



# JDBC

MetaData and PreparedStatement

**MetaData**

**And**

**PreparedStatement**



# Agenda

---

1

**Exploring DatabaseMetaData**

2

**Exploring ResultSetMetaData**

3

**Using PreparedStatement Object**

# Objectives

At the end of this module, you will be able to:

- Analyze how to use the Metadata objects to retrieve more information about the database or the result set
- Create and execute a query using PreparedStatement object

# The DatabaseMetaData Object

- Metadata is data about data
- DatabaseMetaData is an interface to get comprehensive information about the database as a whole
- The Connection object can be used to get a DatabaseMetaData object
- Use the Connection.getMetaData() method to return a DatabaseMetaData object
- This object provides more than 100 methods to obtain information about the database

# The DatabaseMetaData Object

The following are some examples of DatabaseMetaData methods:

- ***getColumnPrivileges()***: Get a description of the access rights for a table's columns.
- ***getColumns()***: Get a description of table columns.
- ***getDatabaseProductName()***: Get the name of this database product.
- ***getDriverName()***: Get the name of this JDBC driver.
- ***storesLowerCaseIdentifiers()***: Does the database store mixed-case SQL identifiers in lower case?
- ***supportsAlterTableWithAddColumn()***: Is ALTER TABLE with add column supported?
- ***supportsFullOuterJoins()***: Are full nested outer joins supported?

# How to obtain Database Metadata?

1. To get the DatabaseMetaData Object

```
DatabaseMetaData dbmd = conn.getMetaData();
```

2. Use the object's methods to get the metadata

```
DatabaseMetaData dbmd = conn.getMetaData();  
String s1 = dbmd.getURL();  
String s2 = dbmd.getSQLKeywords();  
boolean b1 = dbmd.supportsTransactions();  
boolean b2 = dbmd.supportsSelectForUpdate();
```

# The DatabaseMetaData Object

- This interface is implemented by driver vendors to let users know the capabilities of a Database Management System (DBMS) in combination with the driver based on JDBC technology ("JDBC driver") that is used with it. Different relational DBMSs often support different features, implement features in different ways, and use different data types. In addition, a driver may implement a feature on top of what the DBMS offers. Information returned by methods in this interface applies to the capabilities of a particular driver and a particular DBMS working together.
- A user for this interface is commonly a tool that needs to discover how to deal with the underlying DBMS. This is especially true for applications that are intended to be used with more than one DBMS.

## **For example:**

- **getURL():** Returns the URL for the DBMS
- **getSQLKeywords():** Retrieves a comma-separated list of all of this database's SQL keywords that are NOT also SQL92 keywords.
- **supportsTransactions():** Retrieves whether this database supports transactions. If not, invoking the method commit is no use, and the isolation level is TRANSACTION\_NONE.
- **supportsSelectForUpdate():** Retrieves whether this database supports SELECT FOR UPDATE statements.



# Example on DatabaseMetaData object

```
import java.sql.*;
import java.io.*;
class MakeConnection3 {
    Connection conn;
    Statement stmt;
    String s1, s2;
    DatabaseMetaData dbmd;
    MakeConnection3() {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            conn=DriverManager.getConnection("Jdbc:Oracle:thin:@localhost:1521:
orcl","scott","tiger");
```

