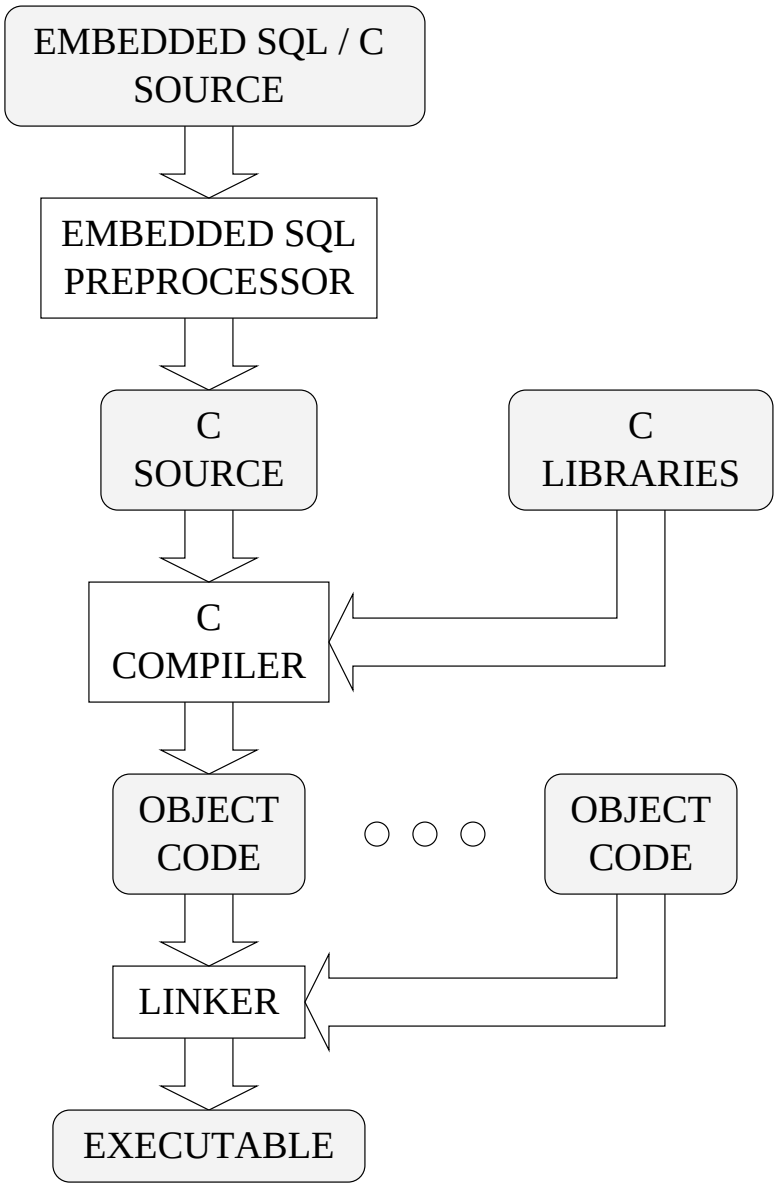
III. Advanced SQL

Lecture Topics

- Introduction to embedded SQL.
- The Open Database Connectivity (ODBC) interface.
- Security in SQL.

Development Process for Applications



A Simple Example

```
#include <stdio.h>
EXEC SQL INCLUDE SQLCA;
#include "sqldef.h"

main()
{
   db_init( &sqlca );

   EXEC SQL WHENEVER SQLERROR GOTO error;

   EXEC SQL CONNECT "admin" IDENTIFIED BY "school";

   EXEC SQL UPDATE Employee
            SET Empname = 'Miller'
            WHERE Empnum = 1056;

   EXEC SQL COMMIT WORK;

   EXEC SQL DISCONNECT;

   db_fini( &sqlca );
   return( 0 );

error:
   printf( "update failed -- sqlcode = %ld\n",
           sqlca.sqlcode );
   return( -1 );
}
```

Ordering on Statements

Textural ordering: order in which statements appear in source file.

Execution ordering: order in which statements are executed.

