Essentials of Information Technology

PC-CS-305

Java Database Connectivity

Topic & Structure of the lesson

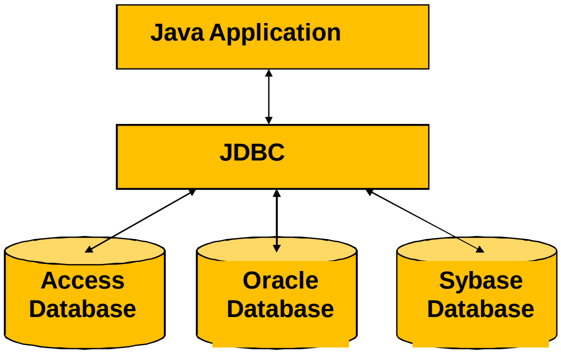
- In this lecture, we will
  - JDBC Concepts
  - Steps during execution of JDBC Application
  - JDBC Component Interaction
  - Steps Involved in Basic JDBC Operations
  - Two-Tier Database Access Model
  - JDBC Driver Types
  - Driver Manager,Connection Object,DSN-less Connection
  - Statement Object
  - ResultSet Object
  - Accessing Data in a ResultSet
  - Getting MetaData for a ResultSet

Java Database Connectivity (JDBC)

- JDBC – provides an interface to Relational Data Sources
- JDBC library provides the means for executing SQL statements to access and operate on a relational database
- JDBC library is implemented in the java.sql package
  - Set of classes and interfaces that provide a uniform API for access to broad range of databases

Talking to Databases

- A JDBC based application is insulated from the characteristics of specific database engines

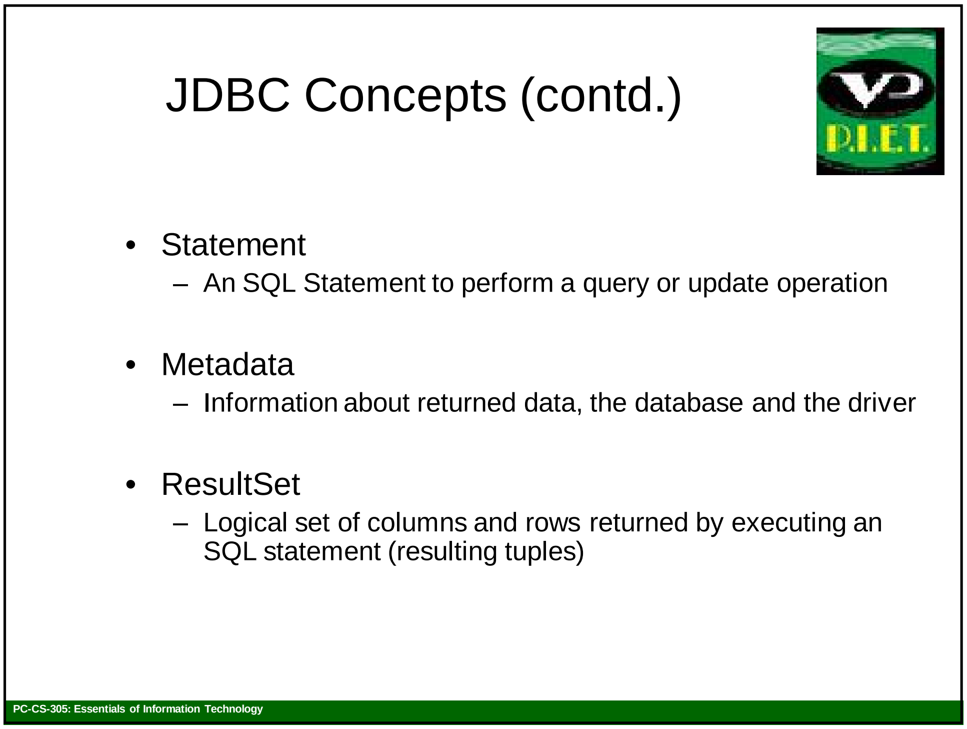


Open Database Connectivity (ODBC)

- is a standard C programming language middleware API for accessing database management systems (DBMS).
- an application written using ODBC can be ported to other platforms,
- ODBC accomplishes DBMS independence by using an ODBC driver as a translation layer between the application and the DBMS.
- The application uses ODBC functions through an ODBC driver manager with which it is linked, and the driver passes the query to the DBMS.

