Lecture (12)
Database I

Dr. Ahmed ElShafee

Agenda

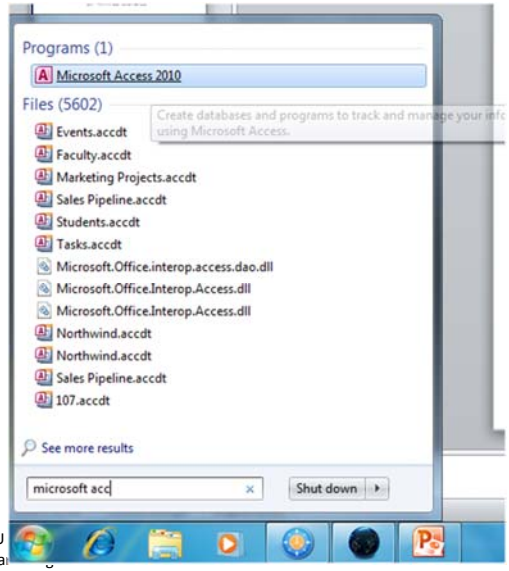
- *
- *
- *

Introduction

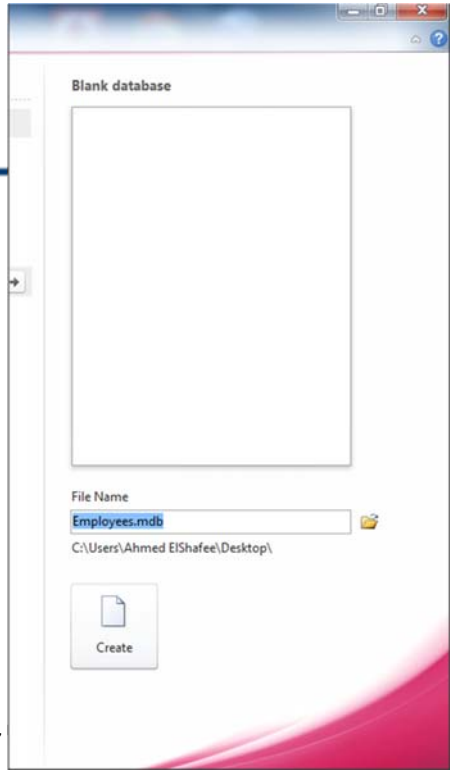
- Java uses something called JDBC (Java Database Connectivity) to connect to databases.
- There's a JDBC API, which is the programming part, and a JDBC Driver Manager, which your programs use to connect to the database.
- JDBC allows you to connect to a wide-range of databases (Oracle, MySQL, etc),
- We are going to build a simple MicroSoft Access database ".mdb" .
- Then access it through java using jdbc driver

Working with MicroSoft Access

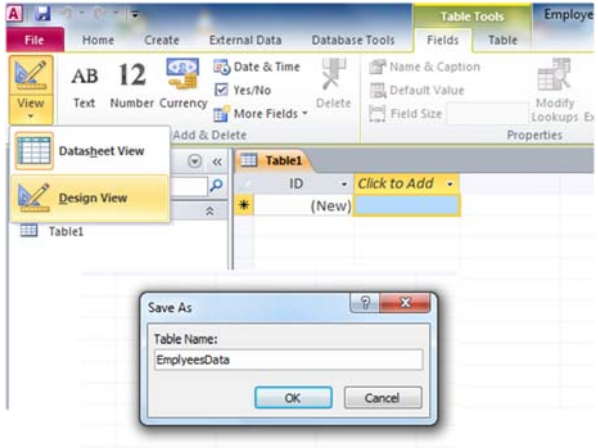
- Open MicroSoft Access



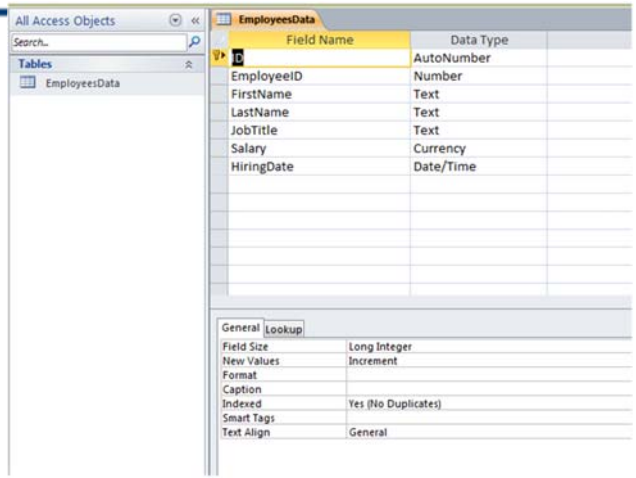
- Select Database file name/type/location
- Then press create