**contd..**

## Example on DatabaseMetadata (Contd.).

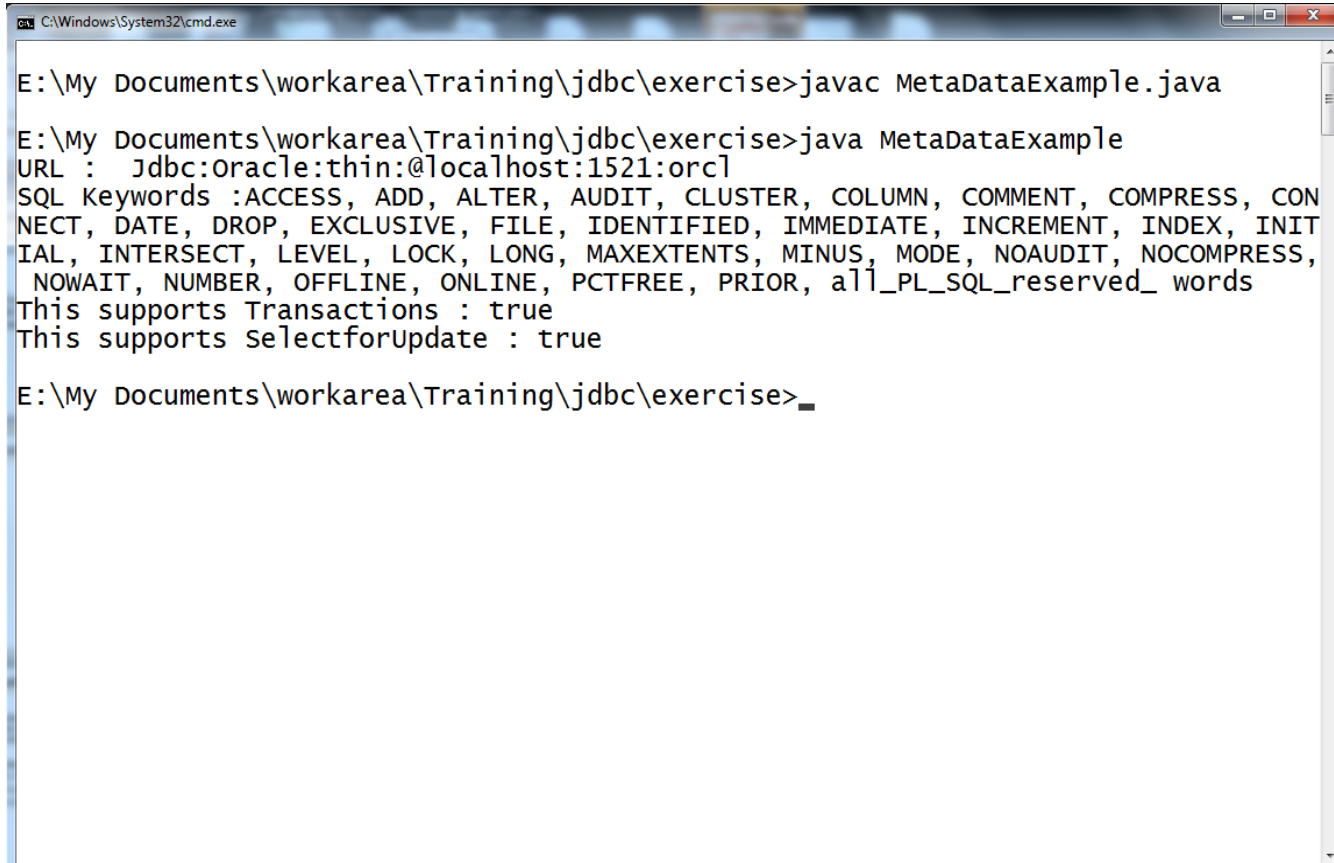
```
dbmd = conn.getMetaData();  
s1 = dbmd.getURL();  
s2 = dbmd.getSQLKeywords();  
boolean b1 = dbmd.supportsTransactions();  
boolean b2 = dbmd.supportsSelectForUpdate();  
System.out.println("URL : "+s1);  
System.out.println("SQL Keywords :"+s2);  
System.out.println("This supports Transactions : "+b1);  
System.out.println("This supports SelectforUpdate : "+b2);  
}
```

**contd..**

## Example on DatabaseMetadata (Contd.).

```
        catch(Exception e) {  
            e.printStackTrace();  
        }  
    }  
}  
  
public class MetaDataExample {  
    public static void main(String args[]) {  
        new MakeConnection3();  
    }  
}
```

# Example on Database Metadata (Contd.).



```
C:\Windows\System32\cmd.exe

E:\My Documents\workarea\Training\jdbc\exercise>javac MetaDataExample.java

E:\My Documents\workarea\Training\jdbc\exercise>java MetaDataExample
URL : jdbc:Oracle:thin:@localhost:1521:orcl
SQL Keywords :ACCESS, ADD, ALTER, AUDIT, CLUSTER, COLUMN, COMMENT, COMPRESS, CON
NECT, DATE, DROP, EXCLUSIVE, FILE, IDENTIFIED, IMMEDIATE, INCREMENT, INDEX, INIT
IAL, INTERSECT, LEVEL, LOCK, LONG, MAXEXTENTS, MINUS, MODE, NOAUDIT, NOCOMPRESS,
NOWAIT, NUMBER, OFFLINE, ONLINE, PCTFREE, PRIOR, all_PL_SQL_reserved_ words
This supports Transactions : true
This supports SelectforUpdate : true

E:\My Documents\workarea\Training\jdbc\exercise>_
```

# The ResultSetMetaData Object

- ResultSetMetaData is an interface which contains methods to get information about the types and properties of the columns in the ResultSet object
- ResultSetMetaData object provides metadata, including:
  - Number of columns in the result set
  - Column type
  - Column name

In JDBC, you use the `ResultSet.getMetaData()` method to return a `ResultSetMetaData` object, which describes the data coming back from a database query. This object can be used to find out about the types and properties of the columns in your `ResultSet`.

