Basic Tutorial

Bok, Jong Soon
javaexpert@nate.com
https://github.com/swacademy

MySQL

- http://www.mysql.com/
- **Developer Zone**
  - http://dev.mysql.com/
- **Documentation**
  - http://dev.mysql.com/doc/
- **MySQL Connector/J Developer Guide**
  - http://dev.mysql.com/doc/connector-j/en/index.html

Oracle

- http://www.oracle.com/index.html
- **Oracle Technology Network**
  - http://www.oracle.com/technetwork/index.html
- **Documentation**
  - http://www.oracle.com/technetwork/indexes/documentation/index.html
- **JDBC Tutorials**
  - http://docs.oracle.com/javase/tutorial/jdbc/basics/index.html

Microsoft

- http://www.microsoft.com
- **Microsoft Developer Network**
  - http://msdn.microsoft.com
- **Microsoft JDBC Driver for SQL Server**
  - http://msdn.microsoft.com/en-us/sqlserver/aa937724.aspx
- **JDBC Driver API Reference**
  - http://msdn.microsoft.com/en-us/library/ms378914(v=sql.110).aspx

PostgreSQL

- http://www.postgresql.org/
- **Documentation**
  - http://www.postgresql.org/docs/
- **The PostgreSQL JDBC Interface**
  - http://jdbc.postgresql.org/documentation/80/index.html

Apache Derby

- http://db.apache.org/derby/
- **Documentation**
  - http://db.apache.org/derby/manuals/index.html
- **Quick Start**
  - http://db.apache.org/derby/quick_start.html
- **Derby JDBC Driver**
  - https://db.apache.org/derby/docs/10.7/devguide/cdevdvlp40653.html

Altibase

- http://kr.altibase.com/
- **Documentation**
  - http://support.altibase.com/manual/en/551b/html/altiman.html
- **JDBC documentation**
  - http://support.altibase.com/manual/en/551b/html/API/ch01.html#N1020D

CUBRID

- http://www.cubrid.org
- **Documentation**
  - http://www.cubrid.org/documentation
- **JDBC Home**
  - http://www.cubrid.org/wiki_apis/entry/cubrid-jdbc-driver

Sqlite3

- **http://www.sqlite.org/**
- **Documentation**
  - http://www.sqlite.org/docs.html
- **JDBC Home**
  - http://en.wikibooks.org/wiki/Java_JDBC_using_SQLite/

JDBC Drivers

- **JDBC-ODBC Bridge Driver**

  If download J2SE, get driver automatically.
- **MySQL JDBC Driver**

  http://dev.mysql.com/downloads/connector/j/
- **Oracle JDBC Driver**

  http://www.oracle.com/technetwork/database/features/jdbc/index-091264.html
- **Microsoft JDBC Driver for SQL Server**

  http://msdn.microsoft.com/en-us/sqlserver/aa937724.aspx
- **PostgreSQL JDBC Driver**

  http://jdbc.postgresql.org/

JDBC Drivers (Cont.)

