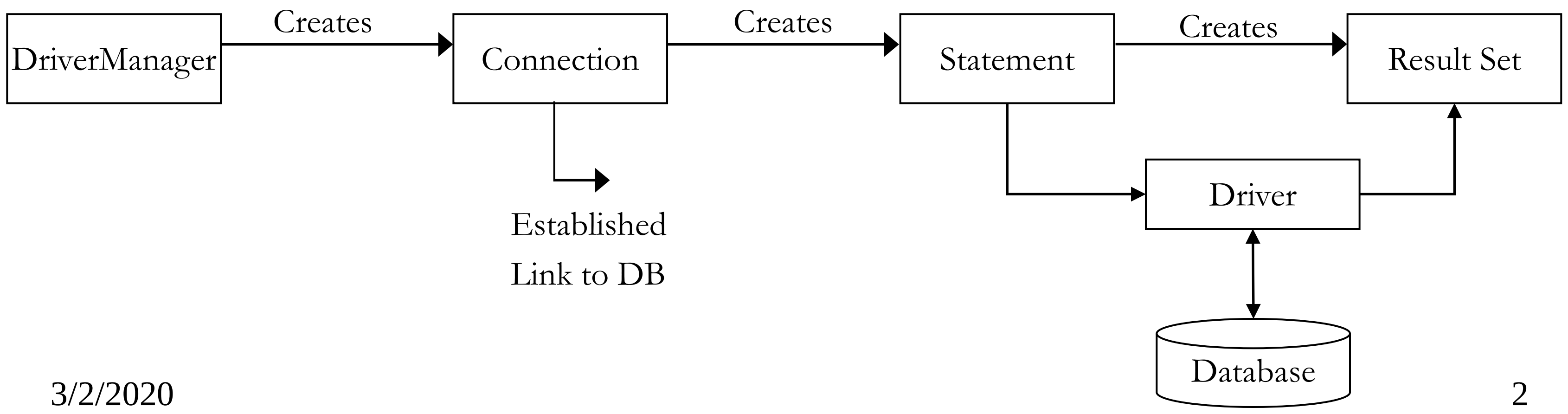
Connecting to Database

JDBC

Conceptual Components

- **Driver Manager:** Loads database drivers and manages connections between the application and the driver
- **Driver:** Translates API calls into operations for specific database
- **Connection:** Session between application and data source
- **Statement:** SQL statement to perform query or update
- **Metadata:** Information about returned data, database, & driver
- **Result Set:** Logical set of columns and rows of data returned by executing a statement



JDBC

Basic Steps

- Import the necessary classes
- Load the **JDBC driver**
- Identify the data source (Define the Connection URL)
- Establish the **Connection**
- Create a **Statement** Object
- Execute query string using **Statement** Object
- Retrieve data from the returned **ResultSet** Object
- Close **ResultSet** & **Statement** & **Connection** Object in order

JDBC

Driver Manager

- DriverManager provides a common access layer on top of different database drivers
  - Responsible for managing the JDBC drivers available to an application
  - Hands out connections to the client code
- Maintains reference to each driver
  - Checks with each driver to determine if it can handle the specified URL
  - The first suitable driver located is used to create a connection
- DriverManager class can not be instantiated
  - All methods of DriverManager are static
  - Constructor is private

JDBC Driver

Loading

- Required prior to communication with a database using JDBC
- It can be loaded
  - dynamically using Class.forName(String *drivername*)
  - System Automatically loads driver using jdbc.drivers system property
- An instance of driver must be registered with DriverManager class
- Each Driver class will typically
  - create an instance of itself and register itself with the driver manager
  - Register that instance automatically by calling RegisterDriver method of the DriverManager class
- Thus the code does not need to create an instance of the class or register explicitly using registerDriver(Driver) class

JDBC Driver

Loading: class.forName()

- Using forName(String) from java.lang.Class instructs the JVM to find, load and link the class identified by the String

```
e.g try {
        Class.forName("COM.cloudscape.core.JDBCDriver");
        } catch (ClassNotFoundException e) {
            System.out.println("Driver not found");
            e.printStackTrace();
        }
```

- At run time the class loader locates the driver class and loads it
  - All static initializations during this loading
  - Note that the name of the driver is a literal string thus the driver does not need to be present at compile time

JDBC Driver

Loading: System Property

- Put the driver name into the jdbc drivers System property
  - When a code calls one of the methods of the driver manager, the driver manager looks for the jdbc.drivers property
  - If the driver is found it is loaded by the Driver Manager
  - Multiple drivers can be specified in the property
  - Each driver is listed by full package specification and class name
  - a colon is used as the delimiter between the each driver

  e.g jdbc.drivers=com.pointbase.jdbc.jdbcUniversalDriver
- For specifying the property on the command line use:
  - java -Djdbc.drivers=com.pointbase.jdbc.jdbcUniversalDriver MyApp
- A list of drivers can also be provided using the Properties file
  - System.setProperty("jdbc.drivers", "COM.cloudscape.core.JDBCDriver");
  - DriverManager only loads classes once so the system property must be set prior to the any DriverManager method being called.

