You run some samples interactively from SQL\*Plus, others from Pro\*C programs. You can experiment with the samples from any Oracle account. However, the Pro\*C examples expect you to use the scott/tiger account.

Before trying the samples, you must create some database tables, then load the tables with data. You do that by running two SQL\*Plus scripts, exampbld and examplod, which are supplied with PL/SQL. You can find these scripts in the PL/SQL demo directory.

The first script builds the database tables processed by the sample programs. The second script loads (or reloads) the database tables. To run the scripts, invoke SQL\*Plus, then issue the following commands:

SQL> START exampbld

...

SQL> START examplod

Sample 1. FOR Loop

The following example uses a simple FOR loop to insert ten rows into a database table. The values of a loop index, counter variable, and either of two character strings are inserted. Which string is inserted depends on the value of the loop index.

Input Table

Not applicable.

PL/SQL Block

-- available online in file 'sample1'

DECLARE

x NUMBER := 100;

BEGIN

FOR i IN 1..10 LOOP

IF MOD(i,2) = 0 THEN -- i is even

INSERT INTO temp VALUES (i, x, 'i is even');

ELSE

INSERT INTO temp VALUES (i, x, 'i is odd');

END IF;

x := x + 100;

END LOOP;

COMMIT;

END;

Output Table

SQL> SELECT \* FROM temp ORDER BY col1;

NUM\_COL1 NUM\_COL2 CHAR\_COL

-------- -------- ---------

1 100 i is odd

2 200 i is even

3 300 i is odd

4 400 i is even

5 500 i is odd

6 600 i is even

7 700 i is odd

8 800 i is even

9 900 i is odd

10 1000 i is even

Sample 2. Cursors

The following example uses a cursor to select the five highest paid employees from the emp table.

Input Table

SQL> SELECT ename, empno, sal FROM emp ORDER BY sal DESC;

ENAME EMPNO SAL

---------- --------- --------

KING 7839 5000

SCOTT 7788 3000

FORD 7902 3000

JONES 7566 2975

BLAKE 7698 2850

CLARK 7782 2450

ALLEN 7499 1600

TURNER 7844 1500

MILLER 7934 1300

WARD 7521 1250

MARTIN 7654 1250

ADAMS 7876 1100

JAMES 7900 950

SMITH 7369 800

PL/SQL Block

-- available online in file 'sample2'

DECLARE

CURSOR c1 is

SELECT ename, empno, sal FROM emp

ORDER BY sal DESC; -- start with highest paid employee

my\_ename VARCHAR2(10);

my\_empno NUMBER(4);

my\_sal NUMBER(7,2);

BEGIN

OPEN c1;

FOR i IN 1..5 LOOP

FETCH c1 INTO my\_ename, my\_empno, my\_sal;

EXIT WHEN c1%NOTFOUND; /\* in case the number requested \*/

/\* is more than the total \*/

/\* number of employees \*/

INSERT INTO temp VALUES (my\_sal, my\_empno, my\_ename);

COMMIT;

END LOOP;

CLOSE c1;

END;

Output Table

SQL> SELECT \* FROM temp ORDER BY col1 DESC;

NUM\_COL1 NUM\_COL2 CHAR\_COL

-------- -------- --------

5000 7839 KING

3000 7902 FORD

3000 7788 SCOTT

2975 7566 JONES

2850 7698 BLAKE

Sample 3. Scoping

The following example illustrates block structure and scope rules. An outer block declares two variables named x and counter and loops four times. Inside this loop is a sub-block that also declares a variable named x. The values inserted into the temp table show that the two x's are indeed different.

Input Table

Not applicable.

PL/SQL Block

-- available online in file 'sample3'

DECLARE

x NUMBER := 0;

counter NUMBER := 0;

BEGIN

FOR i IN 1..4 LOOP

x := x + 1000;

counter := counter + 1;

INSERT INTO temp VALUES (x, counter, 'in OUTER loop');

/\* start an inner block \*/

DECLARE

x NUMBER := 0; -- this is a local version of x

BEGIN

FOR i IN 1..4 LOOP

x := x + 1; -- this increments the local x

counter := counter + 1;

INSERT INTO temp VALUES (x, counter, 'inner loop');

END LOOP;

END;

END LOOP;

COMMIT;

END;

Output Table

SQL> SELECT \* FROM temp ORDER BY col2;

NUM\_COL1 NUM\_COL2 CHAR\_COL

-------- -------- -------------

1000 1 in OUTER loop

1 2 inner loop

2 3 inner loop

3 4 inner loop

4 5 inner loop

2000 6 in OUTER loop

1 7 inner loop

2 8 inner loop

3 9 inner loop

4 10 inner loop

3000 11 in OUTER loop

1 12 inner loop

2 13 inner loop

3 14 inner loop

4 15 inner loop

4000 16 in OUTER loop

1 17 inner loop

2 18 inner loop

3 19 inner loop

4 20 inner loop

Sample 4. Batch Transaction Processing

In the next example the accounts table is modified according to instructions stored in the action table. Each row in the action table contains an account number, an action to be taken (I, U, or D for insert, update, or delete), an amount by which to update the account, and a time tag used to sequence the transactions.

