Some important reminders about DSL for jetbrains/Exposed in Kotlin

DSL stands for *Domain Specific Language*

Dependencies

Make sure you are using the following dependencies and/or repositories in Maven (or equivalent in Gradle):

```
<repositories>
    <repository>
        <id>jcenter</id>
        <name>jcenter</name>
        <url>http://jcenter.bintray.com</url>
    </repository>
    <repository>
        <id>central</id>
        <name>central</name>
        <url>https://repo1.maven.org/maven2/</url>
    </repository>
</repositories>

<dependencies>
    <dependency>
        <groupId>org.jetbrains.exposed</groupId>
        <artifactId>exposed</artifactId>
        <version>0.17.3</version>
    </dependency>

    <dependency>
        <groupId>org.postgresql</groupId>
        <artifactId>postgresql</artifactId>
        <version>42.2.6</version>
    </dependency>
</dependencies>
```

Connection

To stablish a connection to an existing database ─in PostgreSQL─ use the following sintaxis:

```
Database.connect("jdbc:postgresql://di.rec.tion.IP:port/databaseName",
"org.postgresql.Driver", "user", "password")
```

Tables

A DB table is represented by an `object` inherited from `org.jetbrains.exposed.sql.Table` like this:

```
object TableName: Table() {
    val column1: Column<Type> = type(name)
}
```

Where:

- `TableName` has to be an existing relation in the database
- `column1` is the variable to be used in Kotlin (does not need to match the database's relation's column name)
- `Type` is the type to be used in Kotlin
- `type` is the type corresponding to the one used in the database ∈ {varchar(name, length), integer(name)}
- `name`, written within quotes (**""**) has to match the database's relation's column name

Datatypes

in Kotlin function	translates to DB type
`integer`	`INT`
`long`	`BIGINT`
`float`	`FLOAT`
`decimal`	`DECIMAL` *with scale and precision*
`bool`	`BOOLEAN`
`char`	`CHAR`
`varchar`	`VARCHAR` *with length*
`text`	`TEXT`
`enumeration`	`INT` *ordinal value*
`enumerationByName`	`VARCHAR`
`customEnumeration`	`see` *additional section*
`date`	`DATETIME`
`datetime`	`DATETIME`
`blob`	`BLOB`
`binary`	`VARBINARY` *with length*
`uuid`	`BINARY(16)`
`reference`	a foreign key

obtained from here