Lecture 12
Java SE Database Programming
Cristian Toma – Business Card

Cristian Toma
IT&C Security Master

Dorobantilor Ave., No. 15-17
010572 Bucharest - Romania
http://ism.ase.ro

Cristian.toma@ie.ase.ro
T +40 21 319 19 00 - 310
F +40 21 319 19 00
Agenda for Lecture 12

1. JDBC, RDBMS & NoSQL Concepts
2. JDBC & NoSQL Programming
3. Exchange Ideas
1. JDBC Concepts

JDBC Concepts:

Java Database Connectivity (JDBC) is an Application Programming Interface (API) used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, MySQL and SQL Server (even SQLite). JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows Java program to execute SQL statement and retrieve result from database.

What's new in JDBC 4.0

JDBC 4.0 is new and advance specification of JDBC. It provides the following advance features:

- Connection Management
- Auto loading of Driver Interface.
- Better exception handling
- Support for large object
- Annotation in SQL query.
1. **JDBC Concepts**

**JDBC Driver Types:**

JDBC Driver is required to process SQL requests and generate result. The following are the different types of driver available in JDBC.

- **Type-1 Driver** or JDBC-ODBC bridge
- **Type-2 Driver** or Native API Partly Java Driver
- **Type-3 Driver** or Network Protocol Driver
- **Type-4 Driver** or Thin Driver

- [http://www.studytionight.com/java/introduction-to-jdbc](http://www.studytionight.com/java/introduction-to-jdbc)
1. JDBC Concepts

**JDBC Driver 1:**

**JDBC-ODBC bridge**

**Type-1 Driver** act as a bridge between JDBC and other database connectivity mechanism (ODBC). This driver converts JDBC calls into ODBC calls and redirects the request to the ODBC driver.

**Advantage**
- Easy to use
- Allow easy connectivity to all database supported by the ODBC Driver.

**Disadvantage**
- Slow execution time
- Dependent on ODBC Driver.
- Uses Java Native Interface (JNI) to make ODBC call.

http://www.studytonight.com/java/introduction-to-jdbc
1. JDBC Concepts

JDBC Driver 2:

Native API Driver

This type of driver make use of Java Native Interface (JNI) call on database specific native client API. These native client API are usually written in C and C++.

Advantage
- faster as compared to Type-1 Driver
- Contains additional features.

Disadvantage
- Requires native library
- Increased cost of Application

http://www.studytonight.com/java/introduction-to-jdbc
1. JDBC Concepts

**JDBC Driver 3:**

**Network Protocol Driver**

This driver translates the JDBC calls into a database server independent and middleware server-specific calls. Middleware server further translates JDBC calls into database-specific calls.

---

**Advantage**

- Does not require any native library to be installed.
- Database independence.
- Provide facility to switch over from one database to another database.

**Disadvantage**

- Slow due to increase number of network calls.

http://www.studytonight.com/java/introduction-to-jdbc
1. **JDBC Concepts**

**JDBC Driver 4:**

**Thin Driver**

This is a driver called Pure Java Driver because it interacts directly with the database. It does not require any native database library, which is why it is also known as Thin Driver.

**Advantage**

- Does not require any native library.
- Does not require any Middleware server.
- Better performance than other drivers.

**Disadvantage**

- Slow due to the increased number of network calls.

[Diagram showing the interaction between Java App, JDBC Call, Type-4 Driver, and Database Server]

[Source: http://www.studytonight.com/java/introduction-to-jdbc]
1. JDBC Concepts

**JDBC API:**

**java.sql package**

This package includes classes and interfaces to perform almost all JDBC operations, such as creating and executing SQL queries.