- **Sqlite3 JDBC Driver**

  https://bitbucket.org/xerial/sqlite-jdbc
- **CUBRID JDBC Driver**

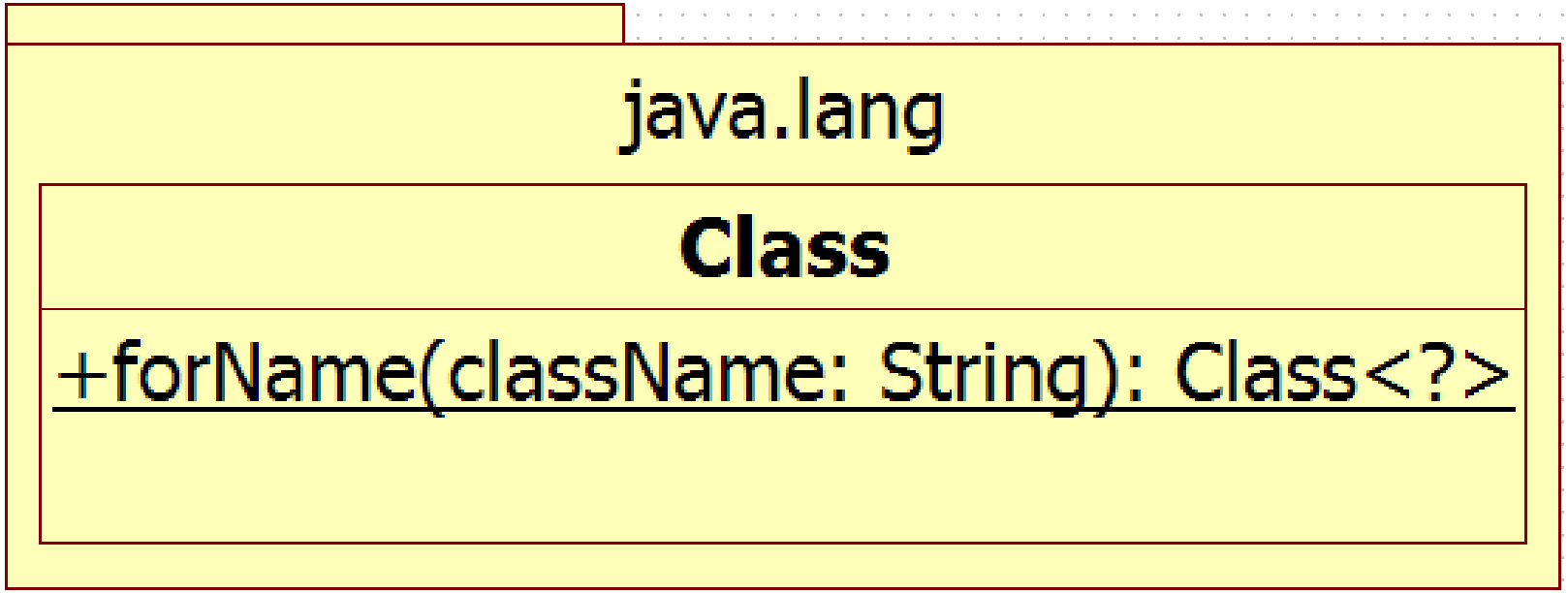
  http://www.cubrid.org/wiki_apis/entry/cubrid-jdbc-driver
- **Altibase JDBC Driver**

  http://aid.altibase.com/pages/viewpage.action?pageId=2988202
- **Apache Derby**

  http://db.apache.org/derby/derby_downloads.html

Loading Drivers

- `public static Class forName(String className)throws classNotFoundException`



Loading Drivers (Cont.)

- **JDBC-ODBC Bridge Driver**

  `sun.jdbc.odbc.JdbcOdbcDriver`

```
try{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- **MySQL Driver**

  com.mysql.jdbc.Driver

```
try{
    Class.forName("com.mysql.jdbc.Driver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- **Oracle Driver**

  `oracle.jdbc.driver.OracleDriver`

```
try{
    Class.forName("oracle.jdbc.driver.OracleDriver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- Microsoft JDBC Driver 4.0 for SQL Server

  com.microsoft.sqlserver.jdbc.SQLServerDriver

```
try{
    Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- **PostgreSQL JDBC Driver**

  `org.postgresql.Driver`

```
try{
    Class.forName("org.postgresql.Driver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- **Sqlite3 JDBC Driver**

  `org.sqlite.JDBC`

```
try{
    Class.forName("org.sqlite.JDBC");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- **CUBRID JDBC Driver**

  cubrid.jdbc.driver.CUBRIDDriver

```
try{
    Class.forName("cubrid.jdbc.driver.CUBRIDDriver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- **Altibase JDBC Driver**

  `Altibase.jdbc.driver.AltibaseDriver`

