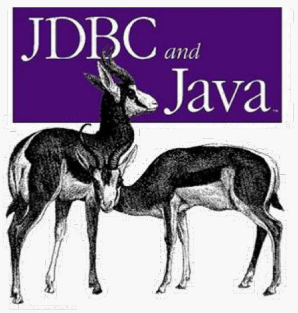
Introduction to SQL Programming Techniques

CSC 375, Fall 2017

The Six Phases of a Project:

Enthusiasm
Disillusionment
Panic
Search for the Guilty
Punishment of the Innocent
Praise for non-participants



Part I: Stored Procedures

Stored Procedures in MySQL

- A stored procedure contains a sequence of SQL commands stored in the database catalog so that it can be invoked later by a program
- Stored procedures are declared using the following syntax:

  Create Procedure <proc-name>

  (param_spec$_1$, param_spec$_2$, …, param_spec$_n$)

  begin

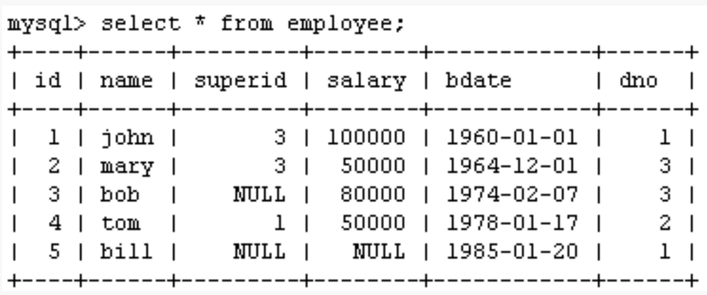
  -- execution code

  end;

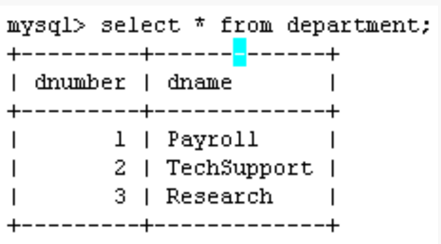
  where each param_spec is of the form:

  [in | out | inout] <param_name> <param_type>
  - in mode: allows you to pass values into the procedure,
  - out mode: allows you to pass value back from procedure to the calling program

Example

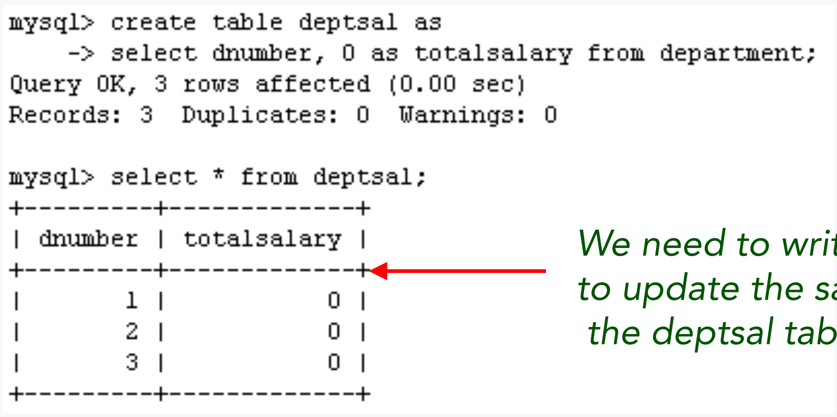


```
mysql> select * from employee;
+----+------+---------+--------+------------+------+
| id | name | superid | salary | bdate      | dno  |
+----+------+---------+--------+------------+------+
|  1 | john |       3 | 100000 | 1960-01-01 |    1 |
|  2 | mary |       3 |  50000 | 1964-12-01 |    3 |
|  3 | bob  |    NULL |  80000 | 1974-02-07 |    3 |
|  4 | tom  |       1 |  50000 | 1978-01-17 |    2 |
|  5 | bill |    NULL |   NULL | 1985-01-20 |    1 |
+----+------+---------+--------+------------+------+
```



```
mysql> select * from department;
+---------+-------------+
| dnumber | dname       |
+---------+-------------+
|       1 | Payroll     |
|       2 | TechSupport |
|       3 | Research    |
+---------+-------------+
```

- Suppose we want to keep track of the total salaries of employees working for each department