**Important classes and interface of java.sql package**

<table>
<thead>
<tr>
<th>classes/interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.sql.BLOB</td>
<td>Provide support for BLOB(Binary Large Object) SQL type.</td>
</tr>
<tr>
<td>java.sql.Connection</td>
<td>Creates a connection with a specific database.</td>
</tr>
<tr>
<td>java.sql.CallableStatement</td>
<td>Execute stored procedures.</td>
</tr>
<tr>
<td>java.sql.CLOB</td>
<td>Provide support for CLOB(Character Large Object) SQL type.</td>
</tr>
<tr>
<td>java.sql.Date</td>
<td>Provide support for Date SQL type.</td>
</tr>
<tr>
<td>java.sql.Driver</td>
<td>Create an instance of a driver with the DriverManager.</td>
</tr>
<tr>
<td>java.sql.DriverManager</td>
<td>This class manages database drivers.</td>
</tr>
<tr>
<td>java.sql.PreparedStatement</td>
<td>Used to create and execute parameterized queries.</td>
</tr>
<tr>
<td>java.sql.ResultSet</td>
<td>It is an interface that provides methods to access the result row-by-row.</td>
</tr>
<tr>
<td>java.sql.Savepoint</td>
<td>Specify savepoint in a transaction.</td>
</tr>
<tr>
<td>java.sql.SQLException</td>
<td>Encapsulate all JDBC-related exceptions.</td>
</tr>
<tr>
<td>java.sql.Statement</td>
<td>This interface is used to execute SQL statements.</td>
</tr>
</tbody>
</table>
1. JDBC Concepts

**JDBC API:**

**jdbc.sql package**

This package is also known as JDBC extension API. It provides classes and interface to access server-side data.

<table>
<thead>
<tr>
<th>classes/interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>javax.sql.ConnectionEvent</code></td>
<td>Provide information about occurrence of event.</td>
</tr>
<tr>
<td><code>javax.sql.ConnectionEventListener</code></td>
<td>Used to register event generated by <code>PooledConnection</code> object.</td>
</tr>
<tr>
<td><code>javax.sql.DataSource</code></td>
<td>Represent the <code>DataSource</code> interface used in an application.</td>
</tr>
<tr>
<td><code>javax.sql.PooledConnection</code></td>
<td>provide object to manage connection pools.</td>
</tr>
</tbody>
</table>
1. JDBC Concepts

**JDBC Steps:**

**Steps to connect a Java Application to Database**

The following 5 steps are the basic steps involve in connecting a Java application with Database using JDBC.

1. Register the Driver
2. Create a Connection
3. Create SQL Statement
4. Execute SQL Statement
5. Closing the connection

http://www.studytonight.com/java/introduction-to-jdbc
1. JDBC Concepts

1 - JDBC Steps:

**Register the Driver**

`Class.forName()` is used to load the driver class explicitly.

Example to register with JDBC-ODBC Driver

```java
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

**Create a Connection**

`getConnection()` method of `DriverManager` class is used to create a connection.

**Syntax**

```java
getConnection(String url)
getConnection(String url, String username, String password)
getConnection(String url, Properties info)
```

**Example establish connection with Oracle Driver**

```java
Connection con = DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE","username","password");
```

1. JDBC Concepts

2 - JDBC Steps:

Create SQL Statement

createStatement() method is invoked on current Connection object to create a SQL Statement.

Syntax

public Statement createStatement() throws SQLException

Example to create a SQL statement

Statement s = con.createStatement();
1. **JDBC Concepts**

3 - **JDBC Steps:**

**Execute SQL Statement**

The `executeQuery()` method of the `Statement` interface is used to execute SQL statements.

**Syntax**

```
public ResultSet executeQuery(String query) throws SQLException
```

**Example to execute a SQL statement**

```java
ResultSet rs = s.executeQuery("select * from user");
while (rs.next())
{
    System.out.println(rs.getString(1) + " " + rs.getString(2));
}
```
1. **JDBC Concepts**

4 - **JDBC Steps:**

**Closing the connection**

After executing SQL statement you need to close the connection and release the session. The `close()` method of `Connection` interface is used to close the connection.

**Syntax**

```java
public void close() throws SQLException
```

**Example of closing a connection**

