DB2 Universal Database
Advanced Programming

BSD/SEC DM Team
Software Group, IBM China

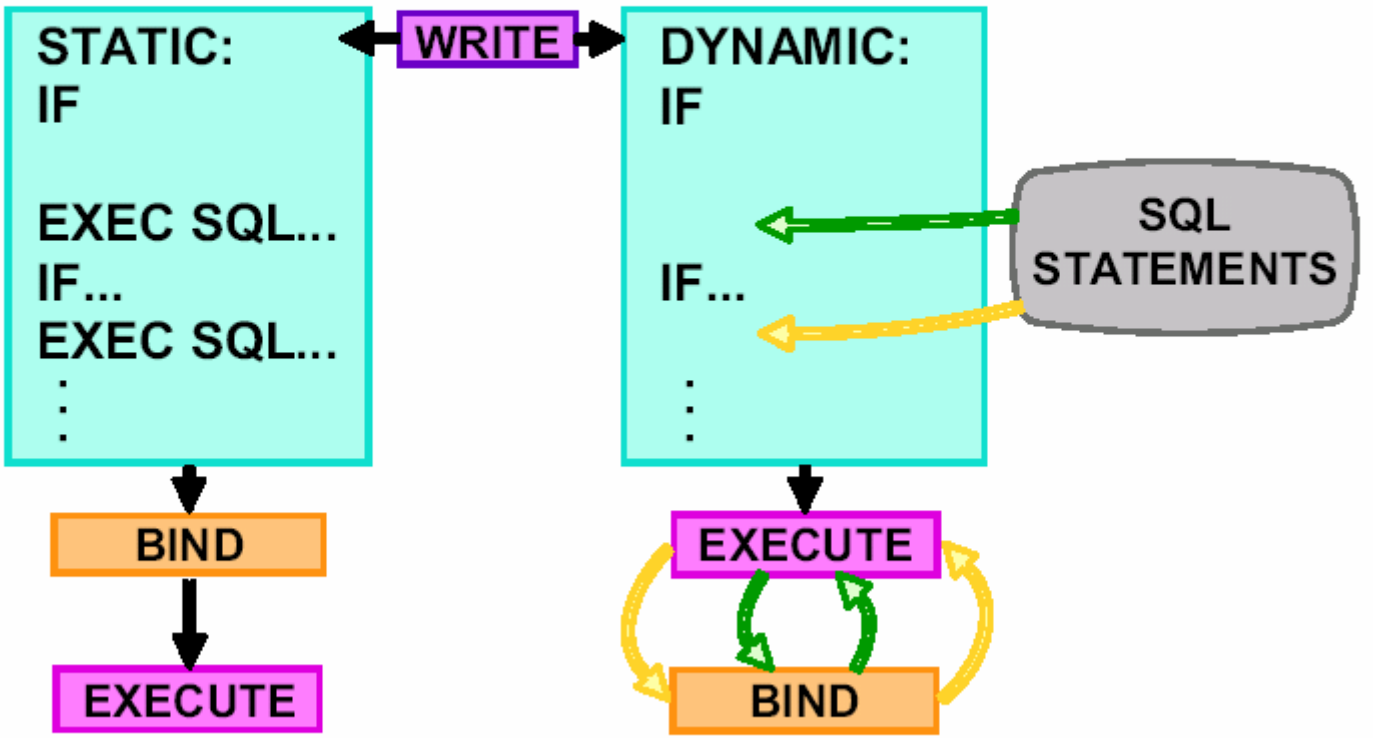
Unit 3. Dynamic SQL

Unit Objectives

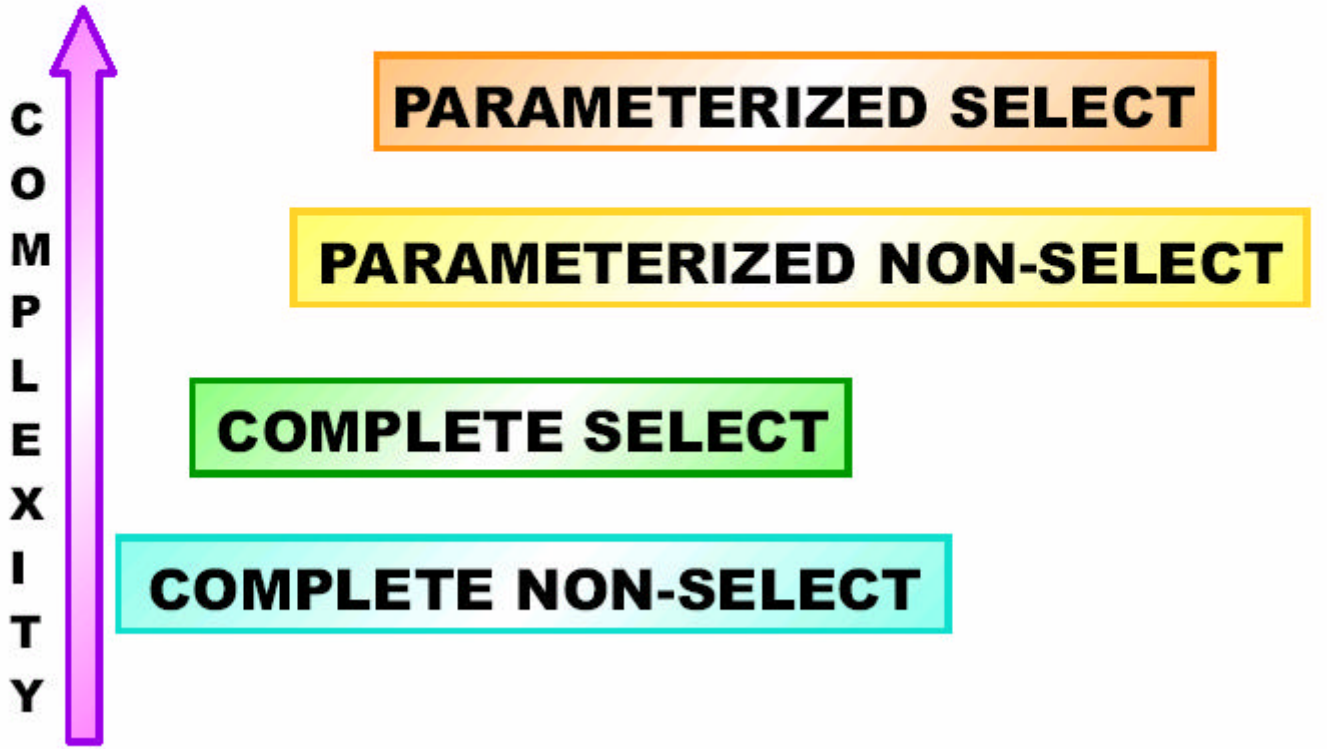
After completing this unit, you should be able to:

- Code dynamic SQL statements to support
  - Complete non-SELECT statements
  - Complete Fixed-List SELECT statements
  - Complete Varying-List SELECT statements
- Use the SQLDA Block to communicate with the database manager

Static SQL versus Dynamic SQL



Types of Dynamic SQL Statements



Scenario

Roberto
Requirements

Patti
Programmer

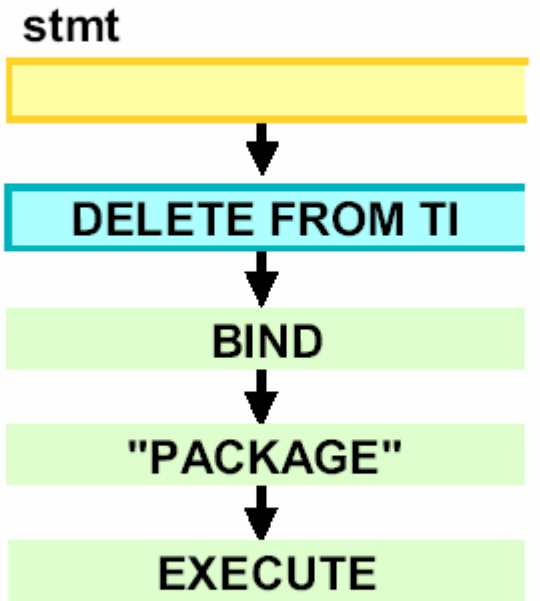
Only Insert / Update / Delete **No Selects**

Complete Non-SELECT

```
char stmt [255];
```

```
<READ SQL STATEMENT
  INTO stmt >
```

```
EXEC SQL
  EXECUTE IMMEDIATE :stmt;
```



<CHECK SQLCA>

Scenario

Roberto
Requirements

Patti
Programmer

Select known columns only / Variable Where predicates

Complete Fixed-List SELECT

```
char stmt [255];
char name [21];
char phone [9];
```


```
<READ STATEMENT INTO stmt>
        <OR FORMULATE>
```

```
SELECT NAME, PHONE
FROM T1
WHERE NAME LIKE 'A%'
```

```
EXEC SQL
  DECLARE C1 CURSOR FOR AOK;
```

BIND

```
EXEC SQL
  PREPARE AOK FROM :stmt;
```

"PACKAGE" AOK

```
<CHECK SQLCA>
```

(to be continued)

Complete Fixed-List SELECT (Cont)

(continued)

```
EXEC SQL
   OPEN C1;
  <CHECK SQLCA>
```

AARON	890-4311
AMES	671-8843
ABLE	549-1375

```
EXEC SQL
   FETCH C1 INTO :name, :phone;
  <CHECK SQLCA>
```

name	phone
AARON	890-4311
.	.
.	.
.	.

```
EXEC SQL
   CLOSE C1;
  <CHECK SQLCA>
```

Scenario

Roberto
Requirements

Patti
Programmer

Any form of DML

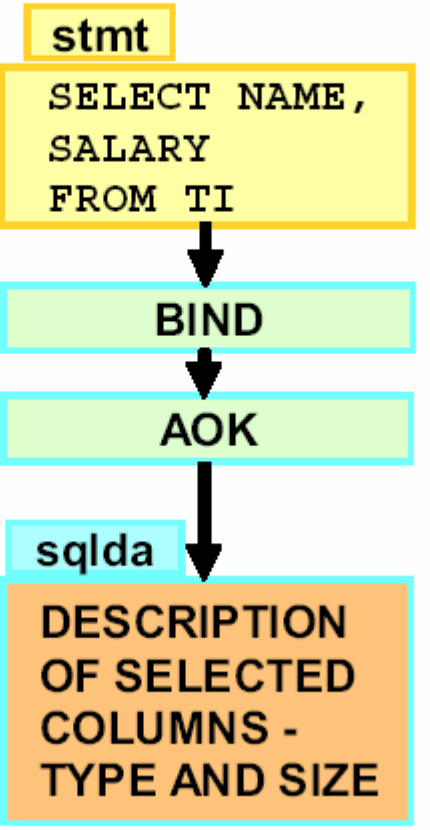
Complete SELECT

```
char stmt [255]
<READ STATEMENT INTO stmt>
```

```
EXEC SQL
   PREPARE AOK INTO :sqlda
                FROM :stmt;
```

```
EXEC SQL
   PREPARE AOK FROM :stmt;
```

```
EXEC SQL
   DESCRIBE AOK INTO :sqlda;
```



(to be continued)

Complete SELECT (Cont)

