הבהרה:

במידה ונרצה להוסיף עמודה חדשה לאחר שבנינו את הטבלה בJAVA נשתמש בקונספט של drop(מחיקת טבלה) ו create teble מכיוון שאין אפשרות להוסיף עמודה לטבלה קיימת(**למרות שבתוכנה של SQLite יש את האפשרות הזו בפועל אין אותה בג'אווה)**. בג'אווה פונקציה זו נקראית uprgrade והיא מופעלת כאשר משתנית גרסת הבסיס נתונים כמו שיתבאר בהמשך.(בכל מקרה נוכל תמיד בטבלאות שניצור להוסיף עוד 2-3 עמודות ריקות רזרבה למקרה שנצטרך להוסיף נתון חדש לאובייקט קיים).

SQLite+Java

על מנת להתחיל עבודה משותפת של ג'אווה עם SQLite , עלינו ליצור 2 קלאסים(הפרד ומשול):

1. DBHelper
2. DBhendler

הראשון נועד על מנת ליצור את בסיס הנתונים במידה ואינו קיים, או - במידה שכן קיים (לאחר הפעלה ראשונית של האפליקציה) - להחזיק בתוכו רפרנס שאיתו ניצור קשר עם הDataBase .

DBHelper

לקלאסס זה נעשה extends SQLiteOpenHelper דבר שמאפשר לנו לכתוב ולמשוך נתונים מבסיס נתונים. אפשרויות האלה מגיעים לקלאסס זה בירושה מקלאסס – sqliteDatabase (לא אמור לעניין אותנו איך כמה ולמה)

```
public class DBHelper extends SQLiteOpenHelper{

}
```

קלאס זה יכיל 3 מטודות(הם נדרשות ליישום, אין צורך לזכור, רק יש לבחור את הקונסטרקטור הנכון):

1. DBHelper (constructur)
2. onCreate()
3. onUpgrade

DBHelper

עם קונסטרקטור זה אנו מייצרים פעם אחת בלבד את בסיס הנתונים שלנו.

למרות שפקודה זו תיקרא בכל הפעלה של האפליקציה, היא לא תיצור בסיס נתונים חדש אלא תייצר רפרנס לקיים.

```
public DBHelper(Context context, String name, CursorFactory factory,
                int version) {
            super(context, name, factory, version);

}
```

הסבר על הקונסטרקטור:

Context - מי הקלאסס שקרא לי?

Name - שם של בסיס הנתונים("books.db" צריך לכתוב בתוך סוגריים כסטרינג .)

לפי קריטריון זה בעצם מזהים אם יש בסיס נתונים קיים בשם זה או לא.

Factory - בלי להרחיב, נרשום null בשדה זה.

Version – מספר שלם שמייצג את גירסת בסיס הנתונים. כאשר נוסיף\נוריד עמודות לדוגמא מחלק מהטבלאות ונרצה שגם מי שמשתמש בבסיס הנתונים הישן שלנו יקבל את הטבלאות החדשות. בעת החלפת מספר הגרסה מ1 ל-2 לדוגמא, תתבצע פונקציית onUpgrade שתמחק את בסיס הנתונים הישן, תבנה את החדש במקומו. ותציב את הנתונים הישנים בטבלאות החדשות.

onCreate

```
@Override
	public void onCreate(SQLiteDatabase db) {
		String cmd = "CREATE TABLE book (_id INTEGER PRIMARY KEY, name TEXT, author
TEXT, price REAL);";
		try{
			db.execSQL(cmd);
		}catch(SQLiteException e){
			e.getCause();
		}
	}
```

פונקציה זו מקבלת לתוכה את בסיס הנתונים שיצרנו בDBHelper ומייצרת טבלאות כפי שנדרוש בשורת ה cmd בתחביר של SQL פשוט. מכיוון שכל מה שקשור לבסיס הנתונים נדרשת זהירות משנה, אנו נשים לב שכמעט בכל מקום יש לנו try כדי לוודא מה יקרה במצב קריסה. צריך להתרגל להתייחס לבסיס הנתונים בצורה רגישה ביותר.

Upgrade

```
@Override
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {

	}
```

תחליט פונקציה זו הוסבר בפונקציית DBHelper בקטגורית version של הקונסטרקטור.