```java
con.close();
```
1. NoSQL Concepts

**MongoDB** is an open-source document database and leading NoSQL database. MongoDB is written in C++. MongoDB is a cross-platform, document oriented database that provides, high performance, high availability, and easy scalability. MongoDB works on concept of collection and document.

**Database**

Database is a physical container for collections. Each database gets its own set of files on the file system. A single MongoDB server typically has multiple databases.

**Collection**

Collection is a group of MongoDB documents. It is the equivalent of an RDBMS table. A collection exists within a single database. Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose.

**Document**

A document is a set of key-value pairs. Documents have dynamic schema. Dynamic schema means that documents in the same collection do not need to have the same set of fields or structure, and common fields in a collection's documents may hold different types of data.
1. **NoSQL Concepts**

The following table shows the relationship of RDBMS terminology with MongoDB.

<table>
<thead>
<tr>
<th>RDBMS</th>
<th>MongoDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Database</td>
</tr>
<tr>
<td>Table</td>
<td>Collection</td>
</tr>
<tr>
<td>Tuple/Row</td>
<td>Document</td>
</tr>
<tr>
<td>column</td>
<td>Field</td>
</tr>
<tr>
<td>Table Join</td>
<td>Embedded Documents</td>
</tr>
<tr>
<td>Primary Key</td>
<td>Primary Key (Default key _id provided by mongodb itself)</td>
</tr>
</tbody>
</table>

**Database Server and Client**

<table>
<thead>
<tr>
<th>Database Server and Client</th>
<th>mongod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mysql/Oracle</td>
<td></td>
</tr>
<tr>
<td>mysql/sqlplus</td>
<td>mongo</td>
</tr>
</tbody>
</table>

https://www.tutorialspoint.com/mongodb/index.htm
Sample Document

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

```json
{
    _id: ObjectId('7df78ad8902c'),
    title: 'MongoDB Overview',
    description: 'MongoDB is no sql database',
    by: 'tutorials point',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100,
    comments: [
        { user: 'user1', message: 'My first comment', dateCreated: new Date(2011,1,20,2,15), like: 0 },
        { user: 'user2', message: 'My second comments', dateCreated: new Date(2011,1,25,7,45), like: 5 }
    ]
}
```

_id is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.
1. NoSQL Concepts

Any relational database has a typical schema design that shows number of tables and the relationship between these tables. While in MongoDB, there is no concept of relationship.

Advantages of MongoDB over RDBMS

• Schema less – MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
• Structure of a single object is clear.
• No complex joins.
• Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
• Tuning.
• Ease of scale-out – MongoDB is easy to scale.
• Conversion/mapping of application objects to database objects not needed.
• Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

• Document Oriented Storage – Data is stored in the form of JSON style documents.
• Index on any attribute
• Replication and high availability
• Rich queries
• Fast in-place updates
• Professional support by MongoDB

Where to Use MongoDB?

• Big Data
• Content Management and Delivery
• Mobile and Social Infrastructure
• User Data Management
• Data Hub

https://www.tutorialspoint.com/mongodb/index.htm
1. NoSQL Concepts

Install MongoDB in Ubuntu

```bash
$ sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 0C49F3730359A14518585931BC711F9BA15703C6

$ echo "deb http://repo.mongodb.org/apt/ubuntu trusty/mongodb-org/testing multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.4.list