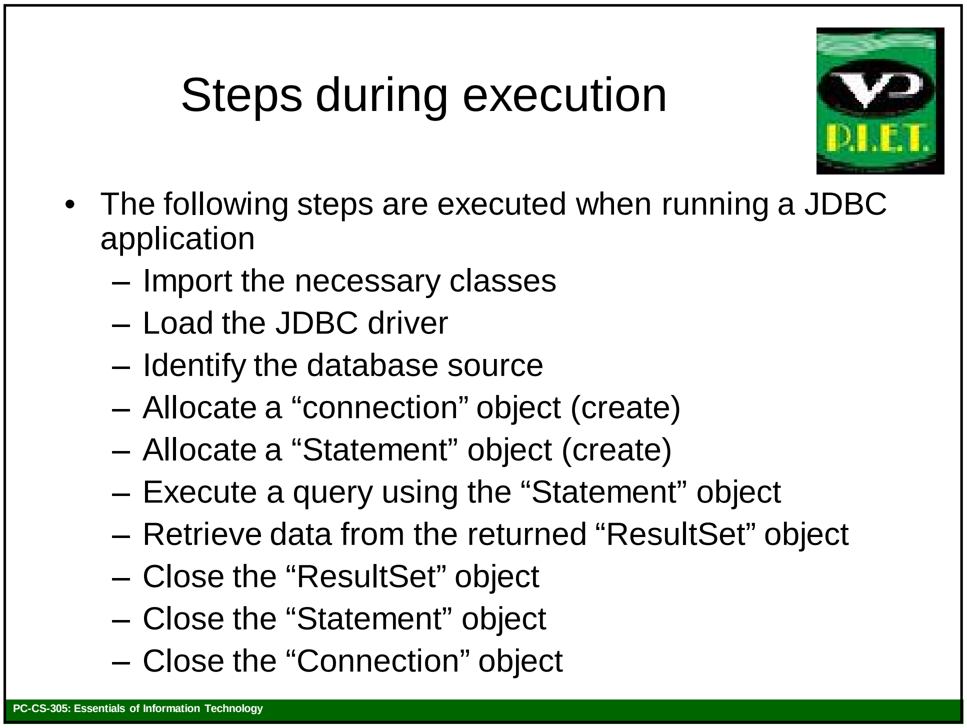
JDBC Concepts

- JDBC's design is very similar to the design of ODBC
- Driver Manager
  - Loads database drivers, and manages the connection between the application and the driver
- Driver
  - Translates API calls into operations for a specific data source
- Connection
  - A session between an application and a database



JDBC Concepts (contd.)

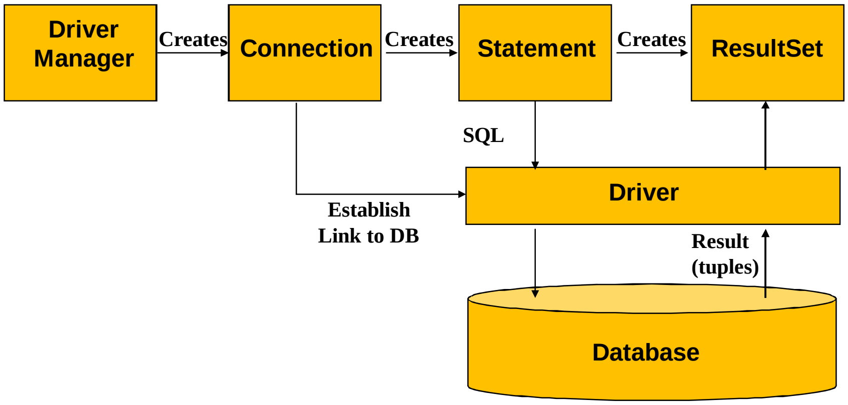
- Statement
  - An SQL Statement to perform a query or update operation
- Metadata
  - Information about returned data, the database and the driver
- ResultSet
  - Logical set of columns and rows returned by executing an SQL statement (resulting tuples)



Steps during execution

- The following steps are executed when running a JDBC application
  - Import the necessary classes
  - Load the JDBC driver
  - Identify the database source
  - Allocate a “connection” object (create)
  - Allocate a “Statement” object (create)
  - Execute a query using the “Statement” object
  - Retrieve data from the returned “ResultSet” object
  - Close the “ResultSet” object
  - Close the “Statement” object
  - Close the “Connection” object

