# INTRODUCTION TO JAVA PROGRAMMING VIII

CSI-3130

# JDBC
# CONNECT JAVA TO SQL DB

https://docs.oracle.com/javase/tutorial/essential/io/index.html
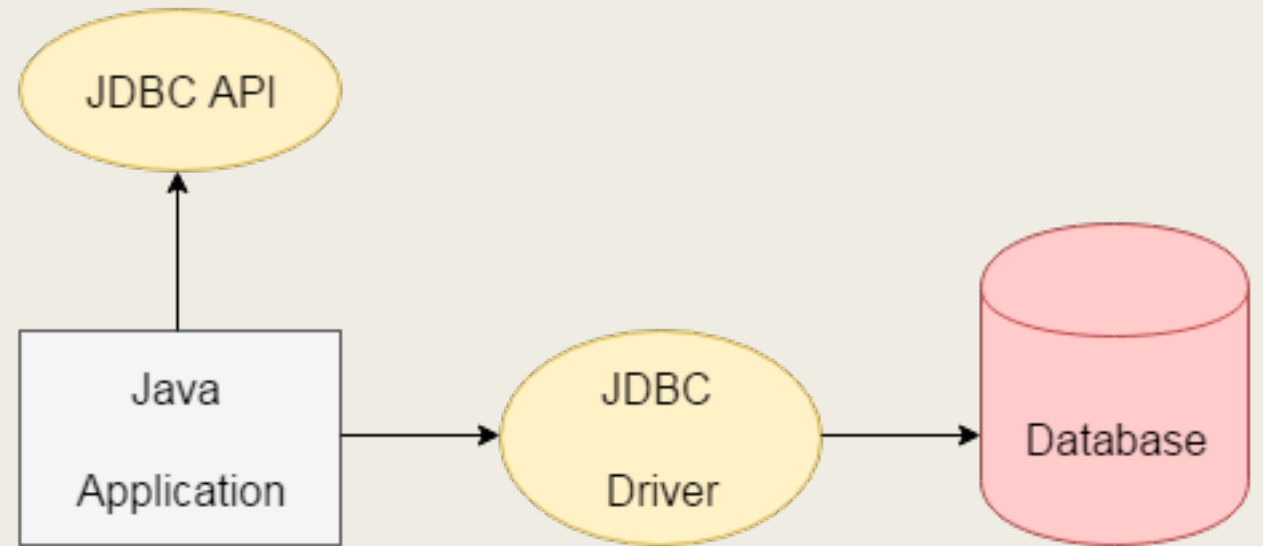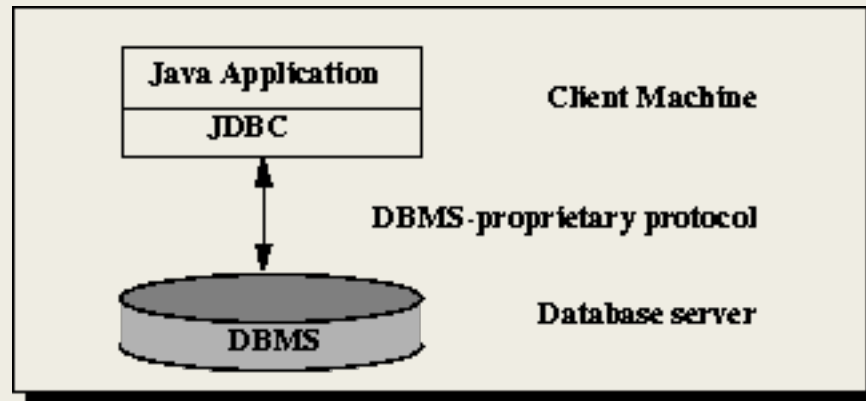
# JDBC

- ■ The JDBC API is a Java API that can access any kind of tabular data, especially data stored in a Relational Databases

- ■ A database is a means of storing information in such a way that information can be retrieved from it.

# JDBC

- In simplest terms, a relational database shows information in tables with rows & columns.

- A **table** is referred to as a **relation**

  - *in the sense that it is a collection of objects of the same type (rows).*

- Data in a table can be related according to common keys or concepts, and the ability to retrieve related data from a table is the basis for the term relational database.