```
try{
    Class.forName("Altibase.jdbc.driver.AltibaseDriver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

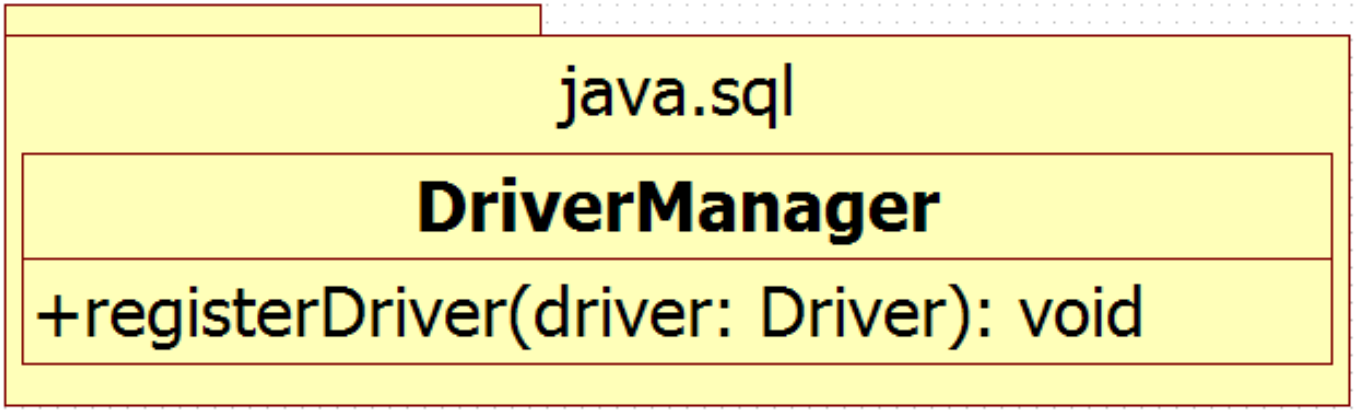
- **Apache Derby JDBC Driver**

  `org.apache.derby.jdbc.EmbeddedDriver`

```
try{
    Class.forName("org.apache.derby.jdbc.EmbeddedDriver");
}catch(ClassNotFoundException ex){
    ex.printStackTrace();
}
```

Loading Drivers (Cont.)

- `public static void registerDriver(Driver driver) throws SQLException`




```
try{
    DriverManager.registerDriver(new com.mysql.jdbc.Driver());
    System.out.println("Class Loading Success");
}catch(SQLException ex){
    System.out.println("Class Not Found");
}
```

Loading Drivers (Cont.)

- **Driver Locations**
  - Jdbc-Odbc Bridge Driver : no actions
  - Oracle, MySQL, MS SQL Server, PostgreSQL, Altibase, CUBRID, Derby, Sqlite3, etc…

    `%JAVA_HOME%\jre\lib\ext`

Loading Drivers (Cont.)

- **Driver Files**
  - MySQL
    - *<installation directory>/MySQL Connection J/*mysql-connector-java-5.1.30-bin.jar
  - Oracle
    - *<oracle_home>/jdbc/lib/*ojdbc6.jar
  - Microsoft SQL Server
    - *<installation directory>*\sqljdbc_*<version>**<language>*\sqljdbc.jar or
    - *<installation directory>*\sqljdbc_*<version>**<language>*\sqljdbc4.jar
  - PostgreSQL
    - Postgresql-9.3-1101.jdbc41.jar

Loading Drivers (Cont.)