- Select design view from view menu
- Type table name, then press “OK”



- Add fields as shown in figure



- Go to database view

- Add the following data

Connecting DB using java code

- Note that the main methods “throws SQLException”

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

- Create a string variable contains the database file path as follows

```
String database = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=D:\\Employees.mdb";
```

- Note that “DBQ=“ refers to database file path.

- Java provide two classes called “Connection” , and “DriverManager” that used in connecting databses as follows

```
Connection con = DriverManager.getConnection( host, username, password );
```

```
Connection conn = DriverManager.getConnection(database, "", "");
```

- The blanks followed by database parameter is the username and password if required to connect to the database file.
- Now create an object from connection class that returns from connection object created in the last step

```
Statement s = conn.createStatement();
```

- Now you are connected to the database
- The next step ot to brows database files tables

- When you write the following statement

- As you can see in the image above, there is a wavy underline for the Connection code.
- The reason for this is because we haven’t trapped a specific error that will be thrown up when connecting to a database – the **SQLException error.**
- It’s the DriverManager that attempts to connect to the database.
- If it fails (incorrect host address, for example) then it will hand you back a **SQLException** error.

- You need to write code to deal with this potential error.
- In the code below, we’re trapping the error in **catch part of the try … catch statement:**

```
try {
}
catch ( SQLException err ) {
System.out.println( err.getMessage( ) );
}
```

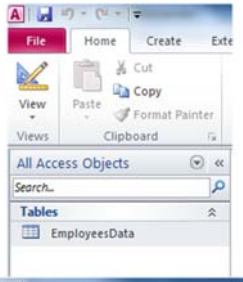
- In between the round brackets of catch, we’ve set up a SQLException object called **err.**
- We can then use the **getMessage** method of this **err** object**.**
- Add the above **try …catch** block to your own code, and move your four connection lines of code to the **try** part**.**

- Code will be as follows

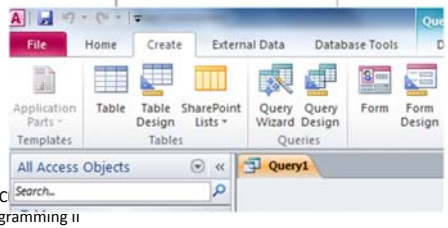
```
/**...*/
public static void main(String[] args) {
    // TODO code application logic here
    try{
            Connection con = DriverManager.getConnection( host, uName, uPass );
    }
    catch(SQLException err){
    System.out.println(err.getMessage());
    }
```


Basic SQL statements

- SQL "Sequential Query Language" is a programming language that is designed to deal with database files.
- Returning to Miscosoft Access.
- Open "Employees.mdb" file again.
- Open the "EmployeesData" table:

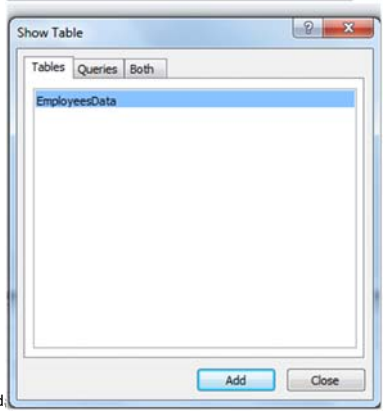


- Open create tab



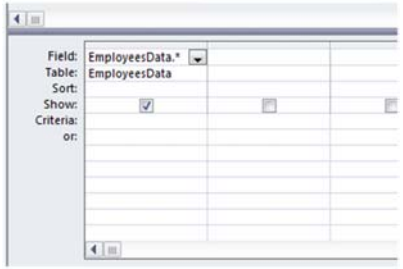
- Open query design

- Select table of that query, press add, then press close

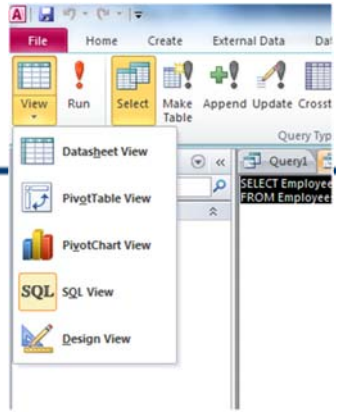


- Select fields you want to display
- Or double click in "*" to display all fields