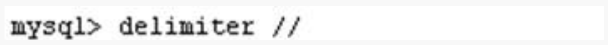


```
mysql> create table deptsal as
    -> select dnumber, 0 as totalsalary from department;
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |           0 |
|       2 |           0 |
|       3 |           0 |
+---------+-------------+
```

We need to write a procedure to update the salaries in the deptsal table

Example

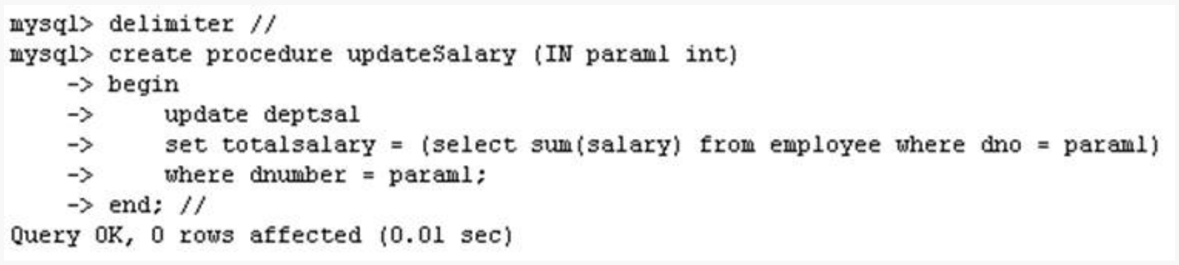


```
mysql> delimiter //
```

Step 1:

Change the delimiter (i.e., terminating character) of SQL statement from semicolon (;) to something else (e.g., //) So that you can distinguish between the semicolon of the SQL statements in the procedure and the terminating character of the procedure definition

Example

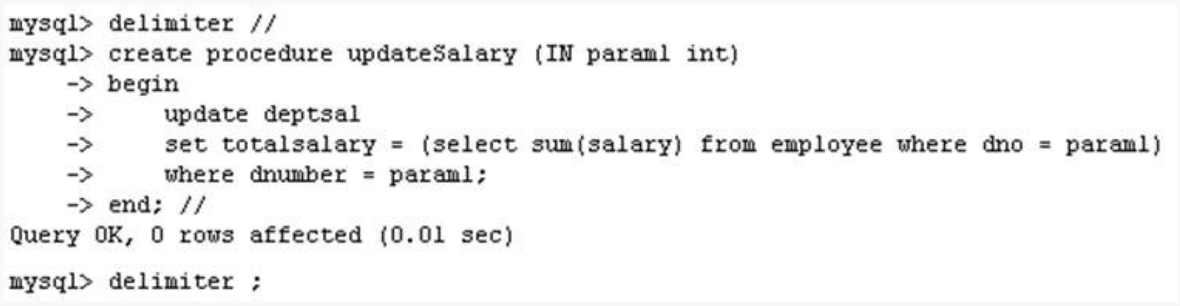


```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)
```

Step 2:

1. Define a procedure called updateSalary which takes as input a department number.
2. The body of the procedure is an SQL command to update the totalsalary column of the deptsal table.
3. Terminate the procedure definition using the delimiter you had defined in step 1 (//)

Example

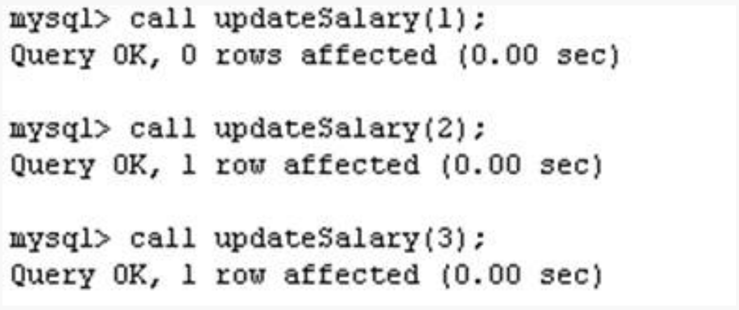


```
mysql> delimiter //
mysql> create procedure updateSalary (IN param1 int)
    -> begin
    ->     update deptsal
    ->     set totalsalary = (select sum(salary) from employee where dno = param1)
    ->     where dnumber = param1;
    -> end; //
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;
```

