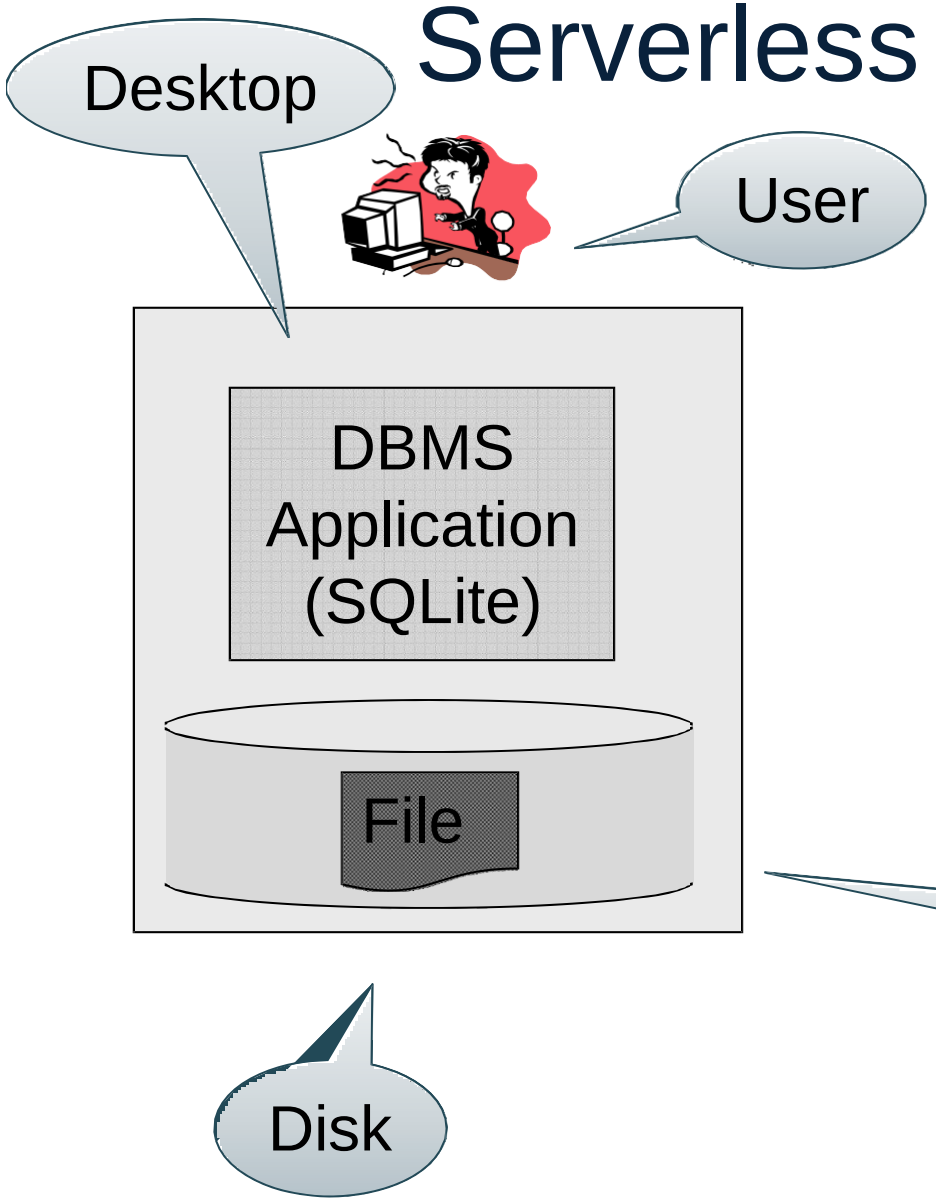
# Introduction to Database Systems

Lecture 10: SQL in Server Environment

# Serverless Architecture

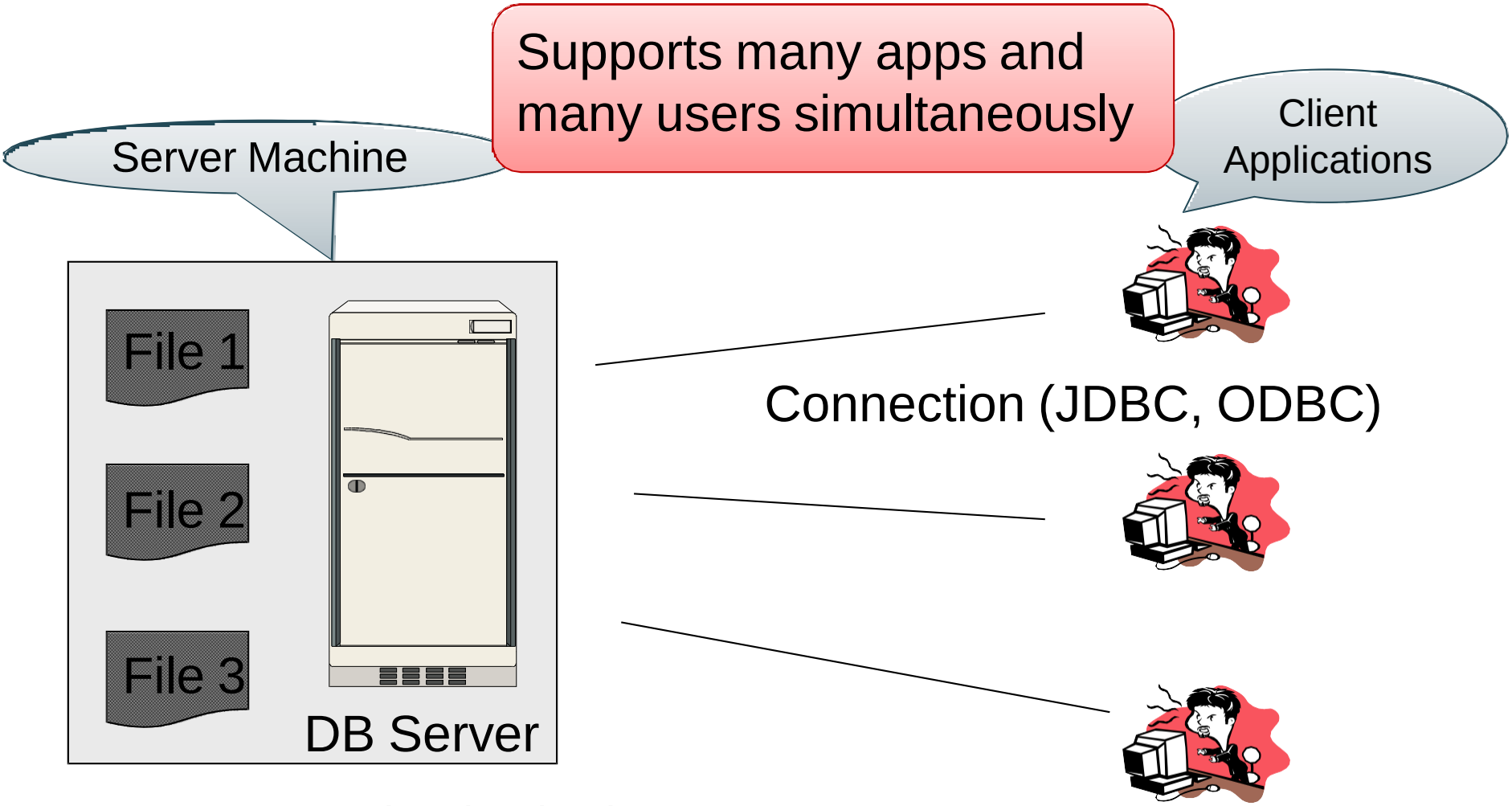



SQLite:

- One data file
- One user
- One DBMS application

- Scales well!
- But only a limited number of scenarios work with such model
- (Can be in browser / phone!)