JDBC Component Interaction



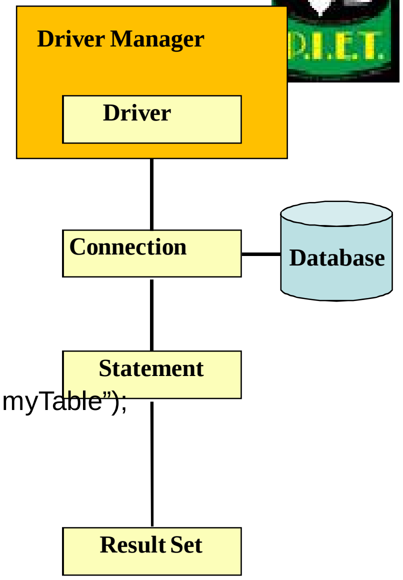
Steps Involved in Basic JDBC Operations

1. Load the JDBC driver class:
   Class.forName("driverName");

2. Open a database connection:
   DriverManager.getConnection ("jdbc:xxx:datasource");

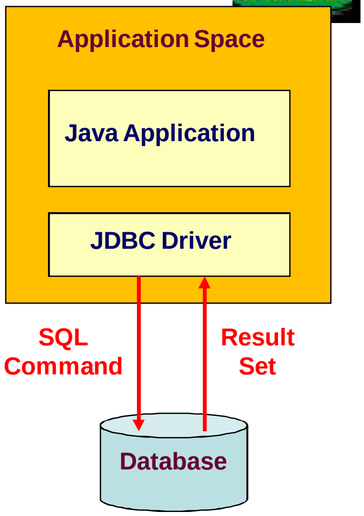
3. Issue SQL statements:
   stmt = con.createStatement();
   stmt.executeQuery ("Select * from myTable");

4. Process result set:
   while (rs.next()) {
   name = rs.getString("name");
   amount = rs.getInt("amt"); }



Two-Tier Database Access Model

- Java Application talks directly to the database
- Accomplished through the JDBC driver which sends commands directly to the database
- Results sent back directly to the application

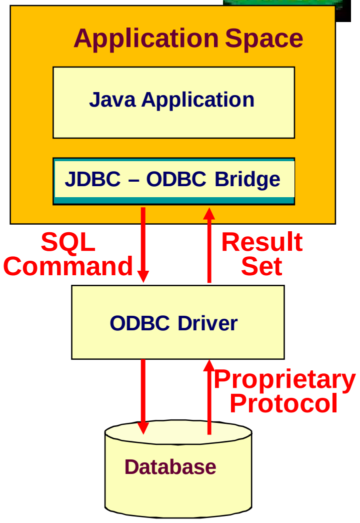


JDBC Driver Types

- **JDBC-ODBC Bridge, plus ODBC driver (Type 1)**
  - Simplest
  - JDBC methods -> Translate JDBC methods to ODBC methods -> ODBC to native methods -> Native methods API
- **Native-API, partly Java driver (Type 2)**
  - JDBC methods -> Map JDBC methods to native methods (calls to vendor library) -> Native methods API (vendor library)
- **JDBC-net, pure Java driver (Type 3)**
  - JDBC methods -> Translate to Native API methods through TCP/IP network -> Native API methods
- **Native-protocol, pure Java driver (Type 4)**
  - Java methods -> Native methods in Java

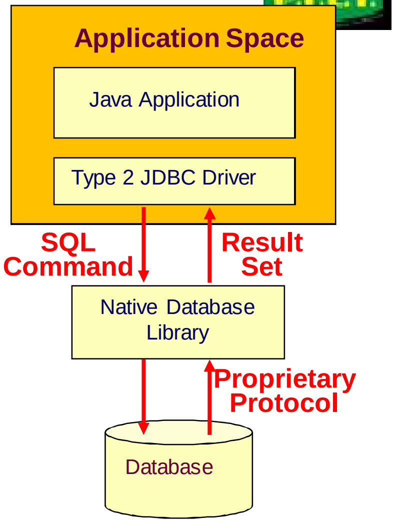
Type 1: JDBC-ODBC Bridge, Plus ODBC Driver

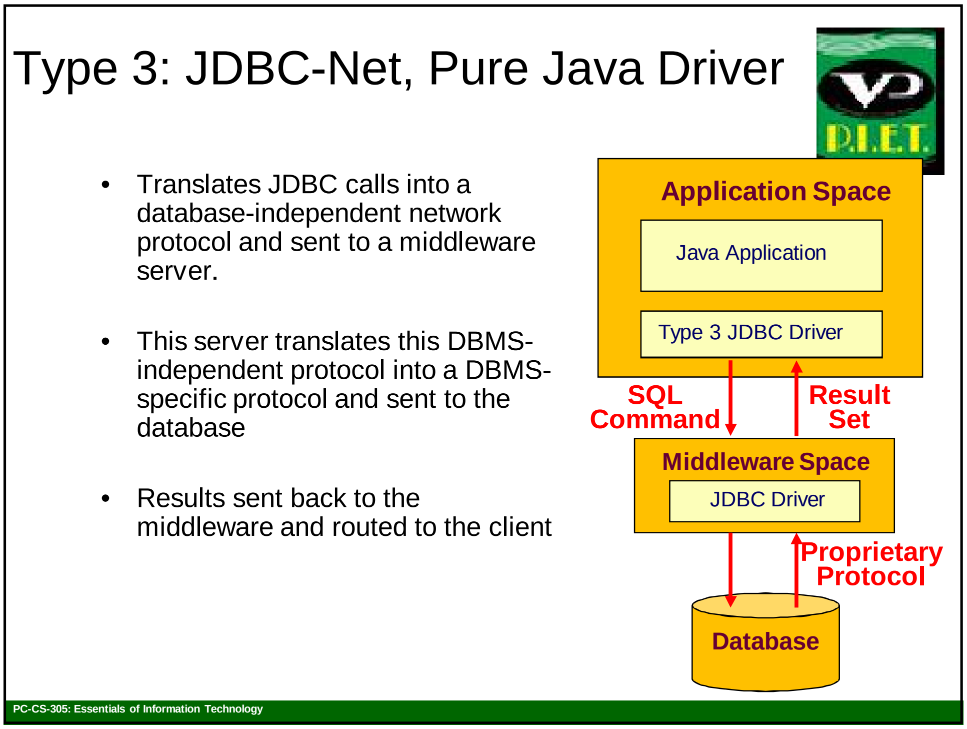
- This driver type is provided by Sun with JDK
- Provides JDBC access to databases through ODBC drivers
- ODBC driver must be configured for the bridge to work
- Only solution if no JDBC driver available for the DBMS