- **Driver Files**
  - Altibase
    - *$ALTIBASE_HOME/lib/*Altibase.jar
  - CUBRID
    - *<installation directory>/jdbc/*cubrid_jdbc.jar
  - Sqlite3
    - *<installation directory>/*sqlite-jdbc4-3.8.2-SNAPSHOT.jar
  - Apache Derby
    - $DERBY_HOME/lib/derby.jar

Making a Connection

DriverManager
+getConnection(url: String): Connection +getConnection(url: String, info: Properties): Connection +getConnection(url: String, user: String, password: String): Connection

JDBC-URL

- **jdbc:sub-protocol:subname**
- **JDBC-ODBC Driver**

  jdbc:odbc:odbc_name;UID=userid;PWD=userpwd
- **MySQL Driver**

  jdbc:mysql://host_name:3306/database_name”
- **Oracle Driver**

  jdbc:oracle:thin:@host_name:1521:SID
- **Microsoft SQL Server**

  jdbc:sqlserver://host_name:1433;database_name=dbname;user=user_name;password=******;

JDBC-URL

- **PostgreSQL**

  jdbc:postgresql:dabase

  jdbc:postgresql://host_name/database

  jdbc:postgresql://host_name:port/database

- **Sqlite3**

  jdbc:sqlite:db_path/database_name

- **CUBRID**

  jdbc:cubrid:host_name:post:database_name

- **Altibase**

  jdbc:Altibase://ip_address:port_no/db_home

- **Apache Derby**

  jdbc:derby:db_path/mydb;create=true

JDBC-URL (Cont.)

- **JDBC-ODBC Driver**

  jdbc:odbc:odbcName;UID=userid;PWD=userpwd

```
Connection conn = null;
String connectionUrl = "jdbc:odbc:myaccess;" +
                       "UID=peter;PWD=password";
try{
    conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
    ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **MySQL Driver**

  jdbc:mysql://localhost:3306/test?user=monty&password=greatestsqldb”

```
Connection conn = null;
String connectionUrl = "jdbc:mysql://localhost/test?" +
                       "user=monty&password=greatestsqldb";
try{
   conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
   ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **Oracle Driver**

  jdbc:oracle:thin:@hostname:port:SID

  SID → **XE**, **ORCL**

```
Connection conn = null;
String connectionUrl = "jdbc:oracle:thin:@localhost:1521:XE";
try{
    conn = DriverManager.getConnection(connectionUrl, "scott", "tiger");
}catch(SQLException ex){
    ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **Microsoft SQL Server**

  jdbc:sqlserver://hostname:1433;databaseName=db_name

```
Connection conn = null;
String connectionUrl = "jdbc:sqlserver://localhost:1433;" +
                       "databaseName=AdventureWorks;" +
                       "user=MyUserName;password=*****;";
try{
   conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
   ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **PostgreSQL**

  `jdbc:postgresql://localhost/test`

```
Connection conn = null;
String connectionUrl = "jdbc:postgresql://localhost/" +
                       "test?user=fred&password=secret" +
                       "&ssl=true";
try{
   conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
   ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **Altibase**

  jdbc:Altibase://server_ip:server_port/dbname

```
Connection conn = null;
String connectionUrl = "jdbc:Altibase:/127.0.0.1:20300/mydb" +
                       "?user=SYS&password=MANAGER" +
                       "&encoding=US7ASCII";
try{
   conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
   ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **CUBRID**

  jdbc:cubrid:<host>:<port>:<db-name>:

  [user-id]:[password]:[?<property> [& <property>]]

```
Connection conn = null;
String connectionUrl = "jdbc:cubrid:192.168.0.1:33000" +
                       ":demodb:public::";
try{
   conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
   ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **Sqlite3**

  jdbc:sqlite:C:/sqliteroom/test.db

  jdbc:sqlite:/home/instructor/SqliteRoom/test.db

```
Connection conn = null;
String connectionUrl = "jdbc:sqlite:test.db";
try{
    conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
    ex.printStackTrace();
}
```

JDBC-URL (Cont.)

- **Apache Derby**

  jdbc:derby:C:/derbyroom/mydb;create=true

  jdbc:derby:/home/instructor/DerbyRoom/mydb;create=true

  jdbc:derby:192.168.0.8:1527/mydb;create=true