JDBC

URLs

- JDBC Urls provide a way to identify a database
- Syntax:

  <protocol>:<subprotocol>:<protocol>

  - Protocol: Protocol used to access database (jdbc here)
  - Subprotocol: Identifies the database driver
  - Subname: Name of the resource
- Example
  - Jdbc:cloudscape:Movies
  - Jdbc:odbc:Movies

Connection

Creation

- Required to communicate with a database via JDBC
- Three separate methods:

```
public static Connection getConnection(String url)
public static Connection getConnection(String url, Properties info)
public static Connection getConnection(String url, String user, String password)
```

- Code Example (Access)

```
try {// Load the driver class
      System.out.println("Loading Class driver");
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      // Define the data source for the driver
      String sourceURL = "jdbc:odbc:music”;
      // Create a connection through the DriverManager class
      System.out.println("Getting Connection");
      Connection databaseConnection = DriverManager.getConnection(sourceURL);
      }
catch (ClassNotFoundException cnfe) {
            System.err.println(cnfe); }
 catch (SQLException sqle) {
            System.err.println(sqle);}
```

Connection

Creation

- Code Example (Oracle)

```
try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
        String sourceURL = "jdbc:oracle:thin:@svvv.edu.in:databasename";
        String user = “Jayendra";
        String password = “SomePassword";
        Connection
    databaseConnection=DriverManager.getConnection(sourceURL,user, password
    );
        System.out.println("Connected Connection"); }
catch (ClassNotFoundException cnfe) {
        System.err.println(cnfe); }
catch (SQLException sqle) {
        System.err.println(sqle);}
```

Connection

Closing

- Each machine has a limited number of connections (separate thread)
  - If connections are not closed the system will run out of resources and freeze
  - Syntax: public void close() throws SQLException

- Naïve Way:

```
try {
      Connection conn
      = DriverManager.getConnection(url);
       // Jdbc Code
      …
      } catch (SQLException sqle) {
         sqle.printStackTrace();
      }
      conn.close();
```

- SQL exception in the Jdbc code will prevent execution to reach conn.close()

- Correct way (Use the finally clause)

```
try{
Connection conn =
     Driver.Manager.getConnection(url);
     // JDBC Code
     } catch (SQLException sqle) {
        sqle.printStackTrace();
     } finally {
        try {
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
     }
```

Statement

Types

- Statements in JDBC abstract the SQL statements
- Primary interface to the tables in the database
- Used to create, retrieve, update & delete data (CRUD) from a table
  - Syntax: Statement statement = connection.createStatement();
- Three types of statements each reflecting a specific SQL statements
  - Statement
  - PreparedStatement
  - CallableStatement

Statement

Syntax

- Statement used to send SQL commands to the database
  - Case 1: ResultSet is non-scrollable and non-updateable

    ```
    public Statement createStatement() throws SQLException
    Statement statement = connection.createStatement();
    ```

  - Case 2: ResultSet is non-scrollable and/or non-updateable

    ```
    public Statement createStatement(int, int) throws SQLException
    Statement statement = connection.createStatement();
    ```

  - Case 3: ResultSet is non-scrollable and/or non-updateable and/or holdable

    ```
    public Statement createStatement(int, int, int) throws SQLException
    Statement statement = connection.createStatement();
    ```

- PreparedStatement

  ```
  public PreparedStatement prepareStatement(String sql) throws SQLException
  PreparedStatement pstatement = prepareStatement(sqlString);
  ```

- CallableStatement used to call stored procedures

  ```
  public CallableStatement prepareCall(String sql) throws SQLException
  ```

Statement

Release

- Statement can be used multiple times for sending a query
- It should be released when it is no longer required
  - Statement.close():
  - It releases the JDBC resources immediately instead of waiting for the statement to close automatically via garbage collection
- Garbage collection is done when an object is unreachable
  - An object is reachable if there is a chain of reference that reaches the object from some root reference
- Closing of the statement should be in the finally clause

```
try{
   Connection conn = Driver.Manager.getConnection(url);
   Statement stmt = conn.getStatement();
   // JDBC Code
   } catch (SQLException sqle) {
   sqle.printStackTrace();
   } finally {
     try {stmt.close();
          conn.close();
     } catch (Exception e) {
          e.printStackTrace();
     }
}
```

JDBC
Logging

- DriverManager provides methods for managing output
  - DriverManagers debug output can be directed to a printwriter

    public static void setLogWriter(PrintWriter pw)
  - PrintWriter can be wrapped for any writer or OutputStream
  - Debug statements from the code can be sent to the log as well.

    public static void println(String s)
- Code

```
FileWriter fw = new FileWriter("mydebug.log");
PrintWriter pw = new PrintWriter(fw);
// Set the debug messages from Driver manager to pw
DriverManager.setLogWriter(pw);
// Send in your own debug messages to pw
DriverManager.println("The name of the database is " + databasename);
```