Type 2: Native-API, Partly Java Driver

- Native-API driver converts JDBC commands into DBMS-specific native calls
- Same restrictions as Type1 – must have some binary code loaded on its machine
- Directly interfaces with the database





Type 3: JDBC-Net, Pure Java Driver

- Translates JDBC calls into a database-independent network protocol and sent to a middleware server.
- This server translates this DBMS-independent protocol into a DBMS-specific protocol and sent to the database
- Results sent back to the middleware and routed to the client

Application Space

Java Application

Type 3 JDBC Driver

SQL Command

Result Set

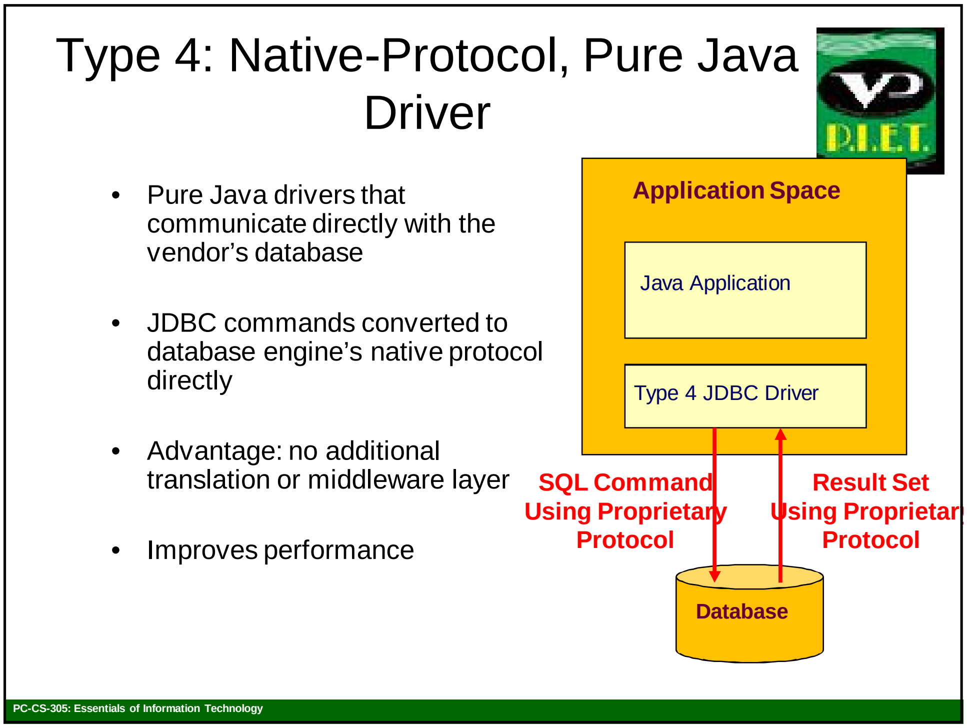
Middleware Space

JDBC Driver

Proprietary Protocol

Database

PC-CS-305: Essentials of Information Technology



Type 4: Native-Protocol, Pure Java Driver

- Pure Java drivers that communicate directly with the vendor's database
- JDBC commands converted to database engine's native protocol directly
- Advantage: no additional translation or middleware layer
- Improves performance