# Client-Server Architecture

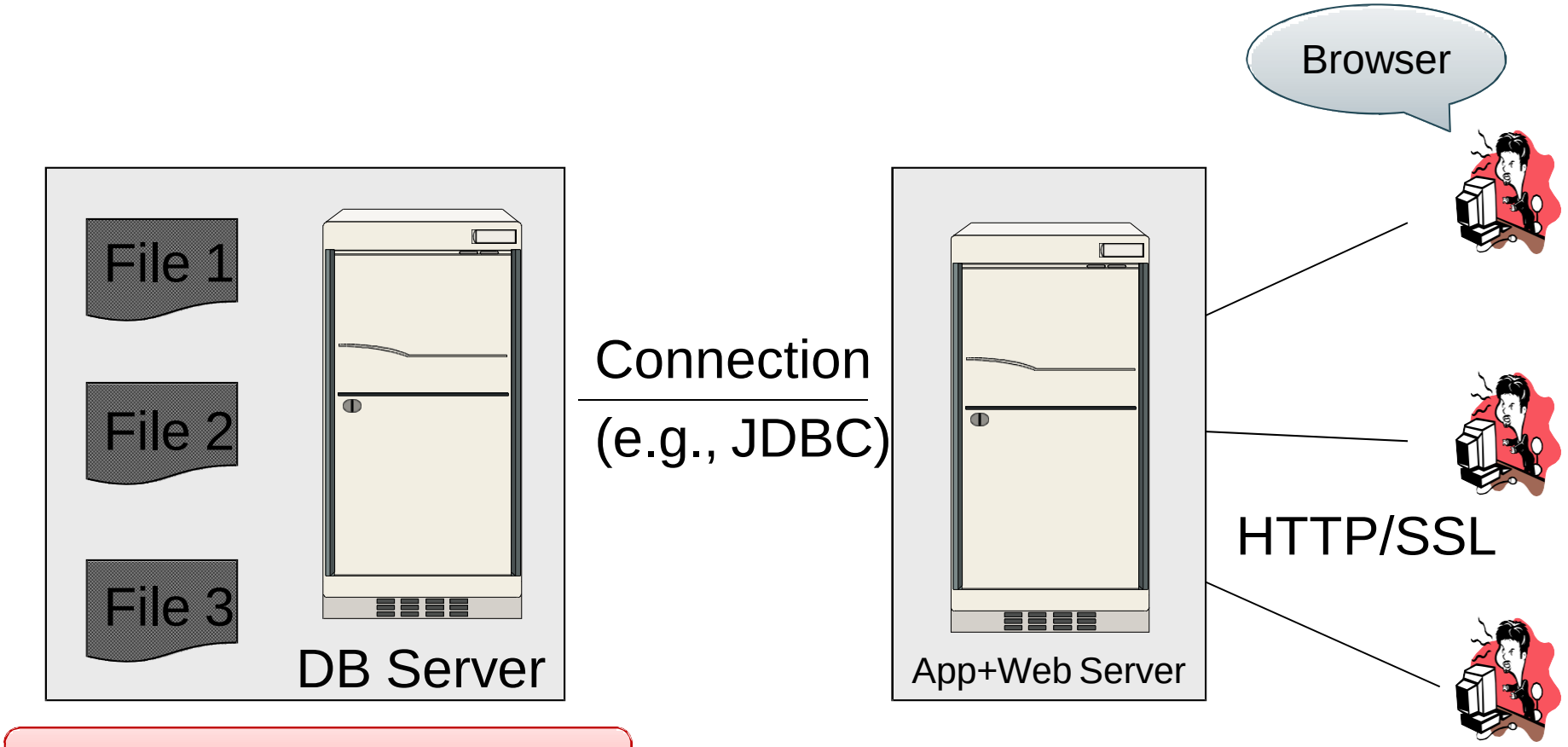



- One server running the database
- Many clients, connecting via the ODBC or JDBC (Java Database Connectivity) protocol

