
  - Po provedení potřebných úkonů je třeba `Cursor` zavřít

```
1. Cursor c = mDb.query(DICTIONARY_TABLE_NAME,
2.                null, //sloupce
3.                null, //klauzule WHERE
4.                null, //parametry,pokud se vyskytují ? ve WHERE
5.                null, //groupBy
6.                null, //having
7.                null);//orderBy
```

query(TABLE_DOCUMENTS, new String[] {KEY_DOCUMENTS_ID, KEY_DOCUMENTS_CREATEDBY_ID }, KEY_ID + "=?“, new String[] {id});

Vyhledávání v DB

- Čtení obsahu DB probíhá až v momentě, kdy zavoláme příslušnou metodu Cursoru (`getString`, `getInt`…)
- Cursor pouze ukazuje na výsledek vyhledávání (šetření paměti)

```
1. Cursor c = mDb.query(DICTIONARY_TABLE_NAME, null, null, null, null, null,
   null);
2. if (c.moveToFirst()) {//posune na začátek, vrátí false, pokud je cursor
   prázdný
3.    int colWord = c.getColumnIndex(KEY_WORD); //číslo sloupce "word"
4.    int colDef = c.getColumnIndex(KEY_DEFINITION); //číslo sloupce "definition"
5.    do {
6.         String word = c.getString(colWord);
7.         String def = c.getString(colDef);
8.         ...
9.    } while (c.moveToNext()); //posune cursor na další řádek
10.   c.close();
11. }
```

Databáze a životní cyklus activity

- DB je nutné po jejím otevření zavřít
  - Open/close v rámci jednoho požadavku
    - Pomalejší
  - V rámci životního cyklu activity
    - Pozor na správné pořadí open/close
- Nezavírat DB, pokud nejsou uzavřeny i cursory
- Obdobně je nutné zavřít i cursory
- Z důvodů paměťové náročnosti se nevyplácí používat různé formy ORM nebo vytvářet pomocné třídy pro obalení dat z databáze

DB otevřená po celou dobu běhu aplikace?

Další metody Cursoru

- `moveToPrevious()`
- `getCount()`
- `getColumnIndexOrThrow(String columnName)`
- `getColumnName(int columnIndex)`
- `getColumnNames()`
- `getColumnCount()`
- `moveToPosition(int position)`
- `getPosition()`
- `isClosed()`
- ...

Update dat v databázi

- Aktualizace dat se provádí pomocí metody `update(table, values, whereClause, whereArgs);`
- Metoda vrací počet aktualizovaných řádků

```
1. ContentValues updatedValues = new ContentValues();
2. updatedValues.put(KEY_DEFINITION, "nová definice");
3.
4. mDb.update(DICTIONARY_TABLE_NAME,
5.         updatedValues,
6.         KEY_WORD+"='word'", //klauzule WHERE
7.         null); //parametry, pokud se vyskytují ? ve WHERE
```

Mazání dat z databáze

- Mazání dat se provádí pomocí metody `delete(table, whereClause, whereArgs);`
- Metoda vrací počet smazaných řádků, pokud `whereClause` není `null`, jinak `0`
- Pro smazání všech řádků a získání počtu předáme do `whereClause` hodnotu `"1"`

```
1. mDb.delete(DICTIONARY_TABLE_NAME,
2.         KEY_WORD+"='word'", //klauzule WHERE
3.         null); //parametry, pokud se vyskytují ? ve WHERE
```

Zámek databáze

- Defaultně se SQLite stará sám o zamykání DB v kritických sekcích a zabraňuje tak poškození dat při zápisu z více vláken
- Pozor na (ne)vrácené výjimky – použít např. frontu nebo jinou synchronizaci přístupu
- Časově náročné
  - Pokud se k DB přistupuje pouze z jednoho vlákna, je dobré zamykání vypnout

```
1. mDb = dbHelper.getWritableDatabase();
2. mDb.setLockingEnabled(false);
```