Application Space

Java Application

Type 4 JDBC Driver

SQL Command Using Proprietary Protocol

Result Set Using Proprietary Protocol

Database

PC-CS-305: Essentials of Information Technology

Driver Manager

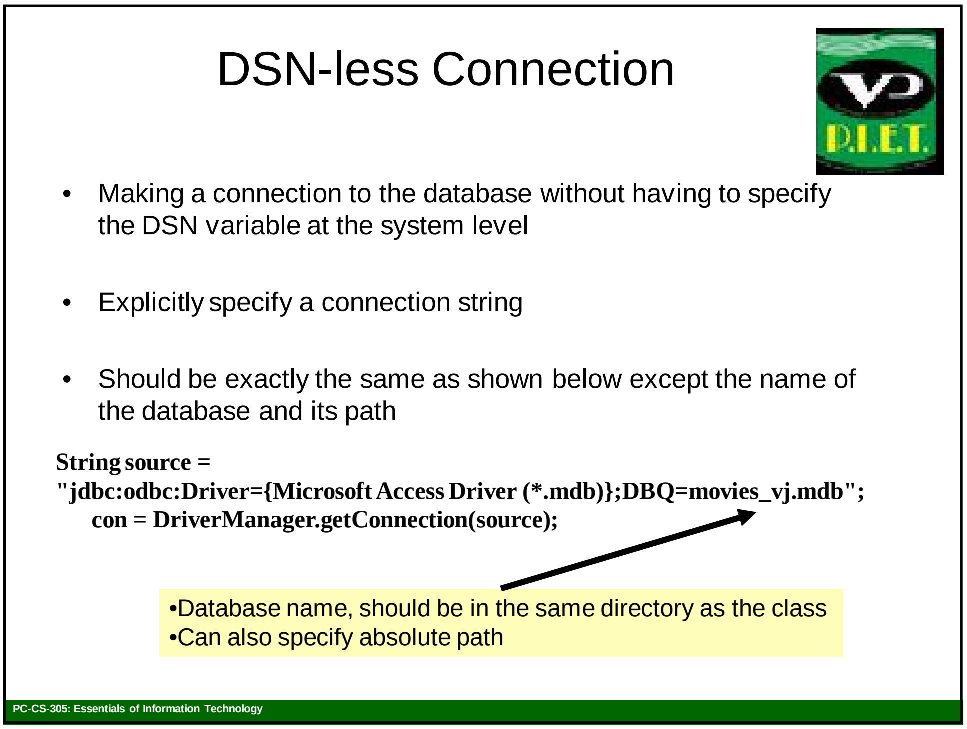
- The DriverManager class is responsible for establishing connections to the data sources, accessed through the JDBC drivers
- The driver can be loaded explicitly by calling the static method “forName” in the “Class” class and pass the driver argument
  - **Eg: Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);**
- The “forName()” method can throw a “ClassNotFoundException” if the driver class is not found. Hence, this function call should be in a try-catch block

Connection Object

- Creating a connection to a data source
- Connection object represents an established connection to a particular data source
- A connection object can also be used to query the data source (data and meta data)
- Different versions of getConnection() method contained in the DriverManager class that returns a connection object:
  - Connection myconn = DriverManager.getConnection(source);
  - Connection myconn = DriverManager.getConnection(source, username, password);
  - **Example**

System DSN name – ODBC data source

String mysource = “jdbc:odbc:technical_library”;
Connection myconn = DriverManager.getConnection(mysource);



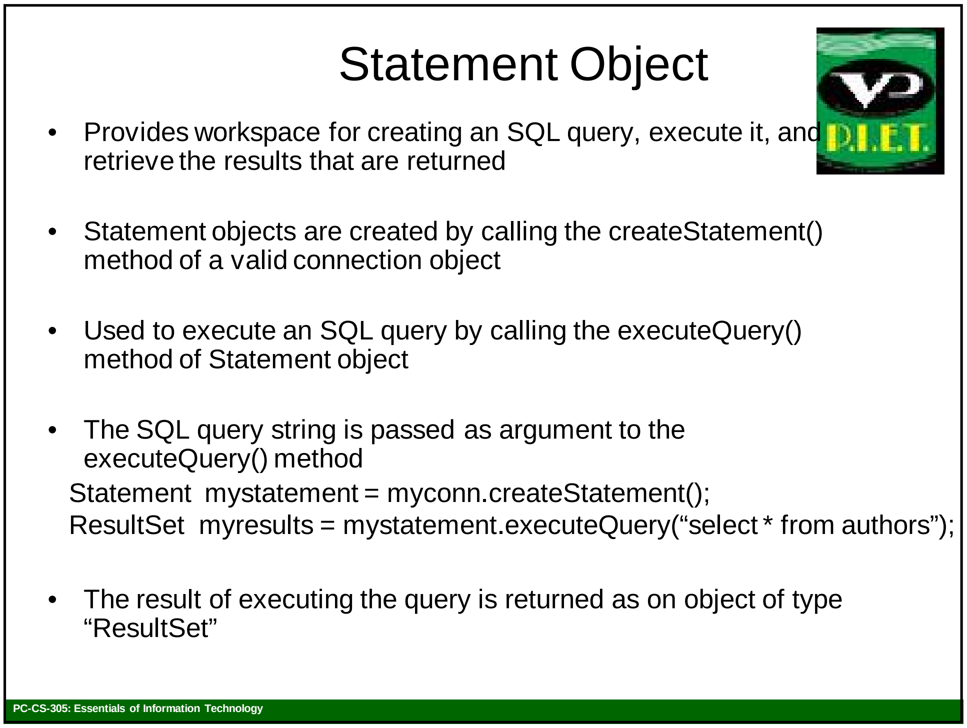
DSN-less Connection

- Making a connection to the database without having to specify the DSN variable at the system level
- Explicitly specify a connection string
- Should be exactly the same as shown below except the name of the database and its path

```
String source =
"jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=movies_vj.mdb";
    con = DriverManager.getConnection(source);
```