On an insert, if the account already exists, an update is done instead. On an update, if the account does not exist, it is created by an insert. On a delete, if the row does not exist, no action is taken.

Input Tables

SQL> SELECT \* FROM accounts ORDER BY account\_id;

ACCOUNT\_ID BAL

---------- -------

1 1000

2 2000

3 1500

4 6500

5 500

SQL> SELECT \* FROM action ORDER BY time\_tag;

ACCOUNT\_ID O NEW\_VALUE STATUS TIME\_TAG

---------- - ---------- -------------------- ---------

3 u 599 18-NOV-88

6 i 20099 18-NOV-88

5 d 18-NOV-88

7 u 1599 18-NOV-88

1 i 399 18-NOV-88

9 d 18-NOV-88

10 x 18-NOV-88

PL/SQL Block

-- available online in file 'sample4'

DECLARE

CURSOR c1 IS

SELECT account\_id, oper\_type, new\_value FROM action

ORDER BY time\_tag

FOR UPDATE OF status;

BEGIN

FOR acct IN c1 LOOP -- process each row one at a time

acct.oper\_type := upper(acct.oper\_type);

/\*----------------------------------------\*/

/\* Process an UPDATE. If the account to \*/

/\* be updated doesn't exist, create a new \*/

/\* account. \*/

/\*----------------------------------------\*/

IF acct.oper\_type = 'U' THEN

UPDATE accounts SET bal = acct.new\_value

WHERE account\_id = acct.account\_id;

IF SQL%NOTFOUND THEN -- account didn't exist. Create it.

INSERT INTO accounts

VALUES (acct.account\_id, acct.new\_value);

UPDATE action SET status =

'Update: ID not found. Value inserted.'

WHERE CURRENT OF c1;

ELSE

UPDATE action SET status = 'Update: Success.'

WHERE CURRENT OF c1;

END IF;

/\*--------------------------------------------\*/

/\* Process an INSERT. If the account already \*/

/\* exists, do an update of the account \*/

/\* instead. \*/

/\*--------------------------------------------\*/

ELSIF acct.oper\_type = 'I' THEN

BEGIN

INSERT INTO accounts

VALUES (acct.account\_id, acct.new\_value);

UPDATE action set status = 'Insert: Success.'

WHERE CURRENT OF c1;

EXCEPTION

WHEN DUP\_VAL\_ON\_INDEX THEN -- account already exists

UPDATE accounts SET bal = acct.new\_value

WHERE account\_id = acct.account\_id;

UPDATE action SET status =

'Insert: Acct exists. Updated instead.'

WHERE CURRENT OF c1;

END;

/\*--------------------------------------------\*/

/\* Process a DELETE. If the account doesn't \*/

/\* exist, set the status field to say that \*/

/\* the account wasn't found. \*/

/\*--------------------------------------------\*/

ELSIF acct.oper\_type = 'D' THEN

DELETE FROM accounts

WHERE account\_id = acct.account\_id;

IF SQL%NOTFOUND THEN -- account didn't exist.

UPDATE action SET status = 'Delete: ID not found.'

WHERE CURRENT OF c1;

ELSE

UPDATE action SET status = 'Delete: Success.'

WHERE CURRENT OF c1;

END IF;

/\*--------------------------------------------\*/

/\* The requested operation is invalid. \*/

/\*--------------------------------------------\*/

ELSE -- oper\_type is invalid

UPDATE action SET status =

'Invalid operation. No action taken.'

WHERE CURRENT OF c1;

END IF;

END LOOP;

COMMIT;

END;

Output Tables

SQL> SELECT \* FROM accounts ORDER BY account\_id;

ACCOUNT\_ID BAL

---------- --------

1 399

2 2000

3 599

4 6500

6 20099

7 1599

SQL> SELECT \* FROM action ORDER BY time\_tag;

ACCOUNT\_ID O NEW\_VALUE STATUS TIME\_TAG

---------- - ---------- --------------------- ---------

3 u 599 Update: Success. 18-NOV-88

6 i 20099 Insert: Success. 18-NOV-88

5 d Delete: Success. 18-NOV-88

7 u 1599 Update: ID not found. 18-NOV-88

Value inserted.

1 i 399 Insert: Acct exists. 18-NOV-88

Updated instead.

9 d Delete: ID not found. 18-NOV-88

10 x Invalid operation. 18-NOV-88

No action taken.

Sample 5. Embedded PL/SQL

The following example shows how you can embed PL/SQL in a high-level host language such as C and demonstrates how a banking debit transaction might be done.