$ sudo apt-get update

$ sudo apt-get install -y mongodb

$ sudo service mongod start

$ sudo service mongod stop

### $ telnet localhost 27017
$ sudo find / -name mongo

$ mongo
> db.stats()
```
1. NoSQL Concepts – Data Modelling

Data in MongoDB has a flexible schema. Documents in the same collection. They do not need to have the same set of fields or structure, and common fields in a collection’s documents may hold different types of data.

Some considerations while designing Schema in MongoDB

• Design your schema according to user requirements.

• Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).

• Duplicate the data (but limited) because disk space is cheap as compare to compute time.

• Do joins while write, not on read.

• Optimize your schema for most frequent use cases.
• Do complex aggregation in the schema.
Data Modelling Example
Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.

• Every post has the unique title, description and URL.
• Every post can have one or more tags.
• Every post has the name of its publisher and total number of likes.
• Every post has comments given by users along with their name, message, data-time and likes.
• On each post, there can be zero or more comments.

In RDBMS schema, design for above requirements will have minimum three tables.
1. NoSQL Hands-On

The use Command – CREATE DATABASE

MongoDB `use DATABASE_NAME` is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

Syntax
Basic syntax of `use DATABASE` statement is as follows – `use DATABASE_NAME`

Example
If you want to create a database with name `<mydb>`, then `use DATABASE` statement would be as follows –

>`use mydb`

switched to db mydb

To check your currently selected database, use the command `db`

>`db mydb`

If you want to check your databases list, use the command `show dbs`.

>`show dbs`

local 0.78125GB
test 0.23012GB

Your created database (mydb) is not present in list. To display database, you need to insert at least one document into it.

>`db.movie.insert({"name":"tutorials point"})`

>`show dbs`

local 0.78125GB
mydb 0.23012GB
test 0.23012GB

In MongoDB default database is test. If you didn't create any database, then collections will be stored in test database.
The dropDatabase() Method – DELETE DATABASE

MongoDB `db.dropDatabase()` command is used to drop an existing database.

Syntax

Basic syntax of `dropDatabase()` command is as follows –
```
db.dropDatabase()
```
This will delete the selected database. If you have not selected any database, then it will delete the default 'test' database.

Example

First, check the list of available databases by using the command, `show dbs`.
```
>show dbs
local 0.78125GB
mydb 0.23012GB
test 0.23012GB
```
If you want to delete new database `<mydb>`, then `dropDatabase()` command would be as follows –
```
>use mydb
switched to db mydb
>db.dropDatabase()
>{ "dropped" : "mydb", "ok" : 1 }
```
Now check list of databases.
```
>show dbs
local 0.78125GB
test 0.23012GB
```
The `createCollection()` Method – CREATE COLLECTION

**“TABLE”**

MongoDB `db.createCollection(name, options)` is used to create a collection. In the command, `name` is the name of the collection to be created. **Options** is a document and is used to specify configuration of the collection.

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>capped</code></td>
<td>Boolean</td>
<td>(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. <strong>If you specify true, you need to specify size parameter also.</strong></td>
</tr>
<tr>
<td><code>autoIndexId</code></td>
<td>Boolean</td>
<td>(Optional) If true, automatically create index on <code>_id</code> field. Default value is false.</td>
</tr>
<tr>
<td><code>size</code></td>
<td>number</td>
<td>(Optional) Specifies a maximum size in bytes for a capped collection. <strong>If capped is true, then you need to specify this field also.</strong></td>
</tr>
<tr>
<td><code>max</code></td>
<td>number</td>
<td>(Optional) Specifies the maximum number of documents allowed in the capped collection.</td>
</tr>
</tbody>
</table>

While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.
The `createCollection()` Method – CREATE COLLECTION <-> “TABLE”

Examples
Basic syntax of `createCollection()` method without options is as follows –
```standard
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 }
```
You can check the created collection by using the command `show collections`.
```standard
>show collections
mycollection
```

The following example shows the syntax of `createCollection()` method with few important options –
```standard
>db.createCollection("mycol", { capped : true, autoIndexId : true, size : 6142800, max : 10000 } )
{ "ok" : 1 }
```

In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.
```standard
>db.tutorialspoint.insert({"name" : "tutorialspoint"})
>show collections
mycol
```

https://www.tutorialspoint.com/mongodb/index.htm
1. NoSQL Hands-On

The drop() Method – DELETE COLLECTION <-> “TABLE”

MongoDB's `db.collection.drop()` is used to drop a collection from the database.

Syntax

Basic syntax of `drop()` command is as follows –

```
DBNAME.COLLECTION_NAME.drop()
```

Example

First, check the available collections into your database `mydb`.

```
> use mydb
switched to db mydb
> show collections
mycol
mycollection
system.indexes
tutorialspoint
```

Now drop the collection with the name `mycollection`.

```
> db.mycollection.drop()
true
```

Again check the list of collections into database.

```
> show collections
mycol
system.indexes
tutorialspoint
```

`drop()` method will return true, if the selected collection is dropped successfully, otherwise it will return false.
MongoDB supports many datatypes. Some of them are –

- **String** – This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** – This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** – This type is used to store a boolean (true/ false) value.
- **Double** – This type is used to store floating point values.
- **Min/ Max keys** – This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** – This type is used to store arrays or list or multiple values into one key.
- **Timestamp** – ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** – This datatype is used for embedded documents.
- **Null** – This type is used to store a Null value.
- **Symbol** – This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** – This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** – This datatype is used to store the document’s ID.
- **Binary data** – This datatype is used to store binary data.
- **Code** – This datatype is used to store JavaScript code into the document.
- **Regular expression** – This datatype is used to store regular expression.

https://www.tutorialspoint.com/mongodb/index.htm
The insert() Method – INSERT DOCUMENT <-> “TUPLE/ROW”
To insert data into MongoDB collection, you need to use MongoDB's `insert()` or `save()` method.

**Syntax**
The basic syntax of `insert()` command is as follows –
> `db.COLLECTION_NAME.insert(document)`

**Example**
> `db.mycoll.insert({  _id: ObjectId('7df78ad8902c'),  title: 'MongoDB Overview',  description: 'MongoDB is no sql database',  by: 'tutorials point',  url: 'http://www.tutorialspoint.com',  tags: ['mongodb', 'database', 'NoSQL'],  likes: 100 })`

Here `mycol` is our collection name, as created in the previous chapter. If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

In the inserted document, if we don't specify the `_id` parameter, then MongoDB assigns a unique ObjectId for this document. `_id` is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows –
> `_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incremeneter)`

To insert multiple documents in a single query, you can pass an array of documents in `insert()` command.
1. NoSQL Hands-On

The insert() Method – INSERT DOCUMENT <-> “TUPLE/ROW”

Example