•Database name, should be in the same directory as the class
•Can also specify absolute path

PC-CS-305: Essentials of Information Technology



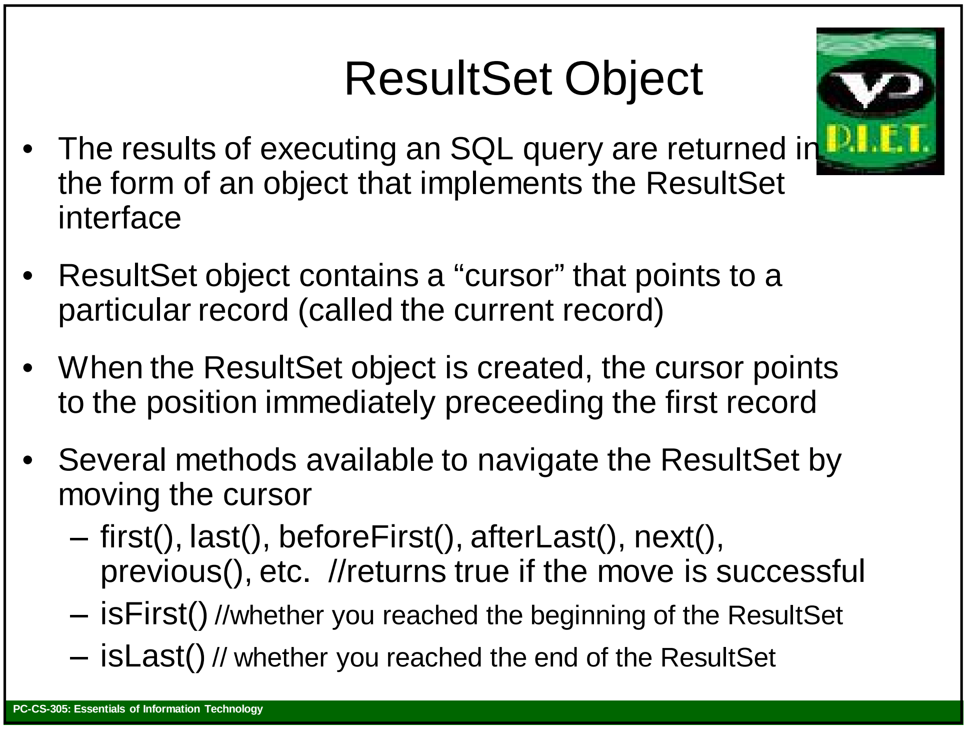
Statement Object

- Provides workspace for creating an SQL query, execute it, and retrieve the results that are returned
- Statement objects are created by calling the createStatement() method of a valid connection object
- Used to execute an SQL query by calling the executeQuery() method of Statement object
- The SQL query string is passed as argument to the executeQuery() method

```
Statement  mystatement = myconn.createStatement();
ResultSet  myresults = mystatement.executeQuery("select * from authors");
```

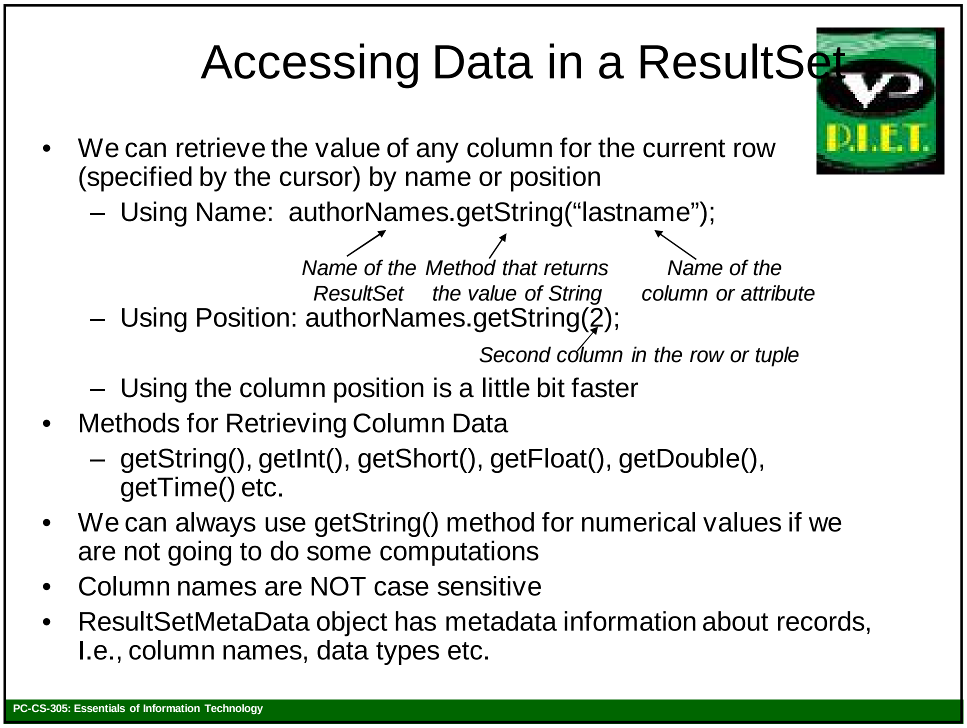
- The result of executing the query is returned as on object of type "ResultSet"