# How to obtain ResultSetMetadata?

1. To get the ResultSetMetadata object

```
ResultSetMetadata rsmd = rset.getMetaData();
```

2. Use the object's methods to get the metadata

```
ResultSetMetadata rsmd = rset.getMetaData();  
for (int i = 1; i <= rsmd.getColumnCount(); i++) {  
    String colname = rsmd.getColumnName(i);  
    int coltype = rsmd.getColumnType(i);  
    ...  
}
```

# Example on ResultSetMetaData

```
import java.sql.*;

class MakeConnection4{

    Connection conn;

    Statement stmt;

    ResultSet rs;

    ResultSetMetaData rsmd;

    MakeConnection4 () {

try{

        Class.forName("oracle.jdbc.driver.OracleDriver");

conn=DriverManager.getConnection("Jdbc:Oracle:thin:@localhost:1521:
orcl","scott","tiger");

        stmt=conn.createStatement();

        rs=stmt.executeQuery("Select * from emp");

    }

}
```

## Example on ResultSetMetaData(Contd.).

```
rsmd = rs.getMetaData();  
int noc = rsmd.getColumnCount();  
System.out.println("Number Of Columns : "+noc);
```

```
for(int i=1;i<=noc;i++) {  
    System.out.print("Column "+i+" ="+rsmd.getColumnName(i)+"; ");  
    System.out.print("Column Type ="+rsmd.getColumnType(i)+"; ");  
    System.out.println("Column Type Name="+rsmd.getColumnTypeName(i)+";");  
}
```

```
    }  
catch(Exception e){  
    e.printStackTrace();  
}  
  
}
```



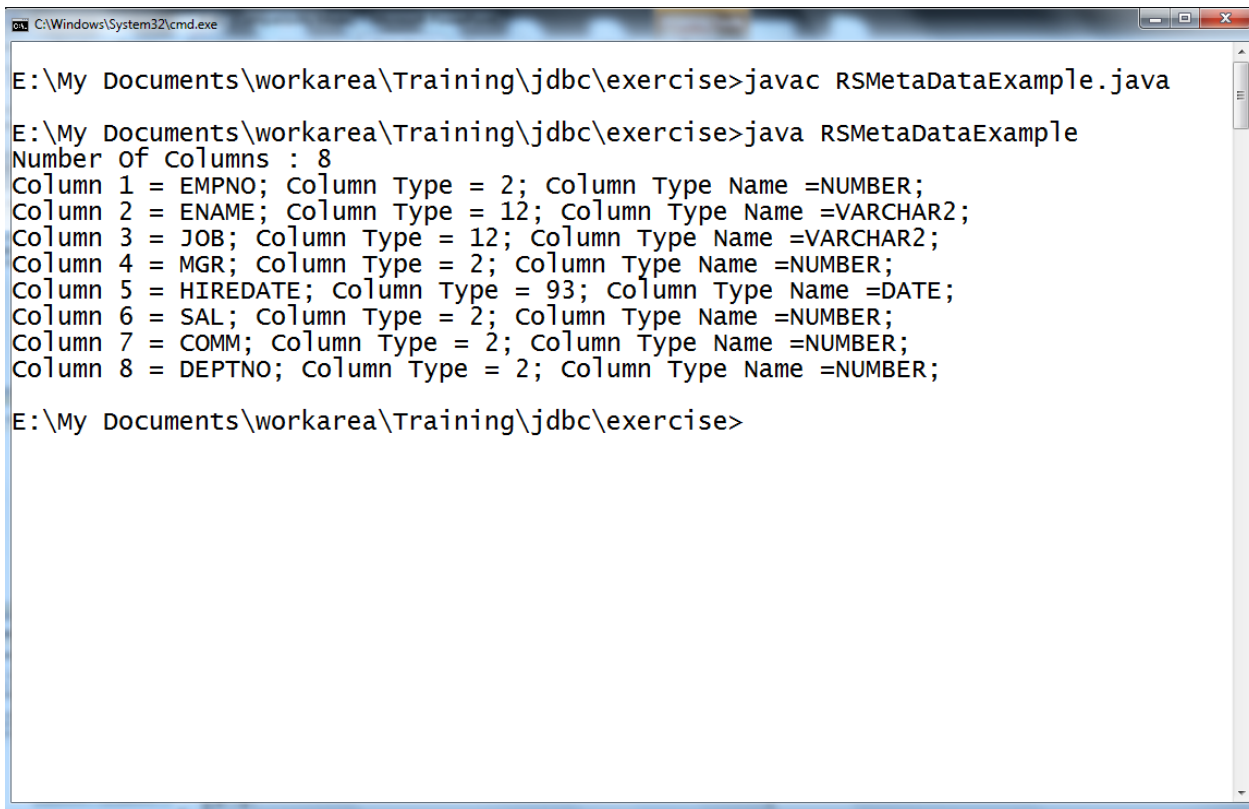
## Example on ResultSetMetaData (Contd.).

```
public class RSMetaDataExample {  
    public static void main(String args[]) {  
        new MakeConnection4();  
    }  
}
```