Programy pro práci s SQLite na desktopu

- SQLite Database Browser
  - http://sqlitebrowser.sourceforge.net
- Sqliteman
  - http://sqliteman.com/
- Pozor na různé verze SQLite a s tím související odlišnosti v syntaxi zápisu – např. NO ACTION

Content Provider

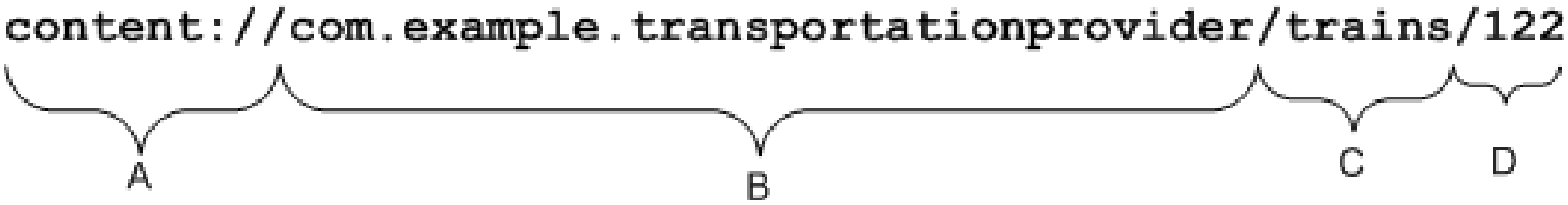
- Prostředek pro poskytování dat
- Data uložena v SQLite DB, souborech, na webu apod.
  - Závisí na konkrétní implementaci
- Jediný možný způsob, jak sdílet data napříč aplikacemi
  - Výjimkou je:
    - Mode - při přístupu k některým souborům
    - IPC - Android Interface Definition Language (AIDL)
- Možné využívat i pro přístup k datům z vlastní aplikace
  - Výhodou je zkrácení kódu

Content Provider

- Každá aplikace může poskytovat svoje data přes `Content Provider` definované pomocí URI
- Pro přístup k datům poskytovaným `ContentProviderem` slouží `ContentResolver`
- Příklady (systémových) content providerů
  - Kontakty
  - SMS
  - Zmeškané hovory
  - Multimedia library

Content Provider – URI

- Musí se definovat unikátní URI



A. Scheme - standardní předpona indikující, že data mají být zpracována Content Providerem

Povinná je pouze část A a B

B. Authority - identifikátor Provideru
   - Musí být deklarována v AndroidManifestu
   - Měla by obsahovat jméno balíčku, aby se zajistila jedinečnost

C. Data type path - vymezení obsahu
   - Může mít několik úrovní (`land/bus, land/train, sea/ship…`)

D. Instance identifier - ID položky, kterou požadujeme
   - Pokud chceme více záznamů, vynecháme

Content Provider – URI

- Definice provideru v AndroidManifestu

```
1. <provider
2.     android:name=".MyProvider"
3.     android:authorities="cz.cvut.exampleprovider" />
```

Privátní CP - android:exported=“false“

- Třída MyProvider.java

```
1. public class MyProvider extends ContentProvider{
2.   public static final Uri CONTENT_URI =
3.       Uri.parse("content://cz.cvut.exampleprovider");
4.   ...
5. }
```

Content Provider

- Je potřeba vždy implementovat 6 základních metod
  - `onCreate` – inicializace provideru, nastavení přístupu např. k DB
  - `getType` – vrácení MIME dat u konkrétní URI
    - MIME formát ve tvaru vnd.X.cursor.dir/Y nebo vnd.X.cursor.item/Y
    - X – název společnosti, projektu apod., Y – název typu oddělený tečkami
  - `query` – výběr dat
  - `insert` – přidání dat
  - `update` – aktualizace dat
  - `delete` – smazání dat