- "EXEC SQL INCLUDE SQLCA;" must be texturally first embedded SQL statement.
- "EXEC SQL CONNECT . . .;" must be first embedded SQL statement executed.

Static Embedded SQL

SQL DML and DDL can be embedded in a C program by prefixing with “EXEC SQL” and suffixing with “;”.

Host variables are used to send and receive values from the database engine.

- Values can be sent by using host variables in place of constants.
- Values can be received by using host variables in an INTO clause.

The SELECT statement is different in Embedded SQL.

Declaring Host Variables

Ex. "Create a new employee with employee number, name, initials and room number given by host variables."

```
EXEC SQL BEGIN DECLARE SECTION;
long int empnum;
char empname[20];
char initials[6];
char roomnum[6];
EXEC SQL END DECLARE SECTION;

/* program fills in variables with appropriate values
*/

EXEC SQL INSERT
   INTO EMPLOYEE( Empnum, Empname, Initials, Roomnum )
   VALUES( :empnum, :empname, :initials, :roomnum );
```

Domain and Type Correspondence

Domain	**C Type**
INTEGER	long int v;
SMALLINT	short int v;
DECIMAL(m,n)	DECL_DECIMAL(m,n) v;
FLOAT	float v;
CHAR(n)	DECL_FIXCHAR(n) v; or char v[n];
VARCHAR(n)	DECL_VARCHAR(n) v;
BIT(n) BIT VARYING(n)	DECL_BINARY(n) v; (same)

Domain and Type Correspondence (cont'd)

Domain	C Type
DATE	DECL_DATETIME v;
TIME	(same)
TIME(i)	(same)
TIMESTAMP	(same)
INTERVAL YEAR/MONTH	(not representable)
INTERVAL DAY/TIME	(not representable)

Querying in Embedded SQL

Two forms

1. SELECT statements that return at most one row.
2. SELECT statements that return multiple rows.

An INTO clause is added in the first case.

A **cursor** must be used in the second case.

Selecting At Most One Tuple

Ex. "A procedure that finds the surname of a student with a given student number."

```
EXEC SQL BEGIN DECLARE SECTION;
long int studnum;
char name[20];
EXEC SQL END DECLARE SECTION;

int GetStudentName( long int student )
{
   studnum = student;

   EXEC SQL
      SELECT Surname INTO :name
      FROM admin.student
      WHERE studnum = :studnum;

   if( SQLCODE == SQLE_NOTFOUND )
      { return( 0 );  /* no such student number */ }
   else if( SQLCODE < 0 )
      { return( -1 ); /* error */ }
   else
      { return( 1 );  /* found */ }
}
```

Indicator Variables

A host variable that indicates null values.

Ex. "A procedure that finds the address of a student with a given student number."

```
EXEC SQL BEGIN DECLARE SECTION;
long int studnum;
char address[27];
short int addind;
EXEC SQL END DECLARE SECTION;

int GetStudentAddress( long int student )
{
   studnum = student;

   EXEC SQL
      SELECT Address INTO :address :addind
      FROM admin.student
      WHERE studnum = :studnum;

   if( SQLCODE == SQLE_NOTFOUND )
      { return( 0 );  /* no such student number */ }
   else if( SQLCODE < 0 )
      { return( -1 ); /* error */ }
   else if( addind < 0 )
      { strcpy( address, "UNKNOWN" );
        return( 1 );  /* found but null */ }
   else
      { return( 1 );  /* found */ }
}
```

Using a Cursor to Scan Multiple Rows

Ex. "A procedure that prints the names of all students."

