Chapter 5

Programs to Access a Database

复习指导

- 嵌入式SQL (ESQL) & 交互式SQL (ISQL)
  - 为什么要引入ESQL？
  - ESQL与ISQL在语言表示上的差别
    - 语句的定界符
    - 主变量 & SQL变量
    - 数据交换方式

- ESQL中扩充的语言成分
  - WHENEVER 语句
  - SELECT……INTO…… 语句
  - 游标 (cursor)：DECLARE, OPEN, FETCH, CLOSE
  - 变量赋值，流程控制语句

Ch5 Programs to Access a Database

5.1 Introduction to Embedded SQL in C

5.2 Condition Handling

5.3 Some Common Embedded SQL Statements

5.4 Programming for Transactions

5.5 The Power of Procedural SQL Programs

5.6 Dynamic SQL

5.7 Some Advanced Programming Concepts

5.1 Introduction to Embedded SQL in C

- **Embedded SQL (ESQL)**
  - **SQL statements embedded i** ***age***
    - **for example: COBOL, C,**

嵌入式
SQL

  - **The idea of SQL**
    - **for end-users to access a database**
    - **shortcoming of SQL**

      Need to know all tables & columns

      Need to know complex SQL syntax

      Too mistakes, especially with updates, deletes...

5.1 Introduction to Embedded SQL in C

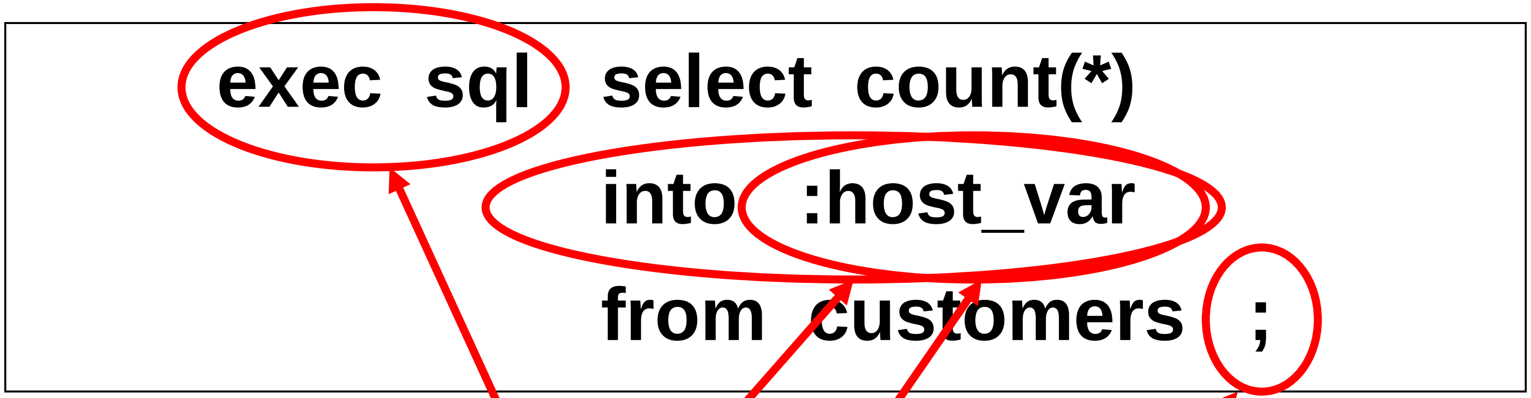
❑ **Solutions**

1. **Embedded SQL (ESQL)**
   - **for *Application Programmers* to develop menu applications**
   - **for *end-users* to access a database through menu applications**
2. **Interactive SQL (ISQL)**
   - **for *Casual Users* to access a database**

❑ ***Application Programmers* and *Casual Users* are proficient in SQL statements. They can spend a lot of time making sure the right SQL statement is used.**

5.1 Introduction to Embedded SQL in C

- An example of ESQL statement in C



```
        exec  sql  select  count(*)
                   into  :host_var
                   from  customers  ;
```

- different syntax than the ISQL
  - start with ‘exec sql’, ended by ‘;’
  - into clause
    - receive the result of select statement with single row
  - host variable (or program variable)

5.1 Introduction to Embedded SQL in C

- An example of ESQL statement in C

```
exec  sql   select  count(*)
            into  :host_var
            from  customers ;
```