- The example shows how to use a `ResultSetMetaData` object to determine the following information about the `ResultSet`:
  - The number of columns in the `ResultSet`.
  - The name of each column
  - The American National Standards Institute (ANSI) SQL type for each column
- **java.sql.Types**

The `java.sql.Types` class defines constants that are used to identify ANSI SQL types. `ResultSetMetaData.getColumnType()` returns an integer value that corresponds to one of these constants.

# Example on ResultSetMetaData (Contd.).



```
C:\Windows\System32\cmd.exe

E:\My Documents\workarea\Training\jdbc\exercise>javac RSMetaDataExample.java

E:\My Documents\workarea\Training\jdbc\exercise>java RSMetaDataExample
Number Of Columns : 8
Column 1 = EMPNO; Column Type = 2; Column Type Name =NUMBER;
Column 2 = ENAME; Column Type = 12; Column Type Name =VARCHAR2;
Column 3 = JOB; Column Type = 12; Column Type Name =VARCHAR2;
Column 4 = MGR; Column Type = 2; Column Type Name =NUMBER;
Column 5 = HIREDATE; Column Type = 93; Column Type Name =DATE;
Column 6 = SAL; Column Type = 2; Column Type Name =NUMBER;
Column 7 = COMM; Column Type = 2; Column Type Name =NUMBER;
Column 8 = DEPTNO; Column Type = 2; Column Type Name =NUMBER;

E:\My Documents\workarea\Training\jdbc\exercise>
```

# Mapping Database Types to Java Types

- `ResultSet` maps database types to Java types.
- In many cases, you can get all the columns in your result set using the `getObject()` or `getString()` methods of `ResultSet`. For performance reasons, or because you want to perform complex calculations, it is sometimes important to have your data in a type that exactly matches the database column.

```
ResultSet rset = stmt.executeQuery  
    ("select ID, DATE_OF_JOIN, SUPERVISOR  
    from STUDENT");
```

```
int id = rset.getInt(1);  
Date rentaldate = rset.getDate(2);  
String status = rset.getString(3);
```

Col Name	Type
ID	NUMBER
DATE_OF_JOIN	DATE
SUPERVISOR	VARCHAR2

# Mapping Database Types to Java Types (Contd.).

SQL data type	Java data type	
	Simply mappable	Object mappable
CHARACTER		String
VARCHAR		String
LONGVARCHAR		String
NUMERIC		java.math.BigDecimal
DECIMAL		java.math.BigDecimal
BIT	boolean	Boolean
TINYINT	byte	Integer
SMALLINT	short	Integer
INTEGER	int	Integer
BIGINT	long	Long
REAL	float	Float
FLOAT	double	Double
DOUBLE PRECISION	double	Double
BINARY		byte[]
VARBINARY		byte[]
LONGVARBINARY		byte[]
DATE		java.sql.Date
TIME		java.sql.Time
TIMESTAMP		java.sql.Timestamp

# Quiz

1. Which of the following java type is mapped to the SQL data type **BIT**
  - a) String
  - b) boolean
  - c) Int
  - d) byte
  
2. What object is returned, when you invoke the getMetaData method on the Connection object
  - a) StatementMetaData
  - b) ResultSetMetaData
  - c) DatabaseMetaData
  - d) ConnectionMetaData

Answers :

1 : b

2 : c

# The PreparedStatement Object

- Using PreparedStatement in place of Statement interface will improve the performance of a JDBC program
- PreparedStatement is inherited from Statement; the difference is that a PreparedStatement holds precompiled SQL statements.
- If you execute a Statement object many times, its SQL statement is compiled each time. PreparedStatement is more efficient because its SQL statement is compiled only once, when you first prepare the PreparedStatement. After that, each time you execute the SQL statement in the PreparedStatement, the SQL statement does not have to be recompiled.
- Therefore, if you need to execute the same SQL statement several times within an application, it is more efficient to use PreparedStatement than Statement.