- A Database Management System (DBMS)

  - *handles the way data is stored, maintained, and retrieved.*

- Relational Database Mgmt System (RDBMS)

# Architecture



■ JDBC helps you to write Java applications that manage these 3 programming activities:

1.  Connect to a data source, like a database

2.  Send queries and update statements to the database

3.  Retrieve and process the results received from the database in answer to your query

# Example

```
public void connectToAndQueryDatabase(String username, String password) {

    Class.forName("xx.xx.Driver"); //egister driver;specific for given DB
    Connection con = DriverManager.getConnection(
            "jdbc:myDriver:myDatabase",
            username, password);

    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT colA, colB FROM Table");

    while (rs.next()) {
        int x = rs.getInt("a");
        String s = rs.getString("b");
        float f = rs.getFloat("c");
    }
}
```

# Many Database providers

■ Divided responsibility

1. *Java API – JDBC*
2. *Database specialties – Driver*
– *Custom driver.jar per a particular database*

# Sample database table

■ Employee

```
table Employee (
    Employee_Number int primary key,
    Fistname varchar(255),
    Lastname varchar(255),
    Date_of_Birth date,
    Car_Number int foreigh key
)
```

```
class Employee {
    int employeeNumber;
    String fistname;
    String lastname;
    Date dob;
    int carNumber;
}
```

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|---|---|---|---|---|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

# SQL

■ Structured Query Language (SQL) is a domain-specific language used in programming and designed for managing data held in a relational database management system (RDBMS)


– *SELECT [columns] FROM [table] WHERE [condition]*

– *SELECT [columns] FROM [table1] INNER JOIN [table2] ON (cond.)..*

– *INSERT INTO [table] VALUES ([values])*

– *UPDATE  [table] SET [column1 = value1,.. ] WHERE [condition]*

– *DELETE FROM  [table] WHERE  [condition]*

# Lets apply SQL

- Employee

```
table Employee (
    Employee_Number int primary key,
    Fistname varchar(255),
    Lastname varchar(255),
    Date_of_Birth date,
    Car_Number int foreigh key
)
```

```
class Employee {
    int employeeNumber;
    String fistname;
    String lastname;
    Date dob;
    int carNumber;
}
```

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|---|---|---|---|---|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

# SQL Select Statement

SELECT * FROM Employees

- Results:

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|---|---|---|---|---|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

# SQL

- Why do not we do it in Java?

- Too slow

- Java is programming language
- *not a retrieval/modification/processing language*

- SQL more efficient

- Mathematical model

- FAST!

- Does not know the concept of objects/polymorphism ☹

# Select Statement

```
SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number IS NOT NULL
```

- Results:

| FIRST_NAME | LAST_NAME |
|------------|-----------|
| Axel | Washington |
| Florence | Wojokowski |

Original data:

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|-----------------|------------|-----------|---------------|------------|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

# Select Statement

SELECT First_Name, Last_Name
FROM Employees
WHERE Last_Name LIKE 'Washington%'

- Results:

| FIRST_NAME | LAST_NAME |
|------------|-----------|
| Axel | Washington |
| Sean | Washington |

Original data:

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|-----------------|-----------|-----------|---------------|------------|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

# Select Statement

SELECT First_Name, Last_Name
FROM Employees
WHERE Car_Number = 12

■ Results:

| FIRST_NAME | LAST_NAME |
|---|---|
| Florence | Wojokowski |

Original data:

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|---|---|---|---|---|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

# Join – connect two tables (over ID)

- Cars:

| Car_Number | Make | Model | Year |
|---|---|---|---|
| 5 | Honda | Civic DX | 1996 |
| 12 | Toyota | Corolla | 1999 |

Employee:

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|---|---|---|---|---|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

```
SELECT Employees.First_Name,
  Employees.Last_Name, Cars.Make,
  Cars.Model, Cars.Year FROM Employees, Cars
WHERE Employees.Car_Number = Cars.Car_Number
```