Content Provider - UriMatcher

- Pokud Provider poskytuje více typů dat, je potřeba pro každý typ vytvořit proměnnou a nastavit `UriMatcher`

```
content://cz.cvut.exampleprovider/auta
content://cz.cvut.exampleprovider/auta/3
content://cz.cvut.exampleprovider/lode
content://cz.cvut.exampleprovider/lode/7
```

```
1.  private static final int AUTA_ALL_ROWS = 1;
2.  private static final int AUTA_SINGLE_ROW = 2;
3.  private static final int LODE_ALL_ROWS = 3;
4.  private static final int LODE_SINGLE_ROW = 4;
5.
6.  private static final UriMatcher uriMatcher;
7.  static {
8.        uriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
9.        uriMatcher.addURI("cz.cvut.exampleprovider", "auta", AUTA_ALL_ROWS);
10.       uriMatcher.addURI("cz.cvut.exampleprovider", "auta/#", AUTA_SINGLE_ROW);
11.       uriMatcher.addURI("cz.cvut.exampleprovider", "lode", LODE_ALL_ROWS);
12.       uriMatcher.addURI("cz.cvut.exampleprovider", "lode/#", LODE_SINGLE_ROW);
13. }
```

Content Provider - getType

- Vrací MIME požadavku v aktuální URI
- Formát:
  - 1 položka: `vnd.<autor>.cursor.item/<typ obsahu>`
  - Všechny položky: `vnd.<autor>.cursor.dir/<typ obsahu>`

```
1.   @Override
2.   public String getType(Uri uri) {
3.     switch (uriMatcher.match(uri)) {
4.     case AUTA_ALL_ROWS:
5.          return "vnd.cvut.cursor.dir/auta";
6.     case AUTA_SINGLE_ROW:
7.          return "vnd.cvut.cursor.item/auta";
8.     case LODE_ALL_ROWS:
9.          return "vnd.cvut.cursor.dir/lode";
10.    case LODE_SINGLE_ROW:
11.         return "vnd.cvut.cursor.item/lode";
12.    default:
13.         throw new IllegalArgumentException("Unsupported URI: " + uri);
14.    }
15.  }
```

Content Provider - query

- Vrací Cursor s výsledky dotazu
- Analogicky se chovají ostatní metody (`insert`, `update`, `delete`)

```
1.   @Override
2.   public Cursor query(Uri uri, String[] projection, String selection,
3.   String[] selectionArgs, String sortOrder) {
4.    switch (uriMatcher.match(uri)) {
5.    case AUTA_ALL_ROWS:
6.         return db.fetchAllCars();
7.    case AUTA_SINGLE_ROW:
8.         String rowNumber = uri.getLastPathSegment();
9.         int i = Integer.parseInt(rowNumber);
10.        return db.fetchCar(i);
11.  ...
12.  }
```

Většinou se používá SQLiteQueryBuilder

Content Provider - query

- Získaní Cursoru za použití SQLiteQueryBuilderu

```
1.  @Override
2.  public Cursor query(Uri uri, String[] projection, String selection,
3.  String[] selectionArgs, String sortOrder) {
4.  SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
5.   switch (uriMatcher.match(uri)) {
6.   case AUTA_ALL_ROWS:
7.        queryBuilder.setTables(TABLE_CAR);
8.        break;
9.   case AUTA_SINGLE_ROW:
10.       String rowNumber = uri.getLastPathSegment();
11.       queryBuilder.setTables(TABLE_CAR);
12.       queryBuilder.appendWhere(ID + "=" + rowNumber);
13.       break;
14. ...//přístup do DB atd.
15. return queryBuilder.query(db, projection, selection, selectionArgs,
    orderBy, null, sortOrder);
16. }
```

Content Resolver

- Slouží k přístupu do ContentProvideru
- Referenci získáme `context.getContentResolver();`
- Chová se podobně jako DB
  - Update, insert, delete a query jsou stejné s výjimkou prvního parametru
  - První parametr je URI požadovaného obsahu

```
1.  Cursor c = getContentResolver().query(Contacts.CONTENT_URI, null, null, null, null);
2.  startManagingCursor(c);
3.  int colName = c.getColumnIndex(Contacts.DISPLAY_NAME);
4.  int colId = c.getColumnIndex(Contacts._ID);
5.  if(c.moveToFirst()){
6.     do{
7.           String jmeno = c.getString(colName);
8.           String id = c.getString(colId);
9.           ...
10.    }while(c.moveToNext());
11. }
12. stopManagingCursor(c);
```

Content Resolver

Start/StopManagingCursor() je deprecated!