- different syntax than the ISQL (cont.)
  - host variable (or program variable)
    - prefix(colon ’:’) of variable shows DBMS this is a program variable
  - host variable can be used to

    ①receive the value of column produced by DBMS and using in host language statements

    ②store the value produced by host language and using in SQL statement

5.1 Introduction to Embedded SQL in C

- **A Simple Program Using ESQL**
  - **The Declare Section**
    - **declare host variables using in ESQL statements**
  - **Condition Handling**
    - **control execution in the face of errors and other conditions**
  - **SQL Connect to Database**
  - **Main Body of Application Program**
    - **user interactions through host language**
    - **access a database through ESQL statements**

      select, insert, update, ...

      selecting multiple rows with a cursor
  - **SQL Disconnect**

The Declare Section

- An example of ESQL statement

```
exec  sql  select  cname, discnt
                into  :cust_name, :cust_discnt
                from  customers
                where  cid = :cust_id ;
```

- in order to use these host variable in an ESQL statement, they must first be declared. Why ?

The Declare Section

- **The usage of Declare Section**
  - ① check data types of column & host variable in compiling
  - ② pre-allocate memory space
    - the host variables can be filled in with values from DBMS
      - Number types
        - » float cust_discnt;
        - » int cust_discnt;
      - Character Strings
        - » in C: with null terminal character
        - » in DB: no terminator, fixed length

return different values

The Declare Section

```
exec sql begin declare section;
    char cust_id[5];
    char cust_name[14];
    float cust_discnt;
    char user_name[20], user_pwd[20];
exec sql end declare section;
```