Step 3: Change the delimiter back to semicolon (;)

Example



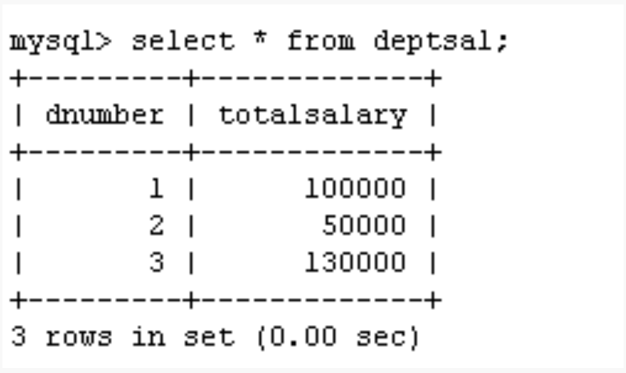
```
mysql> call updateSalary(1);
Query OK, 0 rows affected (0.00 sec)

mysql> call updateSalary(2);
Query OK, 1 row affected (0.00 sec)

mysql> call updateSalary(3);
Query OK, 1 row affected (0.00 sec)
```

Step 4: Call the procedure to update the totalsalary for each department

Example

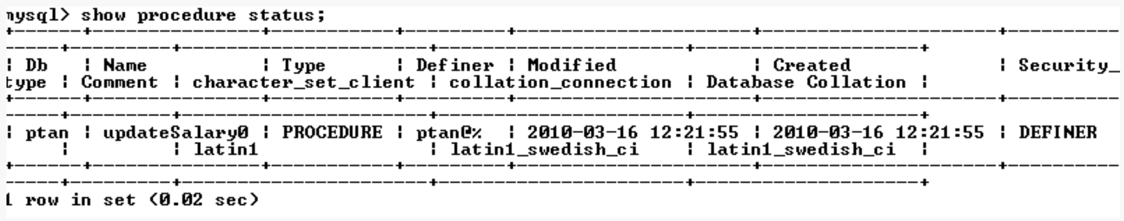


```
mysql> select * from deptsal;
+---------+-------------+
| dnumber | totalsalary |
+---------+-------------+
|       1 |      100000 |
|       2 |       50000 |
|       3 |      130000 |
+---------+-------------+
3 rows in set (0.00 sec)
```

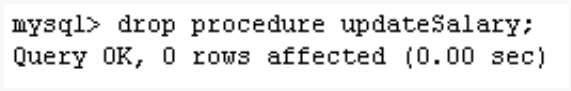
Step 5: Show the updated total salary in the deptsal table

Stored Procedures in MySQL

- Use show procedure status to display the list of stored procedures you have created



```
nysql> show procedure status;
+------+-------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| Db   | Name        | Type      | Definer | Modified            | Created             | Security_
type | Comment | character_set_client | collation_connection | Database Collation |
+------+-------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
| ptan | updateSalary0 | PROCEDURE | ptan@% | 2010-03-16 12:21:55 | 2010-03-16 12:21:55 | DEFINER
     |         | latin1               | latin1_swedish_ci    | latin1_swedish_ci  |
+------+-------------+-----------+---------+---------------------+---------------------+----------
-----+---------+----------------------+----------------------+--------------------+
l row in set (0.02 sec)
```



```
mysql> drop procedure updateSalary;
Query OK, 0 rows affected (0.00 sec)
```

- Use drop procedure to remove a stored procedure

Stored Procedures in MySQL

- You can declare variables in stored procedures
- You can use flow control statements (conditional IF-THEN-ELSE or loops such as WHILE and REPEAT)
- MySQL also supports cursors in stored procedures.
  - A cursor is used to iterate through a set of rows returned by a query so that we can process each individual row.
- To learn more about stored procedures, go to:
  http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

JDBC and SQLJ

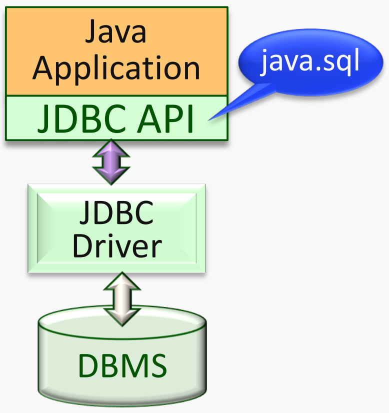
SQL in Application Code

- SQL commands can be called from within a host language (e.g., C++ or Java) program.
  - SQL statements can refer to host variables (including special variables used to return status).
  - Must include a statement to *connect* to the right database.
- Two main integration approaches:
  - Embed SQL in the host language (Embedded SQL, SQLJ)
  - Create special API to call SQL commands (JDBC)

