ICS 321 Fall 2011
SQL in a Server Environment

Asst. Prof. Lipyeow Lim
Information & Computer Science Department
University of Hawaii at Manoa
Three Tier Architecture

- Commonly used in large internet enterprises

- Webserver
  - Eg. Apache/Tomcat
  - Connects clients to database systems

- Application Server
  - Eg. IBM Websphere Application Server, Jboss, SAP Netweaver, etc.
  - Performs business logic like shopping cart, checkout etc

- Database Server
  - Eg. IBM DB2, Oracle, MS SQL Server
  - Runs DBMS, performs queries and updates from app server
SQL Environment

- Schemas: tables, views, assertions, triggers
  - CREATE SCHEMA <schema name>
  - Your login id is your default schema
  - SET SCHEMA <schema>
  - A fully qualified table name is <schema>.<table>

- Catalogs: collection of schemas
  - Corresponds to “databases” in DB2

- Clusters: collection of catalogs
  - Corresponds to “database instance” in DB2
Client-Server Model

- **CONNECT TO <server> AS <connection name> AUTHORIZATION**
- **DISCONNECT/CONNECT RESET/TERMINATE**
- Session – SQL operations performed while a connection is active

**Programming API**
- Generic SQL Interface
- Embedded SQL in a host language
- True Modules. Eg. Stored procedures.

Can be on same machine or different machines
SQL & Other Programming Languages

Two extremes of the integration spectrum:

• Highly integrated eg. Microsoft linq
  – Compiler checking of database operations

• Loosely integrated eg. ODBC & JDBC
  – Provides a way to call SQL from host language
  – Host language compiler doesn’t understand database operations.

• Requirements:
  – Perform DB operations from host language
  – DB operations need to access variables in host language
Each network “card” has a unique MAC address.

- **IP address** assigned by network provider: static or DHCP
- **Port number** usually fixed by application type
- **MAC address**

**Client Application**
- **Higher level protocols**
- **Port number**
- **IP address**

**DBMS Server**
- **Higher level protocols**
- **Port number**
- **IP address**

**Internet**

- **DBMS servers use their own protocols** (e.g., DRDA)
- **Servers use a port that is known by its clients**
- **Servers use static IP address + DNS name**

- **Eg. http URLs, DNS**

- **Eg.** DBMS servers use their own protocols (e.g., DRDA)
- **Eg.** Servers use a port that is known by its clients
- **Eg.** Servers use static IP address + DNS name
Remote Client Access

• Applications run on a machine that is separate from the DB server

• DBMS “thin” client
  – Libraries to link your app to
  – App needs to know how to talk to DBMS server via network

• DBMS “full” client layer
  – Need to pre-configure the thick client layer to talk to DBMS server
  – Your app talks to a DBMS client layer as if it is talking to the server

What information is needed for 2 machines to talk over a network?
Configuring DBMS Client Layer

• Tell the client where to find the server
  
  `db2 CATALOG TCPIP NODE mydbsrv REMOTE 123.3.4.12 SERVER 50001`

• Tell the client where to find the server
  
  `db2 CATALOG DATABASE bookdb AS mybookdb AT NODE mydbsrv`

  - Give a name for this node
  - Specify the IP address/hostname and the port number of the DB server machine
  - Specify the name of the database on the server
  - Give a local alias for the database
  - Specify the name of the node that is associated with this database
Embedded SQL in C Programs

- DBMS-specific Preprocessor translates special macros to DB-specific function calls
- Pre-processor needs access to DBMS instance for validation.
- Executable needs to be bound to a specific database in a DBMS in order to execute

Diagram:

- .sqc
- DBMS-specific Precompiler
  - .c
  - C Compiler
    - .o
    - DBMS-specific libraries
    - C Linker
      - .exe
      - package
      - Database
  - .bnd
    - DBMS-specific Binder
      - .exe
Connecting SQL & Host Language