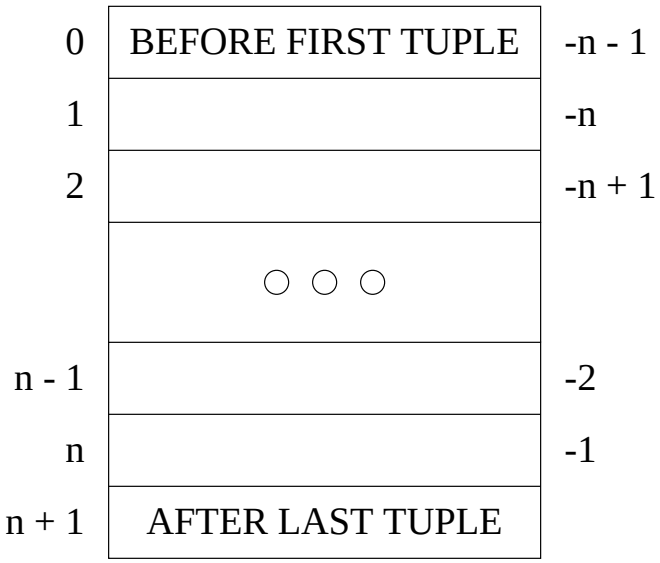
```
void PrintStudentNames( void )
{
   EXEC SQL BEGIN DECLARE SECTION;
   char name[27];
   EXEC SQL END DECLARE SECTION;

   EXEC SQL DECLARE C1 CURSOR FOR
      SELECT Initials || ' ' || Surname
      FROM admin.student;
   EXEC SQL OPEN C1;
   for( ;; ) {
      EXEC SQL FETCH NEXT C1 INTO :name;
      if( SQLCODE == SQLE_NOTFOUND )
         { break; /* have completed scan */ }
      else if( SQLCODE < 0 )
         { break; /* error */ }
      printf( "Name: %s\n", name );
   }
   EXEC SQL CLOSE C1;
}
```

Cursor Positioning

Can be in one of three places

1. before first tuple
2. on a tuple
3. after last tuple

Tuple position numbering



The FETCH Command Syntax

FETCH [*location*] *cursor* [INTO *host-var* {, *host-var* }]

Possible locations

- NEXT
- PRIOR
- FIRST
- LAST
- ABSOLUTE n
- RELATIVE n

Dynamic Embedded SQL

Must be used when tables, columns or predicates are not known when the application is written.

Example applications

- Interactive human interfaces (ISQL)
- Other software tools

Two more commands

- PREPARE (statement preparation)
- EXECUTE (statement evaluation)

Dynamic Embedded SQL (cont'd)

Simplest case: executing a complete non-SELECT statement

```
s = "INSERT INTO students VALUES (1234, 'Weddell', ... )";

EXEC SQL EXECUTE IMMEDIATE :s;
```

To factor cost of "preparing to execute"

```
EXEC SQL PREPARE S1 FROM :s;
EXEC SQL EXECUTE S1;
EXEC SQL EXECUTE S1;
```

Dynamic Embedded SQL (cont'd)

To reference host variables

```
s = "INSERT INTO students VALUES (:?, :?, ... )";

EXEC SQL PREPARE S1 FROM :s;
studnum = 1234
surname = "Weddell''
...
EXEC SQL EXECUTE S1 USING :studnum, :surname, ... ;
```

Dynamic Embedded SQL (cont'd)

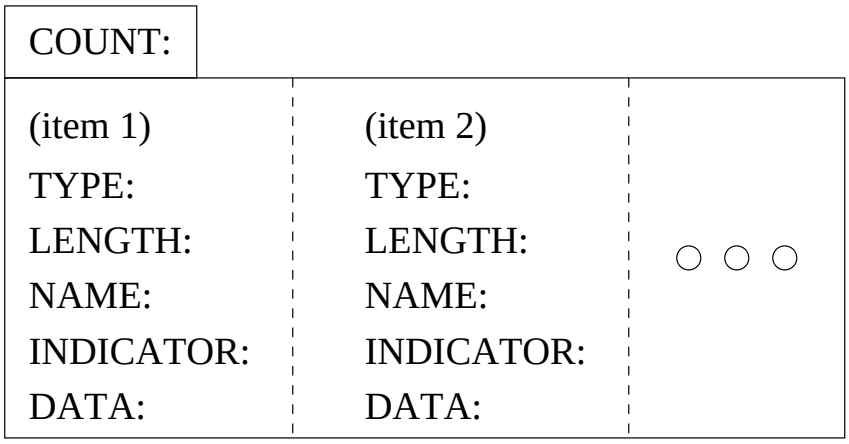
If the number and type of host variables is not known, then a dynamic descriptor area is needed.