(continued)

```
<ALLOCATE HOST VARIABLES TO
RECEIVE ONE SELECTED ROW>
```

host1

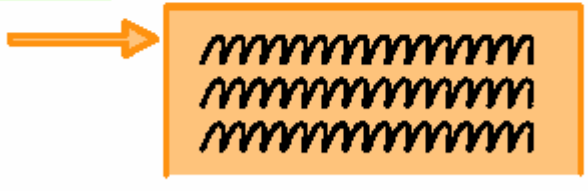
host2

```
<PLACE ADDRESSES OF HOST
VARIABLES INTO sqlda>
```

sqlda
ADDRESSES OF
host1 AND host2;

```
EXEC SQL
  DECLARE CI CURSOR FOR AOK;
```

```
EXEC SQL
  OPEN CI;
```

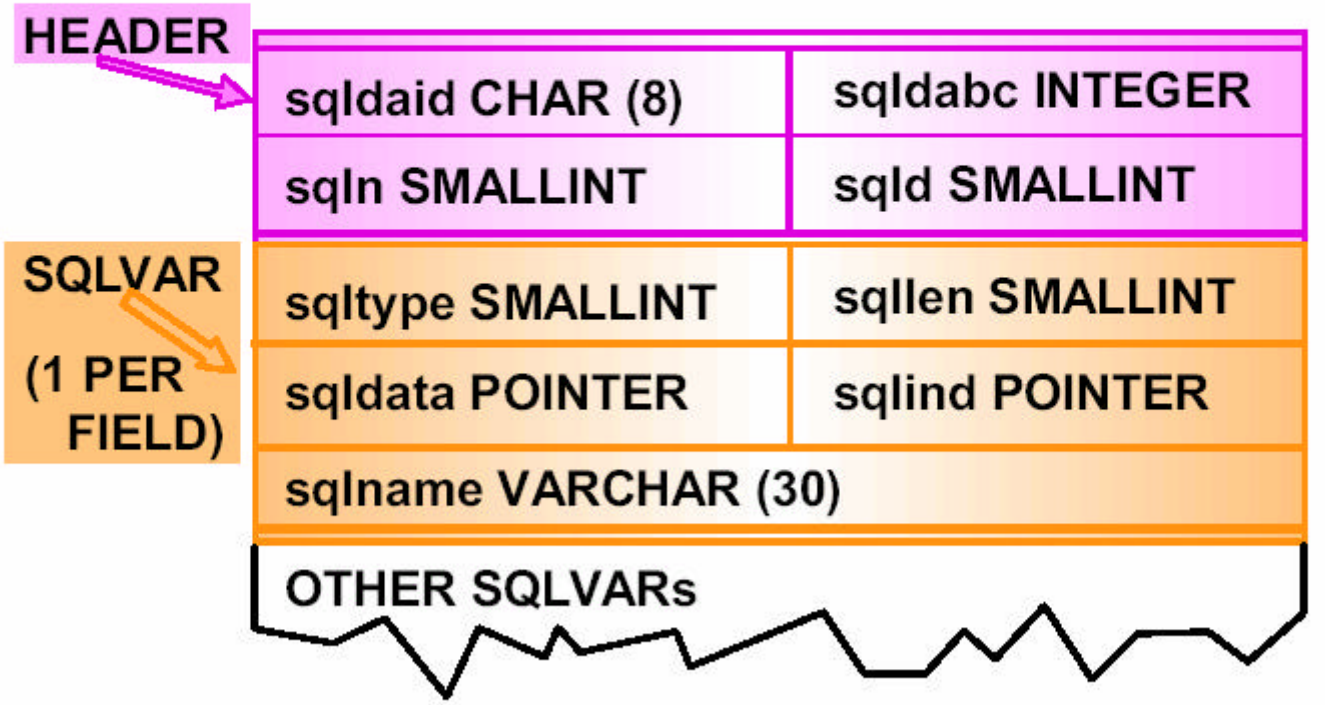


```
EXEC SQL
  FETCH CI USING
    DESCRIPTOR :sqlda;
```

host1
FREE

host2
1000000

SQLDA - Format

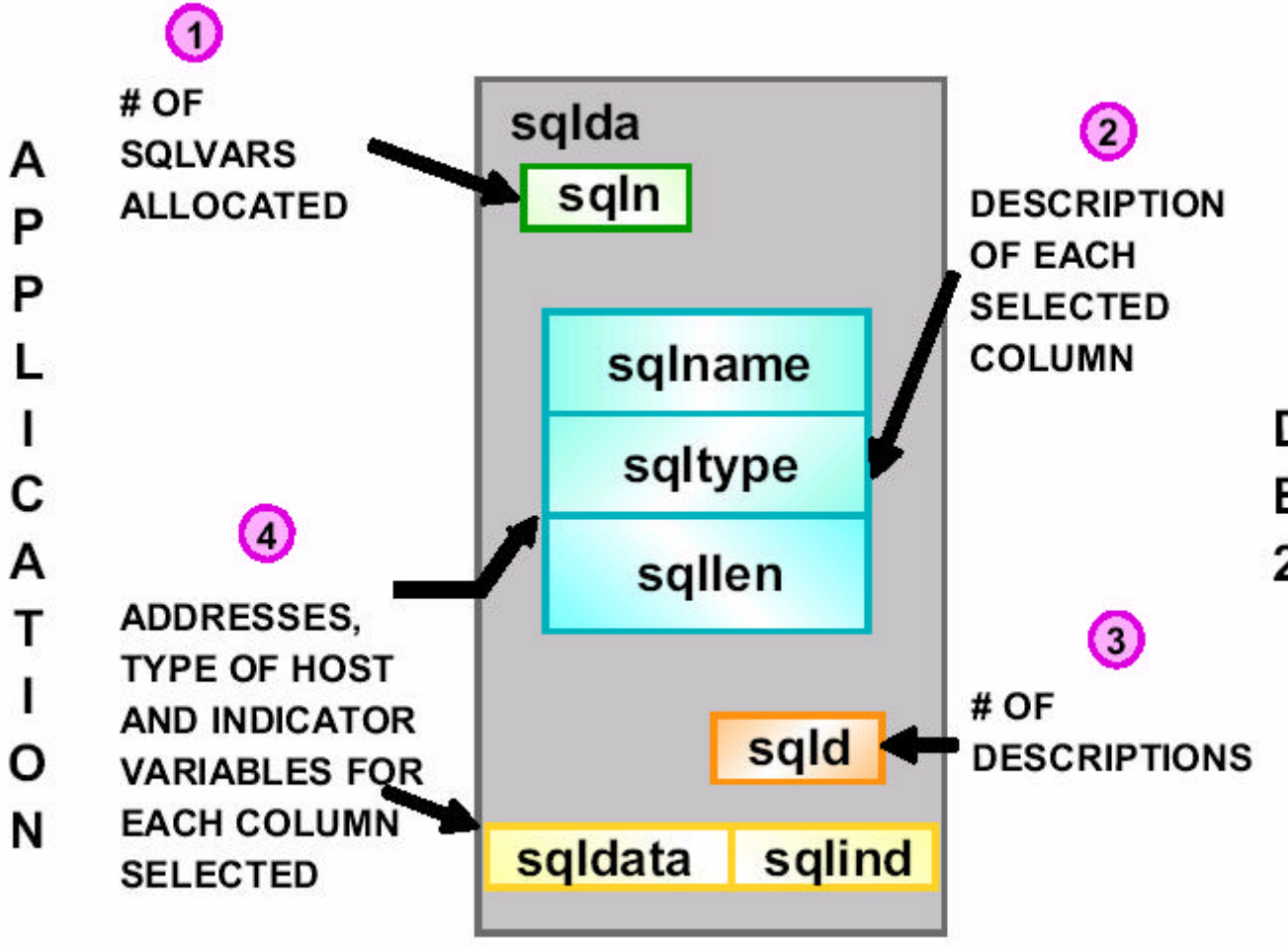


SQLDA - Column Description

sqltype	DATA TYPE	LENGTH	
448	VARCHAR,NOT NULL	MAX	
452	CHAR,NOT NULL	LENGTH	
480	FLOAT,NOT NULL	8	
484	DECIMAL,NOT NULL	PREC.	SCALE
496	INTEGER,NOT NULL	4	
500	SMALLINT,NOT NULL	2	

CODE + 1 = SAME AS CODE, WITH NULLS ALLOWED

SQLDA Usage



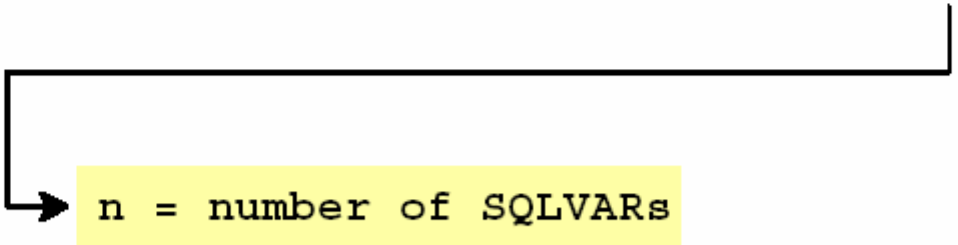
sqln < sqld → sqlcode +236, SQLSTATE '01005'

Declaring and Allocating the SQLDA

```
# include <sqlda.h>      - or -      EXEC SQL
                                       INCLUDE SQLDA;
```

```
struct sqlda * sqldaptr ;
```

```
sqldaptr = (struct sqlda *) malloc(SQLDASIZE(n));
```




```
sqldaptr->sqln = n;

EXEC SQL PREPARE AOK INTO :*sqldaptr FROM :stmt;
```

Complete SELECT Statement Pseudocode

```
      < READ or FORMULATE statement in stmt >

      < ALLOCATE sqlda and SET sqln equal
        to the number of COLUMNS supported  >

 EXEC SQL PREPARE AOK INTO :*sqldaptr FROM :stmt;

      < CHECK sqlca >

 if (sqldaptr -> sqld == 0)
{EXEC SQL EXECUTE AOK; return;}
 if (sqldaptr -> sqln <  sqldaptr -> sqld)
  {
  < ALLOCATE sqlda of sufficient size and reset sqln >
  EXEC SQL DESCRIBE AOK INTO :*sqlda;
  }
 EXEC SQL DECLARE c1 CURSOR FOR AOK;
 < ALLOCATE host variable areas >
 EXEC SQL OPEN c1;

 EXEC SQL FETCH c1 USING DESCRIPTOR :*sqldaptr;

 EXEC SQL CLOSE c1;
```

Dynamic Bind Summary

Bind Information:

- Statement used: PREPARE or EXECUTE IMMEDIATE
- Bind invoked
  - By execution of the SQL statement
  - When the SQL statement executes
- Bind analyzes the individual statement
- Strategy is not stored (*)
- Authorization is checked for the user executing the statement

Adjusting Qualifier and Owner

Jill issues:

```
db2 bind myapp.bnd  qualifier u1  owner u2  dynamicrules bind
```

- Unqualified SQL uses u1 schema
- u2 owns package
  - Can drop, rebind, grant privileges
  - u2's privilege checked for dynamic SQL

