Database Programming

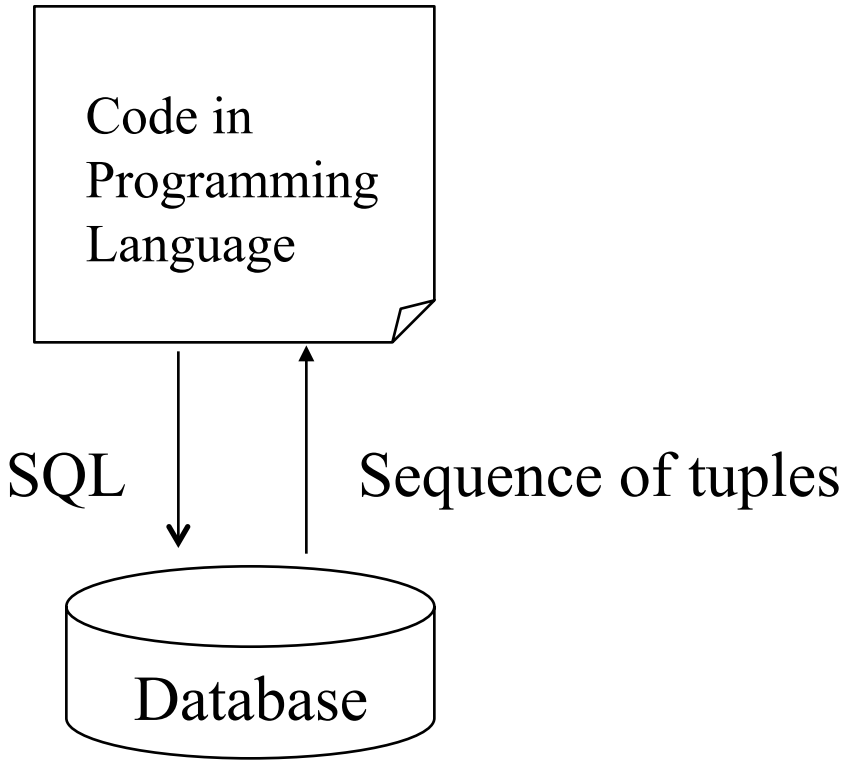
Week 9

*Some of the slides in this lecture are created by Prof. Ian Horrocks from University of Oxford

SQL in Real Programs

- We have seen only how SQL is used at the generic query interface --- an environment where we sit at a terminal and ask queries of a database.
- Reality is almost always different.
  - Programs in a conventional language like C are written to access a database by "calls" to SQL statements.

Database Programming



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)

SQL in Application Code (Con't)

- Impedance mismatch
  - SQL relations are (multi-) sets of records, with no *a priori* bound on the number of records. Typically, no such data structure in programming languages such as C/ C++ (Though now: STL).
  - SQL supports a mechanism called a *cursor* to handle this.

Embedded SQL

- Approach: Embed SQL in the host language.
  - A preprocessor converts/translates the SQL statements into special API calls.
  - Then a regular compiler is used to compile the code.

Embedded SQL

- Language constructs:
  - Connecting to a database

```
EXEC SQL CONNECT :usr_pwd;

// the host variable usr_pwd contains your user
name and password separated by '/'
```

  - Declaring variables

```
EXEC SQL BEGIN DECLARE SECTION

EXEC SQL END DECLARE SECTION
```

  - Statements

```
EXEC SQL Statements;
```

Variable Declaration

- Can use host-language variables in SQL statements
  - Must be prefixed by a colon (:)
  - Must be declared between

```
EXEC SQL BEGIN DECLARE SECTION
                .
                .
                .
EXEC SQL END DECLARE SECTION
```

Variable Declaration in C

Variables in host program

```
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20];
long c_sid;
short c_rating;
float c_age;
EXEC SQL END DECLARE SECTION
```

Embedded SQL: "Error" Variables

Two special variables for reporting errors:

- SQLCODE (older)
  - A negative value to indicate a particular error condition
  - The appropriate C type is *long*
- SQLSTATE (SQL-92 standard)
  - Predefined codes for common errors
  - Appropriate C type is char[6] (a character string of five letters long with a null character at the end to terminate the string)
- One of these two variables must be declared. We assume SQLSTATE

Embedded SQL

- All SQL statements embedded within a host program must be clearly marked.
- In C, SQL statements must be prefixed by EXEC SQL:

```
EXEC SQL
INSERT INTO Sailors
VALUES(:c_sname,:c_sid,:c_rating,:c_age);
```