# The PreparedStatement Object

- A prepared statement can contain variables that you supply each time you execute the statement
- A PreparedStatement does not have to execute exactly the same query each time. You can specify parameters in the PreparedStatement SQL string and supply the actual values for these parameters when the statement is executed.
- The following slide shows how to supply parameters and execute a PreparedStatement.

# How to Create a PreparedStatement?

1. Register the driver and create the database connection
2. Create the prepared statement, identifying variables with a question mark (?)

```
PreparedStatement pstmt =  
conn.prepareStatement("update STUDENT  
set SUPERVISOR = ? where ID = ?");
```

```
PreparedStatement pstmt =  
conn.prepareStatement("select SUPERVISOR from  
STUDENT where ID = ?");
```

Once the connection object is obtained, the `prepareStatement` method is called on it to obtain the `PreparedStatement` object. However, in this case, while creating it, itself, the SQL statement is provided as a parameter to the method. The variable portions of the SQL statement are provided as a question mark (?) so that the values can be supplied dynamically before execution of the statement.



# How to execute PreparedStatement?

1. Supply values for the variables

```
pstmt.setXXX(index, valu);
```

## **Specifying Values for the Bind Variables**

You use the `PreparedStatement.setXXX()` methods to supply values for the variables in a prepared statement. There is one `setXXX()` method for each Java type: `setString()`, `setInt()`, and so on.

You must use the `setXXX()` method that is compatible with the SQL type of the variable. In the example on the slide, the first variable is updating a `VARCHAR` column, so we need to use `setString()` to supply a value for the variable. You can use `setObject()` with any variable type.

Each variable has an index. The index of the first variable in the prepared statement is 1, the index of the second is 2, and so on. If there is only one variable, its index is one. The index of a variable is passed to the `setXXX()` method.

# How to execute PreparedStatement?

## 2. Execute the statement

```
pstmt.executeQuery();  
pstmt.executeUpdate();
```

```
PreparedStatement pstmt =  
    conn.prepareStatement("update STUDENT  
    set SUPERVISOR = ? Where ID = ?");  
pstmt.setString(1, "Jeetendra");  
pstmt.setInt(2, id);  
pstmt.executeUpdate();
```

## **Closing a Prepared Statement**

If you close a prepared statement, you will have to prepare it again.

# Example 1 on PreparedStatement

/\* This class is executed in the following manner :

if you want to create the table, you will execute as `java JCreate table1` where `table1` is the name of the table. The table `table1` is created with the following columns `empid`, `empname`, `dept`, `joindate`, `salary`

\*/

```
import java.sql.*;
```

```
class JCreate {
```

```
    public static void main(String args[]) throws SQLException {
```

```
        JdbcCalls e = new JdbcCalls();
```

```
        e.create(args);
```

```
    }
```

```
}
```

# Example 1 on PreparedStatement(Contd.).

```
import java.sql.*;

class ConnectionClass {
    Connection con;

    Connection connectionFactory() {
        try {
            Class.forName("oracle.jdbc.driver.OracleDriver");
            con=DriverManager.getConnection
            ("Jdbc:Oracle:thin:@localhost:1521:ORCL","scott","tiger");
        }
        catch(Exception e) {
            System.out.println(e);
        }
        return con;
    }
}
```

# Example 1 on PreparedStatement (Contd.).

```
class JdbcCalls {  
    Connection con;  
    JdbcCalls() {  
        ConnectionClass x = new ConnectionClass();  
        con=x.connectionFactory();  
    }  
    void create(String[] args) throws SQLException {  
        String tablename = args[0];  
        PreparedStatement pst = con.prepareStatement("Create table "+tablename+" (empid  
number(4), empname varchar(20), dept varchar2(10), joindate date, salary  
number(10,2))");  
        pst.executeUpdate();  
        System.out.println("Table created successfully");  
    }  
}
```

# Example 1 on PreparedStatement (Contd.).

```
C:\Windows\System32\cmd.exe
E:\My Documents\workarea\Training\jdbc\exercise>javac JCreate.java
E:\My Documents\workarea\Training\jdbc\exercise>java JCreate jdbcdemotable
Table created successfully
E:\My Documents\workarea\Training\jdbc\exercise>_
```

```
SQL Plus

SQL*Plus: Release 11.1.0.6.0 - Production on Wed Nov 19 12:15:03 2014
Copyright (c) 1982, 2007, Oracle. All rights reserved.
Enter user-name: scott
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> desc jdbcdemotable;