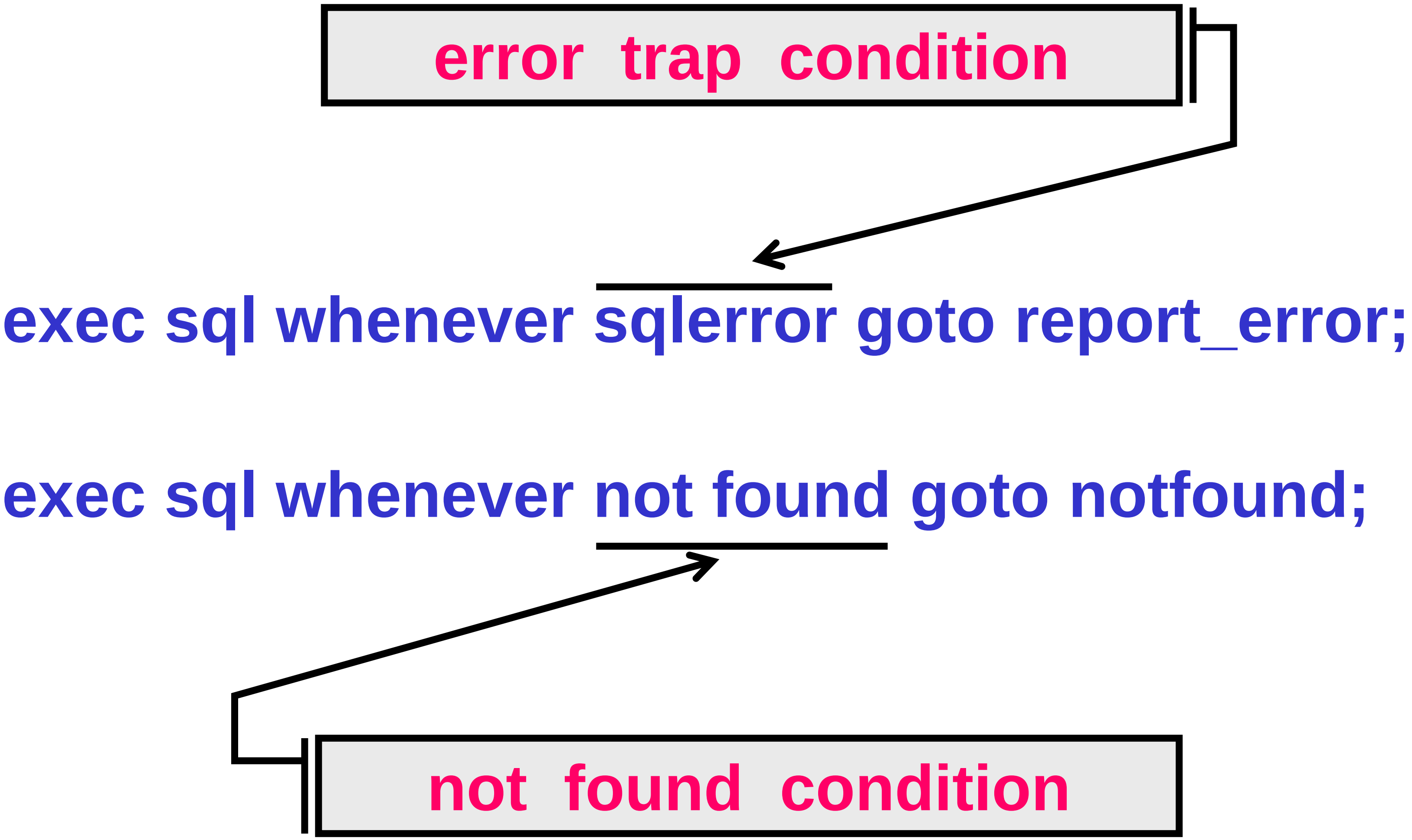
Begin declare SQL host variables

host variables for cno, four characters and a null terminator

End of declare section

host variables for user name and password

Condition Handling



error trap condition

```
exec sql whenever sqlerror goto report_error;

exec sql whenever not found goto notfound;
```

not found condition

SQL Connect Statement

- SQL99

```
EXEC  SQL  CONNECT  TO  target-server
    [AS  connect-name]  [USER  username] ;
```

```
EXEC  SQL  CONNECT  TO  DEFAULT ;
```

- target-server
  - the name of database supplied by DBA
- connect-name
  - the name of the connection session
  - may have more than one connection open at once
- username
  - identify yourself as database user

SQL Connect Statement

- **Oracle**

```
EXEC  SQL  CONNECT  TO  :user_name
         IDENTIFIED BY :user_pwd ;
```

- **user_name**
  - **the host variable of Oracle user name**
- **user_pwd**
  - **the host variable of Oracle user password**

- **no database name in Oracle connect statement**

User interactions & Access a database

```
while (prompt(cid_prompt, 1, cust_id, 4) >= 0)
{
   exec  sql  select  cname,  discnt
                 into  :cust_name, :cust_discnt
                 from  customers
                 where  cid = :cust_id;
    exec sql commit work;
    printf("CUSTOMER'S NAM
   DISCNT IS  %5.1f\n",  cu
    continue;
notfound:
   printf("Can't find custome                         ",
   cust_id);
}
```

根据输入的客户编号(cid)查询其姓名(cname)和折扣(discnt)

SQL Disconnect Statement

- SQL99

```
EXEC  SQL  DISCONNECT  connect-name ;
```

or

```
EXEC  SQL  DISCONNECT  CURRENT ;
```

- Before the Disconnect statement can ben used, it is necessary to use the Commit statement, for successful completion, or Rollback statement, to undo any partial work in an unsuccessful task.

```
EXEC  SQL  COMMIT  WORK ;
```

```
EXEC  SQL  ROLLBACK  WORK ;
```

SQL Disconnect Statement

- Oracle

```
exec  sql  commit  release ;
```

or

```
exec  sql  rollback  release ;
```

a commit statement followed by a disconnect statement, for successful completion

a rollback statement followed by a disconnect statement, to undo any partial work in an unsuccessful task

5.1 Introduction to Embedded SQL in C

- **A Simple Program Using ESQL**
  - **Figure 5.1** ▶

- **Programming with ESQL and C language**
  - **Precompiler**

    convert ESQL statements into C function calls into the database engine.
  - **C-Compiler**
  - **Executable Code**

Selecting Multiple Rows with a Cursor

❑ **Cursor(游标)：One-Row-at-a-Time Principle**

①**declare a cursor**

- **define a cursor with an ESQL select statement which may return multiple rows**

②**open the cursor**

- **execute the select statement and open the result set**

③**fetch a row by the cursor**

- **loop to fetch rows**
- **fetch one row at a time**

④**close the cursor**

- **release the result set**

Selecting Multiple Rows with a Cursor

- declare a cursor

define the cursor name

```
EXEC SQL DECLARE agent_dollars CURSOR FOR
    select  aid, sum(dollars)
    from    orders
    where   cid = :cust_id
    group by  aid ;
```

means multiple rows in result set

search by customer's id(stored in host variable cust_id) when open the cursor agent_dollars

Selecting Multiple Rows with a Cursor

- open the cursor

Before open the cursor, you must place cno value of customer’s id in the host variable cust_id using in the declare statement of cursor agent_dollars.

```
……
EXEC  SQL  OPEN  agent_dollars ;
……
```

execute the select statement

After open the cursor, the pointer of the cursor has been placed in the position before the first row in result set.