Database API Approaches

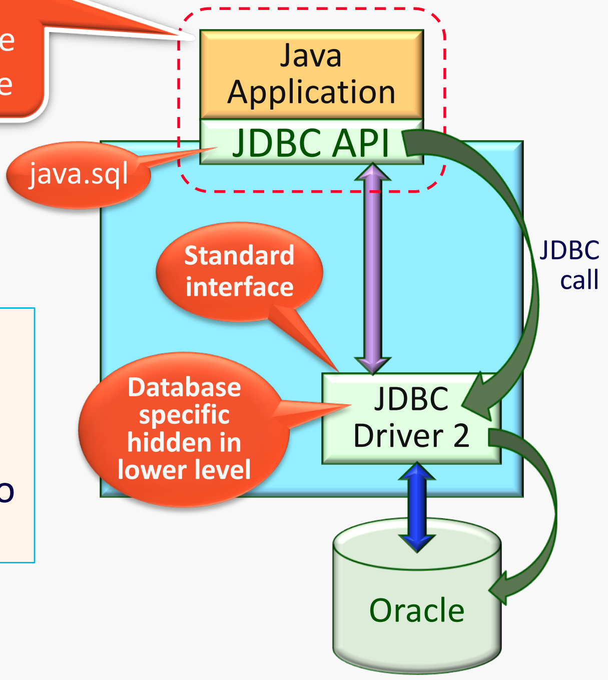
ODBC = Open DataBase Connectivity
JDBC = Java DataBase Connectivity

- JDBC is a collection of Java classes and interface that enables database access
- JDBC contains methods for
  - connecting to a remote data source,
  - executing SQL statements,
  - receiving SQL results
  - transaction management, and
  - exception handling
- The classes and interfaces are part of the java.sql package



Advantage of API Approach

Applications using ODBC or JDBC are DBMS-independent at the source code level and at the level of the executable



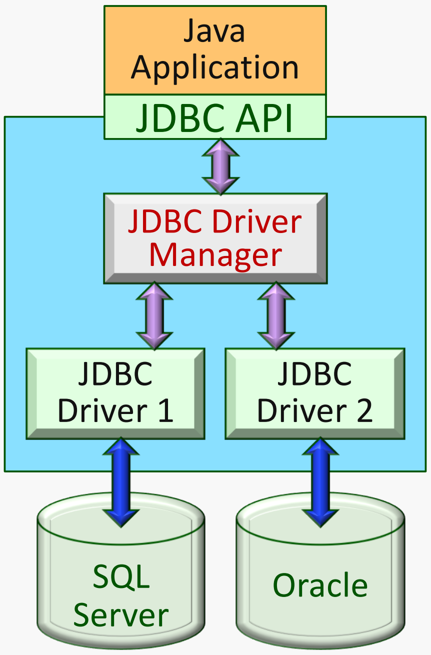
This is achieved by introducing an extra level of indirection

- A DBMS-specific “driver” traps the calls and translates them into DBMS-specific code

Driver Manager

Drivers are registered with a **driver manager**

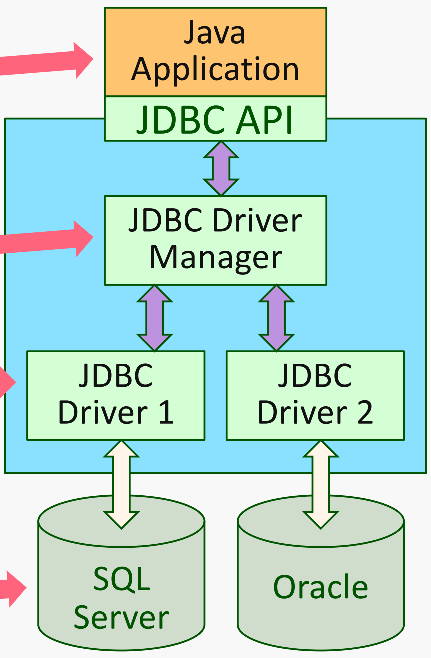
- Drivers are loaded dynamically on demand
- The application can access several different DBMS’s simultaneously



