

Accessing a Database in Java

By:

Hafez Seleim

Data Storage

- Memory (Small (1 - 16 GB), Non-persistent)
- Text Files (inefficient, difficult to organize)
- Databases
 - Manages possibly huge quantities of data
 - Facilitates fast and easy access
 - Makes data integrity guarantees
 - Implementations (MySQL, Microsoft SQL Server, , Oracle,)

Database Organization

- A single database has multiple tables
- A table has multiple rows
- Each row has multiple columns
- Each column represents a different data category

Table of actors:

id	name	date of birth	gender
0	Tom Cruise	07/03/1962	M
1	Katie Holmes	12/18/1978	F
2	Chris Farley	02/15/1964	M
3	Tina Fey	05/18/1970	F

Database Operations (CRUD)

- **Create** data in a table
- **Read** data from a table
- **Update** data in a table
- **Delete** data in a table
- **Structured Query Language (SQL)** : language used to interface with a database, allows a user to perform CRUD operations on a particular database

Tables

```
create table Courses (  
  courseId char(5),  
  subjectId char(4) not null,  
  courseNumber integer,  
  title varchar(50) not null,  
  numOfCredits integer,  
  primary key (courseId)  
);
```

```
create table Enrollment (  
  ssn char(9),  
  courseId char(5),  
  dateRegistered date,  
  grade char(1),  
  primary key (ssn, courseId),  
  foreign key (ssn) references Students (ssn),  
  foreign key (courseId) references Courses  
(courseId)  
);
```

```
create table Students (  
  ssn char(9),  
  firstName varchar(25),  
  lastName varchar(25),  
  birthDate date,  
  phone char(11),  
  primary key (ssn)  
);
```

```
insert into Courses (courseId,  
  subjectId, courseNumber, title,  
  numOfCredits) values ('11113', 'CSCI',  
  '3720', 'Database Systems', 3);
```

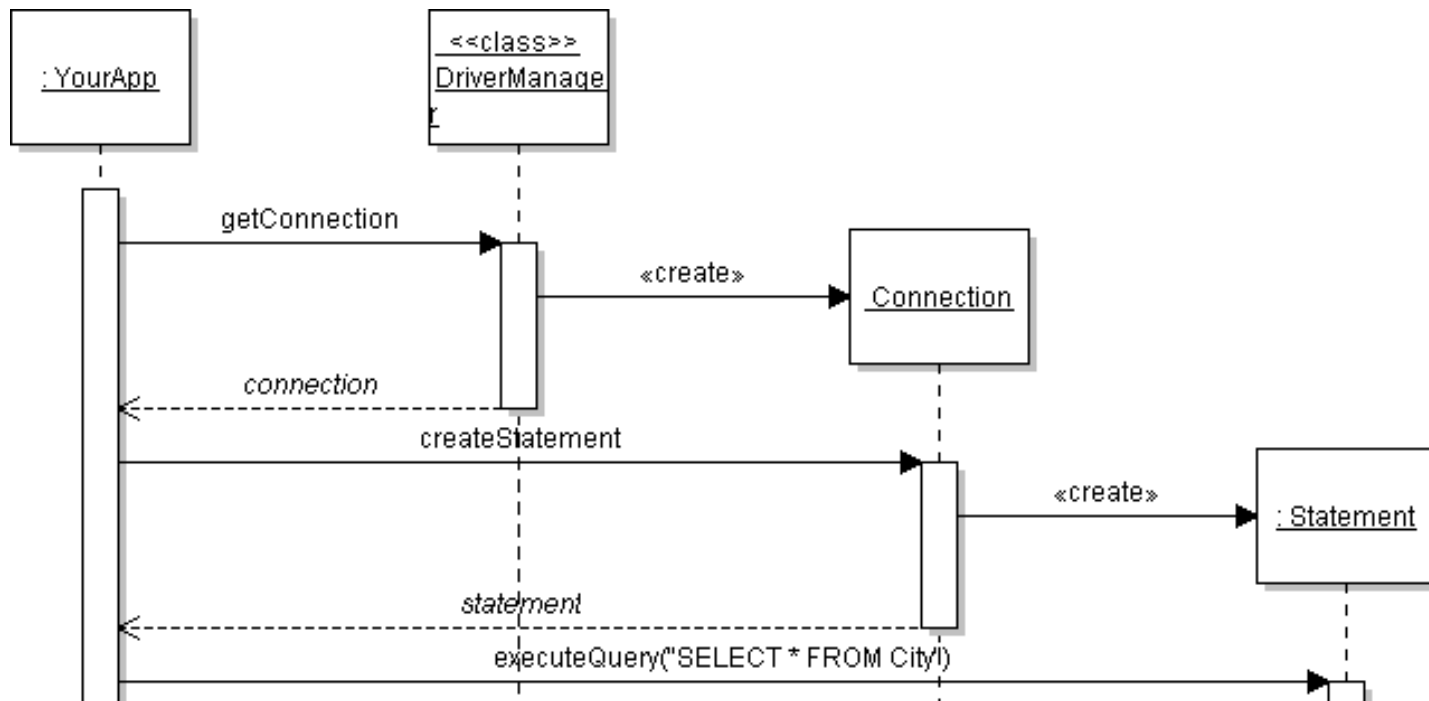
```
select * from Enrollment;
```

```
UPDATE Courses SET title = 'DB'  
  WHERE title = "Database  
  Systems";
```

- DELETE FROM Courses WHERE
 title = 'DB' ;

JDBC Overview

1. JDBC – **Java Database Connectivity**
2. Get a **Connection** to the database.
3. Create a **Statement** using the **Connection**.
4. **Execute** the **Statement** with SQL string.
5. Use the results.