- Cars:

| Car_Number | Make | Model | Year |
|---|---|---|---|
| 5 | Honda | Civic DX | 1996 |
| 12 | Toyota | Corolla | 1999 |

| FIRST_NAME | LAST_NAME | MAKE | MODEL | YEAR |
|---|---|---|---|---|
| Axel | Washington | Honda | Civic DX | 1996 |
| Florence | Wojokowski | Toyota | Corolla | 1999 |

Employee:

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|---|---|---|---|---|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

17

```
SELECT Employees.First_Name,
  Employees.Last_Name, Cars.Make,
  Cars.Model, Cars.Year FROM Employees JOIN Cars
ON (Employees.Car_Number = Cars.Car_Number)
WHERE Cars.Make like 'Hon%'
```

■ Cars:

| Car_Number | Make | Model | Year |
|---|---|---|---|
| 5 | Honda | Civic DX | 1996 |
| 12 | Toyota | Corolla | 1999 |

| FIRST_NAME | LAST_NAME | MAKE | MODEL | YEAR |
|---|---|---|---|---|
| Axel | Washington | Honda | Civic DX | 1996 |

Employee:

| Employee_Number | First_name | Last_Name | Date_of_Birth | Car_Number |
|---|---|---|---|---|
| 10001 | Axel | Washington | 28-Aug-43 | 5 |
| 10083 | Arvid | Sharma | 24-Nov-54 | null |
| 10120 | Jonas | Ginsberg | 01-Jan-69 | null |
| 10005 | Florence | Wojokowski | 04-Jul-71 | 12 |
| 10099 | Sean | Washington | 21-Sep-66 | null |
| 10035 | Elizabeth | Yamaguchi | 24-Dec-59 | null |

18

# Common SQL commands

- **Data Modification:**

- SELECT — used to query and display data from a database. The SELECT statement specifies which columns to include in the result set. The vast majority of the SQL commands used in applications are SELECT statements.

- INSERT — adds new rows to a table. INSERT is used to populate a newly created table or to add a new row (or rows) to an already-existing table.

- DELETE — removes a specified row or set of rows from a table

- UPDATE — changes an existing value in a column or group of columns in a table

# Common SQL commands

- **Data Definition:**

- CREATE TABLE — creates a table with the column names the user provides. The user also needs to specify a type for the data in each column. Data types vary. CREATE TABLE is normally used less often than the data manipulation commands because a table is created only once, whereas adding or deleting rows or changing individual values generally occurs more frequently.

- DROP TABLE — deletes all rows and removes the table definition from the database.

- ALTER TABLE — adds or removes a column from a table. It also adds or drops table constraints and alters column attributes

# JDBC structures

- **Result Sets**
  - *The rows that satisfy the conditions of a query are called the result set.*
  - *The number of rows returned in a result set can be zero, one, or many.*
- **Cursors**
  - *A user can access the data in a result set one row at a time, and a cursor provides the means to do that.*
  - *A cursor can be thought of as a pointer into a file that contains the rows of the result set, and that pointer has the ability to keep track of which row is currently being accessed.*

# Example

```
public void connectToAndQueryDatabase(String username, String password) {

    Connection con = DriverManager.getConnection(
            "jdbc:myDriver:myDatabase",
            username,
            password);

    Statement stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1");

    while (rs.next()) {
        int x = rs.getInt("a");
        String s = rs.getString("b");
        float f = rs.getFloat("c");
    }
}
```

```java
public static void viewTable(Connection con, String dbName) throws SQLException {
    Statement stmt = null;
    String query = "select COF_NAME, SUP_ID, PRICE, SALES, TOTAL "
                    + "from " + dbName + ".COFFEES";
        try {
         stmt = con.createStatement();
         ResultSet rs = stmt.executeQuery(query);
         while (rs.next()) {
             String coffeeName = rs.getString("COF_NAME");
             int supplierID = rs.getInt("SUP_ID");
             float price = rs.getFloat("PRICE");
             int sales = rs.getInt("SALES");
             int total = rs.getInt("TOTAL");
             System.out.println(coffeeName + "\t" + supplierID +
                 "\t" + price + "\t" + sales + "\t" + total);
         }
    } catch (SQLException e ) {
         JDBCTutorialUtilities.printSQLException(e);
    } finally {
         if (stmt != null){ stmt.close(); }
    }
}
```

# JDBC execute a query

- To execute a query, call an execute method from Statement such as the following:

- `executeQuery`: **Returns one ResultSet object.**

- `executeUpdate`: Returns an integer representing the number of rows affected by the SQL statement. Use this method if you are using **INSERT, DELETE, or UPDATE** SQL statements.