- Java embedding (SQLJ) uses # SQL { …. };

SELECT - Retrieving Single Row

```
EXEC SQL SELECT S.sname, S.age
         INTO :c_sname, :c_age
        FROM Sailors S
         WHERE S.sid = :c_sid;
```

SELECT - Retrieving Multiple Rows

- What if we want to embed the following query?

```
SELECT S.sname, S.age
FROM   Sailors
WHERE S.rating > :c_minrating
```

- Potentially, multiple rows will be retrieved
- How do we store the set of rows?
  - No equivalent data type in host languages like C

Cursors

- Can *declare* a cursor on a relation or query statement (which generates a relation).
- Can *open* a cursor, and repeatedly *fetch* a tuple then *move* the cursor, until all tuples have been retrieved.
  - Can use a special clause, called ORDER BY, in queries that are accessed through a cursor, to control the order in which tuples are returned.
    - Fields in ORDER BY clause must also appear in SELECT clause.
  - The ORDER BY clause, which orders answer tuples, is *only* allowed in the context of a cursor.
- Can also modify/delete tuple pointed to by a cursor.

Declaring a Cursor

- Cursor that gets names and ages of sailors whose ratings are greater than "minrating", in alphabetical order

```
EXEC SQL DECLARE sinfo CURSOR FOR
         SELECT  S.sname, S.age
         FROM  Sailors S
         WHERE S.rating > :c_minrating
         ORDER BY S.sname
```

Opening/Fetching a Cursor

- To open the cursor (executed at run-time):
  - **`OPEN sinfo;`**
  - The cursor is initially positioned just before the first row
- To read the current row that the cursor is pointing to:
  - **`FETCH sinfo INTO :c_sname, :c_age`**
- When FETCH is executed, the cursor is positioned to point at the next row
  - Can put the FETCH statement in a loop to retrieve multiple rows, one row at a time

Closing a Cursor

- When we're done with the cursor, we can close it:
  - `CLOSE sinfo;`
- We can re-open the cursor again. However, the rows retrieved might be different (depending on the value(s) of the associated variable(s) when cursor is opened)
  - Ex. If :c_minrating is now set to a different value, then the rows retrieved will be different

Embedding SQL in C: An Example

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
   char c_sname[20]; short c_minrating; float c_age;
EXEC SQL END DECLARE SECTION
c_minrating = random();
EXEC SQL DECLARE sinfo CURSOR FOR  // declare cursor
         SELECT S.sname, S.age
         FROM Sailors S
         WHERE S.rating > :c_minrating
         ORDER BY S.sname;
EXEC SQL OPEN sinfo;                    // open cursor
do {
   EXEC SQL FETCH sinfo INTO :c_sname, :c_age;
   printf("%s is %d years old\n", c_sname, c_age);
} while (SQLSTATE != '02000');    // end of file
EXEC SQL CLOSE sinfo;                   // close cursor
```

Update/Delete Commands

- Modify the rating value of the row currently pointed to by cursor sinfo

```
UPDATE  Sailors S
SET S.rating = S.rating + 1
WHERE CURRENT of sinfo;
```

- Delete the row currently pointed to by cursor sinfo

```
DELETE Sailors S
FROM   CURRENT of sinfo;
```

Dynamic SQL

- SQL query strings are not always known at compile time
  - Such application must accept commands from the user; and based on what the user needs, generate appropriate SQL statements
  - The **SQL statements are constructed on-the-fly**
- Dynamic SQL allows programs to construct and submit SQL queries at run time.
- Example of the use of dynamic SQL from within a C program.

Dynamic SQL - Example