PC-CS-305: Essentials of Information Technology



ResultSet Object

- The results of executing an SQL query are returned in the form of an object that implements the ResultSet interface
- ResultSet object contains a “cursor” that points to a particular record (called the current record)
- When the ResultSet object is created, the cursor points to the position immediately preceeding the first record
- Several methods available to navigate the ResultSet by moving the cursor
  - first(), last(), beforeFirst(), afterLast(), next(), previous(), etc. //returns true if the move is successful
  - isFirst() //whether you reached the beginning of the ResultSet
  - isLast() // whether you reached the end of the ResultSet

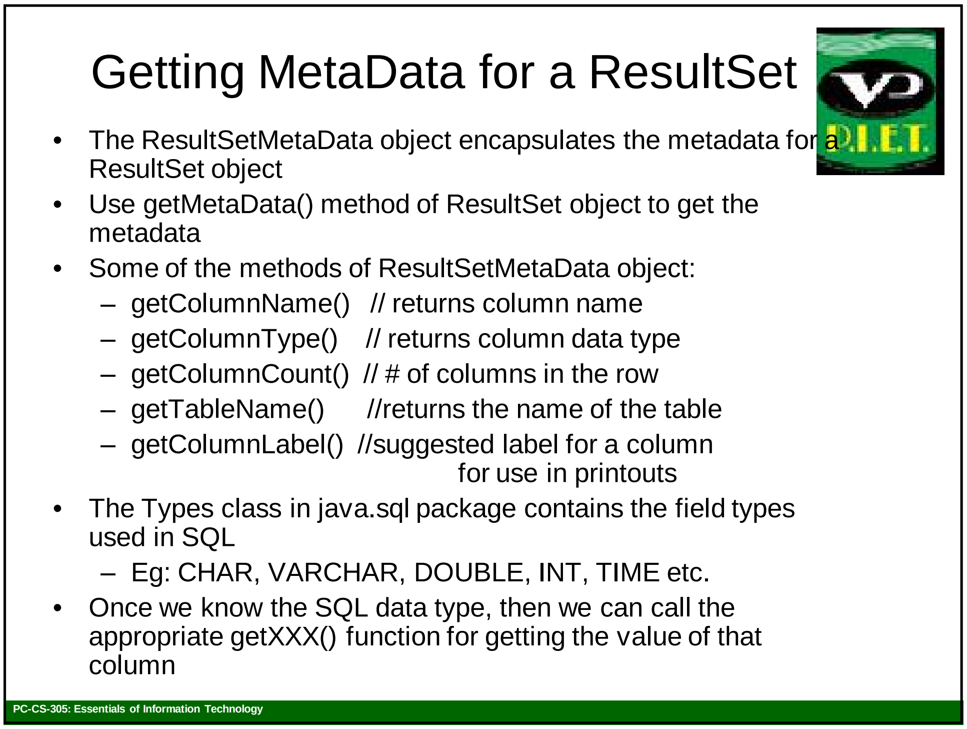


Accessing Data in a ResultSet

- We can retrieve the value of any column for the current row (specified by the cursor) by name or position
  - Using Name: authorNames.getString(“lastname”);