JDBC Code

```
static final String URL = "jdbc:mysql://dbserver/world";
static final String USER = "student";
static final String PASSWORD = "secret";

// 1. Get a Connection to the database.
Connection connection =
    DriverManager.getConnection( URL, USER, PASSWORD );

// 2. Create a Statement
Statement statement = connection.createStatement();

// 3. Execute the Statement with SQL command.
ResultSet rs = statement.executeQuery("SELECT * FROM ...");

// 4. Use the Result.
while ( rs.next( ) ) {
    String name = rs.getString("name");
```

Database URL

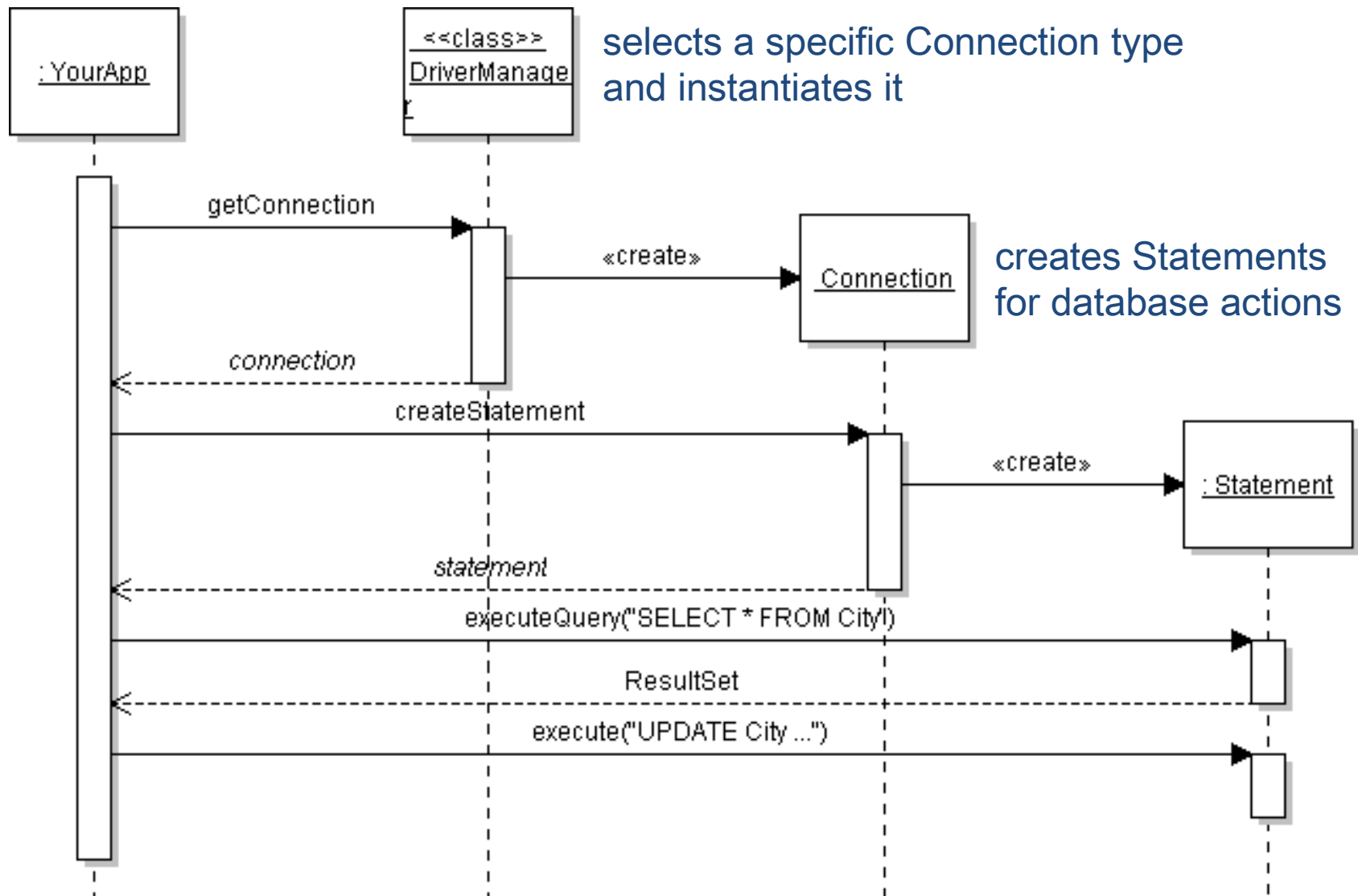
The format of a database URL is:

```
String DB_URL = "jdbc:mysql://dbserver:3306/world";
```

Protocol Sub-protocol Hostname Port DatabaseName

- ❑ **Port** is the TCP port number where the database server is listening.
 - **3306** is the **default port** for MySQL
- ❑ Use hostname "**localhost**" for the local machine.