```

Name	Null?	Type
EMPID		NUMBER (4)
EMPNAME		VARCHAR2 (20)
DEPT		VARCHAR2 (10)
JOINDATE		DATE
SALARY		NUMBER (10, 2)

```
SQL>
```

# Example 2 on PreparedStatement

/\* This class is executed in the following manner :

If you want to insert a row within the table, you will execute as `java JInsert jdbcdemotable 1001 anish admin 23-dec-2008 50000.00` \*/

```
import java.sql.*;

class JInsert {
    public static void main(String args[]){
        try {
            JdbcCalls e = new JdbcCalls();
            e.insert(args);
        }
        catch (SQLException e) {
            System.out.println(e.toString());
        }
    }
}
```

## Example 2 on PreparedStatement (Contd.).

```
class JdbcCalls {  
    Connection con;  
    JdbcCalls() {  
        ConnectionClass x = new ConnectionClass();  
        con=x.connectionFactory();  
    }  
    void insert(String[] args) throws SQLException {  
        String tablename = args[0];  
        int empid = Integer.parseInt(args[1]);  
        String empname = args[2];  
        String dept = args[3];  
        String dat=args[4];  
        Float salary = Float.parseFloat(args[5]);  
    }  
}
```

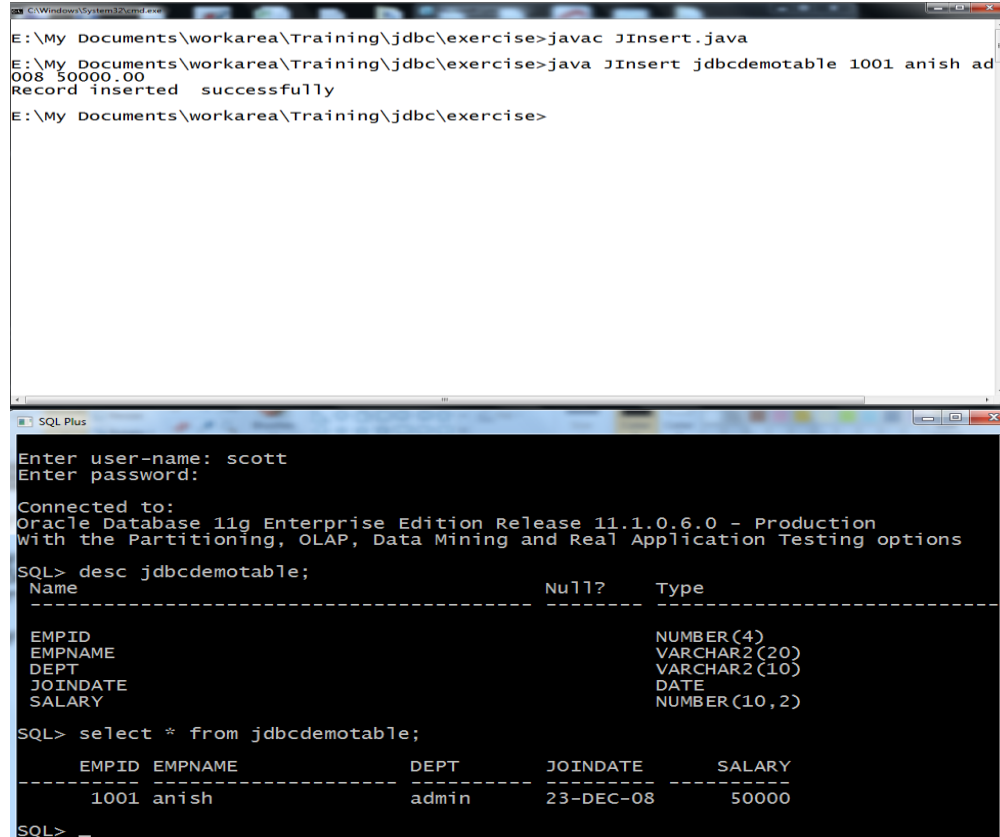


## Example 2 on PreparedStatement (Contd.).

```
PreparedStatement pst = con.prepareStatement("insert into "+tablename+"
values(?,?,?,?,?)");

pst.setInt(1, empid);
pst.setString(2, empname);
pst.setString(3, dept);
pst.setString(4, dat);
pst.setFloat(5, salary);
pst.executeUpdate();
System.out.println("Record inserted successfully");
}
}
```

