ICS 321 Fall 2009

DBMS Application Programming

Asst. Prof. Lipyeow Lim

Information & Computer Science Department

University of Hawaii at Manoa

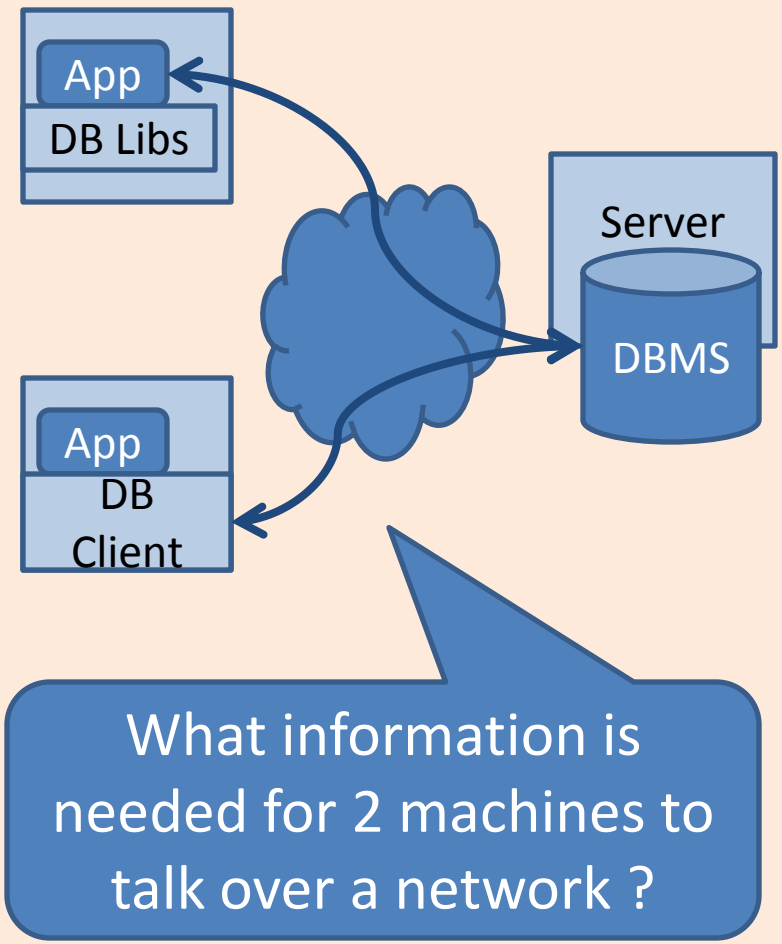
SQL & Other Programming Languages

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

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 thin client layer to talk to DBMS server
  - Your app talks to a DBMS client layer as if it is talking to the server



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

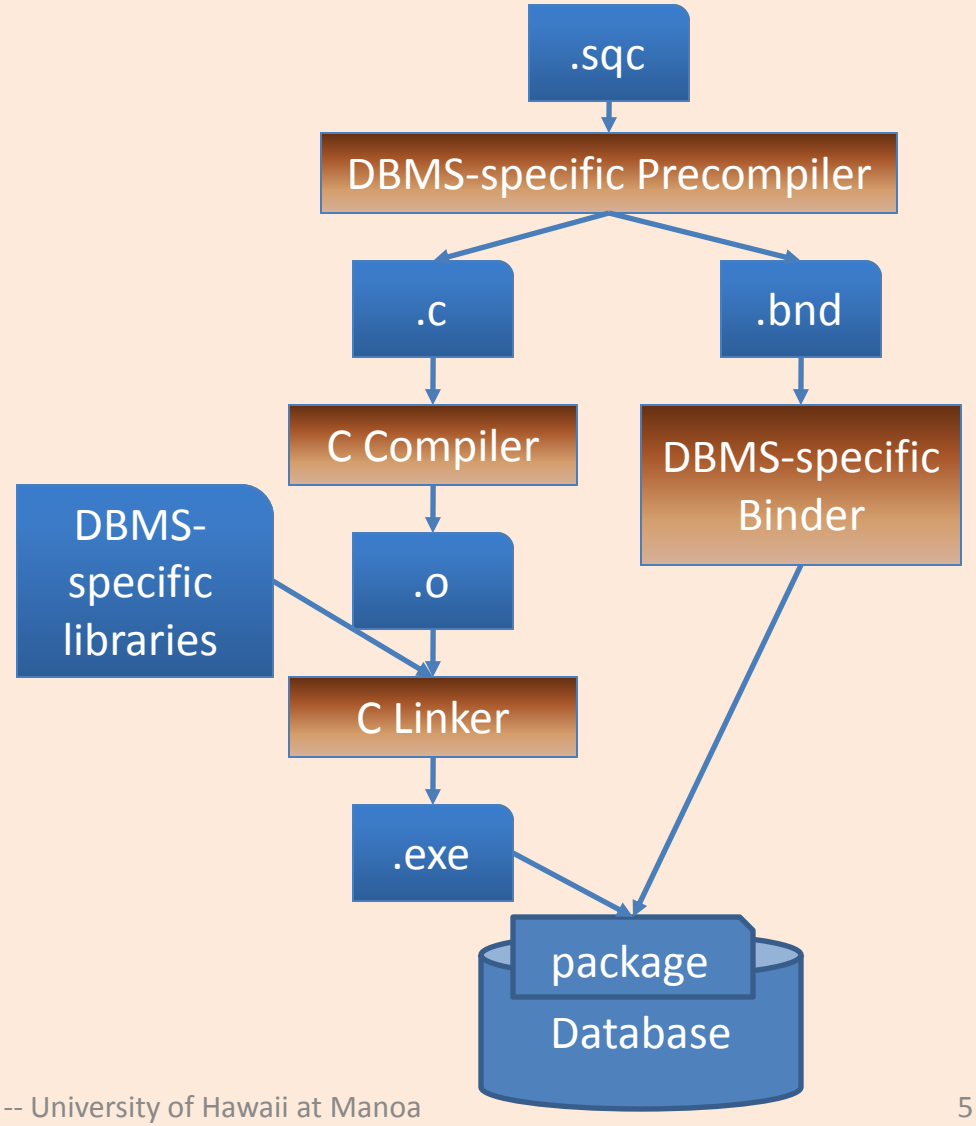
Specify the name of the database on the server

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



An Example of Embedded SQL C Program

```
#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 ?

Static vs Dynamic SQL

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

```
// 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 note completely specified at compile time. Eg.

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