• Need a way for host language to **get data** from SQL environment
• Need a way to **pass values** from host language to SQL environment
• Shared variables
  – **DECLARE SECTION**
  – In SQL, refer using :Salary, :EmployeeNo

```sql
EXEC SQL BEGIN DECLARE SECTION;
char EmployeeNo[7];
char LastName[16];
double Salary;
short SalaryNI;
EXEC SQL END DECLARE SECTION;
```
An Example of Embedded SQL C Program

```c
#include <stdio.h>
#include <string.h>
#include <sql.h>

int main()
{
    // Include The SQLCA Data Structure Variable
    EXEC SQL INCLUDE SQLCA;

    // Define The SQL Host Variables Needed
    EXEC SQL BEGIN DECLARE SECTION;
    char EmployeeNo[7];
    char LastName[16];
    double Salary;
    short SalaryNI;
    EXEC SQL END DECLARE SECTION;

    // Connect To The Appropriate Database
    EXEC SQL CONNECT TO SAMPLE USER
db2admin USING ibmdb2;

    // Declare A Static Cursor
    EXEC SQL DECLARE C1 CURSOR FOR
    SELECT EMPNO, LASTNAME, DOUBLE(SALARY)
    FROM EMPLOYEE
    WHERE JOB = 'DESIGNER';

    // Open The Cursor
    EXEC SQL OPEN C1;
```
An Example of Embedded SQL C Program

// If The Cursor Was Opened Successfully, while (sqlca.sqlcode == SQL_RC_OK)
{
    EXEC SQL FETCH C1 INTO :EmployeeNo, :LastName, :Salary, :SalaryNI;

    // Display The Record Retrieved
    if (sqlca.sqlcode == SQL_RC_OK)
    {
        printf("%-8s %-16s ", EmployeeNo, LastName);
        if (SalaryNI >= 0)
            printf("%lf\n", Salary);
        else
            printf("Unknown\n");
    }
}

// Close The Open Cursor
EXEC SQL CLOSE C1;

// Commit The Transaction
EXEC SQL COMMIT;

// Terminate The Database Connection
EXEC SQL DISCONNECT CURRENT;

// Return Control To The Operating System
return(0);

• A cursor is an iterator for looping through a relation instance.

• Why is a cursor construct necessary?
Updates

- SQL syntax except where clause require current of <cursor>

EXEC SQL BEGIN DECLARE SECTION;
  int certNo, worth;
  char execName[31],
  execName[31],
  execAddr [256],
  SQLSTATE [6];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE execCursor CURSOR FOR MovieExec;
EXEC SQL OPEN execCursor
while (1) {
  EXEC SQL FETCH FROM execCursor INTO :
    execName, :execAddr, :certNo, :worth;
  if (NO_MORE_TUPLES) break;
  if ( worth < 1000)
    EXEC SQL DELETE FROM MovieExec
      WHERE CURRENT OF execCursor;
  else
    EXEC SQL UPDATE MovieExec
      SET netWorth=2*netWorth
      WHERE CURRENT OF execCursor;
}
EXEC SQL CLOSE execCursor
Static vs Dynamic SQL

• Static SQL refers to SQL queries that are completely specified at compile time. Eg.

```sql
// Declare A Static Cursor
EXEC SQL DECLARE C1 CURSOR FOR
SELECT EMPNO, LASTNAME, DOUBLE(SALARY)
FROM EMPLOYEE
WHERE JOB = 'DESIGNER';
```

• Dynamic SQL refers to SQL queries that are not completely specified at compile time. Eg.

```c
strcpy(SQLStmt, "SELECT * FROM EMPLOYEE WHERE JOB=");
strcat(SQLStmt, argv[1]);
EXEC SQL PREPARE SQL_STMT FROM :SQLStmt;
EXEC SQL EXECUTE SQL_STMT;
```
Alternative to Embedded SQL

• What if we want to compile an application without the need for a DBMS-specific pre-compiler?
• Use a library of database calls
  – Standardized (non-DBMS-specific) API
  – Pass SQL-strings from host language and presents result sets in a language friendly way
  – Eg. ODBC for C/C++ and JDBC for Java
  – DBMS-neutral
    • A driver traps the calls and translates them into DBMS-specific code
ODBC/JDBC Architecture

• Application
  – Initiates connections
  – Submits SQL statements
  – Terminates connections
• Driver Manager
  – Loads the right JDBC driver
• Driver
  – Connects to the data source,
  – Transmits requests,
  – Returns results and error codes
• Data Source
  – DBMS
4 Types of Drivers

• Type I: Bridge
  – Translate SQL commands to non-native API
  – eg. JDBC-ODBC bridge. JDBC is translated to ODBC to access an ODBC compliant data source.

