Why Embedded SQL

- C : Logic Control, User interface
- SQL: Database access
- Embed SQL into C => Pro*C Language
- Compilation Procedure
  - proc .pc => .c
  - cc .c => .o

Using Oracle on acmex

- Only acmex!
- Login as your prism user (gtxxxxx)
- Shell to use -- sh, ksh, bash
- At login prompt, run:
  - bash$ **. oraenv**[ENTER]
  - (Note the period followed by the space)
  - It will prompt you:
  - ORACLE_SID = [gte916j] ? **publ**[ENTER]

SQL*Plus

- Now you will be able to run SQL*Plus
  - bash$ **sqlplus /**[ENTER]
- SQL*Plus is a command-line tool that comes with Oracle to interact with the database
- Refer to the web page for some simple commands available with SQL*Plus.

Basic SQL*Plus commands

- SQL> quit
- SQL> SELECT user FROM dual;
- SQL> CREATE TABLE test (…);
- SQL> START script
- - where ‘script.sql’ is a file containing SQL statements.
- SQL> HELP <command>;

Use Proc*C on acmex

- Sample Makefile:
  - /db1/oracle/proc/demo/proc.mk
- Compilation:
  - copy proc.mk into your working directory
  - vi myfile.pc (and other source files if necessary)
  - make -f proc.mk EXE=myfile OBJS="myfile.o …"
- Sample Pro*C programs
  - /db1/oracle/proc/demo/

Host Variables

- Two types of variables in Pro*C script
  - Ordinary C variables : used by C only
  - Host variables: communication between C and SQL
- Defining Host Variables:

SQL Data Type	*HV data type*
gender CHAR	char gender;
cname CHAR(30)	char cname[31];
dob DATE	char dob[10];
num1 NUMBER(6)	char num1[6]; int num1;
num2 NUMBER(10)	char num1[10]; long num2;
num3 DECIMAL(6,2)	float num3;
synopsis VARCHAR(30)	varchar synopsis[31];

Using Host Variables

```
EXEC SQL BEGIN DECLARE SECTION;
        int     cno;
        char    phone[31];
EXEC SQL  END DECLARE SECTION;
…
scanf("%d", &cno);
EXEC SQL
        SELECT phone
        INTO  :phone
        FROM customers
        WHERE cno = :cno ;
printf("Selected phone number is %s\n", phone);
…
```

varchar

- In Pro*C
  - varchar address[61];
- After precompilation

```
struct {
    unsigned short len;
    unsigned char arr[61];
} address;
```

Using varchar

```
EXEC SQL BEGIN DECLARE SECTION;
        int         cno;
        char        phone[31];
        varchar address[61];
EXEC SQL  END DECLARE SECTION;
…
strncpy((char *) address.arr, ”323448 Gatech Station”, 60);
address.len = strlen((char *) address.arr);
strcpy(phone, “404-984-8733”);
cno = 1234;
EXEC SQL
        INSERT INTO customers
        VALUES(:cno, :phone, :address);
…
```

Indicator Variable

- Purpose: deal with null value
- Use: in pair with Host Variable
- Definition

```
EXEC SQL BEGIN DECLARE SECTION;
        int     cno;
        char    phone[31];
        short   phone_ind;
EXEC SQL  END DECLARE SECTION;
```

Test NULL Value

```
scanf("%d", &cno);
EXEC SQL
        SELECT  phone
        INTO :phone INDICATOR  :phone_ind
        FROM customers
        WHERE cno = :cno;
if  (phone_ind == -1)
        printf("The phone number is NULL");
else                /* phone_ind == 0 */
        printf("The phone number is not NULL");
```

Set NULL value

```
cno = 1234;
phone_ind = -1;

EXEC SQL
        UPDATE  customers
        SET phone = :phone INDICATOR :phone_ind
        WHERE cno = :cno;
```