Set Current Schema

```
connect to musicdb user keith
select * from employee
```

will select from KEITH.EMPLOYEE

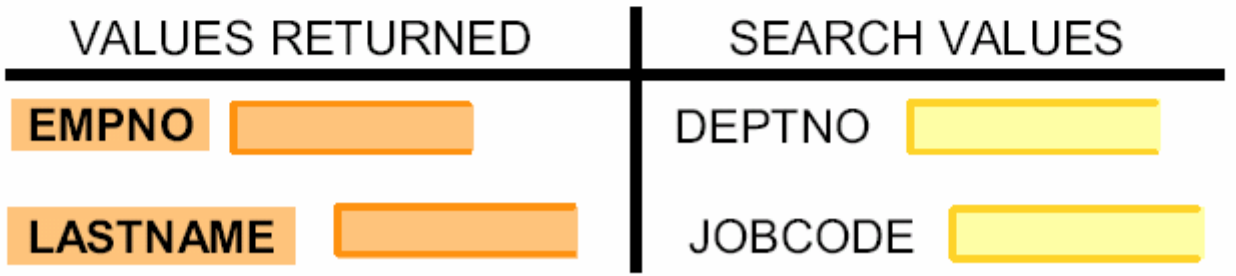
```
set current schema = 'PAYROLL'
select * from employee
```

will select from PAYROLL.EMPLOYEE

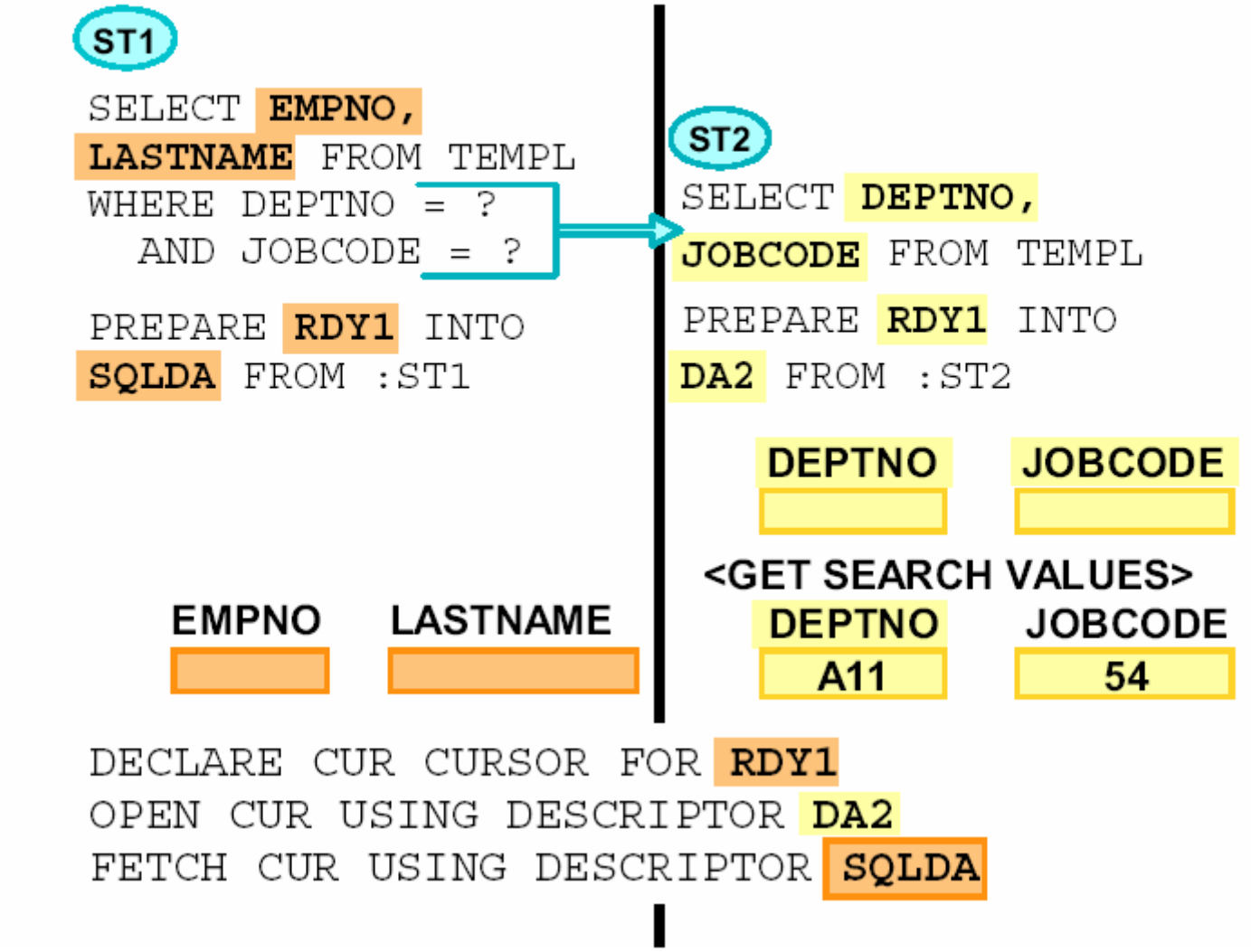
Parameterized SELECT Statement

```
SELECT EMPNO, LASTNAME
   FROM TEMPL
WHERE DEPTNO = ? AND JOBCODE = ?
```

AREAS NEEDED FOR VARIABLES:



Parameterized SELECT Statement (Cont)



Unit Summary

Since completing this unit, you should be able to:

- Code dynamic SQL statements to support
  - Complete non-SELECT statements
  - Complete Fixed-List SELECT statements
  - Complete Varying-List SELECT statements
- Use the SQLDA Block to communicate with the database manager

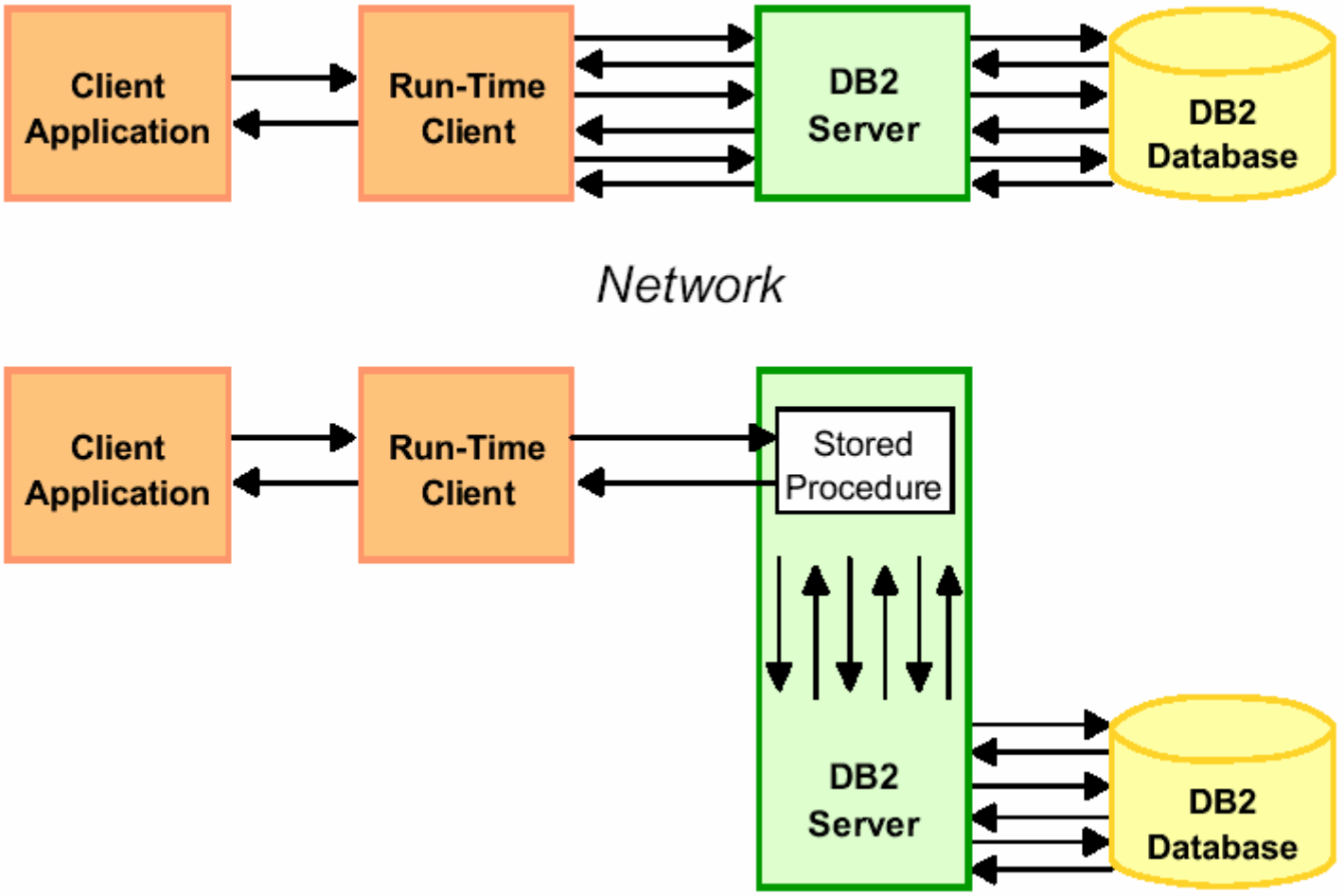
Unit 4. Stored Procedures

Unit Objectives

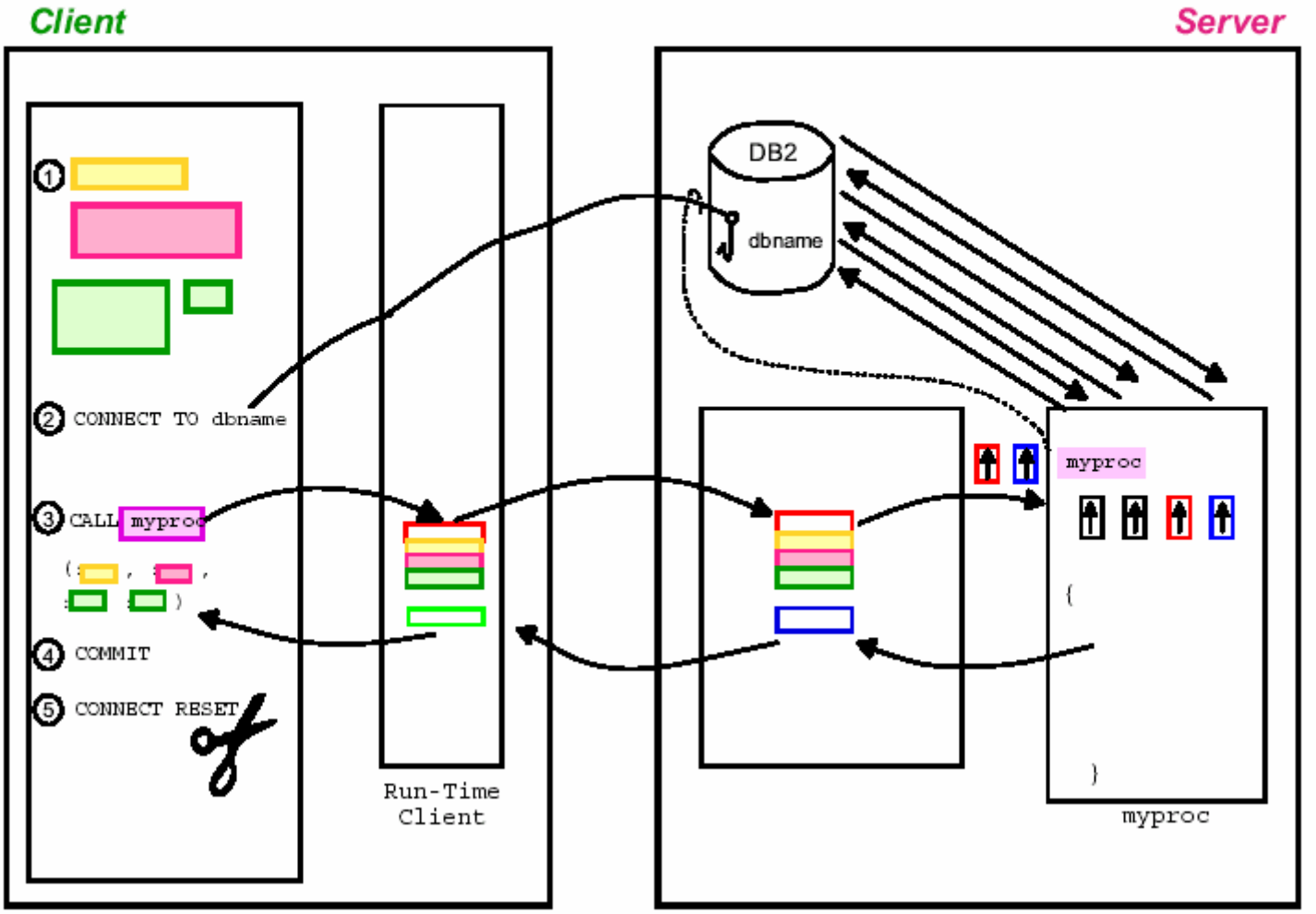
After completing this unit, you should be able to:

- Describe when the use of stored procedures is appropriate
- Describe DB2's implementation of stored procedures
- List the characteristics and specification requirements of the client application and the server procedure
- Describe the communication structures used with stored procedures
- Write stored procedures

