Baze podataka

SQLite

SQLite[1] je biblioteka otvorenog koda koja omogućava rad sa relacionim bazama podataka. Male je veličine i laka za podešavanje te joj je najčešća primena kao lokalno skladište za ugrađene (eng. embedded) uređaje i pametne telefone.

Relacione baze podataka organizuju resurse u tabele čije vrste predstavljaju objekte (elemente). Jedna baza može sadržati jednu ili više tabela koje mogu da budu u relaciji. Objekti sadrže polja koja predstavljaju atribute (osobine) tog objekta i oni su definisani u kolonama tabele. Svaki objekat mora da bude unikatan, što znači da mora postojati bar jedno polje koje de jedinstveno identifikovati svaki element, kako bi se zaobišli konflikti prilikom selekcije željenih elemenata iz tabele.

id	firstName	lastName	age
0	Pera	Perić	30
1	Mika	Mikić	25
2	Jovan	Jovanović	27

Tabela 1: Primer tabele sa 3 objekta; nazivi kolone označavaju atribute objekata

Tipovi podataka za skladištenje

SQLite tipovi podataka specificiraju tip resursa koji se može čuvati u jednoj koloni tabele. Kolona predstavlja atribut koji opisuje sve objekte u tabeli. Svaka kolona može čuvati atribut različitog tipa. Postoji pet tipova podataka podržanih u SQLite bazama podataka:

1. NULL - prazno polje, informacija je trenutno nepoznata
2. INTEGER - označeni ceo broj
3. REAL - decimalna vrednost
4. TEXT - niz karaktera
5. BLOB (Binary Large OBject) - niz binarnih podataka.

Napomena: SQLite ne podržava boolean tip te se za tu potrebu može iskoristiti INTEGER (0 - false, 1 - true). Takođe, SQLite ne podržava datum (date) tip, za čiju alternativu se može koristiti TEXT tip.

[1] Na linku https://www.sqlite.org/download.html se može preuzeti *command-line* sqlite alat (odabrati ***sqlite-tools*** opciju). Na linku https://sqlitebrowser.org/ se može preuzeti aplikacija za pregled i upravljanje bazom podataka

SQLite upiti

Upravljanje SQLite bazom se vrši pomoću komandi navedenih u tabeli - Tabela 3. Ove komande se kombinuju sa odgovarajućim parametrima, formirajući SQL upit (*engl.* query) koji obavlja neku od definisanih operacija nad bazom podataka. SQLite sintaksa je neosetljiva na veličinu slova (*engl.* case insensitive).

Komanda	**Opis**
CREATE	Kreira novu tabelu.
SELECT	Filtrira podatke iz jedne ili više tabela.
INSERT	Formira novi element i upisuje ga u tabelu.
UPDATE	Menja element u tabeli.
DELETE	Briše element iz tabele.
DROP	Briše tabelu.

Tabela 3: SQLite komande

1. Kreiranje tabele

```
CREATE TABLE <table> (column1 TYPE, column2 TYPE…)
```

Primer kreiranja tabele STUDENT sa poljima `ID`, `Name`, i `Age`:

```
CREATE TABLE STUDENT (ID INTEGER, Name TEXT, Age INTEGER);
```

Izgled tabele nakon izvršenja komande **CREATE**:

StudentId	**Name**	**Age**

2. Unos elemenata u tabelu

```
INSERT INTO <table> VALUES (val1, val2,…)
```

Primer unosa elemenata u tabelu STUDENT:

```
INSERT INTO STUDENT VALUES (0, "John", 23);
INSERT INTO STUDENT VALUES (1, "Mike", 30);
INSERT INTO STUDENT VALUES (2, "Trevor", 19);
```

Izgled tabele nakon izvršenja komande **INSERT**:

StudentId	Name	Age
0	John	23
1	Mike	30
2	Trevor	19

3. Brisanje elemenata iz tabele

```
DELETE FROM <table> WHERE <search condition>
```

Primer brisanja elemenata kojima je vrednost atributa `Age` 23:

```
DELETE FROM STUDENT WHERE Age=23;
```

Uslovi za formiranje upita (search condition) mogu da se kombinuju logičkim relacijama. Primer brisanja elemenata kojima je vrednost atributa `Age` 23, a vrednost `Name` atributa John:

```
DELETE FROM STUDENT WHERE Age=23 AND Name=John;
```

4. Filtriranje elemenata u tabeli

```
SELECT <column1,column2,…> FROM <table>
   [WHERE <search condition>]
   [GROUP BY <column1, column2,…>]
   [ORDER BY <column1, column2,…> ASC|DESC]
```

Primeri korišćenja komande SELECT:

```
SELECT Name FROM STUDENT;
SELECT Name, Age FROM STUDENT WHERE Age=23 OR Age=30;
SELECT * FROM STUDENT;
```

Korišćenjem specijalnog karaktera ***** dobavljamo SVE kolone iz tabele.