JDBC: Architecture

Four architectural components:

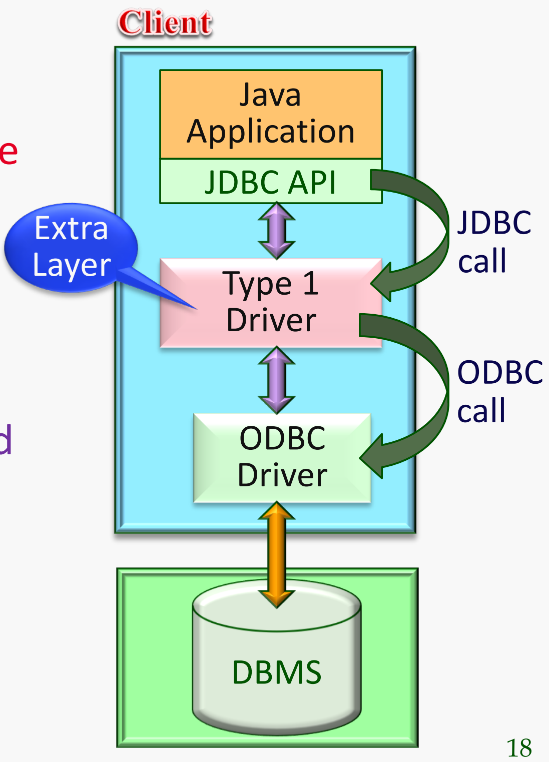
- Application (initiates and terminates connections, submits SQL statements)
- Driver manager (loads JDBC driver and passes function calls)
- Driver (connects to data source, transmits requests and returns/translates results and error codes)
- Data source (processes SQL statements)



JDBC: Four Types of Drivers (1)

Bridge:

- Translates JDBC function calls into function calls of another non-native API such as ODBC.
- The application can use JDBC calls to access an ODBC compliant data source.
- Advantage: no new drivers needed
- Disadvantage:
  - The additional layer affects performance
  - Client requires the ODBC installation
  - Not good for the Web

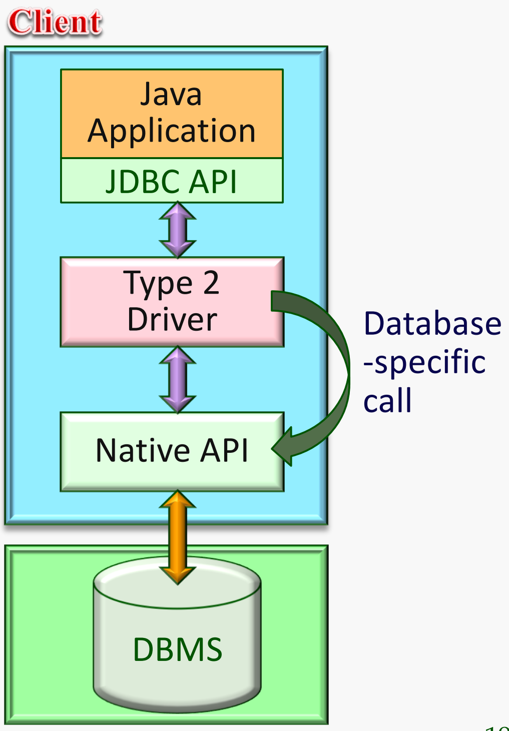


JDBC: Four Types of Drivers (2)

Direct translation to native API via non-Java driver:

Convert JDBC calls into database-specific calls (e.g., Oracle native API)