DBhendler

קלאס זה אינו יורש (extends) מאף אחד

עם קלאס זה נעשה את כל המניפולציות שנרצה על בסיס הנתונים

```
public class DBHendler {

private DBHelper helper;
private SQLiteDatabase db;

}
```

בראש הקלאס נצהיר על אובייקט אחד גלובלי (**DBHelper**) שאיתו נעבוד בכל הפונקציות(למרות שבדוגמא כאן יש שניים רק כדי לרכז במקום אחד)

מיד לאחר מכן ניצור את הקונסטרקטור:

```
public DBHendler(Context con) {

		helper = new DBHelper(con,"notes.db",null,1);
	}
```

הסבר על נתונים אלו אפשר לראות למעלה בדף DBHelper.

1. DBHelper – בסיס הנתונים שלנו (גלומי)
2. SQLiteDatabase – לתוך אובייקט זה נמשוך את בסיס הנתונים שלנו כך שיהיה זמין למניפולציות שונות. אנו נעבוד עם 2 מטודות עיקריות:
   - db = helper.get**Writable**Database();
   - db = helper.get**Readable**Database();

לפי השמות ניתן להבין שעם הפונקציה הראשונה אנו מושכים את בסיס הנתונים באופן שניתן להוספה,מחיקה ועדכון של נתונים. ואילו בשניה רק אפשרות להציג את הנתונים(במקרים שנרצה שלא תהיה אפשרות אפילו בטעות לעשות שינוי בבסיס נתונים)

get**Writable**Database()

לאחר שמשכנו את בסיס הנתונים בצורה הזו נוכל לבצע את הפעולות הבאות:

- `db.insertOrThrow("book",null, cv);`
- `db.delete("book",null, null);`
- `db.delete("book", "_id=?", new String[]{id});`
- `db.update("book", cv, "_id=?", new String[]{id});`

לפני שנפרט רק נשים לב להקבלה של הפונקציות לפי הסדר כמו שהם ב SQLite שיהיה יותר קל להבין:

- `INSERT INTO book VALUES (_id INTEGER PRIMARY KEY, name TEXT);`
- `DELETE FROM book (מחיקת כל הטבלה)`
- `DELETE FROM book WHERE id="3";(מחיקת שורה מסויימת בטבלה)`
- `UPDATE book SET name="moshe" WHERE id="3";`

חשוב לציין שבג'אווה, האובייקט cv – (ContentValues) מכיל בתוכו "keys="values כפי שנראה בהמשך.

<u>הסבר על כל פונקציה:</u>

1. הוספת פריט לבסיס נתונים(הסבר זה ירחיב גם על דברים נוספים שתקפים לגבי שאר הפונקציות):

```
public void addBook(String name,String author,String price){
        SQLiteDatabase db = helper.getWritableDatabase();
        try{
                ContentValues cv= new ContentValues();
                cv.put("name", name);
                cv.put("author", author);
                cv.put("price", price);
                db.insertOrThrow("book",null, cv);
        }catch(SQLiteException e){
                e.getCause();
        }finally{
                if (db.isOpen())
                        db.close();
        }
    }
```

הסבר על השורה הצהובה:

Book – שם הטבלה שאליה נרצה להזין נתונים

Null – מקום זה קובע איך להתייחס לשדות ריקים. כשקובעים null בעצם זה הערך שנכנס לתוך שדה ריק.

Cv – הנתונים שברצוננו להוסיף. אפשר לראות שהפקודה cv.put("key",value); חוזרת 3 פעמים עבור 3 שדות שונים.

בתחביר של SQL היה עלינו לרשום כך(מבלי לציין את שמות השדות מכיוון שזה בהתאמה לפי הסדר):

INSERT INTO book VALUES (name,author,price);

הסבר על השורה הירוקה(נכון גם לדבי דוגמת העדכון בהמשך):

מכיוון שנרצה להזין מספר נתונים לטבלה אנו נשתמש באובייקט ContentValues שהפונקציה insertOrThrow יודעת לאכול בלי מלח...

לגבי השורה הכחולה רק נציין שוב שכאן נמשך הבסיס נתונים ונעשה 'נזיל' לעריכה.