JDBC Overview



Where is the Database Driver?

Driver is in a **JAR file**.

JAR file must be on the **CLASSPATH**.

Use one of these:

1. add as an *external jar file* to your IDE project
2. add the JAR to your CLASSPATH
`CLASSPATH = /my/path/mysql-connector.jar;.`
3. add JAR using the Java command line:
`java -cp /my/path/mysql-connector.jar ...`
4. Put JAR file in the **JRE/lib/ext** directory:
`C:/java/jre1.6.0/lib/ext/mysql-connector.jar`

How to Execute SQL Commands

The Statement interface defines many execute :

```
ResultSet rs =
```

```
    statement.executeQuery("SELECT ...");
```

- use for statements that return data values (SELECT)

```
int count =
```

```
    statement.executeUpdate("UPDATE ...");
```

- use for INSERT, UPDATE, and DELETE

```
boolean b =
```

```
    statement.execute("DROP TABLE test");
```

- use to execute any SQL statement(s)

Object-Relational Mapping

Purpose

- save object as a row in a database table
- create object using data from table
- save and create *associations* between objects

Design Goals

- separate O-R mapping service from our application
- localize the impact of change in database

Goal

- Applications need to save data to *persistent storage*.
- Persistent storage can be database, directory service, XML files, spreadsheet, ...
- For O-O programming, we'd like to save and retrieve *objects* to/from storage.

How to do Object Persistence

Choices for How to do Object Persistence?

1. write your own DAO using JDBC

2. Use an Object-Relational Mapping (ORM) Framework

- Hibernate, TopLink, MyBatis, Apache Cayenne

3. Use a Standard Persistence API.

- Java Persistence Architecture (JPA)

- standard used in JavaEE
- implemented by EclipseLink, Hibernate, OpenJPA

The Problem with Databases

Object-Relational Paradigm Mismatch

- Database stores data as *rows* in *tables*, which are not like objects.
- Objects have *associations* and *collections* databases have *relations* between tables.
- Objects are *unique*, database row data is *copied* each time you query it.

Mapping an Object

ku : Location

id = 101

name = "Kasetsart University"

address = "90 Pahonyotin ..."

object diagram

save()

LOCATIONS

id	name	address
101	Kasetsart University	90 Pahonyotin ...
102	Seacon Square	120 Srinakarin ...

4 Approaches to ORM

1. No ORM -- JDBC in my code.

No Layers! Put the JDBC right in your app code.

2. Do It Myself.

Write your own DAO using JDBC.

3. Use a Framework.

Hibernate, MyBatis, TopLink, or other.

4. Use a Standard.

Java Persistence Architecture (JPA) or Java Data Objects (JDO) provide a *standard API* that have *many implementations*.

Bean and DAO

```
public class StudentBean {  
    private Integer age;  
    private String name;  
    private Integer id;  
  
    public void setAge(Integer age) {  
        this.age = age;}  
  
    public Integer getAge() {  
        return age;}  
  
    public void setName(String name) {  
        this.name = name;}  
  
    public String getName() {  
        return name;}  
  
    public void setId(Integer id) {  
        this.id = id;}  
  
    public Integer getId() {  
        return id;  
    }  
}
```

```
public class StudentDAO {  
  
    public void create(String name, Integer age) {  
        String SQL = "insert into Student (name, age) values (?, ?)";  
        // send the command to the database  
        System.out.println("Created Record Name = " + name + " Age = " +  
age);  
    }  
  
    public Student getStudent(Integer id) {  
        String SQL = "select * from Student where id = ?";  
        //get the data and fill it in new object  
        return student;  
    }  
  
    public List<Student> listStudents() {  
        String SQL = "select * from Student";  
        return students;  
    }  
  
    public void delete(Integer id) {  
        String SQL = "delete from Student where id = ?";  
    }  
  
    public void update(Integer id, Integer age){  
        String SQL = "update Student set age = ? where id = ?";  
        System.out.println("Updated Record with ID = " + id );  
    }  
}
```

Persistence Frameworks

Hibernate - most popular open-source persistence framework for Java. **NHibernate** for .Net.

Uses POJOs and object-query language. Completely decouple Java from database.
Can **reverse engineer**.

MyBatis - simple, uses SQL maps. Database schema not transparent to Java code.

Cayenne - Apache project, has GUI modeler that eliminates need to write xml. Can **reverse engineer** database or generate database schema & Java code.

TopLink (Oracle)

Torque (Apache DB)