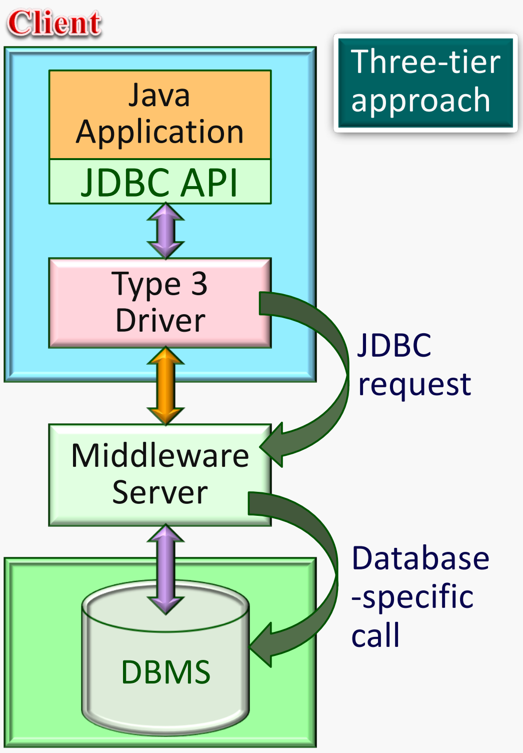
- **Advantage**: Better performance
- **Disadvantage**:
  - Native API must be installed in client
  - Not good for the Web



JDBC: Four Type of Drivers (3)

Network bridge:

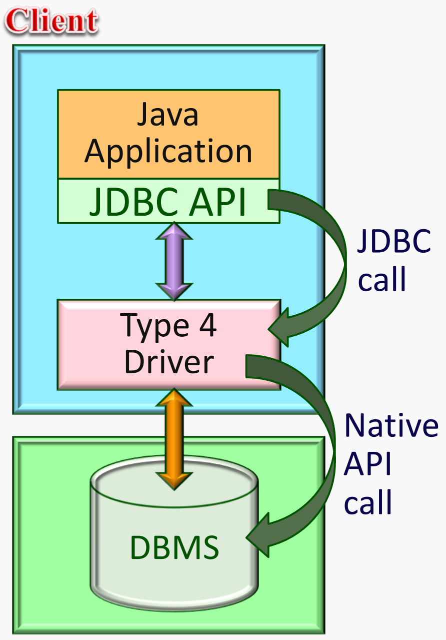
- The driver sends commands over the network to a middleware server
- The middleware server translates the JDBC requests into database-specific calls
- Advantage: Needs only small JDBC driver at each client
- Disadvantage: Need to maintain another server



JDBC: Four Type of Drivers (4)

Direct translation to the Native API via Java Driver:

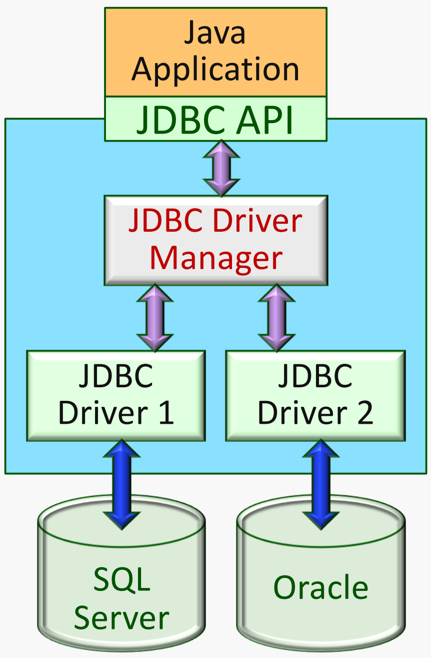
- The driver translates JDBC calls into the native API of the database system
- The driver uses java networking libraries to communicate directly with the database server (i.e., java sockets)
- Advantage:
  - Implementation is all Java
  - Performance is usually quite good
  - Most suitable for Internet access
- Disadvantage: Need a different driver for each database



JDBC Classes and Interfaces

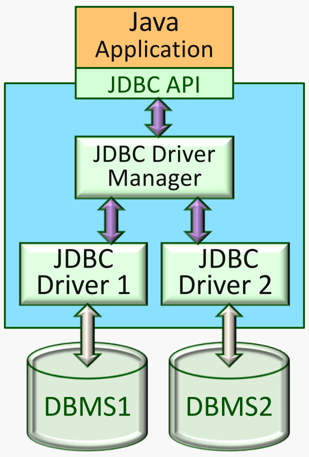
Steps to submit a database query:

1. Load the JDBC driver
2. Connect to the data source
3. Execute SQL statements



JDBC Driver Management

- DriverManager class:
  - Maintains a list of currently loaded drivers
  - Has methods to enable dynamic addition and deletion of drivers