Selecting Multiple Rows with a Cursor

- fetch the result rows

```
while (TRUE) { /* loop to fetch rows */
    exec  sql  fetch agent_dollars
                into :agent_id, :dollar_sum;
    printf("%s %11.2f\n",agent_id,dollar_sum);
}   /* end fetch loop */
```

1) Move the pointer of cursor to the next row, then the next row is current row

2) Fetch the current row’s value into host variables: agent’s id to agent_id, summation of dollars to dollar_sum

Selecting Multiple Rows with a Cursor

- close the cursor

```
......
EXEC  SQL  CLOSE  agent_dollars ;
......
```

1) Close the cursor, and release the result set and other resource in DBMS

2) After close the cursor, it can be opened again.

- A simple program to retrieve multiple rows   Figure 5.2

Selecting Multiple Rows with a Cursor

- end fetch loop

```
exec sql whenever not  found  goto  finish;
……
while (TRUE) {
    exec  sql  fetch …… into ……;
    ……
}
……
finish:    exec  sql  close  agent_dollars;
```

declare ‘not found’ event processing

execute this statement after fetch loop when ‘not found’event is occur

5.2 Condition Handling

❑The Whenever Statement

```
EXEC SQL WHENEVER  condition  action;
```

- ➢set up a ‘condition trap’ for testing an error condition which arise from ESQL statement executing.

- ➢The precompiler will insert testing statements after each ESQL statement, such as if (condition) { action }

❑CONDITIONS

① SQLERROR

- arise from a programming error
- it can terminates execution of the program

② NOT FOUND

- No rows are affected following some SQL statement such as Select, Fetch, Insert, Update, or Delete.
- It often be used to end loop, or change the flow of control.

③ SQLWARNING

- a non-error but notable condition, don’t influence execution of the program
- It may need: EXEC SQL INCLUDE sqlca ;

❑ ACTIONS

① CONTINUE

② GOTO label

- Note: override with whenever statement for the same condition

③ STOP

- terminates execution of the program, rollback the current transaction, disconnects from database

④ DO function | BREAK | CONTINUE

- causes a named C function to be called
- On return from this function, flow of control continues from the statement after the ESQL statement that raised the condition.

5.2 Condition Handling

❑ Whenever Statement: Scope and Flow of Control

➢ Example 5.2.1

```
main() {
    exec sql whenever sqlerror stop; /* first whenever statement */
        ......
    goto s1;
        ......
    exec sql whenever sqlerror continue; /* override first whenever */
s1:
    exec sql update agents set percent = percent + 1;
        ......
}
```

5.2 Condition Handling

- We must be careful when using the Whenever statement to avoid infinite loops.
  - Example 5.2.2 ▷

```
exec sql whenever sqlerror goto handle_error ;
exec sql create table customers
              (cid char(4) not null, cname ...... ) ;
   ......
handle_error:
    exec sql whenever sqlerror continue ;
    exec sql drop table customers ;
    exec sql disconnect ;
    fprintf(stderr, “Couldn’t create customers table”);
    return  –1;
```

5.2 Condition Handling

❑ Explicit Error Checking

➢ Example 5.2.3

```
exec sql begin declare section;
     char SQLSTATE[6];
exec sql end declare section;
exec sql whenever sqlerror goto handle_error;
     ......
exec sql create table custs(cid char(4) ...... ) ;
 if (strcmp(SQLSTATE, “82100”) == 0)
      { handle this condition }
 else if (strcmp(SQLSTATE, “xxxxx”) == 0)
      { handle this condition }
 ......
```

5.2 Condition Handling

❑ Explicit Error Checking

➢ Example 5.2.3

```
exec sql begin declare section;
     char SQLSTATE[6];
exec sql end declare section;
exec sql whenever sqlerror continue;
     ......
exec sql create table custs(cid char(4) ...... ) ;
 if (strcmp(SQLSTATE, “82100”) == 0)
      { handle this condition }
 else if (strcmp(SQLSTATE, “xxxxx”) == 0)
      { handle this condition }
 ......
```