# Client-Server

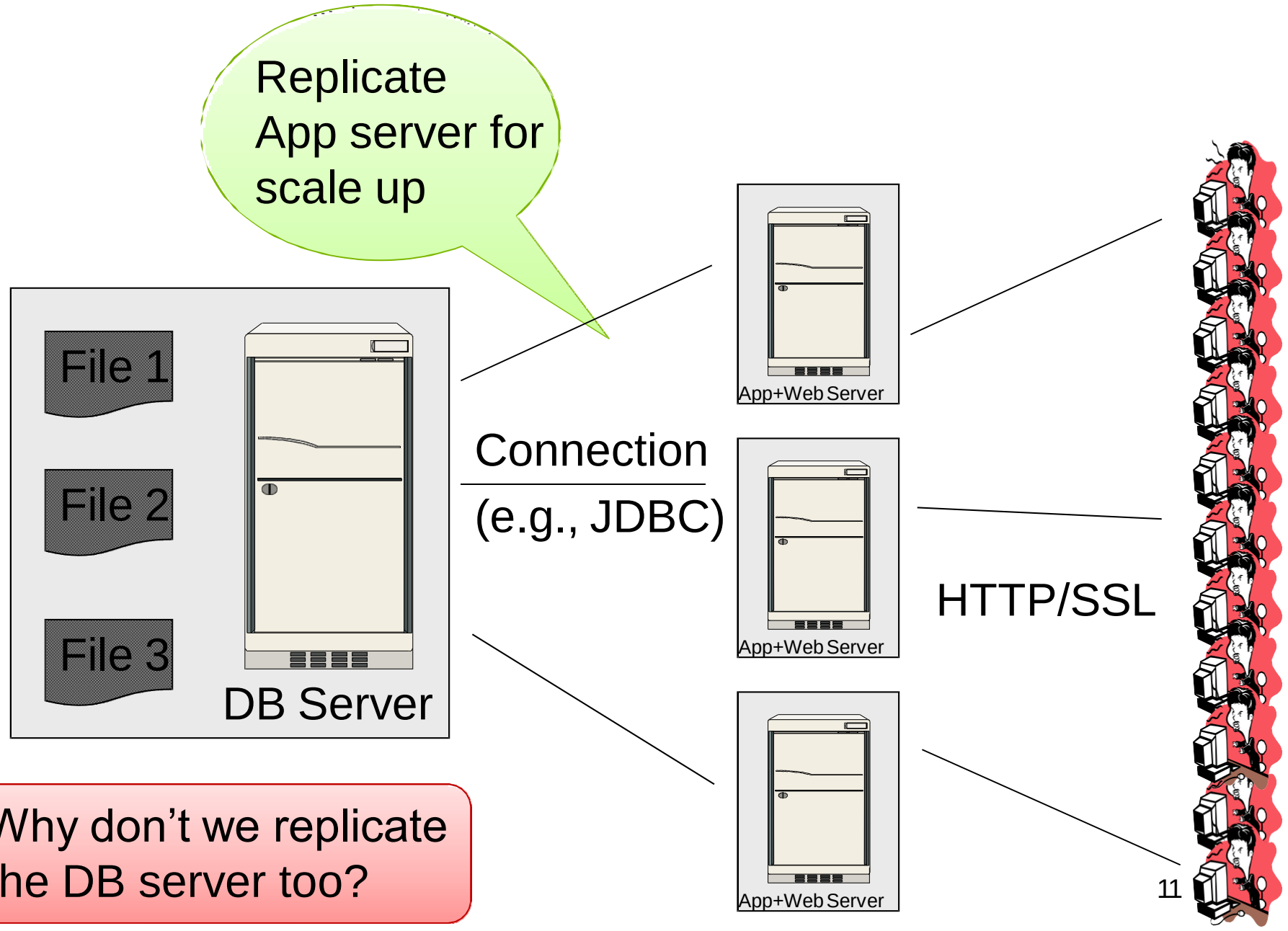
- One *server* that runs the DBMS (or RDBMS):
  - Your own desktop, or
  - Some beefy system, or
  - A cloud service (SQL Azure)
- Many *clients* run apps and connect to DBMS
  - Microsoft's Management Studio (for SQL Server), or
  - psql (for postgres)
  - Some Java program (HW8) or some C++ program
- Clients "talk" to server using JDBC/ODBC protocol

# 3-Tiered Architecture




Web-based applications

# 3-Tiered Architecture




Why don't we replicate the DB server too?