More commands

- ALLOCATE DESCRIPTOR
- DESCRIBE
- GET DESCRIPTOR
- SET DESCRIPTOR

Commercial practice often involves the use of a SQLDA type instead of ALLOCATE, GET and SET commands.

Format of a Dynamic Descriptor Area



Dynamic Embedded SQL (cont'd)

```
s = "INSERT INTO students VALUES (:?, :?, ... )";

EXEC SQL PREPARE S1 FROM :s;
EXEC SQL ALLOCATE DESCRIPTOR I1;
EXEC SQL DESCRIBE INPUT S1 INTO SQL DESCRIPTOR I1;
EXEC SQL GET DESCRIPTOR I1 :n = COUNT;

for( i = 1; i < n; n++ ) {
   EXEC SQL GET DESCRIPTOR I1 VALUE :i
      :t = TYPE, ... ;
   EXEC SQL SET DESCRIPTOR I1 VALUE :i
      DATA = :d, INDICATOR = :ind;
}

EXEC SQL EXECUTE S1 USING SQL DESCRIPTOR I1;
```

Dynamic Embedded SQL (cont'd)

A prepared SELECT statement may be used to open a cursor

```
EXEC SQL DECLARE C1 CURSOR FOR S1;
s = "SELECT studnum, surname FROM student";
EXEC SQL PREPARE S1 FROM :s;
EXEC SQL ALLOCATE DESCRIPTOR O1;
EXEC SQL DESCRIBE OUTPUT S1 INTO SQL DESCRIPTOR O1;
EXEC SQL GET DESCRIPTOR O1 :n = COUNT;

EXEC SQL OPEN C1;
while( ... ) {
   EXEC SQL FETCH FROM C1 INTO SQL DESCRIPTOR O1;
   for( i = 1; i < n; i++ ) {
      EXEC SQL GET DESCRIPTOR O1 VALUE :i
         :t = TYPE, :d = DATA, :ind = INDICATOR;
      /* process column value */
   }
}
```

Dynamic Embedded SQL (cont'd)

If the number of dynamic statements is not known at the time the application is written, then extended dynamic statement names, cursor names, and descriptor area names may be used

```
query = scanf( ... );
queryname = "Q1";
descname = "O1";
cursorname = "C1";

EXEC SQL PREPARE :queryname FROM :query;
EXEC SQL ALLOCATE DESCRIPTOR :descname;
EXEC SQL ALLOCATE :cursorname CURSOR FOR :queryname;
EXEC SQL OPEN :cursorname;
```

The Open Database Connectivity (ODBC) Interface

Defined by Microsoft Corporation as a standard interface to database management systems.

Defined by a set of function calls, called ODBC Application Programming Interface (API).

Database operations specified using SQL statements passed as strings to ODBC functions.

Fundamental Objects in ODBC

There are three fundamental objects used in an ODBC program.

1. Environments
2. Connections
3. Statements

Each object is referenced by a *handle* of type HENV, HDBC and HSTMT, respectively.

Every ODBC application must allocate exactly one environment.

Object Functions in ODBC

For environments

- SQLAllocEnv (allocates object)
- SQLError (obtains diagnostics)
- SQLFreeEnv (frees object)

For connections

- SQLAllocConnect (allocates object)
- SQLConnect (connects to a database)
- SQLSetConnectOption (connection options)
- SQLTransact (used to COMMIT or ROLLBACK changes)
- SQLDisconnect (disconnect from a database)
- SQLError (obtains diagnostics)
- . . .
- SQLFreeConnect (frees object)

Object Functions in ODBC (cont'd)

For statements

- SQLAllocStmt (allocates object)
- SQLExecDirect (execute SQL or create cursor)
- SQLBindCol ("host variables" in ODBC)
- SQLSetParam (initialize a procedure parameter)
- SQLNumResultCols (number of result columns)
- SQLGetData (obtaining values of result columns)
- SQLFetch (cursor access in ODBC)
- SQLError (obtains diagnostics)
- . . .
- SQLFreeStmt (frees object)