```
char c_sqlstring[] = {"DELETE FROM Sailor WHERE rating > 5"};
EXEC SQL PREPARE readytogo FROM :c_sqlstring;
EXEC SQL EXECUTE readytogo
```

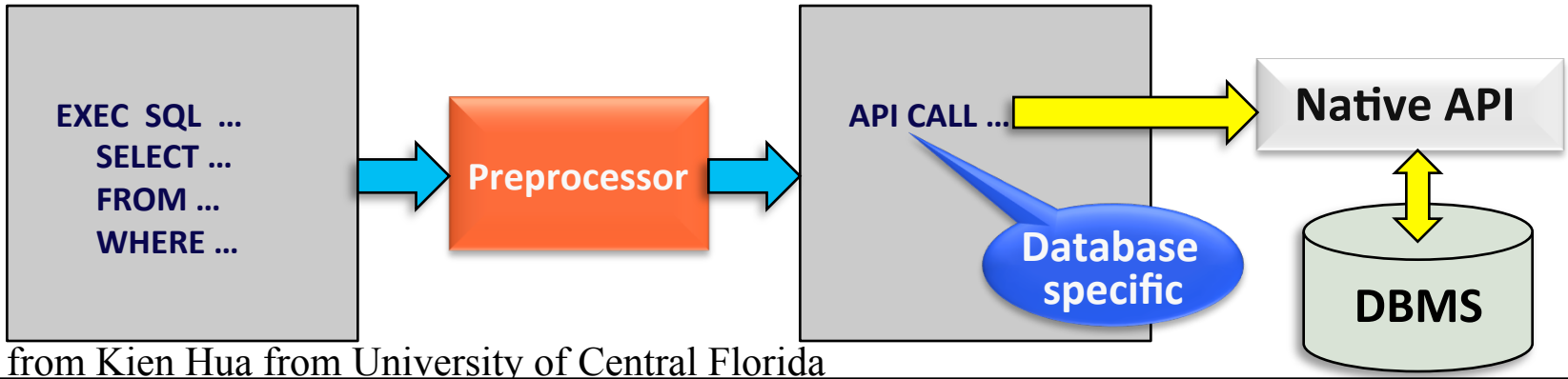
Instruct SQL system to execute the query

Inform SQL system to take the string as query

Limitation of Embedded SQL

- DBMS-specific preprocessor transform the Embedded SQL statements into function calls in the host language
- This translation varies across DBMSs (API calls vary among different DBMSs)
- Even if the source code can be compiled to work with different DBMS's, the final executable works only with one specific DBMS.

  → DBMS-independent only at the source code level

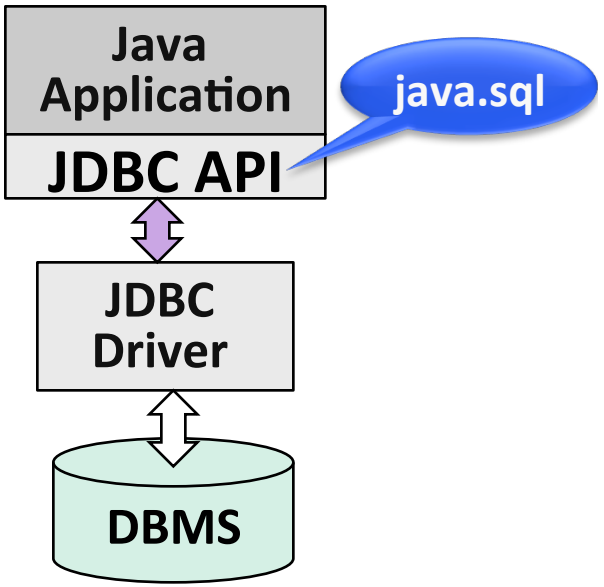


Slide from Kien Hua from University of Central Florida

Database API: Alternative to Embedding

ODBC = Open DataBase Connectivity
JDBC = Java DataBase Connectivity

- Both are API (application-program interface) for a program to interact with a database server
- Application makes calls to
  - Connect with the database server
  - Send SQL commands to the database server
  - Fetch tuples of result one-by-one into program variables
- ODBC (Open Database Connectivity) works with C, C++, C#, and Visual Basic
- JDBC (Java Database Connectivity) works with Java



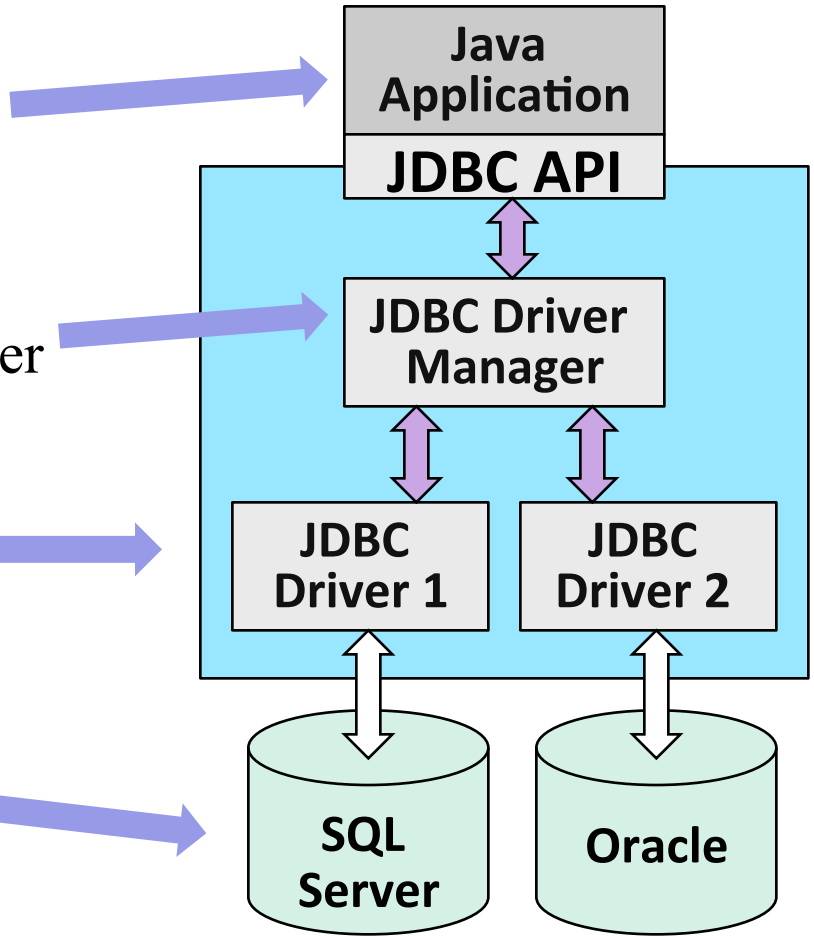
JDBC

- 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

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)



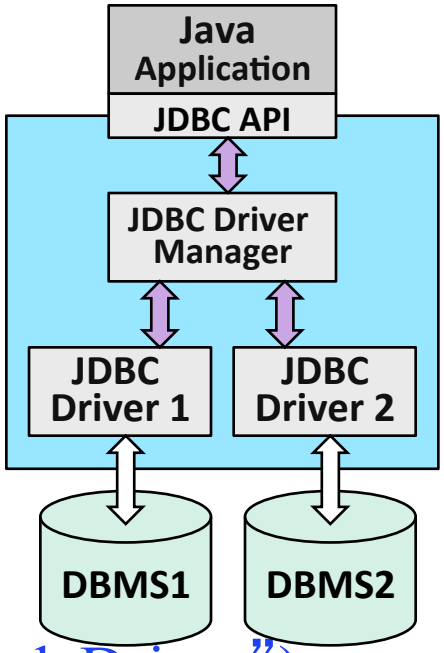
Slide from Kien Hua from University of Central Florida

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
  - The driver we need depends on which DBMS is available to us



- Two ways of loading a JDBC driver:

1. In the Java code:
   Class.forName(<driver name>)
   e.g., Class.forName("oracle.jdbc.driver.OracleDriver");
   // This method loads an instance of the driver class

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

2. Enter at command line when starting the Java application:
   -Djdbc.drivers=oracle/jdbc.driver

JDBC Steps: More Details

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 Object
6. Processing the Result Set
7. Closing the Result Set and Statement Objects
8. Closing the Connection

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. 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);
```