```
Connection conn = null;
String connectionUrl = "jdbc:derby:mydb;create=true;
try{
    conn = DriverManager.getConnection(connectionUrl);
}catch(SQLException ex){
    ex.printStackTrace();
}
```

Creating JDBC Statements

- A `Statement` object is what sends your SQL statement to the DBMS.
- It takes an instance of an active connection to create a `Statement` object.
- `Connection` object create the `Statement` object.

Connection
+createStatement(): Statement +createStatement(resultSetType: int, resultSetConcurrency: int): Statement

Executing Statements

- For a `SELECT` statement, the method to use is `executeQuery`.
- For statements that create or modify tables, the method to use is `executeUpdate`.

Statement
+execute(sql: String): boolean +executeUpdate(sql: String): int +executeQuery(sql: String): ResultSet

Retrieving Values from Result Sets

- If send the `SELECT` statements to a database, get the results.
- The JDBC API returns results in a `ResultSet` object.
- We need to declare an instance of the class `ResultSet` to hold our results.
- E.g.

```
ResultSet rs = stmt.executeQuery
("SELECT..");
```

Using the Method `next`

- The variable `rs`, which is an instance of `ResultSet`, contains the rows.
- Will go to each row and retrieve the values according to their types.
- The method `next` moves what is called a *cursor* to the next row and makes that row (called the current row) the one upon which we can operate.
- Since the *cursor* is initially positioned just above the first row of a `ResultSet` object, the first call to the method `next` moves the *cursor* to the first row and makes it the current row.