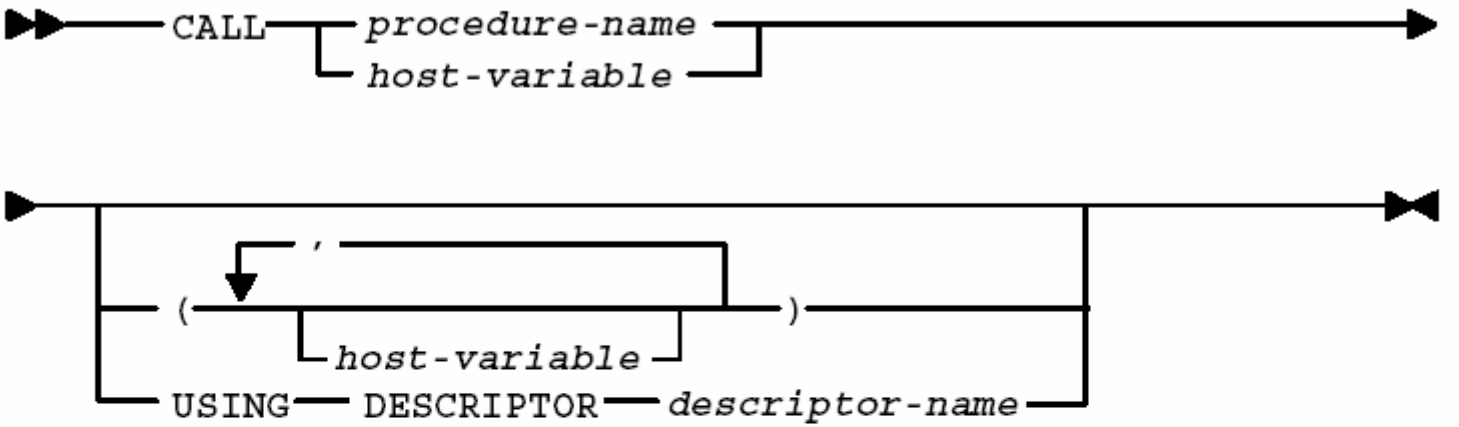
Stored Procedures



Client versus Server Function Responsibility



CALL Syntax



The client application must

- declare
- allocate
- initialize
- pass data area

for each value to be passed either TO or FROM the stored procedure.

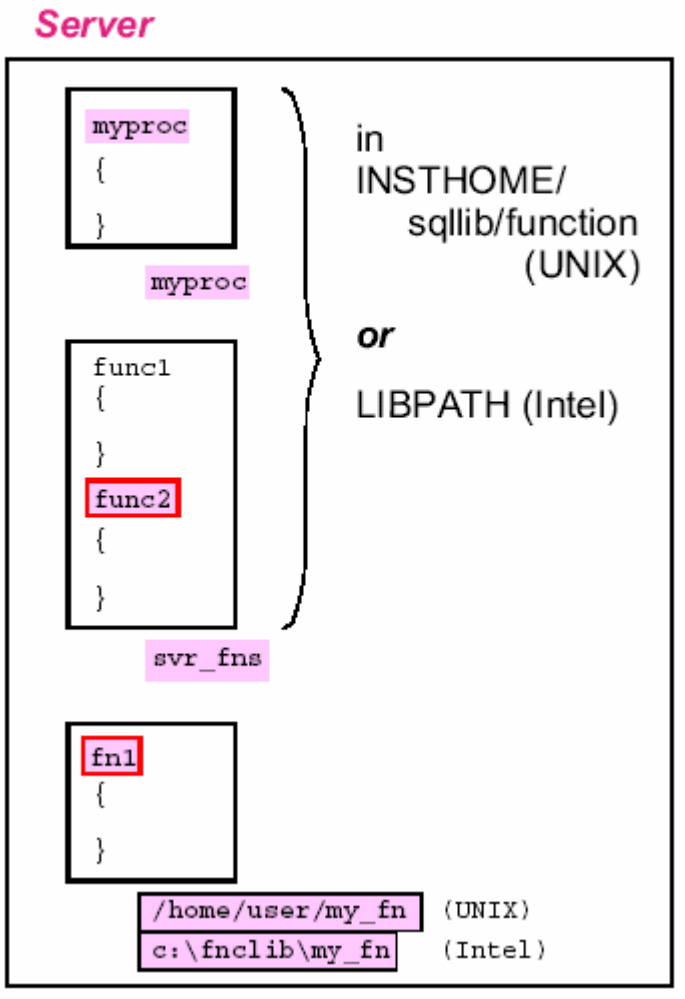
Client Application - Program Name

```
strcpy(procname, "myproc");
```

```
strcpy(procname,"svr_fns!func2")
```

```
UNIX Server:
strcpy(procname, "/home/user1/my_fn!fn1");
Intel Server:
strcpy(procname, "c:\\fnclib\\my_fn!fn1");

EXEC SQL CALL  :procname...
```



Client Application - Passing Data via Host Variables

```
EXEC SQL BEGIN DECLARE SECTION:
  char  host_var1[15];
  float host_var2;
  short ind_var2;
  long  host_var3;
  short ind_var3;
  char procname[254] = "myproc";
EXEC SQL END DECLARE SECTION;

  strcpy(host_var1,"new data");
  host_var2 = 17.6;
  ind_var2 = 0;
  ind_var3  = -1;

  EXEC SQL CALL :procname
     (:host_var1, :host_var2 :ind_var2, :host_var3 :ind_var3);
```

TYPE	INDICATOR
In and Out	Not Needed
Out	-1
In	0
In/Out from the Calling program	

Client Application - Passing Data via SQLDA

```
struct sqlda * inout_sqlda =
               (struct sqlda *) malloc (SQLDASIZE(3));
long  host_var3;
short ind_var3 = -1;
short ind_var2 = 0;

inout_sqlda->sqln = 3;
inout_sqlda->sqld = 3;

inout_sqlda->sqlvar[0].sqltype = SQL_TYP_CSTR;
inout_sqlda->sqlvar[0].sqllen = 16;
inout_sqlda->sqlvar[0].sqldata =
             (char *) malloc (inout_sqlda->sqlvar[0].sqllen);
strcpy(inout_sqlda->sqlvar[0].sqldata,"new data");

inout_sqlda->sqlvar[1].sqltype = SQL_TYP_NFLOAT;
inout_sqlda->sqlvar[1].sqllen = sizeof(float);
inout_sqlda->sqlvar[1].sqldata =
             (char *) malloc (inout_sqlda->sqlvar[1].sqllen);
*(float *)inout_sqlda->sqlvar[1].sqldata = 17.6;
inout_sqlda->sqlvar[1].sqlind = &ind_var2;

inout_sqlda->sqlvar[2].sqltype = SQL_TYP_NINTEGER;
inout_sqlda->sqlvar[2].sqllen = sizeof(long);
inout_sqlda->sqlvar[2].sqldata = (char *) &host_var3;
inout_sqlda->sqlvar[2].sqlind = &ind_var3;

EXEC SQL Call myproc
        USING DESCRIPTOR :*inout_sqlda;
```

CREATE PROCEDURE

```
CREATE PROCEDURE MYPROC (INOUT HOST1 CHAR(15),
          IN HOST2 DOUBLE, OUT HOST3 INTEGER)
          EXTERNAL NAME '/home/user1/myfn!fn1'
          LANGUAGE C
          PARAMETER STYLE DB2DARI

       EXEC SQL CALL MYPROC
           (:host_var1, :host_var2 :ind_var2,
                  :host_var3 :ind_var3);
```

What Gets Passed?

- In → some of Input SQLDA
- Out ← all of SQLCA
  some of Output SQLDA

Building the Stored Procedure Application

Client Application – Normal database manager application

Server Procedure – Must create a library
- Use export file

– Place in INSTHOME/sqllib/function (UNIX)

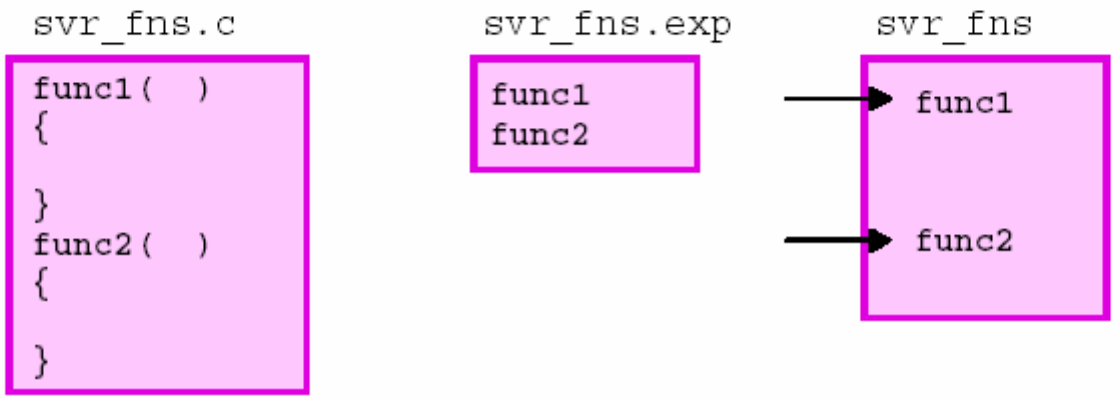
OR

LIBPATH (Intel)

OR

Client supplies full path name

```
xlc  svr_fns.c  -ldb2  -bE:svr_fns.exp
```



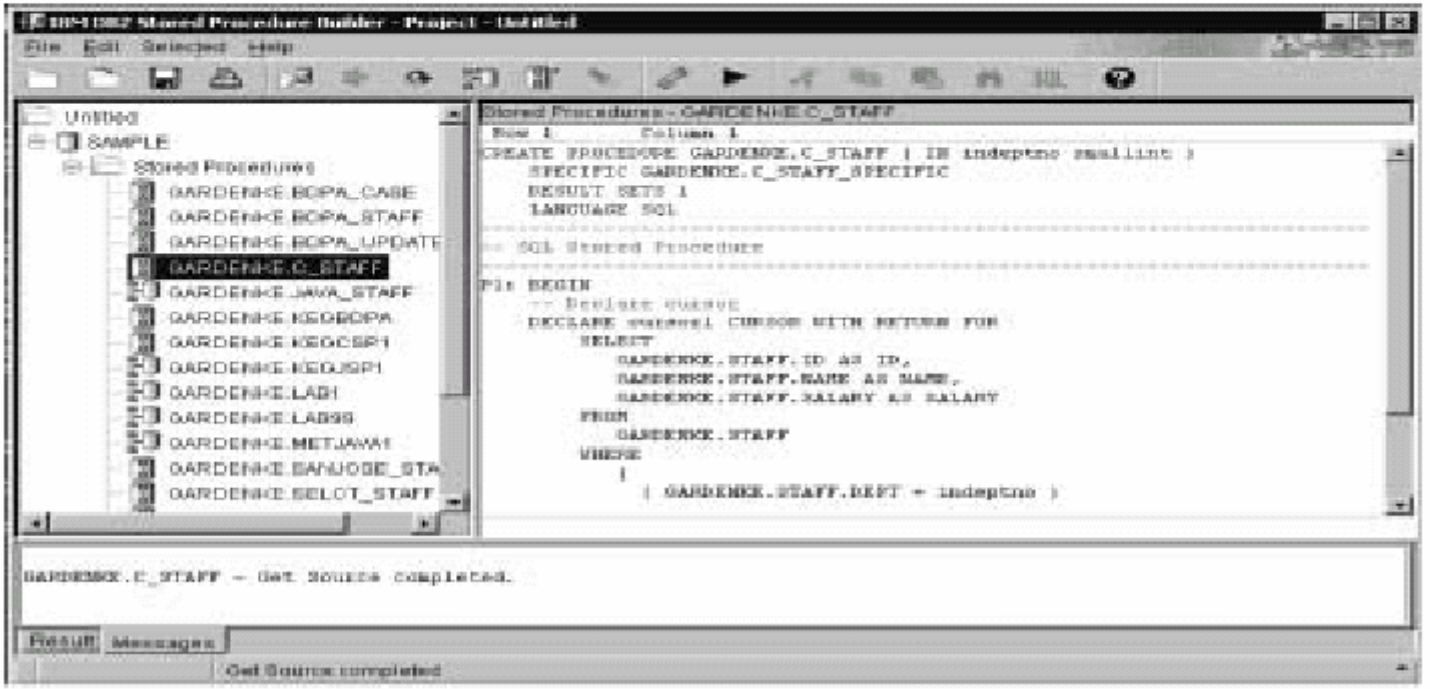
Troubleshooting Checklist

- Check build process of server procedure
- Ensure entry point name matches
- Test with local execution
- If using a descriptor
  - Ensure proper setting of sqln and sqld
  - Ensure indicator variable supplied for sqltypes that allow null
- If using host variables, provide an indicator variable for either input-only or output-only parameters
- Ensure server procedure is in the default directory or the client application process provides a full path
- Check that server passes SQLCA back and client checks for success
- Use `fprintf` for debugging the server code