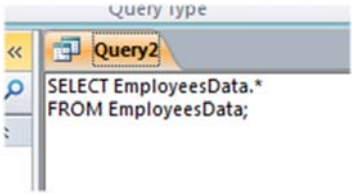
- Selected fields will appear on the following table



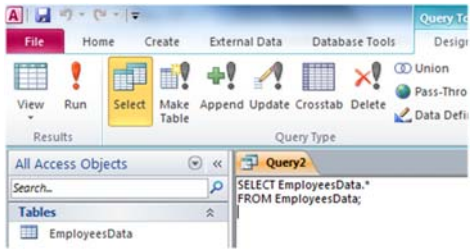
- Go to view tab, select SQL view



- The following SQL statement will appear

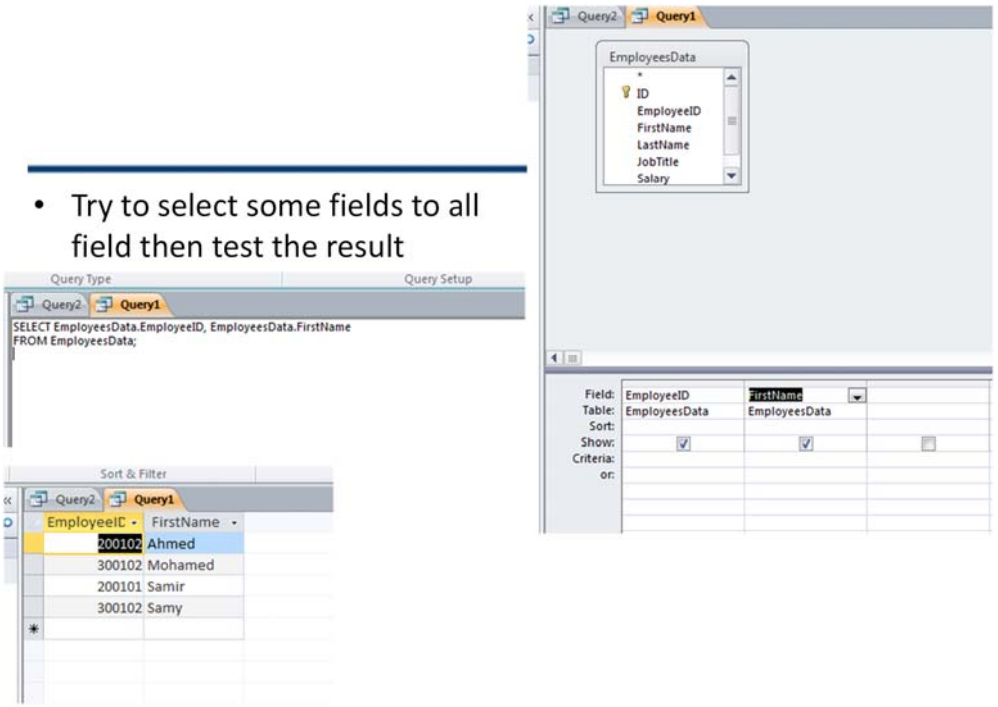


- Press run button



- The following table will appear

- Try to select some fields to all field then test the result



Add SQL statements to your java code

- Now we agreed that to display all fields and all records on the table we can use the following statement

```
SELECT * FROM EmployeesData
```

- To execute that statement, you will use the statement object created in the last section to execute the sql statement

```
String selTable = "SELECT * FROM EmployeesData";
 s.execute(selTable);
```

ResultSets

- A **ResultSet** is a way to store and manipulate the records returned from a SQL query.
- **ResultSets** come in three different types.
- The type you use depends on what you want to do with the data:
- 1) Do you just want to move forward through the records, from beginning to end?
- 2) Do you want to move forward AND backward through the records, as well as detecting any changes made to the records?