```javascript
> db.post.insert([
  {
    title: 'MongoDB Overview',
    description: 'MongoDB is no sql database',
    by: 'tutorials point',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 100
  },
  {
    title: 'NoSQL Database',
    description: 'NoSQL database doesn't have tables',
    by: 'tutorials point',
    url: 'http://www.tutorialspoint.com',
    tags: ['mongodb', 'database', 'NoSQL'],
    likes: 20,
    comments: [
      {
        user: 'user1',
        message: 'My first comment',
        dateCreated: new Date(2013,11,10,2,35),
        like: 0
      }
    ]
  }
])
```

To insert the document you can use `db.post.save(document)` also. If you don't specify `_id` in the document then `save()` method will work same as `insert()` method. If you specify `_id` then it will replace whole data of document containing `_id` as specified in save() method.
1. NoSQL Hands-On

The find() Method – QUERY DOCUMENT <-> "TUPLE/ROW"

To query data from MongoDB collection, you need to use MongoDB's find() method.

Syntax
The basic syntax of find() method is as follows –
>db.COLLECTION_NAME.find()

find() method will display all the documents in a non-structured way.

The pretty() Method
To display the results in a formatted way, you can use pretty() method.

Syntax
>db.mycol.find().pretty()

Example
>db.mycol.find().pretty()
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials point",
  "url": "http://www.tutorialspoint.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}

Apart from find() method, there is findOne() method, that returns only one document.

https://www.tutorialspoint.com/mongodb/index.htm
The find() Method – QUERY DOCUMENT <-> “TUPLE/ROW”

RDBMS Where Clause Equivalents in MongoDB
To query the document on the basis of some condition, you can use following operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
<th>Example</th>
<th>RDBMS Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td><code>{&lt;key&gt;:&lt;value&gt;}</code></td>
<td>db.mycol.find({&quot;by&quot;:&quot;tutorial point&quot;}).pretty()</td>
<td>where by = 'tutorial point'</td>
</tr>
<tr>
<td>Less Than</td>
<td><code>{&lt;key&gt;:{$lt:&lt;value&gt;}}</code></td>
<td>db.mycol.find({&quot;likes&quot;:{$lt:50}}).pretty()</td>
<td>where likes &lt; 50</td>
</tr>
<tr>
<td>Less Than Equals</td>
<td><code>{&lt;key&gt;:{$lte:&lt;value&gt;}}</code></td>
<td>db.mycol.find({&quot;likes&quot;:{$lte:50}}).pretty()</td>
<td>where likes &lt;= 50</td>
</tr>
<tr>
<td>Greater Than</td>
<td><code>{&lt;key&gt;:{$gt:&lt;value&gt;}}</code></td>
<td>db.mycol.find({&quot;likes&quot;:{$gt:50}}).pretty()</td>
<td>where likes &gt; 50</td>
</tr>
<tr>
<td>Greater Than Equals</td>
<td><code>{&lt;key&gt;:{$gte:&lt;value&gt;}}</code></td>
<td>db.mycol.find({&quot;likes&quot;:{$gte:50}}).pretty()</td>
<td>where likes &gt;= 50</td>
</tr>
<tr>
<td>Not Equals</td>
<td><code>{&lt;key&gt;:{$ne:&lt;value&gt;}}</code></td>
<td>db.mycol.find({&quot;likes&quot;:{$ne:50}}).pretty()</td>
<td>where likes != 50</td>
</tr>
</tbody>
</table>
1. NoSQL Hands-On

The find() Method – QUERY DOCUMENT <-> “TUPLE/ROW” with AND

Syntax
In the find() method, if you pass multiple keys by separating them by 'comma', then MongoDB treats it as AND condition. Following is the basic syntax of AND –

```javascript
> db.mycol.find( {$and: [{key1: value1}, {key2:value2}] } ).pretty()
```

Example
Following example will show all the tutorials written by 'tutorials point' and whose title is 'MongoDB Overview'.