Stored Procedure Builder

Functionality:

- Builds the server portion of the Stored Procedure
- Modifies and rebuild existing Procedures
- Test and debug the execution of installed Procedures



SQL Procedure Language

- **SQL Procedures support:**
  - Multiple parameters: input, output, input/output
  - Returning multiple output result sets to client
- **SQL Procedures are defined in DB2 catalog**
- **SQL Procedure source is stored in DB2 catalog**
- **SQL Procedure language is folded to upper case**
  - Exception: delimited values

SQL Procedure Language (Cont)

- **An SQL Procedure consists of:**
  - A CREATE PROCEDURE statement
    - LANGUAGE = SQL
  - A procedure body which may include:
    - Compound statement(s): BEGIN … END
    - Declaration statements
    - Assignment statements
    - Conditional statements
    - Iterative control structure: LOOPs, etc.
    - Exception Handling
    - CALL another stored procedure

Stored Procedure Builder

LAUNCHING

- DB2 UDB program folder
- IBM Visual Age for JAVA
- Microsoft Visual Studio
- Microsoft Visual Basic

Stored Procedure Builder (Cont)

DEVELOPMENT ALTERNATIVES

- JAVA on DB2/UDB for UNIX, Windows and OS/2
  - JDBC and SQLJ
- SQL / PL
  - BASIC-like language

PARAMETER PASSING

- Single result set
- Multiple result sets
- Output parameters

Unit Summary

Since completing this unit, you should be able to:

- Describe when the use of stored procedures is appropriate
- Describe DB2's implementation of stored procedures
- List the characteristics and specification requirements of the client application and the server procedure
- Describe the communication structures used with stored procedures
- Write stored procedures

Unit 5. Introduction to Call Level Interface (CLI)

Unit Objectives

After completing this unit, you should be able to:

- Identify the differences between CLI and embedded SQL
- Identify the advantages of CLI
- Identify the disadvantages of CLI
- Define the primary tasks of an application
- Describe the purpose of handles
- Identify how a transaction is started
- Process results sets returned from stored procedures

What Is CLI?

- IBM's callable SQL interface
- Supported in C and C++
- Uses function calls to pass dynamic SQL to DB2
- Alternative to Embedded SQL
- Incorporates both the ODBC and X/Open CLI functions; aligned with emerging ISO CLI standard
- Can also act as an ODBC driver when loaded by ODBC driver manager

CLI Advantages

- **Portability**
  - Consistent interface
  - No precompile
- **Bind not required on customers' database**
- **SQLCA and SQLDA not needed**
  - Necessary data structures are allocated and provided by DB2 CLI
- **Allows use of arrays of data to be specified on input**
- **Allows retrieval of multiple rows of a result set directly into an array**
- **Consistent interface to query catalog tables across DBMSs**
- **Read-only scrollable cursors**
- **Uses function calls to perform operations instead of SQL statements that must be preprocessed**
- **CLI offers the ability to have the SQL statements provided at run time**

CLI Considerations

- **Performance - Dynamic SQL**
  - Bind at execution
  - Current Statistics
  - Network Traffic
- **Security**
  - Authorizations are validated at run time, for the execution of each individual statement
- **DB2 specific Utility APIs - restricts portability**

CLI and Security

STAT1.SQC

```
EXEC SQL
DELETE FROM
EMP
WHERE ID =:hv;
```

CLI

```
DELETE FROM
EMPL WHERE
ID = '123'
```

```
GRANT EXECUTE ON PACKAGE
STAT1 TO USER  USERA
```

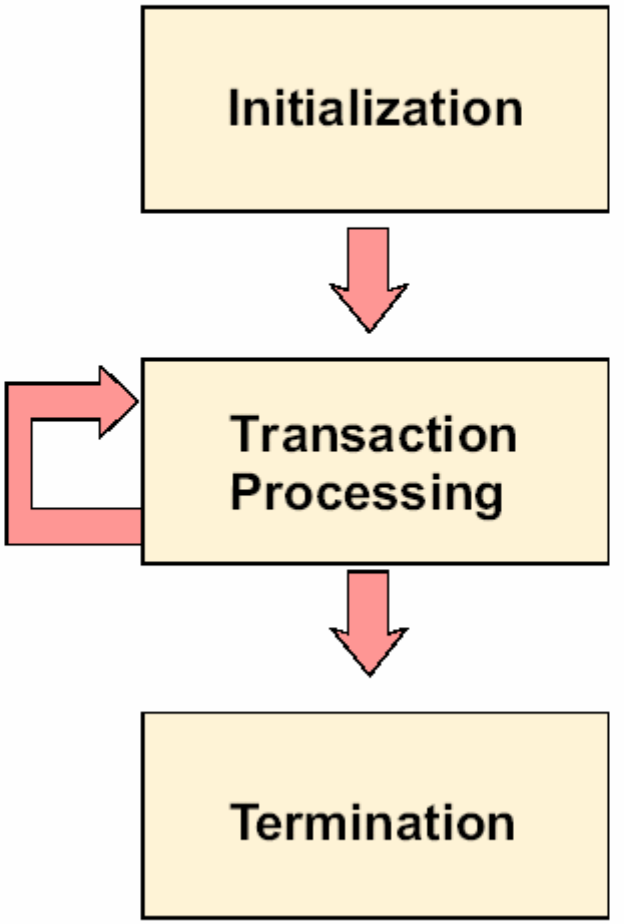
USERA

```
GRANT DELETE ON TABLE
EMPL TO USER  USERA
```

DB2 CLI versus Embedded SQL

- No precompilation
- No explicit cursor declaration or OPEN Cursor
- Parameter markers allowed on EXECUTE IMMEDIATE
- SQLEndTran() instead of COMMIT or ROLLBACK
- Environment handle and statement handles instead of SQLCA and SQLDA
- SQLSTATE instead of SQLCODE
- Multithreaded thread-safe applications without calling context management DB2 APIs Threadsafe

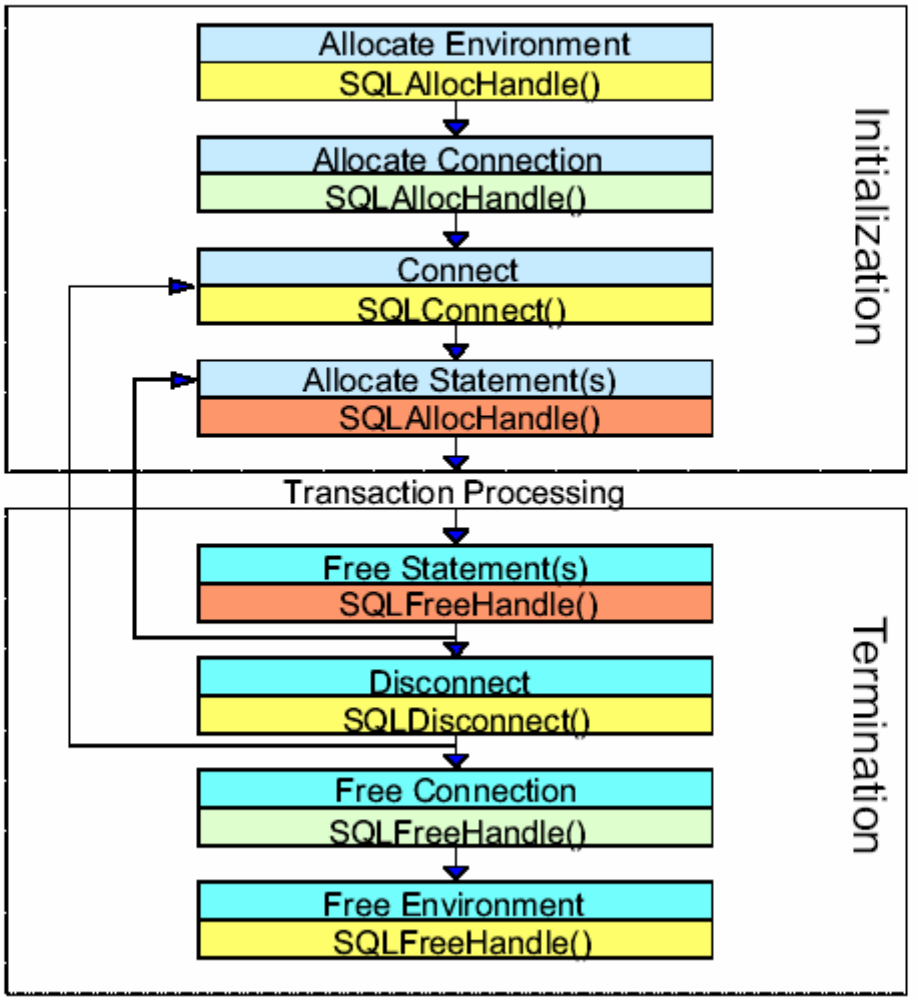
Application Tasks



Application Handles

- **Environment Handle - SQLHENV**
  - Global information regarding the state of the application
  - One per application
- **Connection Handle - SQLHDBC**
  - Information associated with a connection managed by DB2 CLI
  - Multiple per application
- **Statement Handle - SQLHSTMT**
  - Data object that is used to track the execution of a single SQL statement (including statement options, SQL text, cursor information, and more)
  - Multiple per application