- `execute`: Returns true if the first object that the query returns is a ResultSet object. False if an int value or not value is returned. Any kind of statement.

# JDBC result set

```
String query = "select COF_NAME, SUP_ID, PRICE, SALES, TOTAL "
                + "from " + dbName + ".COFFEES";
try {
    stmt = con.createStatement();
    ResultSet rs = stmt.executeQuery(query);
    while (rs.next()) {
        String coffeeName = rs.getString("COF_NAME");
        int supplierID = rs.getInt("SUP_ID");
        float price = rs.getFloat("PRICE");
        int sales = rs.getInt("SALES");
        int total = rs.getInt("TOTAL");
        System.out.println(coffeeName + "\t" + supplierID +
            "\t" + price + "\t" + sales + "\t" + total);
    }
}
```

# JDBC clean up – close the connection

```
} finally {
    if (stmt != null){ stmt.close(); }
}
```

```java
public static void viewTable(Connection con) throws SQLException {

    String query = "select COF_NAME, SUP_ID, PRICE, " +
                   "SALES, TOTAL " + "from COFFEES";

    try (Statement stmt = con.createStatement()) {

        ResultSet rs = stmt.executeQuery(query);

        while (rs.next()) {
            String coffeeName = rs.getString("COF_NAME");
            int supplierID = rs.getInt("SUP_ID");
            float price = rs.getFloat("PRICE");
            int sales = rs.getInt("SALES");
            int total = rs.getInt("TOTAL");
            System.out.println(coffeeName + ", " + supplierID +
                ", " + price + ", " + sales + ", " + total);
        }
    } catch (SQLException e) {
        JDBCTutorialUtilities.printSQLException(e);
    }
}
```

# Create a table

```sql
-- SQL

create table SUPPLIERS (
    SUP_ID integer NOT NULL,
    SUP_NAME varchar(40) NOT NULL,
    STREET varchar(40) NOT NULL,
    CITY varchar(20) NOT NULL,
    STATE char(2) NOT NULL,
    ZIP char(5),
    PRIMARY KEY (SUP_ID));
```

```java
public void createTable() throws
SQLException {
    String createString =
    "create table " + dbName +
        ".SUPPLIERS " +
        "(SUP_ID integer NOT NULL, " +
        "SUP_NAME varchar(40) NOT NULL, " +
        "STREET varchar(40) NOT NULL, " +
        "CITY varchar(20) NOT NULL, " +
        "STATE char(2) NOT NULL, " +
        "ZIP char(5)," +
        "PRIMARY KEY (SUP_ID))";

    Statement stmt = null;
    try{
        stmt = con.createStatement();
        stmt.executeUpdate(createString);
    } catch (SQLException e) {
        JDBCTutorialUtilities
            .printSQLException(e);
    } finally {
        if (stmt != null){ stmt.close(); }
    }
}
```

# Populate table

```sql
insert into SUPPLIERS values(
    49, 'Superior Coffee',
    '1 Party Place','Mendocino',
    'CA', '95460');
insert into SUPPLIERS values(
    101, 'Acme, Inc.',
    '99 Market Street','Grosville',
    'CA','95199');
```

```java
public void populateTable() throws Exception {
    Statement stmt = null;
    try{
            stmt = con.createStatement();
            stmt.executeUpdate(
                "insert into " + dbName +
                ".SUPPLIERS " +
                "values(49, 'Superior Coffee'," +
                "'1 Party Place'," +
                "'Mendocino', 'CA', '95460')");

            stmt.executeUpdate(
                "insert into " + dbName +
                ".SUPPLIERS " +
                "values(101, 'Acme, Inc.'," +
                "'99 Market Street'," +
                "'Groundsville', 'CA', '95199')");
    } catch (SQLException e) {
            JDBCTutorialUtilities
                .printSQLException(e);
    } finally {
        if (stmt != null){ stmt.close();}
    }
}
```