# The four types of JDBC database drivers

| | |
|---|---|
| Type 1 | A *JDBC-ODBC bridge driver* converts JDBC calls into ODBC calls that access the DBMS protocol. The ODBC driver must be installed on the client machine. |
| Type 2 | A *native protocol partly Java driver* converts JDBC calls into calls in the native DBMS protocol. This conversion takes place on the client. |
| Type 3 | A *net protocol all Java driver* converts JDBC calls into a net protocol that's independent of any native DBMS protocol. Then, middleware software running on a server converts the net protocol to the native DBMS protocol. This conversion takes place on the server side. |
| Type 4 | A *native protocol all Java driver* converts JDBC calls into a native DBMS protocol. This conversion takes place on the server side. |

## How to download a database driver

- For MySQL databases, you can download a JDBC driver named Connector/J from the MySQL web site. This driver is an open-source, type-4 driver that's available for free.
- For other databases, you can usually download a type-4 JDBC driver from the database's web site.

## How to make a database driver available to an application

- Before you can use a database driver, you must make it available to your application. One easy way to do that is to copy the JAR file for the driver into the JDK's jre\lib\ext directory.

## Database URL syntax

```
jdbc:subprotocolName:databaseURL
```

## How to connect to a MySQL database with automatic driver loading

```
try
{
    String dbURL = "jdbc:mysql://localhost:3306/murach";
    String username = "root";
    String password = "sesame";
    Connection connection = DriverManager.getConnection(
        dbURL, username, password);
}
catch(SQLException e)
{
    for (Throwable t : e)
        t.printStackTrace();
}
```

## How to connect to an Oracle database with automatic driver loading

```
Connection connection = DriverManager.getConnection(
    "jdbc:oracle:thin@localhost/murach", "scott",
    "tiger");
```

## How to load a MySQL database driver prior to JDBC 4.0

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

# How to connect to a database

- You use the getConnection method of the DriverManager class to return a Connection object.
- When you use the getConnection method, you must supply a URL for the database, a username, and a password. This method throws a SQLException.
- With JDBC 4.0, the SQLException class implements the Iterable interface.
- With JDBC 4.0, the database driver is loaded automatically. This new feature is known as *automatic driver loading*. Prior to JDBC 4.0, you needed to use the forName method of the Class class to load the driver. This method throws a ClassNotFoundException.
- The connection string for each driver is different, so see the documentation for details.
- It's possible (though not typical) to load multiple database drivers and establish connections to multiple types of databases.

## How to create a result set that contains 1 row and 1 column

```
Statement statement = connection.createStatement();
ResultSet userIDResult = statement.executeQuery(
    "SELECT UserID FROM User " +
    "WHERE EmailAddress = 'jsmith@gmail.com'");
```

## How to create a result set that contains multiple columns and rows

```
Statement statement = connection.createStatement();
ResultSet products = statement.executeQuery(
    "SELECT * FROM Product ");
```

## How to move the cursor to the first record in the result set

```
boolean userIDExists = userIDResult.next();
```

## How to loop through a result set

```
while (products.next()) {
    // statements that process each record
}
```

# ResultSet methods for forward-only, read-only result sets

| Method | Description |
|---|---|
| `next()` | Moves the cursor to the next row in the result set. |
| `last()` | Moves the cursor to the last row in the result set. |
| `close()` | Releases the result set's resources. |
| `getRow()` | Returns an int value that identifies the current row of the result set. |

# How to return a result set and move the cursor through it

- To return a *result set*, you use the createStatement method of a Connection object to create a Statement object. Then, you use the executeQuery method of the Statement object to execute a SELECT statement that returns a ResultSet object.
- By default, the createStatement method creates a forward-only, read-only result set. This means that you can only move the *cursor* through it from the first record to the last and that you can't update it.
- When a result set is created, the cursor is positioned before the first row. Then, you can use the methods of the ResultSet object to move the cursor.
- The createStatement, executeQuery, and next methods throw an SQLException. As a result, any code that uses these methods needs to catch or throw this exception.