5.2 Condition Handling

❑ **Exp 5.2.4: Getting Error Messages from the Oracle DB**

```
#define ERRLEN 256        /* maximum length of error
                             message                    */
int errlength = ERRLEN;   /* size of buffer */
int errsize;              /* to contain actual message
                             length                     */
char errbuf[ERRLEN];      /* buffer to receive message */
    ......
sqlglm(errbuf, &errlength, &errsize);   /* get the error
                                           message for
                                           Oracle DB     */
printf("%.*s\n", errsize, errbuf);
```

5.2 Condition Handling

❑ Indicator Variables

- ➢ indicate the null value of column

```
exec sql begin declare section;
      float cust_discnt;
      short int cd_ind;
      ......
exec sql end declare section;
......
exec   sql       select  discnt
                 into     :cust_discnt   :cd_ind
                 from     customers
                 where  cid = :cust_id;
```

may have INDICATOR keyword

5.2 Condition Handling

- **The possible values for indicator variables**
  - **= 0**
    - **A database value, not null, was assigned to the host variable**
  - **> 0**
    - **A truncated database character string was assigned to the host variable**
  - **= -1**
    - **The database value is null, and the host variable value is not a meaningful value**

5.2 Condition Handling

❑ For example

- to set the discnt value to null in a specific row of customers

```
cd_ind = -1;
exec  sql  update  customers
    set  discnt = :cust_discnt INDICATOR :cd_ind
    where  cid = :cust_id;
```

5.3 Some Common ESQL Statement

❑ **Figure 5.3 Basic Embedded SQL Select Form (Single-Row Select)**

```
EXEC SQL
    SELECT [ ALL|DISTINCT ] expression, ......
    INTO  host-variable [ indicator-variable ], ......
    FROM  tableref [corr-name], ......
    [ WHERE  search-condition ]  ;
```

5.3 Some Common ESQL Statement

❑ Figure 5.4 Type Correspondences

Basic SQL type	ORACLE type	DB2 UDB type	C datatype
char(n)	char(n)	char(n)	char arr[n+1]
varchar(n)	varchar(n)	varchar(n)	char array[n+1]
smallint	smallint	smallint	short int
integer, int	integer, int, number(10)	integer, int	int
real	real	real	float
double precision, float	double precision, number, float	double precision, double, float	double

5.3 Some Common ESQL Statement

- Figure 5.5 Embedded SQL Declare Cursor Syntax

```
EXEC SQL
     DECLARE cursor-name CURSOR FOR
     subquery
     [ ORDER BY ...... ]
     [ FOR { READ ONLY |
                 UPDATE [ OF columnname, ...... ] } ]  ;
```

5.3 Some Common ESQL Statement

- Figure 5.6 Embedded Basic SQL Delete Syntax

```
EXEC SQL
    DELETE FROM tablename [ corr_name ]
    [ WHERE search_condition |
      WHERE CURRENT OF cursor_name ]
```

- Figure 5.7&5.8 Embedded Basic SQL Update Syntax

```
EXEC SQL
    UPDATE tablename [ corr_name ]
    SET columnname = expr, ......
    [ WHERE search_condition |
      WHERE CURRENT OF cursor_name ]
```

5.3 Some Common ESQL Statement

- Figure 5.9 Embedded Basic SQL Insert Syntax

```
EXEC SQL
    INSERT INTO tablename[ (column_nme, ... ) ]
         VALUES ( expr, ...... )  |  subquery   ;
```

- The Other ESQL Statement

```
    EXEC SQL CREATE TABLE ...... ;
    EXEC SQL DROP TABLE ...... ;
    EXEC SQL COMMIT WORK ;
    EXEC SQL ROLLBACK WORK ;
    EXEC SQL CONNECT ...... ;
    EXEC SQL DISCONNECT ...... ;
```

5.4 Programming for Transactions

- **The Concept of a Transaction**
  - **group several SQL statements together into a single indivisible, all-or-nothing transactional package.**
  - **Idea of concurrency**
    - **simultaneous access to data by multiple users**
    - **Example 5.4.1**

      Inconsistent view of data

5.4 Programming for Transactions

❑ Process P1

S1 (correct state)

```
update A
set balance = balance - $400.00
where A.aid = ‘A1’;
```

S2 (incorrect state)

```
update A
set balance =  balance + $400.00
where A.aid = ‘A2’
```

S3 (correct state)

A1.balance	A2.balance
$900.00	$100.00
$500.00	$100.00
$500.00	$500.00

5.4 Programming for Transactions

❑ How Transactions Are Specified in Programs

- Start Transaction
  - when first access is made to table after connect or prior commit or abort.
- End Transaction
  - exec sql commit work;

    Successful commit, rows updated, become concurrently visible.
  - exec sql rollback work;

    Unsuccessful abort, row value updates rolled back and become concurrently visible.