## Update

```java
public void updatePrice(float price, String cofName,
            String username, String password)
        throws SQLException{

        Connection con;
        PreparedStatement pstmt;
        try{
            con = ds.getConnection(username, password);
            con.setAutoCommit(false);
            pstmt = con.prepareStatement("UPDATE COFFEES "
                + "SET PRICE = ? " + "WHERE COF_NAME = ?");
            pstmt.setFloat(1, price);
            pstmt.setString(2, cofName);
            pstmt.executeUpdate();

            con.commit();
            pstmt.close();

        } finally {
            if (con != null) {con.close();}
        }
    }
```

# Delete

```
stmt = conn.createStatement();
String sql = "DELETE FROM Registration " + "WHERE id = 101";
stmt.executeUpdate(sql);
```
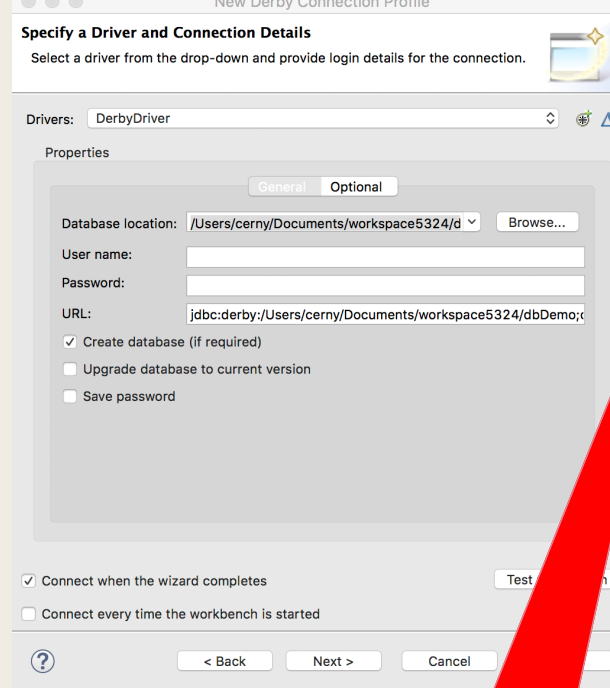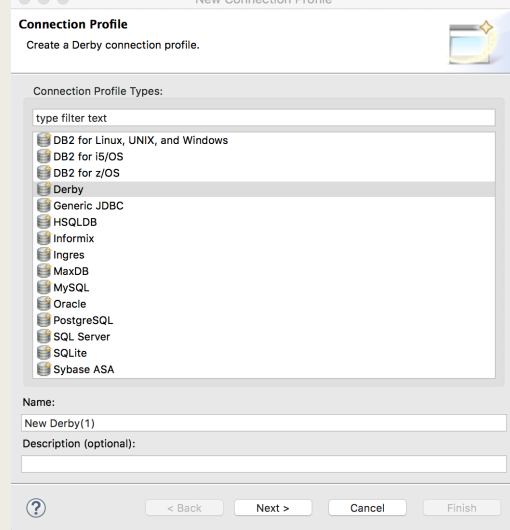
# Apache Derby

- The most simple file-based SQL database!

- Good for practice or standalone apps with a single connection

- Embedded driver does not support multiple connections!

- open source relational database implemented entirely in Java

- Derby has a small footprint -- about 3.5 megabytes

- Derby is based on the Java, JDBC, and SQL standards.