Using the Method `next` (Cont.)

- Successive invocations of the method `next` move the *cursor* forward one row at a time from the first row to the last row.
- This method can be used in a `while` statement because it returns `true` as long as the *cursor* is on a valid row.
- When the *cursor* goes beyond the last row, the method `next` returns `false`, thereby terminating the `while` loop.

Retrieving Column Values

- **Use a getter method (e.g., `getInt`, `getString`, `getDouble`, and so on) of the appropriate type to retrieve the value in each column.**
- **The JDBC API offers two ways to identify the column form which a getter method gets a value.**
  - One way is to give the column name.
  - Second way is to give the column index, with `1` signifying the first column.

Retrieving Column Values (Cont.)

- `public` *type* `getXxx(String columnName) throws SQLException`
- `public` *type* `getXxx(int columnIndex) throws SQLException`

Setting Up Tables

- **COFFEES**

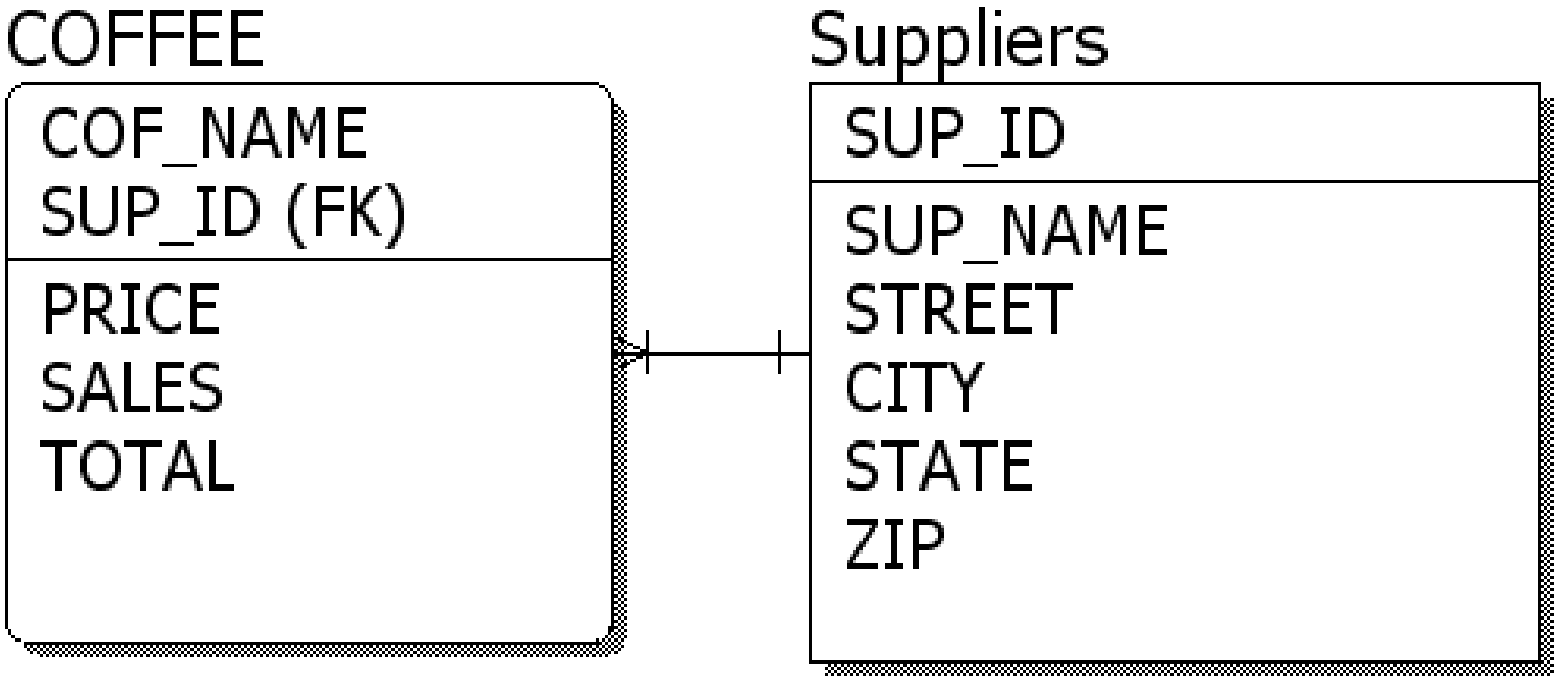
COF_NAME	SUP_ID	PRICE	SALES	TOTAL
Colombian	101	7.99	0	0
French_Roast	49	8.99	0	0
Espresso	150	9.99	0	0
Colombian_Decaf	101	8.99	0	0
French_Roast_Decaf	49	9.99	0	0

- **Suppliers**

SUP_ID	SUP_NAME	STREET	CITY	STATE	ZIP
101	Acme, Inc.	99 Market Street	Groundsville	CA	95199
49	Superior Coffee	1 Party Place	Mendocino	CA	95460
150	The High Ground	100 Coffee Lane	Meadows	CA	93966

Setting Up Tables (Cont.)

- **Table Modeling**



Setting Up Tables (Cont.)

- **Creating tables**