Connecting to Oracle

- SQL Command
  - EXEC SQL CONNECT :*uid* IDENTIFIED BY :*passwd*;
- *uid* and *passwd* are **varchar** type HVs
- Take a look at *sample.pc*
- The *uid* and *passwd* are a problem on acmex
- Set *uid* and *passwd* to “” (the empty string)
- To disconnect from the database
  - EXEC SQL COMMIT RELEASE;

Transaction

- Transaction:
  - A sequence of database statements that must be executed atomically -- either all statements complete successfully or none of them
- Two Types of transactions in Oracle
  - Read Only: Only query statements
  - Read Write: Both query and update statements

Transaction Commands

- Set Transaction Type
  - EXEC SQL SET TRANSACTION READ ONLY;
  - Default is READ/WRITE.
- End Transaction
  - EXEC SQL COMMIT WORK RELEASE;
    - Commits changes and releases locks
  - EXEC SQL ROLLBACK WORK RELEASE;
    - Undoes changes and releases locks

SQLCA.sqlcode

- SQLCA
  - A data structure that contains the status of the execution of the last SQL statement.
  - Refreshed by execution of every SQL statement
  - EXEC SQL INCLUDE SQLCA;
- SQLCA.sqlcode
  - Most commonly used field of SQLCA

    =0 successfully executed

    >0 Statement executed but with an exception

    <0 Error occurred -- statement did not execute

Example 1

- Question : SQLCA.sqlcode = ?

```
EXEC SQL INCLUDE sqlca;
….

cno = 1234;
EXEC SQL
        SELECT phone
        INTO :phone
        FROM customers
        WHERE cno = :cno;
```

Example 1 (cont.)

- Answer
  - case 1: Everything is OK
  - case 2: There’s no table named “customers”
  - case 3: There’s no customer with cno=1234
  - case 4: Some other error occurred!

Example 2

- Purpose:
  - transfer $100 from checking to saving

```
acctno = 1234;
trans_flag = 0;
EXEC SQL SET TRANSACTION READ WRITE;
EXEC SQL
        Update checking
        SET balance = balance - 100
        WHERE acctno = :acctno;
if  (sqlca.sqlcode!=0) trans_flag = 1;
EXEC SQL
        Update saving
        SET balance = balance + 100
        WHERE acctno = :acctno;
if  (sqlca.sqlcode!=0) trans_flag = 1;
if  (trans_flag ==1 ) EXEC SQL  ROLLACK;
else EXEC SQL COMMIT;
```

Cursors

- Limitation of “SELECT ... INTO…”
  - Only one row can be returned
- When multiple rows are returned
  - use cursor as pointer to span rows
- Two types of cursor
  - For Read Only
  - For Updates

```
EXEC SQL DECLARE customer_cur CURSOR FOR
        SELECT * from customers
        WHERE cno >= :cno
FOR READ ONLY;

cno = 1234;
EXEC SQL SET TRANSACTION READ ONLY;
EXEC SQL OPEN customer_cur;
EXEC SQL FETCH customer_cur INTO :cno, :phone;
while (sqlca.sqlcode==0) {
        printf("%d\t%s\n", cno, phone);
        EXEC SQL FETCH customer_cur INTO :cno, :phone;
}
EXEC SQL CLOSE customer_cur;
EXEC SQL COMMIT;
```

```
EXEC SQL DECLARE customer_cur CURSOR FOR
        SELECT  phone FROM customers
        WHERE  cno >= :cno
FOR UPDATE;
cno = 1234;
EXEC SQL SET TRANSACTION READ WRITE;
EXEC SQL OPEN customer_cur;
EXEC SQL  FETCH customer_cur   INTO :phone INDICATOR :phone_ind;
while (sqlca.sqlcode==0) {
        if (phone_ind == -1)
                EXEC SQL UPDATE  customers
                        SET phone = “N/A”
                        WHERE CURRENT OF customer_cur;
        EXEC SQL FETCH customer_cur
                INTO :phone INDICATOR :phone_ind;
}
EXEC SQL CLOSE customer_cur;
EXEC SQL COMMIT;
```

More Topics

- Error handling
- Dynamic SQL