format: **Connection connection = DriverManager.getConnection("jdbc:oracle:thin:@<hostname>:<port>:<sid>","<username>","<password>");**

http://www.oracle.com/technology/tech/java/sqlj_jdbc/htdocs/jdbc_faq.html#02_01

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.setString(4, price);
pstmt.setString(5, 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();
```

Step 5/6, Con't

```
// 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();

}
```

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

Mapping Data Types

- There are data types specified to SQL that need to be mapped to Java data types if the user expects Java to be able to handle them.
- Conversion falls into three categories:
  - SQL type to Java direct equivalents

    SQL INTEGER direct equivalent of Java int data type.
  - SQL type can be converted to a Java equivalent.

    SQL CHAR, VARCHAR, and LONGVARCHAR can all be converted to the Java String data type.
  - SQL data type is unique and requires a special Java data class object to be created specifically for their SQL equivalent.

    SQL DATE converted to the Java Date object that is defined in java.Date especially for this purpose.

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));
```

Use SQLJ to write your program when…

- you want to be able to check your program for errors at translation-time rather than at run-time.
- you want to write an application that you can deploy to another database. Using SQLJ, you can customize the static SQL for that database at deployment-time.
- you are working with a database that contains compiled SQL. You will want to use SQLJ because you cannot compile SQL statements in a JDBC program.

Use JDBC to write your program when…

- your program uses dynamic SQL. For example, you have a program that builds queries on-the-fly or has an interactive component.
- you do not want to have a SQLJ layer during deployment or development.

Useful JDBC Tutorials

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