5. Brisanje tabele

```
DROP TABLE <table>
```

Upravljanje SQLite bazom podataka u Android aplikacijama

Android pruža dve klase radi lakšeg upravljanja SQLite bazom podataka: **SQLiteOpenHelper** i **SQLiteDatabase.**

SQLiteOpenHelper klasa

SQLiteOpenHelper je klasa koju je potrebno naslediti kako bi se na lak način izvršila inicijalizacija baze i abstrahovale sve funkcionalnosti poput kreiranja i ažuriranja baze podataka.

Ova klasa omogućava korišćenje dve metode: **`getWritableDatabase`** i **`getReadableDatabase`** koje vraćaju instancu baze u modu za pisanje i čitanje, respektivno. Ove metode su „thread safe" što znači da se bazi može pristupati sa sigurnošću da će sve niti koje joj pristupaju biti sinhronizovane kako se ne bi desilo da dve niti u isto vreme upisuju ili čitaju podatke, čime se zaobilaze konflikti.

Klasa SQLiteOpenHelper sadrži dve bitne „callback" metode: **`onCreate`** i **`onUpgrade`**. Metoda `onCreate` se poziva kada se prvi put kreira baza podataka i u njoj je potrebno izvršiti kreiranje i inicijalizaciju tabela. Metoda `onUpgrade` se poziva kada se promeni verzija baze podataka te je u ovoj metodi moguće odraditi određenu akciju u zavisnosti od verzije baze.

SQLiteDatabase klasa

SQLiteDatabase klasa omogudava izvršavanje standardnih SQL komandi. Ova klasa sadrži metodu **`execSQL`** koja omogućava izvršavanje SQL komandi. Metodi je potrebno proslediti tekst upita kao String objekat. Kao rezultat, metoda vraća rezultat tražene operacije.

Kreiranje baze podataka

Primer kreiranja baze podataka `STUDENT` sa kolonama `ID` (integer), `Name` (text) i `Age` (integer):

```
db.execSQL(
    "CREATE TABLE STUDENT(ID INTEGER, Name TEXT, Age INTEGER);"
);
```

Dodavanje elemenata u bazu podataka

Dodavanje elemenata u bazu podataka omogućava metoda **`insert`** klase SQLiteDatabase. Kao parametar, metoda `insert` očekuje objekat klase *ContentValues*. U objekat klase ContentValues mogu da se smeste podaci u obliku *ključ-vrednost* pomoću metode `put`.

```
public long insert (String table, String nullCol, ContentValues values)
```

Primer smeštanja podataka u ContentValues objekat i upis istih podataka u bazu `STUDENT`:

```
SQLiteDatabase db = getWrtableDatabase();
ContentValues values = new ContentValues();
values.put("ID", 0);
values.put("Name", "John");
values.put("Age", 23);
db.insert("STUDENT", null, values);
close();
```

Svaki put kada se završi rad sa bazom (dodavanje, čitanje, brisanje elemenata) potrebno je pozvati metodu **`close`** koja zatvara instancu baze.

Brisanje elemenata iz baze podataka

Brisanje elemenata iz baze podataka omogućava metoda **`delete`** klase SQLiteDatabase.

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

Primer brisanja elemenata kojima je vrednost atributa `Age` 23:

```
SQLiteDatabase db = getWritableDatabase();
db.delete("STUDENT", "Age=?", new String[] {"23"});
close();
```

Filtriranje elemenata baze podataka

Metoda koja omogućava izvršavanje upita nad tabelom baze je **`query`**. Ova metoda vraća objekat *Cursor*. Cursor je interfejs koji predstavlja dvodimenzionalnu tabelu baze. Cursor koji je dobijen kao povratna vrednost pokazuje na prvi element iz filtriranih rezultata. Uz pomoć cursor metoda moguće je iterirati i preuzimati željene podatke iz tabele.

```
public Cursor query (
 String table,           // ime tabele
 String[] columns,       // lista kolona za prikaz
 String selection,       // selekcija po kojoj se pretražuje
 String[] selectionArgs, // argumenti selekcije
 String groupBy,         // kolone po kojima se grupišu podaci
 String having,          // uslov HAVING
 String orderBy);        // način sortiranja dobavljenih podataka
```

Primer dobavljanja svih kolona tabele STUDENT grupisanih po atributima `Age` i `Name`, kojima je vrednost atributa `Name` John:

```
SQLIteDatabase db = getReadableDatabase();
Cursor cursor = db.query(
 "STUDENT",
 null,
 "Name=?",
 new String[] {"John"},
 "Age, Name",
 null,
 null);
```