לגבי פונקצית ה try–

אנו מוודאים שבמצב קריסה(טעות בקוד, נגמרה הסוללה, נפל לאוקיינוס וכו') בסיס הנתונים ייסגר בכל מקרה(**finally**)

ניתן לזה משל קטן:

***אדם הולך ברחוב ופותח את הארנק. בארנק יש הרבה שטרות של כסף. במידה והאדם נופל כל הכסף יתפזר על הרצפה.

ומכיוון שלא נרצה שדבר כזה יקרה, ה- finally דואג **שבכל מקרה** גם אם האדם יפול או לא, הארנק יסגר

ולענייננו, מכיוון שבשורה הכחולה הפכנו את הבסיס נתונים לפגיע (ניתן לערוך למחוק וכו') נרצה לוודא שבכל מקרה לא משנה מה יקרה בסיס הנתונים יסגר בסוף הפעולה. כמו כן התנאי if(db.isOpen) בודק האם בסיס הנתונים כבר פתוח, במידה וכן הוא יסגור אותו, שזו בעצם פעולת ההגנה שאנו מחפשים. (אם היינו סוגרים בסיס נתונים שהוא כבר סגור היינו מקבלים אקספשן)

השורה הסגולה דואגת להביא לנו תיאור שגיאה מדוייק לבסיסי נתונים.

2. מחיקת **כל** נתוני הטבלה:

```
public void deleteAll (){
            SQLiteDatabase db = helper.getWritableDatabase();
            try{
                  db.delete("book",null, null);
            }catch(SQLiteException e){
                  e.getCause();
            }finally{
                  if
 (db.isOpen())
                        db.close();
            }
      }
```

Book – הטבלה שממנה נרצה למחוק

כאשר 2 האופציות הבאות מוגדרות כnull זו פקודה למחיקת כל התאים(בעצם אנו לא מציינים לו את מה למחוק).

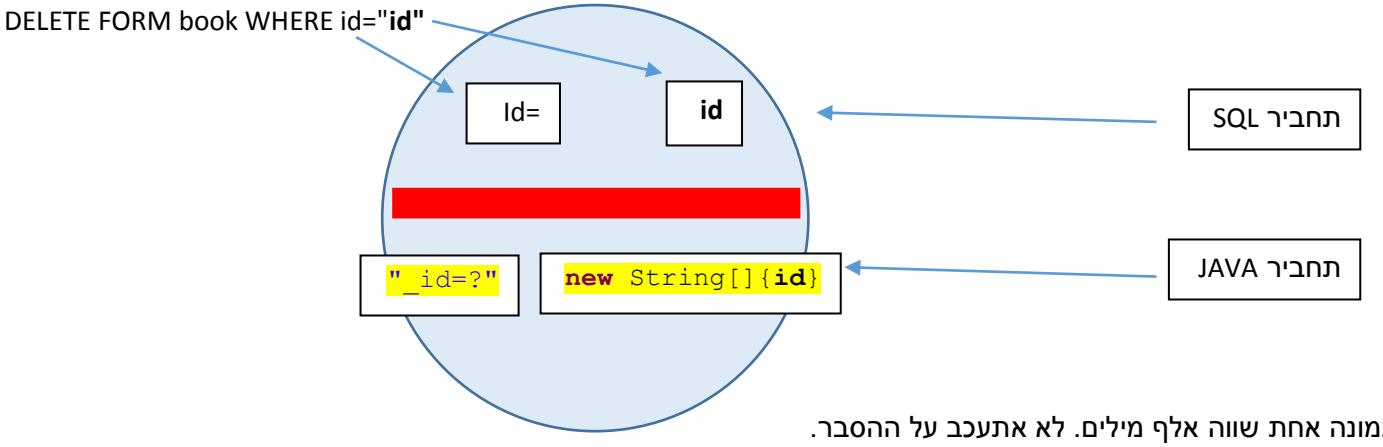
3. מחיקת שורה מסויימת בטבלה.

```
public void deleteBook (String id){
            SQLiteDatabase db = helper.getWritableDatabase();
            try{
                  db.delete("book", "_id=?", new String[]{id});
            }catch(SQLiteException e){

            }finally{
                  if (db.isOpen())
                        db.close();
            }
      }
```

עכשיו במקום 2 null יש לנו את האפשרויות הבאות שהם דבר אחד: `"_id=?", new String[]{id}`

שזה מקביל לפקודת SQL לפי התיאור הבא:



תמונה אחת שווה אלף מילים. לא אתעכב על ההסבר.