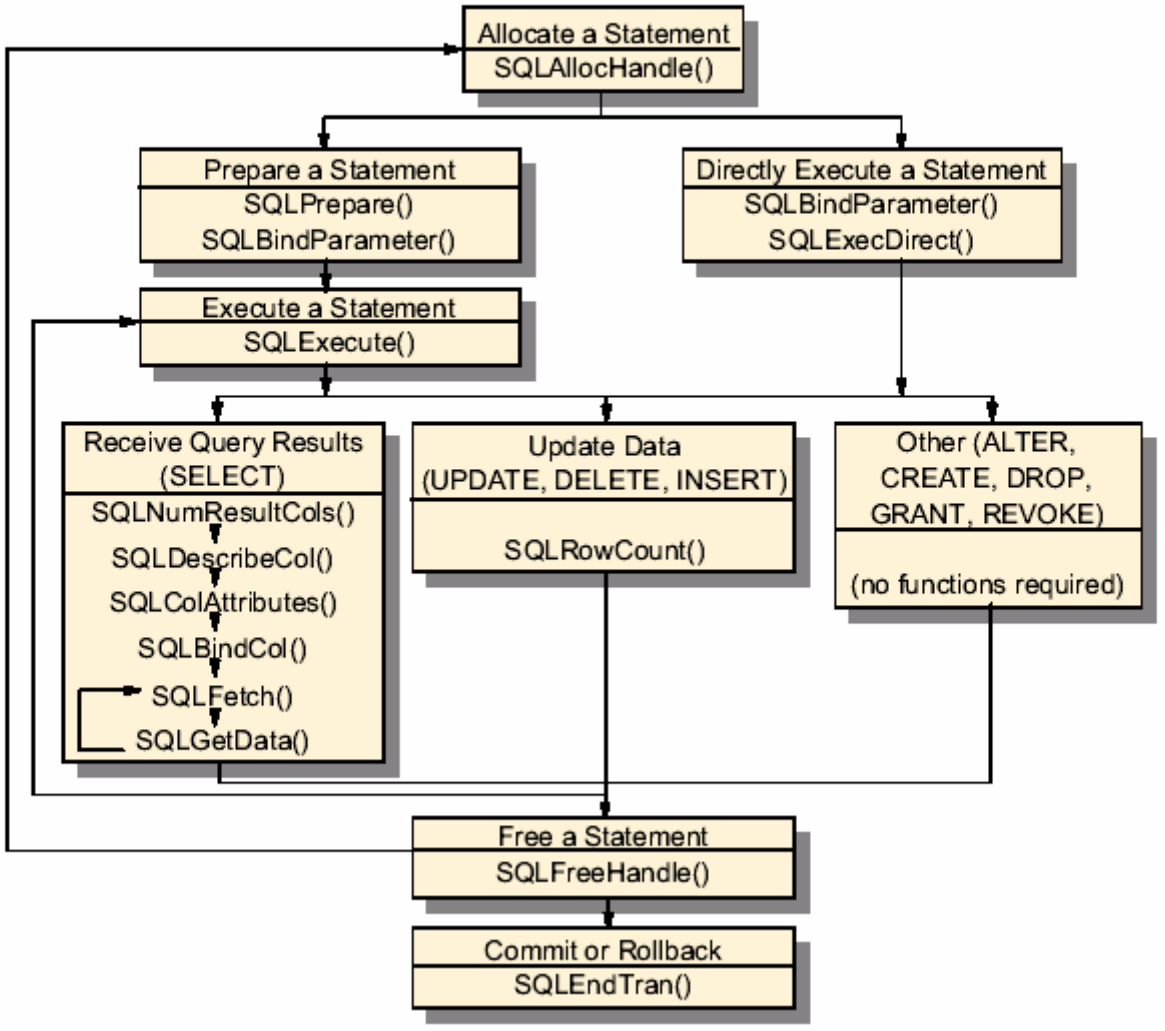
Initialization and Termination



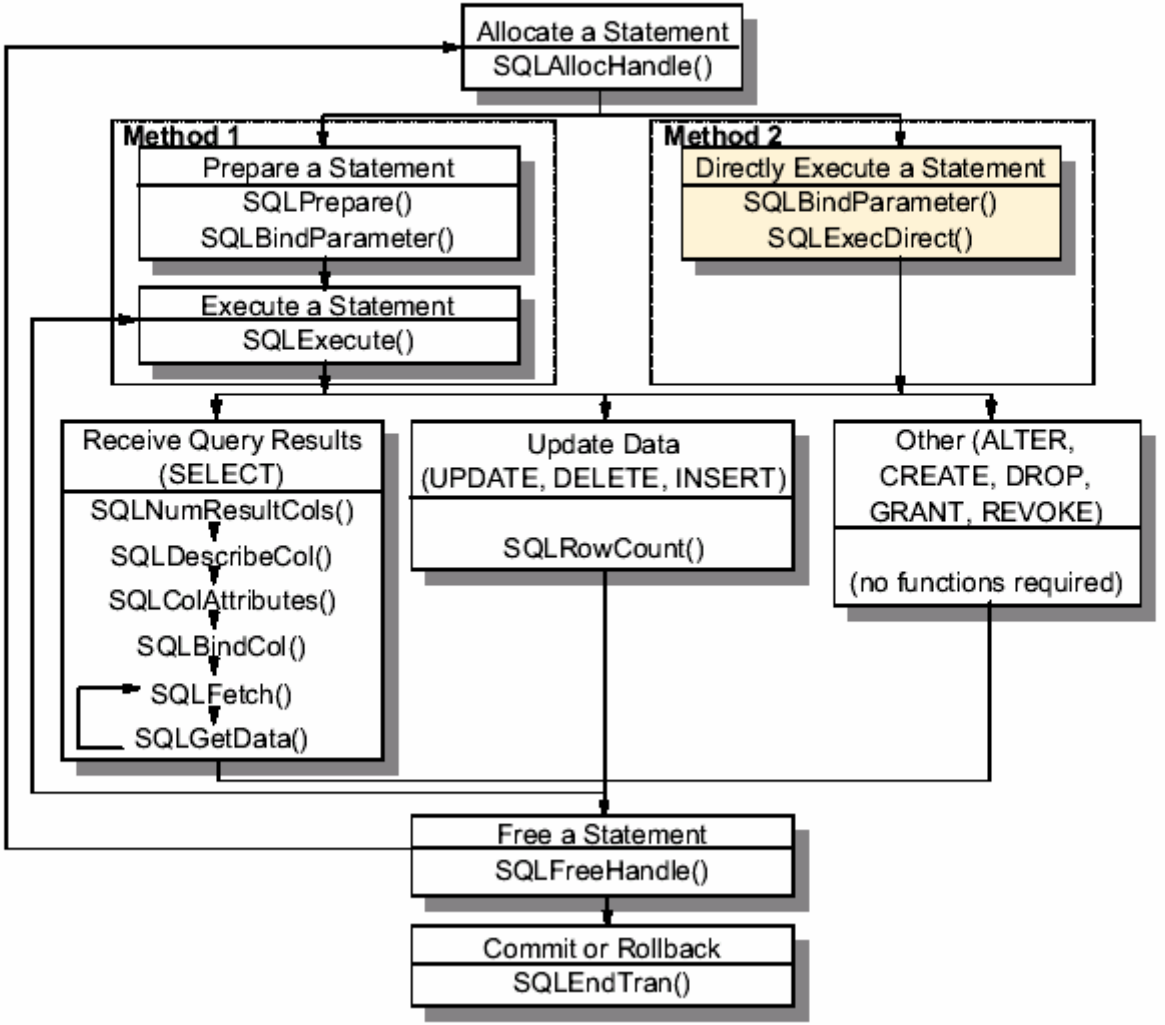
Application Handles

- **Environment Handle - SQLHENV**
  - Global information regarding the state of the application
  - One per application
- **Connection Handle - SQLHDBC**
  - Information associated with the connection managed by DB2 CLI (general status information, transaction status, diagnostic information)
  - Multiple per application - coordinated or concurrent transaction
- **Statement Handle - SQLHSTMT**
  - Data object that is used to track the execution of a single SQL statement (statement options, SQL statement text, dynamic parameters, cursor information, bindings for dynamic arguments and columns, result values, and status information)
  - Multiple per application - limited by overall system resources
- **Descriptor Handle -**
  - Data object that contains information about columns in a result set, and dynamic parameters in an SQL statement

Transaction Processing Overview



Preparing and Executing

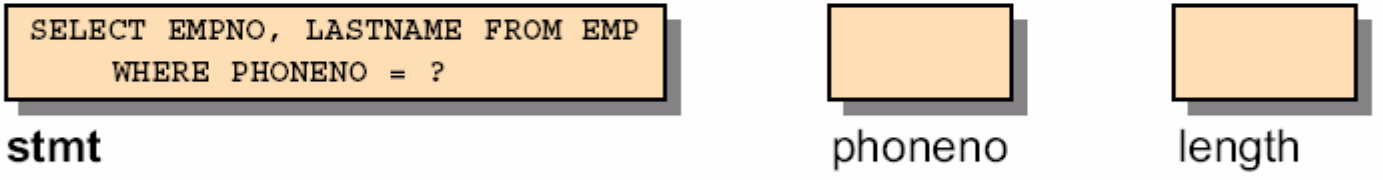


Parameter Markers

Embedded Static SQL

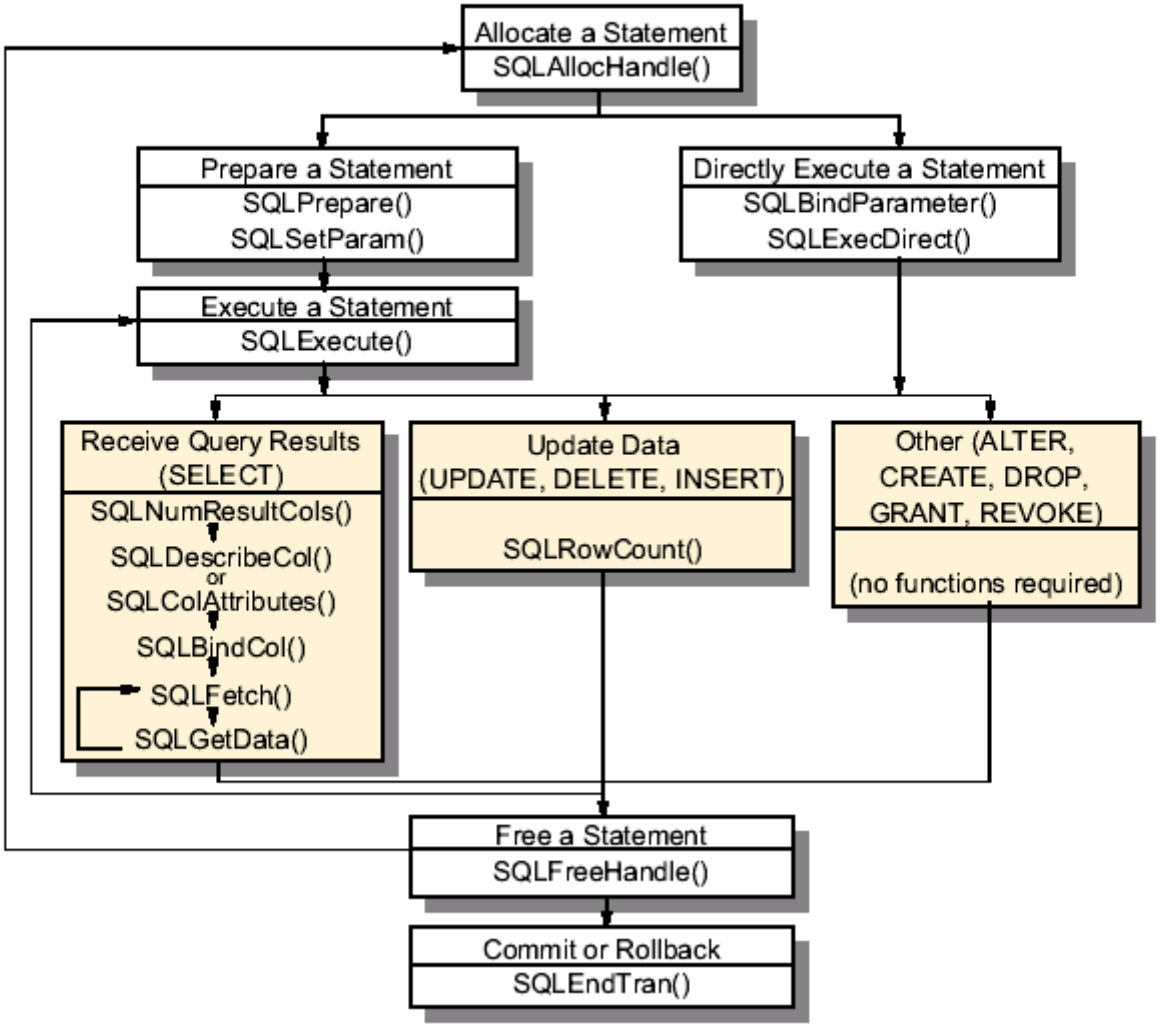
```
EXEC SQL SELECT EMPNO, LASTNAME
INTO :empno, :lastname
FROM EMP
WHERE PHONENO = :phoneno;
```