- `startManagingCursor & stopManagingCursor`
  - Metody které provážou životní cyklus Cursoru s životním cyklem Activity
  - Pokud se activita zastaví, `Cursor` se deaktivuje
  - Když se activita opět obnoví, zavolá se `requery();`
  - Po skončení activity se `Cursor` uzavře

- Pro přístupu k systémovému ContentProvideru je většinou zapotřebí oprávnění

Oprávnění pro přístup ke kontaktům (`AndroidManifest.xml`)

```
1. <uses-permission
2.       android:name="android.permission.READ_CONTACTS" />
```

Soubory

- Podpora standardních `java.io` tříd a metod

```
1. // Vytvorime novy File s adresarem, kde chceme mit soubor
2. File wallpaperDirectory = new File("/sdcard/myDirectory/");
3. // Vytvorime adresare, pokud neexistuji
4. wallpaperDirectory.mkdirs();
5. // Vytvorime novy File se cestou a jmenem souboru
6. File outputFile = new File(wallpaperDirectory,
   "newFile.txt");
7. // Ziskame OutputStream
8. FileOutputStream fos = new FileOutputStream(outputFile);
9. // Zapiseme pozadovany obsah
10.fos.write(new String("text").getBytes());
11.// Zavreme outputstream
12.fos.close();
```

Pokud zapisujeme do externího úložiště, potřebujeme oprávnění v AndroidManifestu

```
1. <uses-permission
2.       android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Soubory

- Jsou 2 základní umístění, kam můžeme ukládat soubory
  - Interní úložiště
    - Umístění ve složce: `/data/data/<jméno aplikace>/files/`
    - IOStreamy získáme pomocí :

```
1. os = openFileOutput("soubor", MODE_PRIVATE);
2. is = openFileInput("soubor");
```

    - Vhodné pro ukládání malých souborů
  - Externí úložiště
    - Doporučené umístění souborů aplikace:
      `/sdcard/Android/data/<jméno aplikace>/files/`
    - IOStreamy získáme pomocí :

Zkontrolovat jestli je externí úložiště připojeno!

```
1.  //null = ./, Environment.DIRECTORY_MUSIC = ./Music, ...
2.  File dir = getExternalFilesDir(null);
3.  File f = new File(dir, "soubor");
4.  os = new FileOutputStream(f);
5.  is = new FileInputStream(f);
```

Soubory - cache

- Pokud potřebujeme uložit soubory, jejichž smazáním nepřijdeme o žádná důležité data, použijeme `cache`
- Umístění bude `/data/data/<jméno aplikace>/cache/` nebo `/sdcard/Android/data/<jméno aplikace>/cache/`
- Pokud bude docházet místo v úložišti, systém automaticky smaže tyto soubory

```
1. File dir = getCacheDir(); // popr. getExternalCacheDir();
2. File f = new File(dir, "souborCache");
3. os = new FileOutputStream(f);
4. is = new FileInputStream(f);
```

Soubory – další užitečné metody

- `getFilesDir()`
  - vrátí absolutní cestu složky vašeho interního úložiště
- `getDir(String name, int mode)`
  - vytvoří(nebo otevře existující) složku ve vašem interním úložišti
- `deleteFile()` – smaže soubor ve vašem interním úložišti
- `fileList()` – vrátí pole aktuálně uložených souborů

Soubory

- Složky `/data/data/<jméno aplikace>/` a `/sdcard/Android/data/<jméno aplikace>/` budou po odinstalování aplikace automaticky smazány

- Uživatel může v nastavení aplikace dané soubory smazat ručně!

- Částečně se ustupuje od používání externího úložiště

- Shared Preferences – 7. přednáška

Další zdroje

- http://www.ibm.com/developerworks/xml/library/x-androidstorage/index.html
- http://kagii.com/post/6828016869/android-sqlite-locking