```javascript
> db.mycol.find( {$and:[{"by":"tutorials point"},{"title": "MongoDB Overview"}]}) ).pretty()
```

```javascript
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database", "by": "tutorials point", "url":
  "http://www.tutorialspoint.com", "tags": ["mongodb", "database", "NoSQL"], "likes": "100"
}
```

For the above given example, equivalent where clause will be 'where by = 'tutorials point' AND title = 'MongoDB Overview' '. You can pass any number of key, value pairs in find clause.

https://www.tutorialspoint.com/mongodb/index.htm
1. NoSQL Hands-On

The find() Method – QUERY DOCUMENT <-> “TUPLE/ROW” with OR

Syntax
To query documents based on the OR condition, you need to use $or keyword. Following is the basic syntax of OR –

```
> db.mycol.find({ $or: [ {key1: value1}, {key2:value2} ] } ).pretty()
```

Example
Following example will show all the tutorials written by 'tutorials point' or whose title is 'MongoDB Overview'.

```
> db.mycol.find({ $or: [ {"by": "tutorials point"}, {"title": "MongoDB Overview"} ] } ).pretty()
{
  "_id": ObjectId(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials point",
  "url": "http://www.tutorialspoint.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
```
The find() Method – QUERY DOCUMENT <-> “TUPLE/ROW” with AND + OR Together

Example
The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is *where likes > 10 AND (by = 'tutorials point' OR title = 'MongoDB Overview')*

```javascript
> db.mycol.find( {
  "likes": {
    "$gt": 10
  },
  $or: [{"by": "tutorials point"}, {
    "title": "MongoDB Overview"
  }]
}).pretty()
```

```json
{
  "_id": ObjectID(7df78ad8902c),
  "title": "MongoDB Overview",
  "description": "MongoDB is no sql database",
  "by": "tutorials point",
  "url": "http://www.tutorialspoint.com",
  "tags": ["mongodb", "database", "NoSQL"],
  "likes": "100"
}
```
1. NoSQL Hands-On

The update() Method - UPDATE DOCUMENT <-> “TUPLE/ROW”

MongoDB's update() and save() methods are used to update document into a collection. The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

The update() method updates the values in the existing document.

Syntax

The basic syntax of **update()** method is as follows –

>db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

Example

Consider the mycol collection has the following data.

```json
{"_id": ObjectId(5983548781331adf45ec5), "title": "MongoDB Overview"}
{"_id": ObjectId(5983548781331adf45ec6), "title": "NoSQL Overview"}
{"_id": ObjectId(5983548781331adf45ec7), "title": "Tutorials Point Overview"}
```

Following example will set the new title 'New MongoDB Tutorial' of the documents whose title is 'MongoDB Overview'.

```javascript
>db.mycol.update({'title': 'MongoDB Overview'}, {$set: {'title': 'New MongoDB Tutorial'}})
>db.mycol.find()
```

```json
{"_id": ObjectId(5983548781331adf45ec5), "title": "New MongoDB Tutorial"}
{"_id": ObjectId(5983548781331adf45ec6), "title": "NoSQL Overview"}
{"_id": ObjectId(5983548781331adf45ec7), "title": "Tutorials Point Overview"}
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```javascript
>db.mycol.update({'title': 'MongoDB Overview'}, {$set: {'title': 'New MongoDB Tutorial'}}, {multi: true})
```

https://www.tutorialspoint.com/mongodb/index.htm
1. NoSQL Hands-On

The **save()** Method – SAVE/UPDATE DOCUMENT <-> “TUPLE/ROW”

The **save()** method replaces the existing document with the new document passed in the save() method.

**Syntax**
The basic syntax of MongoDB **save()** method is shown below –

```
>db.COLLECTION_NAME.save({_id:ObjectId(),NEW_DATA})
```

**Example**

Following example will replace the document with the `_id` '5983548781331adf45ec7'.