DB2 CLI

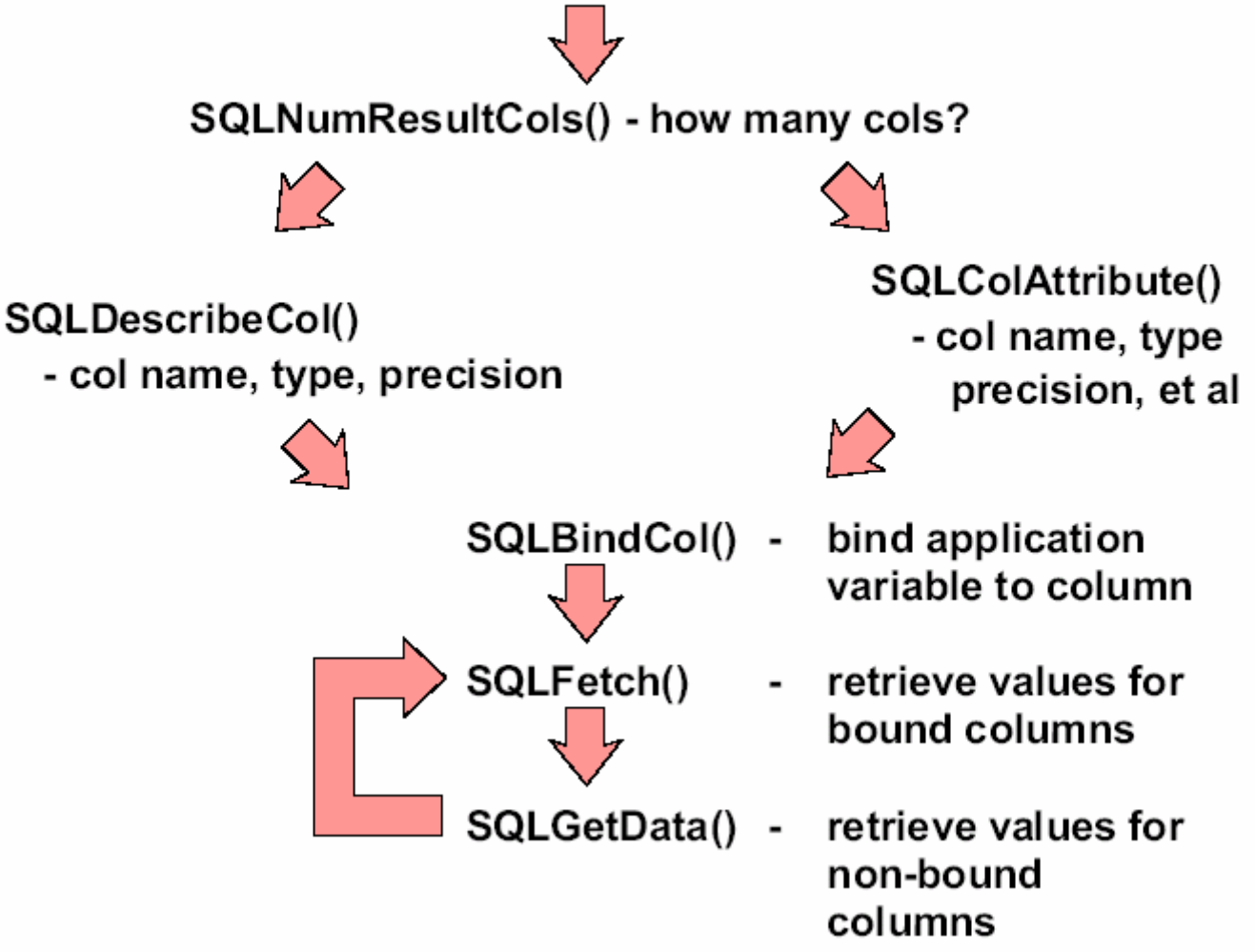



```
SQLPrepare(stmt_handle,stmt,SQL_NTS);
SQLBindParameter(stmt_handle,1,SQL_PARAM_INPUT,
   SQL_INTEGER,0,0,phoneno,0,NULL);
gets (phoneno);
SQLExecute(stmt_handle);
```

Processing Results



Retrieve Query Results (SELECT)



Update Data

- **Check for diagnostic messages**
- **SQLRowCount() - get number of rows affected by SQL statement**
- **Positioned UPDATE or DELETE requires the cursor name**
  - SQLGetCursorName()

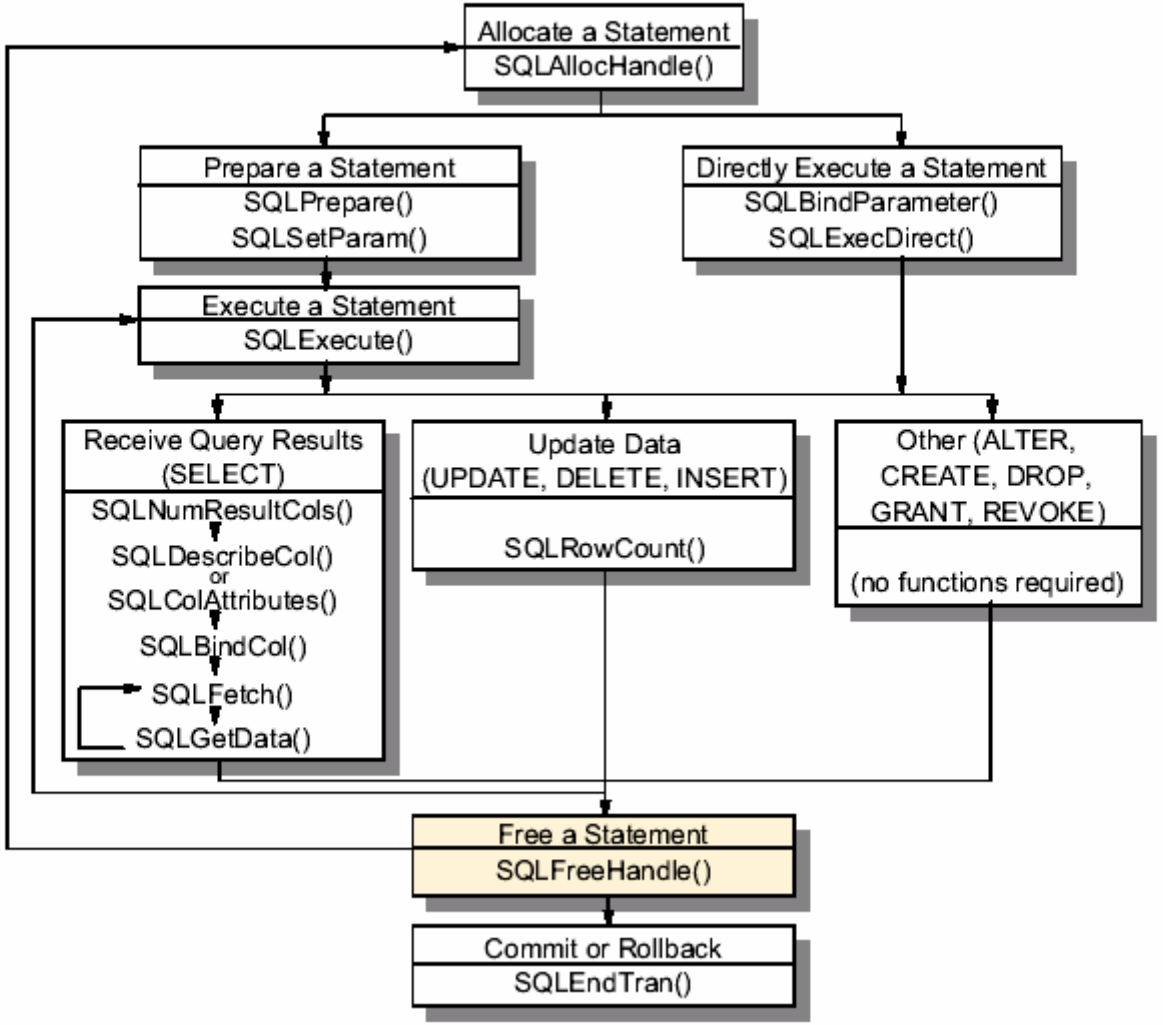
Positioned UPDATE or DELETE

```
SELECT EMPNO, PHONENO FROM EMP FOR UPDATE OF PHONENO
```

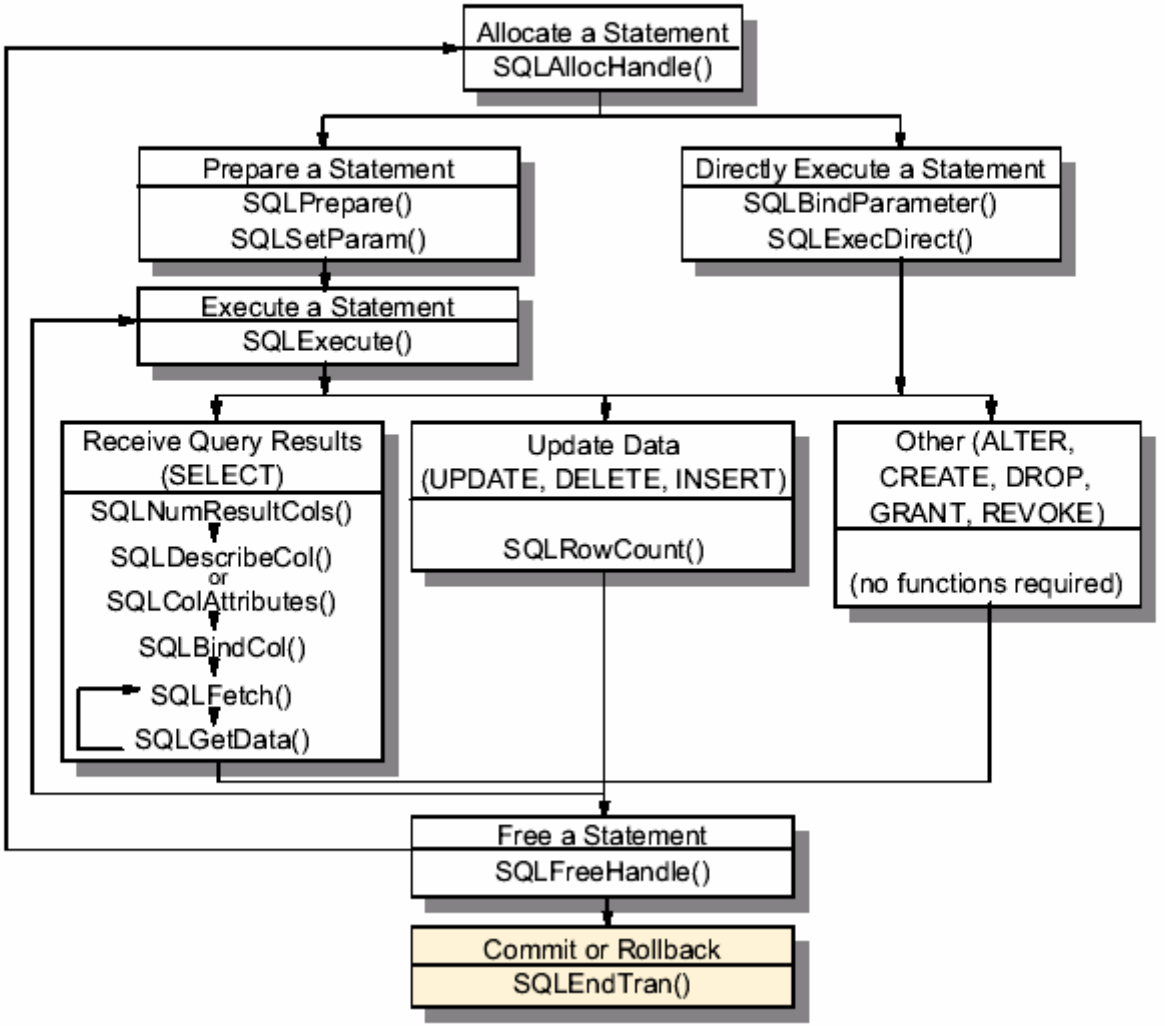
stmt

```
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &sel_hdl);
:
SQLAllocHandle (SQL_HANDLE_STMT, hdbc, &upd_hdl);
:
SQLExecDirect(sel_hdl, stmt, SQL_NTS);
:
SQLGetCursorName(sel_hdl, cursor, size of (cursor), &clength);
:
SQLBindCol(...); /* for each column */
:
SQLFetch(sel_hdl);
:
/* if this one is to be updated */
  sprintf (updstmt, "UPDATE EMP SET PHONE='%s'
         WHERE CURRENT of %s", newphone, cursor);
 SQLExecDirect(upd_hdl, updstmt, SQL_NTS);
:
```