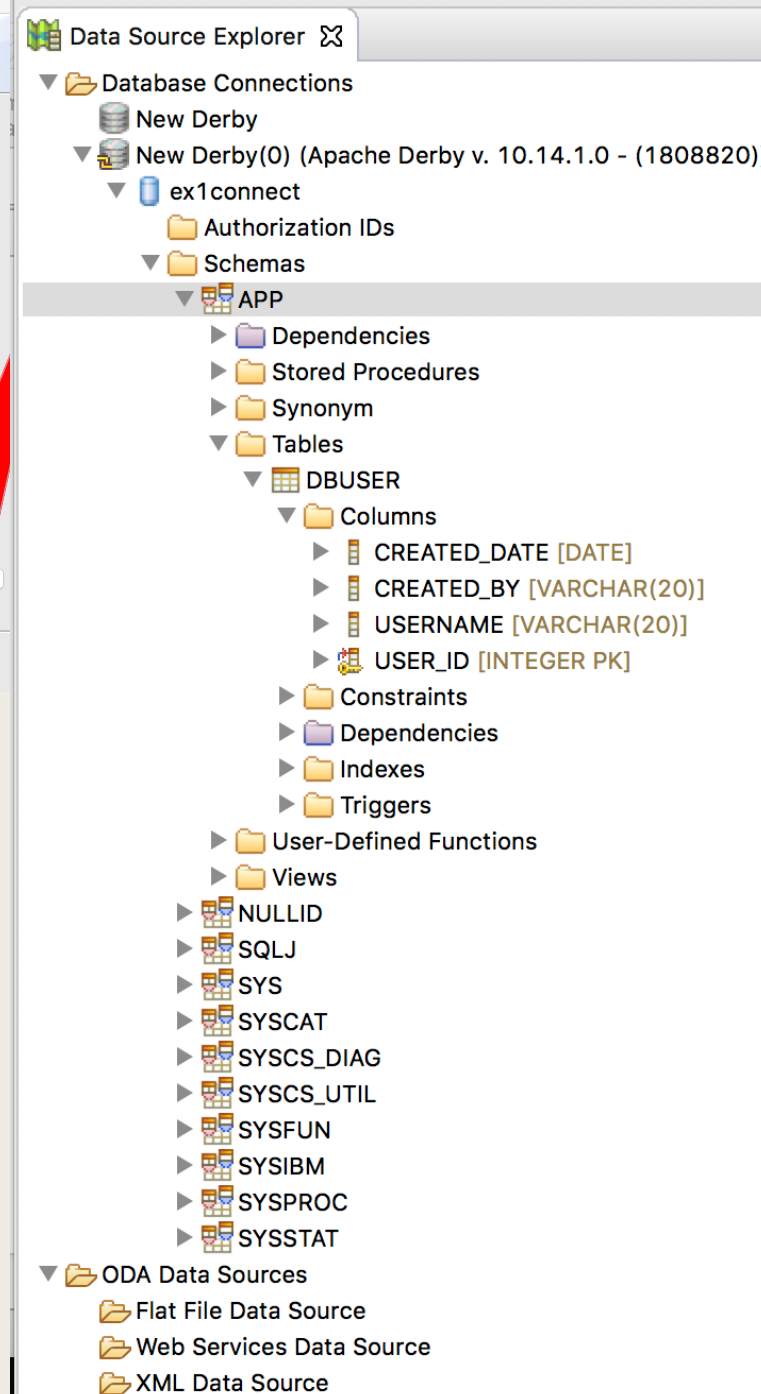- Derby is easy to install, deploy, and use.

# Apache Derby

- ■ Consider dbDemo app from canvas
- – *run*
- ■ Ex1Connect
- ■ Ex1Connect2

- ■ Explain what was the issue

- ■ Run

- ■ Ex2CreateTable

- ■ Ex3InsertRow

# Eclipse

- Menu Window

- Perspective | Open Perspective | Other…

- Pick Databased development

- Right Click Database connections | New… | Derby | Next

- Add driver
  – *pick Embedded latest | call it DerbyDriver |*
  – *| tab JAR list | add JAR : ./lib/derby.jar*
  – *remove any preexisting JAR | hit OK*

- Database location | pick your DB
  – *(e.g. /Users/cerny/Documents/workspace5324/dbDemo/ex1connect)*

- Leave user/pass empty | Next | Finish

Remember to disconnect!!!!

■ Right click DBUser table | Data | Edit



| USER_ID [INTEGER] | USERNAME [VARCHAR(20)] | CREATED_BY [VARCHAR(20)] | CREATED_DATE [DATE] |
|---|---|---|---|
| 1 | testName | system | 2018-03-19 |
| <new row> | | | |

# Example

- Try to call again Ex3InserRow

- What is the issue?


- Call Ex4Select


- Call Ex5Update and again Ex4Select

- Call Ex6Delete and again Ex4Select