- Two ways of loading a JDBC driver:
  1. In the Java code:
     Class.forName("oracle/jdbc.driver.Oracledriver");
     /* This method loads an instance of the driver class
  2. Enter at command line when starting the Java application:
     -Djdbc.drivers=oracle/jdbc.driver

JDBC Steps

1) Importing Packages
2) Registering the JDBC Drivers
3) Opening a Connection to a Database
4) Creating a Statement Object
5) Executing a Query and Returning a Result Set
6) Object
7) Processing the Result Set
8) Closing the Result Set and Statement Objects
9) Closing the Connection

Executing SQL Statements

- Three different ways of executing SQL statements:
  1. Statement (both static and dynamic SQL statements)
  2. PreparedStatement (semi-static SQL statements)
  3. CallableStatment (stored procedures)

- PreparedStatement class:

  Used to create precompiled, **parameterized SQL statements**
  - SQL structure is fixed
  - Values of parameters are determined at run-time

  - Example
    - https://docs.oracle.com/javase/tutorial/jdbc/basics/prepared.html

1: Importing Packages

```
//Import packages
import java.sql.*; //JDBC packages
import java.math.*;
import java.io.*;
import oracle.jdbc.driver.*;
```

2. Registering JDBC Drivers

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

// Load Oracle driver

Class.forName("oracle.jdbc.driver.OracleDriver")

// Or:
//DriverManager.registerDriver (new oracle.jdbc.driver.OracleDriver());
```

3. Connections in JDBC

- We interact with a data source through sessions.
- A session is started through creation of a Connection object
- Each connection identifies a logical session with a data source
- Connections are specified through a URL that uses the jdbc protocol: jdbc:<subprotocol>:<otherParameters>

Example:

```
                               Host              Port
String url="jdbc:oracle:www.bookstore.com:3083";
Connection con;
try{
      con = DriverManager.getConnection(url,userId,password);
  } catch(SQLException excpt) { …}
```

Different drivers have slightly different URL formats – check the documentation

Discuss later

3. Opening Connection to a Database

```
//Prompt user for username and password

String user;
String password;
user = readEntry("username: ");
password = readEntry("password: ");

// Connect to the database

Connection conn = DriverManager.getConnection
(“jdbc:oracle:thin:@apollo.ite.gmu.edu: 1521:ite10g”, user, password);
```

4. Creating a Statement Object

```
// Suppose Books has attributes isbn, title, author,
// quantity, price, year. Initial quantity is always
// zero; ?’s are placeholders

String sql = “INSERT INTO Books VALUES(?,?,?,0,?,?)”;
PreparedStatement pstmt = conn.prepareStatement(sql);

// now instantiate the parameters with values.
// Assume that isbn, title, etc. are Java variables
// that contain the values to be inserted.

pstmt.clearParameters();
pstmt.setString(1, isbn);
pstmt.setString(2, title);
pstmt.setString(3, author);
pstmt.setFloat(5, price);
pstmt.setInt(6, year);
```

5. Executing a Query, Returning Result Set
6. Processing the Result Set

```
// The executeUpdate command is used if the SQL stmt does not return any
// records (e.g. UPDATE, INSERT, ALTER, and DELETE stmts).
// Returns an integer indicating the number of rows the SQL stmt modified.

int numRows = pstmt.executeUpdate();

// If the SQL statement returns data, such as in a SELECT query, we use
executeQuery method

String sqlQuery = “SELECT title, price FROM Books
WHERE author=?”;
PreparedStatement pstmt2 = conn.prepareStatement
(sqlQuery);
pstmt2.setString(1, author);
ResultSet rset = pstmt2.executeQuery ();

// Print query results  the (1) in getString refers to the title value,  and
the (2) refers to the price value

while (rset.next ())
System.out.println (rset.getString (1)+ " ” +
rset.getFloat(2));
```

7. Closing the Result Set and Statement Objects
8. Closing the Connection