<u>עדכון שורה מסוימת(לדוגמא עדכון מחיר):</u>

```
public void updatePrice(String id, String price){
            SQLiteDatabase db = helper.getWritableDatabase();
            ContentValues cv = new ContentValues();
            cv.put("price", price);
            try{
                  db.update("book", cv, "_id=?", new String[]{id});
            }catch(SQLiteException e){
                  e.getCause();
            }finally{
                  if (db.isOpen())
                        db.close();
            }
      }
```

Book – שם הטבלה שאותה נעדכן

Cv – הנתונים שנרצה לעדכן

Id=? + new String[]{id} – מספר השורה שנרצה לעדכן

getReadableDatabase()

לאחר שנמשוך את בסיס הנתונים עם הפונקציה הזו נוכל להציג את הנתונים ע"י adapter בתוך ListView .

```
public ArrayList<String> getAllBooks(){
        Cursor cursor = null;
        SQLiteDatabase db = helper.getReadableDatabase();
        try{
            Cursor cursor = db.query("book", null, null, null, null, null, null);
        }catch(SQLiteException e){
            e.getCause();
        }

        ArrayList<String> list = new ArrayList<String>();
        while (cursor.moveToNext()){
            int id = cursor.getInt(0);
            String name = cursor.getString(1);
            String author = cursor.getString(2);
            String price = cursor.getString(3);
            list.add(id+" "+name+" "+author+" "+price);
        }
        return list;
    }
```

בשורה הכחולה אנו מושכים את הבסיס נתונים במצב קריאה

בשורה הצהובה יש את אובייקט הCursor. שהוא יודע להכיל בתוכו את בסיס הנתונים אך גם כאן, באופן גולמי.

הדרך לרוץ על כל הנתונים יתבצע באופן שמסומן בשורה הירוקה מכיוון שאין אינדקס לאובייקט קורסר (אני חושב...)

ובשורה הסגולה הראל, כאן הקסם... –

בבסיס נתונים לשורות אין אינדקס ומשום כך המעבר בין שורה לשורה מתבצע ע"י MoveToNext()

אך לעמודות **יש אינדקס** לכן אם נסתכל על הטבלה הבאה:

0	1	2	3
_id	Name	author	Price
1	moshe	rabeyno	Tag price
2	david	king	770

השורות הבאות יפנו לנתונים הבאים (במחזור של 2 שורות)

```
int id = cursor.getInt(0); - 1
String name = cursor.getString(1); - "moshe"
String author = cursor.getString(2); - "rabeyno"
String price = cursor.getString(3); - "tag price"
```

ובסיבוב השני (MoveToNext())

```
int id = cursor.getInt(0); - 2
String name = cursor.getString(1); - "david"
String author = cursor.getString(2); - "king"
String price = cursor.getString(3); - "770"
```

בסוף כל איטרציה של הלולאה הנתונים נכנסים לתוך מערך ועם מערך אנו כבר יודעים איך לעבוד עם ListView וכו' וכו'

עכשיו בפניה לנתונים בתוך המערך אנו כבר לא נשתמש ב position (באירוע לחיצה שאנו קושרים לכל השורות בListView) מכיוון שיותר הוא לא יהיה זה ל id של האובייקט. המפתח שאיתו נפנה מעתה לכל שורה יהיה הid של השורה עצמה, דבר זה בטוח אינו משתנה אפילו לאחר מחיקת והוספת נתונים לבסיס נתונים. אפשר להבין ע"פ הדוגמא הבאה:

0	1	2	3
_id	Name	author	Price
1	moshe	rabeyno	Tag price
2	david	king	770
7	moshe	rabeyno	Tag price
9	david	king	770

אמנם יש 4 שורות ורק ב2 השורות הראשונות אפשר לפנות לתא ע"י id+1 אך מהשורה השלישית כבר אין את היכולת. לכן ה id של הבסיס נתונים עצמו יהיה הדבר המדוייק ביותר לפנות איתו לשורה במערך.