# Methods of a ResultSet object that return data from a result set

| Method | Description |
|---|---|
| `getXXX(int columnIndex)` | Returns data from the specified column number. |
| `getXXX(String columnName)` | Returns data from the specified column name. |

## Code that uses column indexes to return fields from the products result set

```
String code = products.getString(1);
String description = products.getString(2);
double price = products.getDouble(3);
```

## Code that uses column names to return the same fields

```
String code = products.getString("ProductCode");
String description
    = products.getString("ProductDescription");
double price = products.getDouble("ProductPrice");
```

## Code that creates a Product object from the products result set

```
Product product = new Product(products.getString(1),
                              products.getString(2),
                              products.getDouble(3));
```

# How to retrieve data from a result set

- The getXXX methods can be used to return all eight primitive types. Examples:
  - getInt
  - getLong
- The getXXX methods can also be used to return strings, dates, and times. Examples:
  - getString
  - getDate
  - getTime
  - getTimestamp

# How to use the executeUpdate method to modify data

## How to add a record

```
String query =
   "INSERT INTO Product (ProductCode,
           ProductDescription, ProductPrice) " +
   "VALUES ('" + product.getCode() + "', " +
           "'" + product.getDescription() + "', " +
           "'" + product.getPrice() + "')";
Statement statement = connection.createStatement();
int rowCount = statement.executeUpdate(query);
```

## How to update a record

```
String query = "UPDATE Product SET " +
   "ProductCode = '" + product.getCode() + "', " +
   "ProductDescription = '" + product.getDescription() +
       "', " +
   "ProductPrice = '" + product.getPrice() + "' " +
   "WHERE ProductCode = '" + product.getCode() + "'";
Statement statement = connection.createStatement();
int rowCount = statement.executeUpdate(query);
```

# How to use the executeUpdate method to modify data (cont.)

## How to delete a record

```
String query = "DELETE FROM Product " +
               "WHERE ProductCode = '" + productCode +
               "'";
Statement statement = connection.createStatement();
int rowCount = statement.executeUpdate(query);
```

# The executeUpdate method…

- Is an older method that works with most JDBC drivers. Although there are some newer methods that require less SQL code, they may not work properly with all JDBC drivers.
- Returns an int value that identifies the number of records that were affected by the SQL statement.

# How to use a prepared statement

## To return a result set

```
String preparedSQL = "SELECT ProductCode,
ProductDescription, ProductPrice "
    + "FROM Product WHERE ProductCode = ?";
PreparedStatement ps =
    connection.prepareStatement(preparedSQL);
ps.setString(1, productCode);
ResultSet product = ps.executeQuery();
```

# How to use a prepared statement (cont.)

## To modify data

```
String preparedSQL = "UPDATE Product SET "
                   + "   ProductCode = ?, "
                   + "   ProductDescription = ?, "
                   + "   ProductPrice = ?"
                   + "WHERE ProductCode = ?";
PreparedStatement ps =
connection.prepareStatement(preparedSQL);
ps.setString(1, product.getCode());
ps.setString(2, product.getDescription());
ps.setDouble(3, product.getPrice());
ps.setString(4, product.getCode());
ps.executeUpdate();
```

# How to use a prepared statement (cont.)

## To insert a record

```
String preparedQuery =
      "INSERT INTO Product (ProductCode,
       ProductDescription, ProductPrice) "
    + "VALUES (?, ?, ?)";
PreparedStatement ps =
    connection.prepareStatement(preparedQuery);
ps.setString(1, product.getCode());
ps.setString(2, product.getDescription());
ps.setDouble(3, product.getPrice());
ps.executeUpdate();
```

# How to use a prepared statement (cont.)

## To delete a record

```
String preparedQuery = "DELETE FROM Product "
                     + "WHERE ProductCode = ?";
PreparedStatement ps =
    connection.prepareStatement(preparedQuery);
ps.setString(1, productCode);
ps.executeUpdate();
```

# How to work with prepared statements

- When you use *prepared statements* in your Java programs, the database server only has to check the syntax and prepare an execution plan once for each SQL statement. This improves the efficiency of the database operations.
- To specify a parameter for a prepared statement, type a question mark (?) in the SQL statement.
- To supply values for the parameters in a prepared statement, use the set methods of the PreparedStatement interface.
- To execute a SELECT statement, use the executeQuery method.
- To execute an INSERT , UPDATE, or DELETE statement, use the executeUpdate method.