```
CREATE TABLE COFFEE
(
        COF_NAME     VARCHAR(32) PRIMARY KEY,
        SUP_ID    INTEGER NOT NULL       REFERENCES Suppliers(sup_id),
        PRICE     FLOAT,
        SALES     INTEGER,
        TOTAL     INTEGER
);
```

```
CREATE TABLE Suppliers
(
        SUP_ID    INTEGER   PRIMARY KEY,
        SUP_NAME     VARCHAR(50),
        STREET    VARCHAR(100),
        CITY      VARCHAR(20),
        STATE     VARCHAR(10),
        ZIP       CHAR(5)
);
```

Setting Up Tables (Cont.)

- **Creating JDBC Statements**
  - For statements that create or modify tables, the method to use is **`executeUpdate`**.

```
try{
    stmt = conn.createStatement();
    String sql = "CREATE TABLE Suppliers ";
    sql += "(SUP_ID INTEGER PRIMARY KEY,";
    sql += "SUP_NAME VARCHAR(50), STREET VARCHAR(100),";
    sql += "CITY VARCHAR(20), STATE VARCHAR(10), ZIP CHAR(5))";
    stmt.executeUpdate(sql);
    System.out.println("Command executes successfully");
}catch(SQLException ex){
    ex.printStackTrace();
}
```

Setting Up Tables (Cont.)

- **Creating JDBC Statements**

```
try{
    stmt = conn.createStatement();
    String sql = "CREATE TABLE COFFEE ";
    sql += "(COF_NAME VARCHAR(32) PRIMARY KEY, ";
    sql += "SUP_ID INTEGER NOT NULL REFERENCES Suppliers(sup_id),";
    sql += "PRICE FLOAT, SALES INTEGER, TOTAL INTEGER)";
    stmt.executeUpdate(sql);
    System.out.println("Command executes successfully");
}catch(SQLException ex){
    ex.printStackTrace();
}
```

Setting Up Tables (Cont.)

- **Entering Data into a Table**

```
try{
    stmt = conn.createStatement();
    String sql = "INSERT INTO Suppliers VALUES (101, 'Acme, Inc.',";
    sql += "'99 Market Street', 'Goundsville', 'CA', '95199')";
    stmt.executeUpdate(sql);
    System.out.println("Command executes Successfully");
}catch(SQLException ex){
    ex.printStackTrace();
```

```
try{
    stmt = conn.createStatement();
    String sql = "INSERT INTO COFFEE ";
    sql += "VALUES ('Colombian', 101, 7.99, 0, 0)";
    stmt.executeUpdate(sql);
    System.out.println("Command executes Successfully");
}catch(SQLException ex){
    ex.printStackTrace();
```

Getting Data from a Table

- **SELECT * FROM COFFEE;**

Select Test

select * from coffee

cof_name	sup_id	price	sales	total
Colombian	101	7.99	0	0
Colombian_Decaf	101	8.99	0	0
Espresso	150	9.99	0	0
French_Roast	49	8.99	0	0

```
COF_NAME             SUP_ID   PRICE    SALES    TOTAL
-----------------    ------   -----    -----    -----
Colombian            101      7.99     0        0
French_Roast         49       8.99     0        0
Espresso             150      9.99     0        0
Colombian_Decaf      101      8.99     0        0
French_Roast_Decaf   49       9.99     0        0
```

Using Prepared Statements

- **Want to execute a `Statement` object many times**
  - Reduce execution time.
- **When it is crated, SQL statement will be sent to the DBMS right away.**
  - It will be compiled.
- **When the `PreparedStatement` is executed, the DBMS can just run the `PreparedStatement`'s SQL statement without having to compile it first.**

Using Prepared Statements (Cont.)

- **Creating a `PreparedStatement` Object.**

```
String sql = "UPDATE COFFEE SET sales = ? ";
sql += "WHERE cof_name = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
```

- **Supplying Values for `PreparedStatement` Parameters.**

```
pstmt.setInt(1, 75);
pstmt.setString(2, "Colombian");
pstmt.executeUpdate();
```

Using Prepared Statements (Cont.)

- **Using a Loop to Set Values.**

```
String sql = "UPDATE COFFEE SET sales = ? ";
sql += "WHERE cof_name = ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
int [] salesForWeek = {175, 150, 60, 155, 90};
String [] coffees = {"Colombian", "French_Roast", "Espresso",
                     "Colombian_Decaf", "French_Roast_Decaf"};
int len = coffees.length;
for(int i = 0 ; i < len ; i++){
    pstmt.setInt(1, salesForWeek[i]);
    pstmt.setString(2, coffees[i]);
    pstmt.executeUpdate();
}
```

Using Joins – Scenario

- Where the owner wants to get a list of the coffees he buys from a particular supplier.
- The names of the suppliers are in the table SUPPLIERS, and the names of the coffees are in the table COFFEES.
- Since both tables have the column SUP_ID, this column can be used in a join.
- It follows that you need some way to distinguish which SUP_ID column you are referring to.
- This is done by preceding the column name with the table name, as in "COFFEES.SUP_ID" to indicate that you mean the column SUP_ID in the table COFFEES.
- The following code, in which *stmt* is a Statement object, selects the coffees bought from Acme, Inc.

Using Joins (Cont.)