Querying the Database

Executing Queries
Methods

- Two primary methods in statement interface used for executing Queries
  - executeQuery Used to retrieve data from a database
  - executeUpdate: Used for creating, updating & deleting data
- executeQuery used to retrieve data from database
  - Primarily uses Select commands
- executeUpdate used for creating, updating & deleting data
  - SQL should contain Update, Insert or Delete commands
- Uset setQueryTimeout to specify a maximum delay to wait for results

Executing Queries

Data Definition Language (DDL)

- Data definition language queries use executeUpdate
- Syntax: int executeUpdate(String sqlString) throws SQLException
  - It returns an integer which is the number of rows updated
  - sqlString should be a valid String else an exception is thrown
- Example 1: Create a new table

```
Statement statement = connection.createStatement();
String sqlString =
“Create Table Catalog”
+ “(Title Varchar(256) Primary Key Not Null,”+
+ “LeadActor Varchar(256) Not Null, LeadActress Varchar(256) Not Null,”
+ “Type Varchar(20) Not Null, ReleaseDate Date Not NULL )”;
Statement.executeUpdate(sqlString);
```

  - executeUpdate returns a zero since no row is updated

Executing Queries

DDL (Example)

- Example 2: Update table

    Statement statement = connection.createStatement();

    String sqlString =

    “Insert into Catalog”

    + “(Title, LeadActor, LeadActress, Type, ReleaseDate)”

    + “Values(‘Gone With The Wind’, ‘Clark Gable’, ‘Vivien Liegh’,”

    + “’Romantic’, ‘02/18/2003’ ”

    Statement.executeUpdate(sqlString);

    – executeUpdate returns a 1 since one row is added

Executing Queries
Data Manipulation Language (DML)

- Data definition language queries use executeQuery
- Syntax

  ResultSet executeQuery(String sqlString) throws SQLException
  - It returns a ResultSet object which contains the results of the Query
- Example 1: Query a table

```
Statement statement = connection.createStatement();
String sqlString = “Select Catalog.Title, Catalog.LeadActor, Catalog.LeadActress,” +
                   “Catalog.Type, Catalog.ReleaseDate From Catalog”;
ResultSet rs = statement.executeQuery(sqlString);
```

ResultSet

Definition

- ResultSet contains the results of the database query that are returned
- Allows the program to scroll through each row and read all columns of data
- ResultSet provides various access methods that take a column index or column name and returns the data
  - All methods may not be applicable to all resultsets depending on the method of creation of the statement.
- When the executeQuery method returns the ResultSet the cursor is placed before the first row of the data
  - Cursor refers to the set of rows returned by a query and is positioned on the row that is being accessed
  - To move the cursor to the first row of data next() method is invoked on the resultset
  - If the next row has a data the next() results true else it returns false and the cursor moves beyond the end of the data
- First column has index 1, not 0

ResultSet

- ResultSet contains the results of the database query that are returned
- Allows the program to scroll through each row and read all the columns of the data
- ResultSet provides various access methods that take a column index or column name and returns the data
  - All methods may not be applicable to all resultsets depending on the method of creation of the statement.
- When the executeQuery method returns the ResultSet the cursor is placed before the first row of the data
  - Cursor is a database term that refers to the set of rows returned by a query
  - The cursor is positioned on the row that is being accessed
  - First column has index 1, not 0
- Depending on the data numerous functions exist
  - getShort(), getInt(), getLong()
  - getFloat(), getDouble()
  - getClob(), getBlob(),
  - getDate(), getTime(), getArray(), getString()

ResultSet

- Examples:
  - Using column Index:

    Syntax:public String getString(int columnIndex) throws SQLException

    e.g. ResultSet rs = statement.executeQuery(sqlString);

    String data = rs.getString(1)
  - Using Column name

    public String getString(String columnName) throws SQLException

    e.g. ResultSet rs = statement.executeQuery(sqlString);

    String data = rs.getString(Name)
- The ResultSet can contain multiple records.
  - To view successive records next() function is used on the ResultSet
  - Example: while(rs.next()) {
  - System.out.println(rs.getString); }

Scrollable ResultSet

- ResultSet obtained from the statement created using the no argument constructor is:
  - Type forward only (non-scrollable)
  - Not updateable
- To create a scrollable ResultSet the following statement constructor is required
  - Statement createStatement(int resultSetType, int resultSetConcurrency)
- ResultSetType determines whether it is scrollable. It can have the following values:
  - ResultSet.TYPE_FORWARD_ONLY
  - ResultSet.TYPE_SCROLL_INSENSITIVE (Unaffected by changes to underlying database)
  - ResultSet.TYPE_SCROLL_SENSITIVE (Reflects changes to underlying database)
- ResultSetConcurrency determines whether data is updateable. Its possible values are
  - CONCUR_READ_ONLY
  - CONCUR_UPDATEABLE
- Not all database drivers may support these functionalities