(3) Do you want to move forward AND backward through the records, but are not bothered about any changes made to the records

- Type number 1 on the list above is called a **TYPE_FORWARD_ONLY ResultSet**.
- Number 2 on the list is a **TYPE_SCROLL_SENSITIVE ResultSet**.
- The third ResultSet option is called **TYPE_SCROLL_INSENSITIVE.**
- The ResultSet type goes between the round brackets of createStement:

```
Statement stmt = con.createStatement( );
```


- Because we've left the round brackets empty, we'll get the default RecordSet, which is **TYPE_FORWARD_ONLY**.
- we'll use one of the other types.
- But you use them like this:

```
Statement stmt =
con.createStatement( RecordSet.TYPE_SCROLL_SENSITIVE );
```

- So you first type the word RecordSet. After a dot, you add the RecordSet type you want to use.

- you also need to specify whether the **ResultSet** is Read Only or whether it is Updatable.
- You do this with two built-in constants: **CONCUR_READ_ONLY and CONCUR_UPDATABLE**.
- Again, these come after the word RecordSet:

```
ResultSet.CONCUR_READ_ONLY
ResultSet.CONCUR_UPDATABLE
```

- The final code will be,…

```
Statement stmt =con.createStatement(
ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
```

- One more thing to get used to with **ResultSets** is something called a **Cursor**.
- A **Cursor** is really just a pointer to a table row.
- When you first load the records into a **ResultSet**, the **Cursor** is pointing to just before the first row in the table.
- You then use methods to manipulate the **Cursor**.
- But the idea is to identify a particular row in your table.

Using a ResultSet

- Once you have all the records in a Results set, there are methods you can use to manipulate your records.
- Here are the methods you'll use most often:

Next	Moves the Cursor to the next row in your table. If there are no more rows in the table, a value of False will be returned.
Previous	Moves the Cursor back one row in your table. If there are no more rows in the table, a value of False will be returned.

first	Moves the Cursor to the first row in your table
last	Moves the Cursor to the last row in your table
Absolute	Moves the Cursor to a particular row in the table. So absolute(5) will move the Cursor to row number 5 in the table

- The ResultSet also has methods you can use to identify a particular column (field) in a row.
- You can do so either by using the name of the column, or by using its index number.
- For our Workers table we set up four columns.
- They had the following names: **ID, First_Name, Last_Name, and Job_Title**.
- The index numbers are therefore 1, 2, 3, 4.

- We set up the ID column to hold Integer values. The method you use to get at integer values in a column is **getInt**:

```
int id_col = rs.getInt("ID");
```

- We could use the Index number instead:

```
int id_col = rs.getInt(1);
```


- For the other three columns in our database table, we set them up to hold Strings.
- We, therefore, need the getString method:

```
String first_name = rs.getString("First_Name");
```

- Or we could use the Index number:

```
String first_name = rs.getString(2);
```


- Because the ResultSet Cursor is pointing to just before the first record when the data is loaded, we need to use the **next** method to move to the first row.
- The following code will get the first record from the table:

```
rs.next( );
int id_col = rs.getInt("ID");
String first_name = rs.getString("First_Name");
String last_name = rs.getString("Last_Name");
String job = rs.getString("Job_Title");
```

- You can add a print line to your code

```
System.out.println( id_col + " " + first_name + " " + last_name + " " + job );
```


- Now it's the time to fetch the results, which in this case the database table contents, fields X rows
- Create an object from "ResultSet" class, which is responsible of fetching data from database tables

```
ResultSet rs = s.getResultSet();
```

- Now get data from records as follows

```
while ((rs != null) && (rs.next())) {
            System.out.println(rs.getString(1) + " : " + rs.getString(2)+" : " +
rs.getString(3)+" : " + rs.getString(4)+" : " + rs.getString(5)+" : " +
rs.getString(6));
        }
```

- Now close the connection and statement to free database

```
s.close();  // Close the statement
        conn.close(); // Close the database. Its no more required
```



```
public static void main(String[] args) {
    // TODO code application logic here
    try{
            Connection con = DriverManager.getConnection( host, uName, uPass );
            Statement stmt =con.createStatement( ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
            String SQL = "SELECT * FROM Workers";
            ResultSet rs = stmt.executeQuery( SQL );
            rs.next( );
             int id_col = rs.getInt("ID");
             String first_name = rs.getString("First_Name");
             String last_name = rs.getString("Last_Name");
             String job = rs.getString("Job_Title");
             System.out.println( id_col + " " + first_name + " " + last_name + " " + job );


    }
    catch(SQLException err){
    System.out.println(err.getMessage());
    }
```


Lecture1201

- Check lab manual

Thanks,

See you next Week, isA