```
>db.mycol.save( { "_id" : ObjectId('5983548781331adf45ec7'), "title" : "Tutorials Point New Topic" , "by" : "Tutorials Point" } )
```

```
>db.mycol.find()
{ "_id" : ObjectId('5983548781331adf45ec5'), "title" : "Tutorials Point New Topic" , "by" : "Tutorials Point" } { "_id" : ObjectId('5983548781331adf45ec6'), "title" : "NoSQL Overview" } { "_id" : ObjectId('5983548781331adf45ec7'), "title" : "Tutorials Point Overview" } >
```
1. NoSQL Hands-On

The remove() Method – DELETE DOCUMENT <-> “TUPLE/ROW”

MongoDB's `remove()` method is used to remove a document from the collection. `remove()` method accepts two parameters. One is deletion criteria and second is justOne flag.

- **deletion criteria** – (Optional) deletion criteria according to documents will be removed.
- **justOne** – (Optional) if set to true or 1, then remove only one document.

**Syntax**

Basic syntax of `remove()` method is as follows –

```javascript
>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)
```

**Example**

Consider the `mycol` collection has the following data.

```json
{
    "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"
}
{
    "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"
}
{
    "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"
}
```

Following example will remove all the documents whose title is 'MongoDB Overview'.

```javascript
>db.mycol.remove({'title':'MongoDB Overview'})
>db.mycol.find()
```

```json
{
    "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"
}
{
    "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"
}
```
The find() Method – PROJECTION of COLLECTION <-> “TABLE”

In MongoDB, projection means selecting only the necessary data rather than selecting whole of the data of a document. If a document has 5 fields and you need to show only 3, then select only 3 fields from them.

MongoDB's find() method, exposed in QUERY DOCUMENT section, accepts second optional parameter that is list of fields that you want to retrieve. In MongoDB, when you execute find() method, then it displays all fields of a document. To limit this, you need to set a list of fields with value 1 or 0. 1 is used to show the field while 0 is used to hide the fields.

Syntax
The basic syntax of find() method with projection is as follows –
>db.COLLECTION_NAME.find( {}, {KEY:1} )

Example
Consider the collection mycol has the following data –

```
{ "_id" : ObjectId(5983548781331adf45ec5), "title":"MongoDB Overview"}
{ "_id" : ObjectId(5983548781331adf45ec6), "title":"NoSQL Overview"}
{ "_id" : ObjectId(5983548781331adf45ec7), "title":"Tutorials Point Overview"}
```

Following example will display the title of the document while querying the document.

```
>db.mycol.find( {}, {"title":1, _id:0} )
```

```
{"title":"MongoDB Overview"}
{"title":"NoSQL Overview"}
{"title":"Tutorials Point Overview"}
```

Please note _id field is always displayed while executing find() method, if you don't want this field, then you need to set it as 0.

https://www.tutorialspoint.com/mongodb/index.htm
Section Conclusion

Fact: Java is suitable for Databases

In few samples it is simple to understand: JDBC API, NoSQL programming and remember databases concepts.
Java SQLite – SQL Insert, Select, Update, Delete + NoSQL - MongoDB

Java SQLite/MySQL JDBC & NoSQL-MongoDB Programming
2. JDBC Programming

```java
// JDBC driver name and database URL
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost:3306/";

public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    try{
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");
        //Class.forName("org.sqlite.JDBC");

        //STEP 3: Open a connection
        System.out.println("Connecting to database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
        //conn = DriverManager.getConnection("jdbc:mysql:test.db");
        //conn.setAutoCommit(false);

        //STEP 4: Execute a query
        System.out.println("Creating database...");
        stmt = conn.createStatement();

        String sqlDrop = "DROP DATABASE STUDENTS";
        stmt.executeUpdate(sqlDrop);

        String sql = "CREATE DATABASE STUDENTS";
        stmt.executeUpdate(sql);
        System.out.println("Database created successfully...");
    }catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
    }catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    }finally{
        //Finally block used to close resources
        try{
            if(stmt!=null) stmt.close();
        }catch(SQLException se2){
            // nothing we can do
        }try{
            if(conn!=null) conn.close();
        }catch(SQLException se){
            // nothing we can do
        }
    }
}
```
2. JDBC Programming