```
String query = "SELECT COFFEE.cof_name " +
               "FROM COFFEE, Suppliers " +
               "WHERE Suppliers.sup_name = 'Acme, Inc.' and " +
               "Suppliers.sup_id = COFFEE.sup_id";
ResultSet rs = stmt.executeQuery(query);
System.out.println("Coffee bought from Acme, Inc. : ");
while(rs.next()){
    String coffeeName = rs.getString("cof_name");
    System.out.println("        " + coffeeName);
}
```

```
---------- Java Interpreting ----------
Coffee bought from Acme, Inc. :
        Colombian
        Colombian_Decaf
```

Using Transactions

- **There are times when you do not want one statement to take effect unless another one also succeeds.**
- **Scenario**
  - When the proprietor of The Coffee Break updates the amount of coffee sold each week, he will also want to update the total amount sold to date.
  - However, he will not want to update one without also updating the other;
  - Otherwise, the data will be inconsistent.
  - The way to be sure that either both actions occur or neither action occurs is to use a *transaction*.
  - A transaction is a set of one or more statements that are executed together as a unit, so either all of the statements are executed or none of them are executed.

Using Transactions (Cont.)

- **Disabling Auto-commit Mode**
  - When a connection is crated, it is auto-commit mode.
  - Each individual SQL statement is treated as a transaction.
  - Will be automatically committed right after it is executed.
  - The way to allow two or more statements to be grouped into a transaction. ➔ to disable auto-commit mode.

```
conn.setAutoCommit(false);
```

Using Transactions (Cont.)

- **Committing a Transaction**
  - Once auto-commit mode is disabled, no SQL statements will be committed until you call the method commit explicitly.
  - All statements executed after the previous call to the method commit will be included in the current transaction and will be committed together as a unit.

Using Transactions (Cont.)

```
conn.setAutoCommit(false);
String sql = "UPDATE COFFEE SET sales = ? " +
                " where cof_name = ? ";
PreparedStatement pstmt = con.prepareStatement(sql);
pstmt.setInt(1, 50);
pstmt.setString(2, "Colombian");
pstmt.executeUpdate();
sql = "UPDATE COFFEE SET total = total + ? ";
        " WHERE cof_name = ? ";
PreparedStatement pstmt1 = con.prepareStatement(sql);
pstmt1.setInt(1, 50);
pstmt1.setString(2, "Colombian");
pstmt1.executeUpdate();
conn.commit();
conn.setAutoCommit(true);
```

Stored Procedures

- Is a group of SQL statements that form a logical unit and perform a particular task.
- Are used to encapsulate a set of operations or queries to execute on a database server.
- Can be compiled and executed with different parameters and results.
- May have any combination of input, output and input/output parameters.
- Are supported by most DBMSs.

Stored Procedures (Cont.)

- **SQL Statements for Creating a Stored Procedure**

```
--for Oracle or MS SQL Server 2000
CREATE PROCEDURE show_Suppliers
AS
SELECT Suppliers.sup_name, coffee.cof_name
FROM Suppliers, COFFEE
WHERE Suppliers.sup_id = COFFEE.sup_id;
EXEC show_Suppliers;
-- for MySQL 5.0
CREATE PROCEDURE show_Suppliers()
SELECT Suppliers.sup_name, coffee.cof_name
FROM Suppliers, COFFEE
WHERE Suppliers.sup_id = COFFEE.sup_id;
CALL show_Suppliers();
```

Stored Procedures (Cont.)

- **Calling a Stored Procedure Using the JDBC API**
  - First, to create a `CallableStatement` object.
  - Is done with an open `Connection` object.
  - Contains a `call` to a stored procedure.

```
String sql = "{ call show_Suppliers() }";
CallableStatement cstmt = conn.prepareCall(sql);
ResultSet rs = cstmt.executeQuery();
```