• Type II: Direct Translation to native API via non-Java driver
  – Translates SQL to native API of data source.
  – Needs DBMS-specific library on each client.

• Type III: Network bridge
  – SQL stmts sent to a middleware server that talks to the data source. Hence small JDBC driver at each client

• Type IV: Direct Translation to native API via Java driver
  – Converts JDBC calls to network protocol used by DBMS.
  – Needs DBMS-specific Java driver at each client.
High Level Steps

1. Load the ODBC/JDBC driver
2. Connect to the data source
3. [optional] Prepare the SQL statements
4. Execute the SQL statements
5. Iterate over the resultset
6. Close the connection
Getting Data to/fro Host Language

- No declaration of shared variables
- Variables in host language is bound to columns of a SQL cursor
- ODBC
  - SQLBindCol – gets data from SQL environment to host variables.
  - SQLBindParameter – gets data from host variables to SQL environment
- JDBC
  - ResultSet class
  - PreparedStatement class
Prepare Statement or Not?

• Executing without preparing statement
  – After DBMS receives SQL statement,
    • The SQL is compiled,
    • An execution plan is chosen by the optimizer,
    • The execution plan is evaluated by the DBMS engine
    • The results are returned

• conn.prepareStatement
  – Compiles and picks an execution plan

• pstmt.executeUpdate
  – Evaluates the execution plan with the parameters and gets the results

String sql="SELECT * FROM books WHERE price < ?";
PreparedStatement pstmt = conn.prepareStatement(sql);
Pstmt.setFloat(1, usermaxprice);
Pstmt.executeUpdate();

cf. Static vs Dynamic SQL
ResultSet

```java
ResultSet rs = stmt.executeQuery(sqlstr);
while( rs.next() ){
    col1val = rs.getString(1); ...
}
```

- Iterate over the results of a SQL statement -- cf. cursor
- Note that types of column values do not need to be known at compile time

<table>
<thead>
<tr>
<th>SQL Type</th>
<th>Java Class</th>
<th>accessor</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>Boolean</td>
<td>getBoolean</td>
</tr>
<tr>
<td>CHAR, VARCHAR</td>
<td>String</td>
<td>getString</td>
</tr>
<tr>
<td>DOUBLE, FLOAT</td>
<td>Double</td>
<td>getDouble</td>
</tr>
<tr>
<td>INTEGER</td>
<td>Integer</td>
<td>getInt</td>
</tr>
<tr>
<td>REAL</td>
<td>Double</td>
<td>getFloat</td>
</tr>
<tr>
<td>DATE</td>
<td>Java.sql.Date</td>
<td>getDate</td>
</tr>
<tr>
<td>TIME</td>
<td>Java.sql.Time</td>
<td>getTime</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Java.sql.Timestamp</td>
<td>getTimestamp</td>
</tr>
</tbody>
</table>
RowSet

- When inserting lots of data, calling an execute statement for each row can be inefficient
  - A message is sent for each execute
- Many APIs provide a rowset implementation
  - A set of rows is maintained in-memory on the client
  - A single execute will then insert the set of rows in a single message
- Pros: high performance
- Cons: data can be lost if client crashes.
- Analogous rowset for reads (ie. ResultSet) also available
Stored Procedures

• What?
  – A procedure that is called and executed via a single SQL statement
  – Executed in the same process space of the DBMS server
  – Can be programmed in SQL, C, java etc
  – The procedure is stored within the DBMS