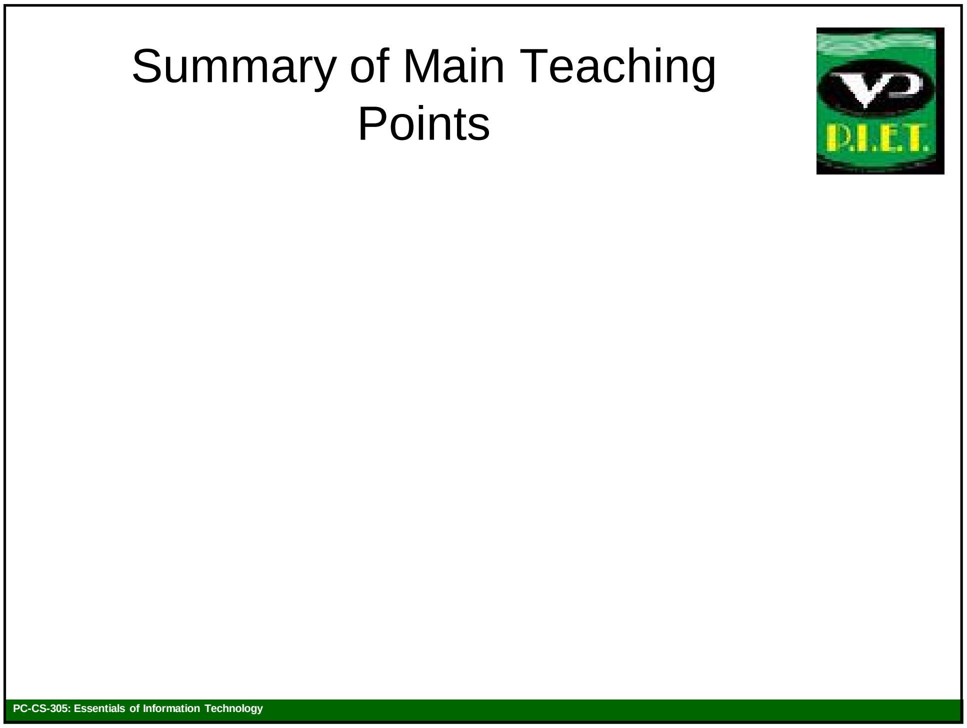
    Name of the ResultSet *Method that returns the value of String* *Name of the column or attribute*
  - Using Position: authorNames.getString(2);

    Second column in the row or tuple
  - Using the column position is a little bit faster
- Methods for Retrieving Column Data
  - getString(), getInt(), getShort(), getFloat(), getDouble(), getTime() etc.
- We can always use getString() method for numerical values if we are not going to do some computations
- Column names are NOT case sensitive
- ResultSetMetaData object has metadata information about records, I.e., column names, data types etc.

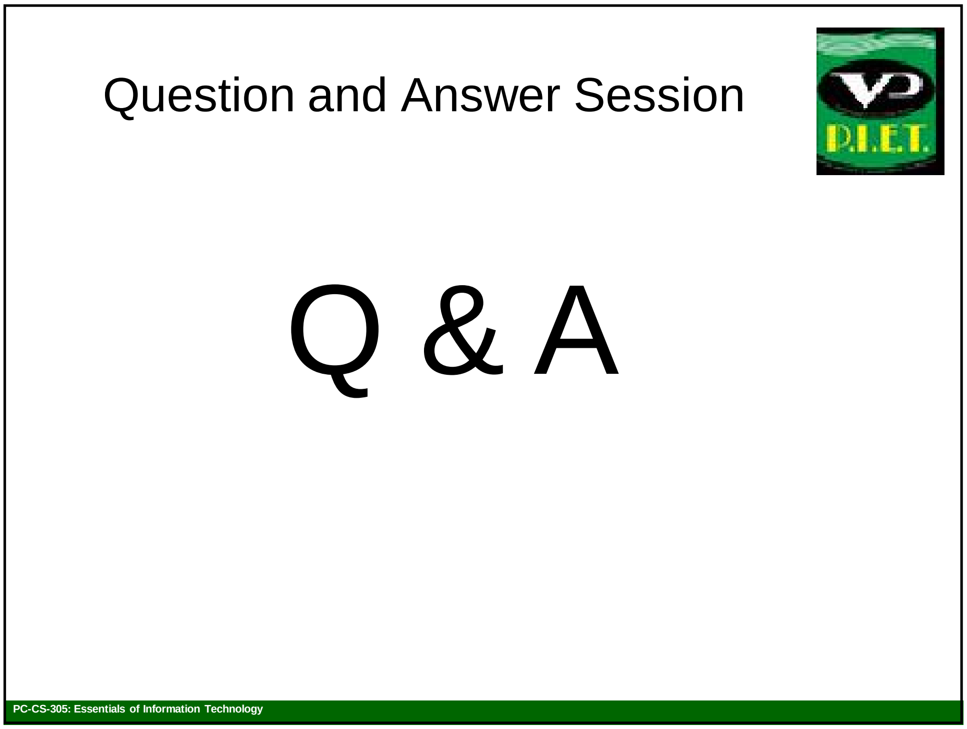


Getting MetaData for a ResultSet

- The ResultSetMetaData object encapsulates the metadata for a ResultSet object
- Use getMetaData() method of ResultSet object to get the metadata
- Some of the methods of ResultSetMetaData object:
  - getColumnName() // returns column name
  - getColumnType() // returns column data type
  - getColumnCount() // # of columns in the row
  - getTableName() //returns the name of the table
  - getColumnLabel() //suggested label for a column for use in printouts
- The Types class in java.sql package contains the field types used in SQL
  - Eg: CHAR, VARCHAR, DOUBLE, INT, TIME etc.
- Once we know the SQL data type, then we can call the appropriate getXXX() function for getting the value of that column



Summary of Main Teaching Points



Question and Answer Session

Q & A

PC-CS-305: Essentials of Information Technology