Freeing Statement Handles

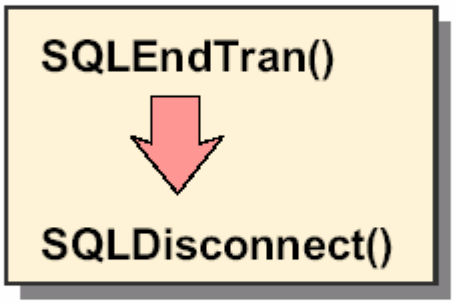


Commit or Rollback



Commit or Rollback (Cont)

- **Transactions are started by:**
  SQLPrepare(), SQLExecDirect(), SQLGetTypeInfo(), or by Meta Data Function calls
- **Transactions are ended by:**



- **After**
  - Prepared SQL statements survive transactions
  - Statement handles are still valid
  - Cursor names, bound parameters, and column bindings are retained
  - Cursor positions are maintained after commit
  - Cursors are closed, results discarded after rollback

Processing Errors

- Return Codes
- Detailed Diagnostics (SQLSTATEs, messages, SQLCA)
- Call SQLGetDiagRec() or SQLGetDiagField() after receiving SQL_SUCCESS_WITH_INFO_or SQL_ERROR from another function

SQL Error and SQLGetDiagRec and SQLGetDiagField

```
SQLGetDiagRec()  and SQLGetDiagField()

main() {
  SQLCHAR buffer[SQL_MAX_MESSAGE_LENGTH+1];
  SQLCHAR sqlstate[SQL_SQLSTATE_SIZE+1];
  SQLINTEGER sqlcode;
  SQLSMALLINT length;
  ...
  ...
  ...
  rc=(SQLExecute(hstmt, sql, SQL_NTS);
  while (SQLGetDiagRec(SQL_HANDLE_STMT,
              hstmt, 1, sqlstate, &sqlcode, buffer, SQL_MAX_MESSAGE_LENGTH+1
              , &LENGTH)) == SQL_SUCCESS) {
              printf("\n SQLSTATE is %s", sqlstate);
              printf("\n SQLCODE is %d", sqlcode);
              printf("\n Message Text %s", buffer);
             i++;
             }
//    or
   while(SQLGetDiagField(SQL_HANDLE_STMT, hstmt, 1,
            SQL_DIAG_MESSAGE_TEXT, buffer;
            SQL_MAX_MESSAGE_LENGTH+1, &length) == SQL_SUCCESS){
            SQLGetDiagField(SQL_HANDLE_STMT,hstmt, 1, SQL_DIAG_SQLSTATE,sqlstate,
                                          SQL_STATE_SIZE+1, &length);
            SQLGetDiagField(SQL_HANDLE_STMT, hstmt, 1, SQL_DIAG_NATIVE, &sqlcode,
                                           SQL_IS_POINTER,0);
            printf("\n SQLSTATE is %s ", sqlstate);
            printf("\n SQLCODE is %d ", sqlcode);
            printf("\n Message Text %s", buffer);
            i++;
            }
   }
```

Metadata Function Calls

SQLTables()

SQLColumns()

SQLPrimaryKey()

SQLForeignKey()

SQLStatistics()

SQLProcedures()

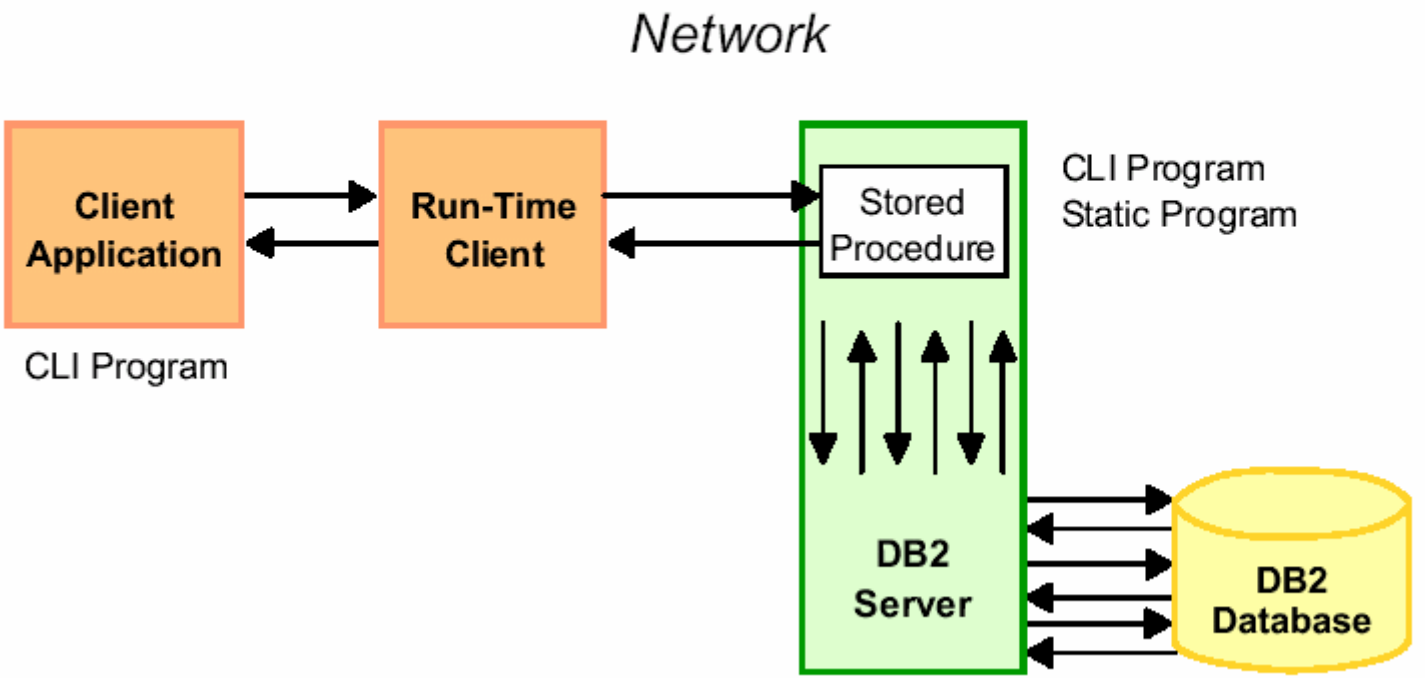
SQLProcedureColumns()

SQLSpecialColumns()

SQLTablePrivileges()

SQLColumnPrivileges()

CLI and Stored Procedures

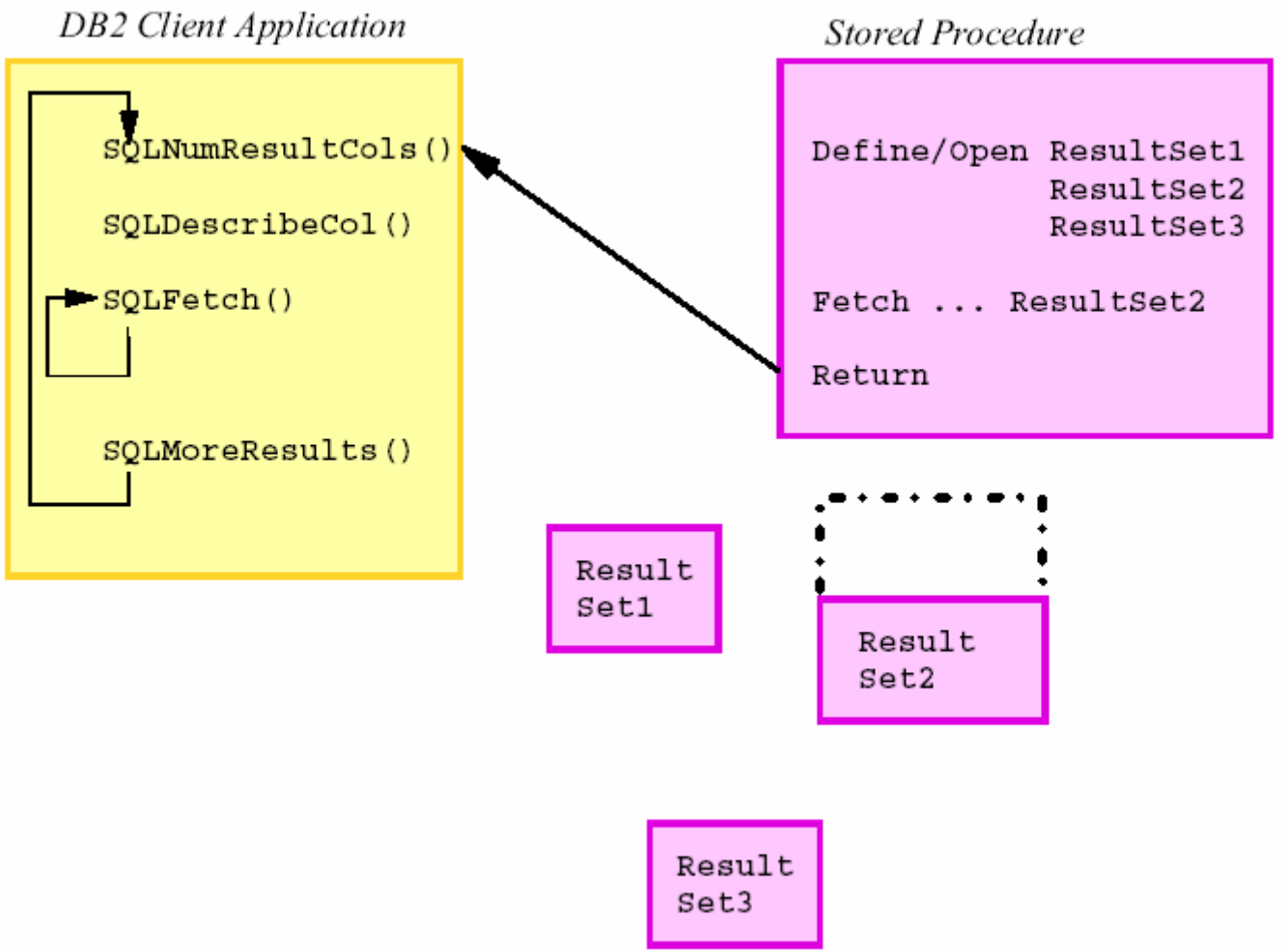


CLI Program as Client Application

```
SQLCHAR stmt[] = "CALL OUTSRV (?, ?)";

SQLPrepare (hstmt, stmt, SQL_NTS);
SQLBindParamter (hstmt, 1, SQL_PARAM_INPUT_OUTPUT,
  SQL_C_DEFAULT, SQL_INTEGER, 0, 0, &host1, 0, NULL);
SQLBindParameter (hstmt, 2, SQL_PARAM_OUTPUT,
  SQL_C_DEFAULT, SQL_CHAR, 20, 0, name, 21, NULL);
SQLExecute (hstmt);
```

Returning Result Sets from Stored Procedures



CLI Calling Application - Multiple Result Sets

Calling Application

```
SQLExecDirect( hstmt, "CALL STOR_PROC(?)", SQL_NTS);

while(rc != SQL_NO_DATA_FOUND)
{
   rc = SQLFetch( hstmt );
   SQLGetData(.....);
}
rc = SQLMoreResults( hstmt );
if(rc != SQL_NO_DATA_FOUND)
   {
   while(rc != SQL_NO_DATA_FOUND)
      {
         rc = SQLFetch( hstmt );
         SQLGetData(.....);
      }
   }
```

CLI Program as Stored Procedure

- Stored procedure runs under same connection and transaction as the client application

- Stored procedure must invoke `SQLConnect()` method with null input parameters to associate with the underlying connection of the client application

```
SQLAllocHandle (SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);

SQLAllocHandle (SQL_HANDLE_DBC, henv, &hdbc);

SQLConnect (hdbc, NULL, SQL_NTS, NULL, SQL_NTS, NULL, SQL_NTS);
```

Unit Summary

Since completing this unit, you should be able to:

- Identify the differences between CLI and embedded SQL
- Identify the advantages of CLI
- Identify the disadvantages of CLI
- Define the primary tasks of an application
- Describe the purpose of handles
- Identify how a transaction is started
- Process results sets returned from stored procedures