• Advantages:
  – Encapsulate application logic while staying close to the data
  – Re-use of application logic by different users
  – Avoid tuple-at-a-time return of records through cursors
SQL Stored Procedures

**CREATE PROCEDURE** `ShowNumReservations`

```
SELECT S.sid, S.sname, COUNT(*)
FROM Sailors S, Reserves R
WHERE S.sid = R.sid
GROUP BY S.sid, S.sname
```

- Parameters modes: IN, OUT, INOUT

**CREATE PROCEDURE** `IncreaseRating`

```
CREATE PROCEDURE IncreaseRating ( IN sailor_sid INTEGER, IN increase INTEGER )
UPDATE Sailors
SET rating = rating + increase
WHERE sid = sailor_sid
```
CREATE PROCEDURE TopSailors ( 
    IN num INTEGER) 
LANGUAGE JAVA 
EXTERNAL NAME
   "file:///c:/storedProcs/rank.jar"
Calling Stored Procedures

- **SQL:**
  
  ```sql
  CALL IncreaseRating(101, 2);
  ```

- **Embedded SQL in C:**
  ```c
  EXEC SQL BEGIN DECLARE SECTION
  int sid; int rating;
  EXEC SQL END DECLARE SECTION
  EXEC SQL CALL IncreaseRating(:sid, :rating);
  ```

- **JDBC**
  ```java
  CallableStatement cstmt = conn.prepareCall("{call Show Sailors}");
  ResultSet rs = cstmt.executeQuery();
  ```

- **ODBC**
  ```c
  SQLCHAR *stmt = (SQLCHAR *)"CALL ShowSailors";
  cliRC = SQLPrepare(hstmt, stmt, SQL_NTS);
  cliRC = SQLExecute(hstmt);
  ```
User Defined Functions (UDFs)

• Extend and add to the support provided by SQL built-in functions

• Three types of UDFs
  – **Scalar**: returns a single-valued answer. Eg. Building SUBSTR()
  – **Column**: returns a single-valued answer from a column of values. Eg. AVG()
  – **Table**: returns a table. Invoked in the FROM clause.

• Programable in SQL, C, JAVA.
Scalar UDFs

- Returns the tangent of a value

```
CREATE FUNCTION TAN (X DOUBLE) RETURNS DOUBLE 
LANGUAGE SQL CONTAINS SQL 
RETURN SIN(X)/COS(X)
```

- Reverses a string

```
CREATE FUNCTION REVERSE(INSTR VARCHAR(4000)) RETURNS VARCHAR(4000) CONTAINS SQL 
BEGIN ATOMIC 
DECLARE REVSTR, RESTSTR VARCHAR(4000) DEFAULT ''; 
DECLARE LEN INT; 
IF INSTR IS NULL THEN 
RETURN NULL; 
END IF; 
SET (RESTSTR, LEN) = (INSTR, LENGTH(INSTR)); 
WHILE LEN > 0 DO 
SET (REVSTR, RESTSTR, LEN) = (SUBSTR(RESTSTR, 1, 1) CONCAT REVSTR, SUBSTR(RESTSTR, 2, LEN - 1), LEN - 1); 
END WHILE; 
RETURN REVSTR; 
END
```
Table UDFs

- returns the employees in a specified department number.

```sql
CREATE FUNCTION DEPTEMPLOYEES (DEPTNO CHAR(3))
    RETURNS TABLE (
        EMPNO CHAR(6),
        LASTNAME VARCHAR(15),
        FIRSTNAME VARCHAR(12))
    LANGUAGE SQL
    READS SQL DATA
    RETURN
    SELECT EMPNO, LASTNAME, FIRSTNAME
    FROM EMPLOYEE
    WHERE EMPLOYEE.WORKDEPT = DEPTEMPLOYEES.DEPTNO
```
Java UDFs

```java
import COM.ibm.db2.app.UDF;

public void tableUDF(
    double inSalaryFactor,
    String outName,
    String outJob,
    double outNewSalary)
    throws Exception {
    int intRow = 0;
    ...
} // tableUDF } // UDFsrv class
```