## Example 2 on PreparedStatement (Contd.).



The screenshot shows two windows. The top window is a Windows command prompt with the following text:

```
E:\My Documents\workarea\Training\jdbc\exercise>javac JInsert.java
E:\My Documents\workarea\Training\jdbc\exercise>java JInsert jdbcdemotable 1001 anish ad
008 50000.00
Record inserted successfully
E:\My Documents\workarea\Training\jdbc\exercise>
```

The bottom window is SQL Plus, showing the connection to an Oracle database and the execution of two SQL queries:

```
Enter user-name: scott
Enter password:
Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> desc jdbcdemotable;
Name Null? Type
-----
EMPID NUMBER(4)
EMPNAME VARCHAR2(20)
DEPT VARCHAR2(10)
JOINDATE DATE
SALARY NUMBER(10,2)

SQL> select * from jdbcdemotable;
EMPID EMPNAME DEPT JOINDATE SALARY
-----
1001 anish admin 23-DEC-08 50000

SQL>
```

# Example 3 on PreparedStatement

/\* This class is executed in the following manner :

If you want to display all the rows, you will execute as java JDisplay jdbcdemotable

\*/

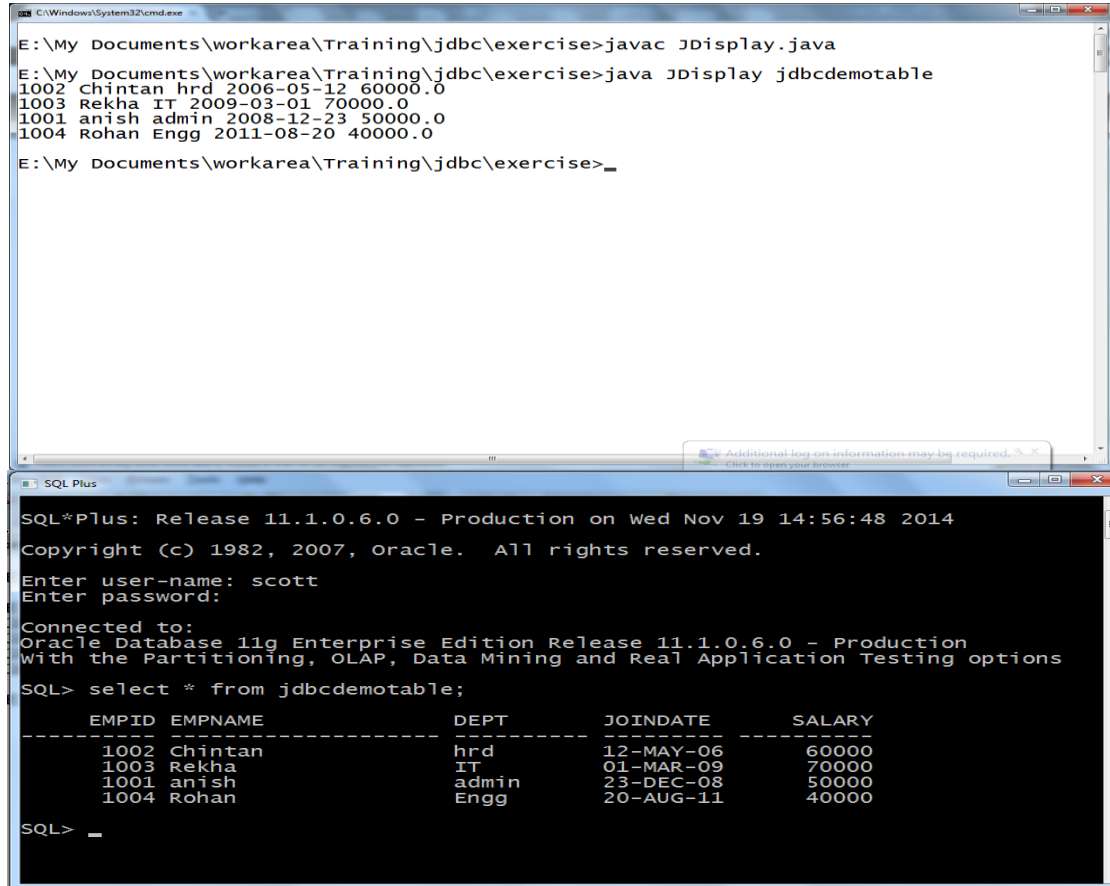
```
import java.sql.*;

class JDisplay {
    public static void main(String args[]) {
        try {
            JdbcCalls e = new JdbcCalls();
            e.display(args);
        }
        catch(Exception e) {
            System.out.println(e);
        }
    }
}
```