The ODBC Interface (cont'd)

Ex. "Delete all marks for student 86004."

```
{
   HENV env;
   HDBC dbc;
   HSTMT stmt;

   SQLAllocEnv( &env );
   SQLAllocConnect( env, &dbc );
   SQLConnect( dbc, "Sample", SQL_NTS,
               "admin", SQL_NTS, "school", SQL_NTS );
   SQLSetConnectOption( dbc, SQL_AUTOCOMMIT, FALSE );
   SQLAllocStmt( dbc, &stmt );

   SQLExecDirect( stmt,
   "delete from mark where studnum = 86004", SQL_NTS );

   SQLTransact( env, dbc, SQL_COMMIT );

   SQLFreeStmt( stmt, SQL_DROP );
   SQLDisconnect( dbc );
   SQLFreeConnect( dbc );
   SQLFreeEnv( env );
}
```

The ODBC Interface (cont'd)

Ex. "Print number and surnames of all students."

```
{
   HDBC dbc;
   HSTMT stmt;
   RETCODE retcode;
   long int num;
   char name[20];

   SQLAllocStmt( dbc, &stmt );
   SQLExecDirect( stmt,
   "select studnum, surname from student", SQL_NTS );
   SQLBindCol( stmt, 1, SQL_C_LONG, &num, sizeof(num), NULL );
   SQLBindCol( stmt, 2, SQL_C_CHAR, &name, SIZEOF(name), NULL ),

   for( ;; ) {
      retcode = SQLFetch( stmt );
      if( retcode == SQL_NO_DATA_FOUND ) break;
      printf( "Student: %n, surname: %s\n", num, name );
   }

   SQLFreeStmt( stmt, SQL_CLOSE };
```

Note: using SQL_CLOSE in last statement results in closing cursor but not freeing statement.

Security in SQL

The GRANT and REVOKE statements are used to

- maintain users and user groups for a database
- maintain DDL privileges for users and user groups
- maintain DML privileges for users and user groups

Ex. “Add a benefits group to the database with access to the employee table.”

```
GRANT CONNECT TO benefits;
GRANT GROUP TO benefits;
GRANT ALL PRIVILEGES ON Employee TO benefits;
```

Ex. “Add Weddell to the benefits group, with password abc.”

```
GRANT CONNECT TO Weddell IDENTIFIED BY ”abc”;
GRANT MEMBERSHIP IN GROUP benefits TO Weddell;
```

Security in SQL (cont'd)

Ex. "Create a new user called George, password xyz, with the authority to execute SQL DDL statements."

```
GRANT CONNECT TO George IDENTIFIED BY xyz;
GRANT RESOURSE TO George;
```

Ex. "Make Mary the database administrator, with passwords: change quickly."

```
GRANT CONNECT TO Mary IDENTIFIED BY "change quickly";
GRANT DBA TO Mary;
```

Ex. "Have Mary change her password and revoke Weddell's membership in benefits, but still allow him to query the employee table."

```
CONNECT Mary IDENTIFIED BY "change quickly";
GRANT CONNECT TO Mary IDENTIFIED BY "xvqmt";
REVOKE MEMBERSHIP IN benefits FROM Weddell;
GRANT SELECT ON Employee TO Weddell;
```

Privileges on Tables and Views

INSERT (may insert new tuples in named table or view)

DELETE (may delete existing tuples from named table or view)

SELECT (may query existing tuples in named table or view)

UPDATE [*column-name-list*]
(may update indicated columns of existing tuples in named table or view)

REFERENCES
(may create a foreign key constraints to named table)

Changes in SQL2

A USAGE privilege is introduced in order to control use of a domain, collation, character set, or translation.

Ex. "Grant use of the money domain to Weddell."

GRANT USAGE ON DOMAIN money TO Weddell;

The INSERT privilege can now be granted on individual columns.

Ex. "Do not allow benefits to enter new tutors."

REVOKE ALL PRIVILEGES ON Employee FROM benefits;
GRANT INSERT(Empnum, Empname, Initials, Roomnum)
TO benefits;
GRANT DELETE, SELECT ON Employee TO benefits;