```java
public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    try {
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //STEP 3: Open a connection
        System.out.println("Connecting to a selected database...");
        conn = DriverManager.getConnection("DB URL, USER, PASS");
        System.out.println("Connected database successfully...");

        //STEP 4: Execute a query
        System.out.println("Creating table in given database...");
        stmt = conn.createStatement();

        String sql = "CREATE TABLE REGISTRATION " +
                     "(id INTEGER not NULL, " +
                     "first VARCHAR(255), " +
                     "last VARCHAR(255), " +
                     "age INTEGER, " +
                     "PRIMARY KEY (id))";

        stmt.executeUpdate(sql);
        System.out.println("Created table in given database...");
    } catch (SQLException se) {
        //Handle errors for JDBC
        se.printStackTrace();
    } catch (Exception e) {
        //Handle errors for Class.forName
        e.printStackTrace();
    } finally {
        //finally block used to close resources
        try {
            if(stmt!=null)
                conn.close();
        } catch (SQLException se) {
            // do nothing
            if(conn!=null)
                conn.close();
        }
    }
}
```
2. JDBC Programming

```java
// JDBC driver name and database URL
static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
static final String DB_URL = "jdbc:mysql://localhost:3306/STUDENTS";

// Database credentials
static final String USER = "root"; //username
static final String PASS = "stud"; //password

public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    try{
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //STEP 3: Open a connection
        System.out.println("Connecting to selected database...");
        conn = DriverManager.getConnection(DB_URL, USER, PASS);
        System.out.println("Connected database successfully...");

        //STEP 4: Execute a query
        System.out.println("Inserting records into the table...");
        stmt = conn.createStatement();
        String sql = "INSERT INTO REGISTRATION " + "+ VALUES (100, 'Zara', 'Ali', 18)";
        stmt.executeUpdate(sql);
        sql = "INSERT INTO REGISTRATION " + "+ VALUES (101, 'Mahnaiz', 'Fatma', 25)";
        stmt.executeUpdate(sql);
        sql = "INSERT INTO REGISTRATION " + "+ VALUES (102, 'Zaid', 'Khan', 30)";
        stmt.executeUpdate(sql);
        sql = "INSERT INTO REGISTRATION " + "+ VALUES (103, 'Sumit', 'Mittal', 28)";
        stmt.executeUpdate(sql);
        System.out.println("Inserted records into the table...");
    }
    catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
    }
    catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    }
    finally{
        //finally block used to close resources
    }
```
2. JDBC Programming

```java
public static void main(String[] args) {
    Connection conn = null;
    Statement stmt = null;
    try {
        //STEP 2: Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        //STEP 3: Open a connection
        System.out.println("Connecting to a selected database...");
        conn = DriverManager.getConnection("DB URL", USER, PASS);
        System.out.println("Connected database successfully...");

        //STEP 4: Execute a query
        System.out.println("Creating statement...");
        stmt = conn.createStatement();

        String sql = "SELECT id, first, last, age FROM REGISTRATION";
        ResultSet rs = stmt.executeQuery(sql);
        //STEP 5: Extract data from result set
        while(rs.next()){
            //Retrieve by column name
            int id = rs.getInt("id");
            int age = rs.getInt("age");
            String first = rs.getString("first");
            String last = rs.getString("last");

            //Display values
            System.out.print("ID: " + id);
            System.out.print("Age: " + age);
            System.out.println("First: " + first);
            System.out.println("Last: " + last);
        }
        rs.close();
    } catch(SQLException se){
        //Handle errors for JDBC
        se.printStackTrace();
    } catch(Exception e){
        //Handle errors for Class.forName
        e.printStackTrace();
    }
    finally{
        //finally block used to close resources
        try{
            if(stmt!=null)
                conn.close();
        } catch(SQLException se){
        }
    }
}
```
Section Conclusions

Please review the JDBC API, NoSQL library and Database Programming.
Java Relational & NoSQL

Communicate & Exchange Ideas
Questions & Answers!

But wait...
There’s More!
Thanks!

Java SE – Java Standard Edition Programming
End of Lecture 12 – Database Programming