## Example 3 on PreparedStatement (Contd.).

```
class JdbcCalls {
    Connection con;
    JdbcCalls() {
        ConnectionClass x = new ConnectionClass();
        con=x.connectionFactory();
    }
    void display(String[] args) throws SQLException {
        String tablename = args[0];
        PreparedStatement pst = con.prepareStatement("select * from "+tablename);
        ResultSet rs= pst.executeQuery();
        while(rs.next()) {
            System.out.println(rs.getInt(1)+" "+rs.getString(2)+" "+rs.getString(3)+"
            "+rs.getDate(4)+" "+rs.getFloat(5)); }
        con.close();
    } }
```

# Example 3 on PreparedStatement (Contd.).



The screenshot shows two windows. The top window is a Windows command prompt with the following text:

```
C:\Windows\System32\cmd.exe

E:\My Documents\workarea\Training\jdbc\exercise>javac JDisplay.java
E:\My Documents\workarea\Training\jdbc\exercise>java JDisplay jdbcdemotable
1002 Chintan hrd 2006-05-12 60000.0
1003 Rekha IT 2009-03-01 70000.0
1001 anish admin 2008-12-23 50000.0
1004 Rohan Engg 2011-08-20 40000.0
E:\My Documents\workarea\Training\jdbc\exercise>_
```

The bottom window is an SQL Plus terminal with the following text:

```
SQL*Plus: Release 11.1.0.6.0 - Production on Wed Nov 19 14:56:48 2014
Copyright (c) 1982, 2007, Oracle. All rights reserved.

Enter user-name: scott
Enter password:

Connected to:
Oracle Database 11g Enterprise Edition Release 11.1.0.6.0 - Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> select * from jdbcdemotable;

   EMPID EMPNAME      DEPT      JOINDATE      SALARY
-----
   1002  Chintan      hrd      12-MAY-06      60000
   1003  Rekha        IT       01-MAR-09      70000
   1001  anish       admin    23-DEC-08      50000
   1004  Rohan       Engg     20-AUG-11      40000

SQL> _
```

# Example on Modifying the row

/\* This class is executed in the following manner :

If you want to modify a row, you will execute as Java JModify table1 1001 60000.00 where modifying a row will allow you to change the salary

```
*/  
  
import java.sql.*;  
  
class JModify {  
    public static void main(String args[]) {  
        try {  
            JdbcCalls e = new JdbcCalls();  
            e.modify(args);  
        }  
        catch (SQLException e) {  
            System.out.println(e);  
        }  
    }  
}
```

## Example on Modifying the row (Contd.).

```
class JdbcCalls {  
    Connection con;  
    JdbcCalls() {  
        ConnectionClass x = new ConnectionClass();  
        con=x.connectionFactory();  
    }  
    void modify(String[] args) throws SQLException{  
        String tablename = args[0];  
        int empid = Integer.parseInt(args[1]);  
        Float sal = Float.parseFloat(args[2]);  
        PreparedStatement pst = con.prepareStatement("update "+tablename+" set  
salary="+sal+" where empid="+empid);  
        int i= pst.executeUpdate();  
        con.close();  
    }  
}
```

## Example on Deleting a row

/\* This class is executed in the following manner : If you want to delete a row, you will execute as java JDelete table1 1001

\*/

```
import java.sql.*;
```

```
class JDelete{
```

```
    public static void main(String args[]) {
```

```
        try {
```

```
            JdbcCalls e = new JdbcCalls();
```

```
            e.delete(args);
```

```
        }
```

```
        catch (SQLException e) {
```

```
            System.out.println(e);
```

```
        }
```

```
    }
```

```
}
```



## Example on Deleting a row (Contd.).

```
class JdbcCalls {  
    Connection con;  
    JdbcCalls() {  
        ConnectionClass x = new ConnectionClass();  
        con=x.connectionFactory();  
    }  
    void delete(String[] args) throws SQLException {  
        String tablename = args[0];  
        int empid = Integer.parseInt(args[1]);  
        PreparedStatement pst = con.prepareStatement("delete from "+tablename+"  
where empid="+empid);  
        int i= pst.executeUpdate();  
        con.close();  
    }  
}
```

# Summary

In this module, you were able to:

- Analyze how to use the Metadata objects to retrieve more information about the database or the result set
- Create and execute a query using PreparedStatement object



**Thank You**