5.4 Programming for Transactions

❑ **A Transaction Example** **(Figure 5.13, pg. 211)**

```
#include <stdio.h>
#include “prompt.h”
int main()
{
    exec sql begin declare section;
        char acctfrom[11], acctto[11];
        double dollars;
    exec sql end declare section;
    char dollarstr[20];

    exec sql connect to default;
exec sql set transaction isolation level serializable;
```

```
    while (1) {
        ……
        exec sql whenever sqlerror goto do_rollback;
        exec sql update accounts set balance =
          balance - :dollars where acct = :acctfrom;
        exec sql update accounts set balance =
          balance + :dollars where acct = :acctto;
        exec sql commit work;
        printf(“Transfer complete.\n”);
        continue;
do_rollback:
        exec sql rollback work;
        printf(“Trans failed.\n”);
    }
    exec sql disconnect current;
    return 0;
}
```

5.6 Dynamic SQL

- ❑ allow us to construct a character string in a host variable to be used as an SQL statement.

- ❑ three type

  1) Execute Immediate

     EXECUTE IMMEDIATE :host_var;

  2) Prepare, Execute, and Using

     ① PREPARE handle FROM :stmt_string;

     - ➢ use the '?' marking the dynamic parameter

     ② EXECUTE handle USING :host_var;

  3) Dynamic Select

     - ▪ The Describe Statement and the SQLDA

Execute Immediate (Figure 5.23, pg. 221)

```
#include <stdio.h>
exec sql include sqlca;
exec sql begin declare section;
    char user_name[]=“scott”; char user_pwd[]=“tiger”;
    char sqltext[]=“delete from customers where cid=\’c006\’”;
exec sql end declare section;
int main()
{
    exec sql whenever sqlerror goto report_error;
    exec sql connect :user_name identified by :user_pwd;
    exec sql execute immediate :sqltext;
    exec sql commit release;
    return 0;
report_error:
    print_dberror();
    exec sql rollback release;
    return 1;
}
```

Prepare and Execute Statements (Figure 5.25, pg. 223)

```
#include <stdio.h>
exec sql include sqlca;
exec sql begin declare section;
    char cust_id[5], sqltext[256];
exec sql end declare section;
int main()
{
    strcpy(sqltext, “delete from customers where cid = ?”);
    exec sql whenever sqlerror goto report_error;
    exec sql connect to testdb;
    exec sql prepare delcust from :sqltext;
    while (1) {
        ......   /* input customer’s id to cust_id */
        exec sql execute delcust using :cust_id;
        exec sql commit work;
    }
    ......
}
```

5.6 Dynamic SQL

- **Dynamic Select**
  - **the number of column values to be retrieved may be unknown.**
  - **Figure 5.26, pg. 225**

```
exec sql include sqlca;
exec sql include sqlda;

sqlda = sqlald(...);
exec sql prepare stmt from :sqltext;
exec sql describe stmt into sqlda;

exec sql declare crs cursor for stmt;

exec sql fetch crs using descriptor sqlda;
```

5.7 Some Advanced Programming Concepts

❑ Scrollable Cursors

```
   EXEC  SQL  DECLARE  cursor_name
      [ INSENSITIVE ]  [ SCROLL ]
      CURSOR  [ WITH  HOLD ]  FOR
         subquery  { UNION  subquery }
         [ ORDER  BY ...... ]
      [ FOR  READ  ONLY  |
         FOR  UPDATE  OF  columnname ...... ];
```

```
 EXEC  SQL  FETCH
   [ { NEXT | PRIOR | FIRST | LAST |
   { ABSOLUTE | RELATIVE } value_spec } FROM ]
   cursor_name INTO ......;
```

```
exec  sql        select  count(*)
             into  :host_var
             from  customers ;
```

```
exec  sql  select  cname, discnt
             into  :cust_name, :cust_discnt
             from  customers
             where  cid = :cust_id ;
```