```
// close the result set, statement,
// and the connection

rset.close();
pstmt.close();
pstmt2.close();
conn.close();
}
```

Connection Class Interface (1)

- void setTransactionIsolation(int *level*)
  Sets isolation level for the current connection
- public int getTransactionIsolation()
  Get isolation level of the current connection
- void setReadOnly(boolean b)
  Specifies whether transactions are read-only
- public boolean getReadOnly()
  Tests if transaction mode is read-only
- void setAutoCommit(boolean b)
  - If autocommit is set, then each SQL statement is considered its own transaction.
  - Otherwise, a transaction is committed using commit(), or aborted using rollback().
- public boolean getAutoCommit()
  Test if autocommit is set

Connection Class Interface (2)

- public boolean isClosed()
  Checks whether connection is still open.
- connectionname.close()
  Close the connection connectionname

PreparedStatement

Place holder

```
String sql="INSERT INTO Sailors VALUES(?,?,?,?)";
PreparedStatment pstmt=con.prepareStatement(sql);
pstmt.clearParameters();
pstmt.setInt(1,sid);
pstmt.setString(2,sname);
pstmt.setInt(3, rating);
pstmt.setFloat(4,age);

int numRows = pstmt.executeUpdate();
```

Connection name

Good style to always clear

Setting parameter values
sid, sname, rating, age are java variables

Number of rows modified

Use executeUpdate() when no rows are returned

ResultSet Example

- PreparedStatement.executeUpdate only returns the number of affected records
- PreparedStatement.executeQuery returns data, encapsulated in a ResultSet object
  - ResultSet is similar to a cursor
  - Allows us to read one row at a time
  - Intially, the ResultSet is positioned before the first row
  - Use next() to read the next row
  - next() returns false if there are no more rows

Common ResultSet Methods (1)

POSITIONING THE CURSOR	
next()	Move to next row
previous()	Moves back one row
absolute(int num)	Moves to the row with the specified number
relative(int num)	Moves forward or backward (if negative)
first()	Moves to the first row
Last()	Moves to the last row

Common ResultSet Methods (2)

RETRIEVE VALUES FROM COLUMNS	
getString(string columnName):	Retrieves the value of designated column in current row
getString(int columnIndex)	Retrieves the value of designated column in current row
getFloat (string columnName)	Retrieves the value of designated column in current row

Matching Java and SQL Data Types

SQL Type	Java class	ResultSet get method
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

SQL Data Types

BIT	A boolean value
CHAR(*n*)	A character string of fixed length *n*
VARCHAR(*n*)	A variable-length character string with a maximum length *n*
DOUBLE	A double-precision floating point value
FLOAT(*p*)	A floating point value with a precision value *p*
INTEGER	A 32-bit signed integer value
REAL	A high precision numeric value
DATE	A day/month/year value
TIME	A time of day (hour, minutes, second) value
TIMESTAMP	A day/month/year/hour/minute/second value

SQLJ

- Embedded SQL for Java
- SQLJ is similar to existing extensions for SQL that are provided for C, FORTRAN, and other programming languages.
- IBM, Oracle, and several other companies have proposed SQLJ as a standard and as a simpler and easier-to-use alternative to JDBC.

SQLJ

```
#sql { … } ;
```

- SQL can span multiple lines
- Java host expressions in SQL statement

SQLJ Example

```
String title; Float price; String author(“Lee”);

// declare iterator class

#sql iterator Books(String title, Float price);
Books books;

// initialize the iterator object books; sets the
// author, execute query and open the cursor

#sql books =
{SELECT title, price INTO :title, :price
FROM Books WHERE author=:author };
// retrieve results
while(books.next()){
System.out.println(books.title()+”,”+books.price());
books.close();
```

JDBC Equivalent

```
String sqlQuery = “SELECT title, price FROM Books
WHERE author=?”;
PreparedStatement pstmt2 = conn.prepareStatement(sqlQuery);
pstmt2.setString(1, author);
ResultSet rset = pstmt2.executeQuery ();

// Print query results. The (1) in getString refers
// to the title value, and the (2) refers to the
// price value

while (rset.next ())
System.out.println (rset.getString (1)+ " ” +
rset.getFloat(2));
```

SQLJ Advantage

- Can check for program's errors at translation-time rather than at run-time
- Can write an application that is deployable to other databases
  - SQLJ allows users to customize the static SQL for that database at deployment-time.
- Can work with a database that contains compiled SQL
  - Cannot compile SQL statements in a JDBC program.

JDBC Tutorials

- Check
  - http://java.sun.com/docs/books/tutorial/jdbc/basics/
  - http://infolab.stanford.edu/~ullman/fcdb/oracle/or-jdbc.html