Input Table

SQL> SELECT \* FROM accounts ORDER BY account\_id;

ACCOUNT\_ID BAL

---------- --------

1 1000

2 2000

3 1500

4 6500

5 500

PL/SQL Block in a C Program

/\* available online in file 'sample5' \*/

#include <stdio.h>

char buf[20];

EXEC SQL BEGIN DECLARE SECTION;

int acct;

double debit;

double new\_bal;

VARCHAR status[65];

VARCHAR uid[20];

VARCHAR pwd[20];

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE SQLCA;

main()

{

extern double atof();

strcpy (uid.arr,"scott");

uid.len=strlen(uid.arr);

strcpy (pwd.arr,"tiger");

pwd.len=strlen(pwd.arr);

printf("\n\n\tEmbedded PL/SQL Debit Transaction Demo\n\n");

printf("Trying to connect...");

EXEC SQL WHENEVER SQLERROR GOTO errprint;

EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;

printf(" connected.\n");

for (;;) /\* Loop infinitely \*/

{

printf("\n\*\* Debit which account number? (-1 to end) ");

gets(buf);

acct = atoi(buf);

if (acct == -1) /\* Need to disconnect from Oracle \*/

{ /\* and exit loop if account is -1 \*/

EXEC SQL COMMIT RELEASE;

exit(0);

}

printf(" What is the debit amount? ");

gets(buf);

debit = atof(buf);

/\* ---------------------------------- \*/

/\* ----- Begin the PL/SQL block ----- \*/

/\* ---------------------------------- \*/

EXEC SQL EXECUTE

DECLARE

insufficient\_funds EXCEPTION;

old\_bal NUMBER;

min\_bal CONSTANT NUMBER := 500;

BEGIN

SELECT bal INTO old\_bal FROM accounts

WHERE account\_id = :acct;

-- If the account doesn't exist, the NO\_DATA\_FOUND

-- exception will be automatically raised.

:new\_bal := old\_bal - :debit;

IF :new\_bal >= min\_bal THEN

UPDATE accounts SET bal = :new\_bal

WHERE account\_id = :acct;

INSERT INTO journal

VALUES (:acct, 'Debit', :debit, SYSDATE);

:status := 'Transaction completed.';

ELSE

RAISE insufficient\_funds;

END IF;

COMMIT;

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

:status := 'Account not found.';

:new\_bal := -1;

WHEN insufficient\_funds THEN

:status := 'Insufficient funds.';

:new\_bal := old\_bal;

WHEN OTHERS THEN

ROLLBACK;

:status := 'Error: ' || SQLERRM(SQLCODE);

:new\_bal := -1;

END;

END-EXEC;

/\* -------------------------------- \*/

/\* ----- End the PL/SQL block ----- \*/

/\* -------------------------------- \*/

status.arr[status.len] = '\0'; /\* null-terminate \*/

/\* the string \*/

printf("\n\n Status: %s\n", status.arr);

if (new\_bal >= 0)

printf(" Balance is now: $%.2f\n", new\_bal);

} /\* End of loop \*/

errprint:

EXEC SQL WHENEVER SQLERROR CONTINUE;

printf("\n\n>>>>> Error during execution:\n");

printf("%s\n",sqlca.sqlerrm.sqlerrmc);

EXEC SQL ROLLBACK RELEASE;

exit(1);

}

Interactive Session

Embedded PL/SQL Debit Transaction Demo

Trying to connect... connected.

\*\* Debit which account number? (-1 to end) 1

What is the debit amount? 300

Status: Transaction completed.

Balance is now: $700.00

\*\* Debit which account number? (-1 to end) 1

What is the debit amount? 900

Status: Insufficient funds.

Balance is now: $700.00

\*\* Debit which account number? (-1 to end) 2

What is the debit amount? 500

Status: Transaction completed.

Balance is now: $1500.00

\*\* Debit which account number? (-1 to end) 2

What is the debit amount? 100

Status: Transaction completed.

Balance is now: $1400.00

\*\* Debit which account number? (-1 to end) 99

What is the debit amount? 100

Status: Account not found.

\*\* Debit which account number? (-1 to end) -1

Output Tables

SQL> SELECT \* FROM accounts ORDER BY account\_id;

ACCOUNT\_ID BAL

---------- ------

1 700

2 1400

3 1500

4 6500

5 500

SQL> SELECT \* FROM journal ORDER BY date\_tag;

ACCOUNT\_ID ACTION AMOUNT DATE\_TAG

---------- -------------------- --------- ---------

1 Debit 300 28-NOV-88

2 Debit 500 28-NOV-88

2 Debit 100 28-NOV-88

Sample 6. Calling a Stored Procedure

This Pro\*C program connects to Oracle, prompts the user for a department number, then calls procedure get\_employees, which is stored in package personnel. The procedure declares three index-by tables as OUT formal parameters, then fetches a batch of employee data into the index-by tables. The matching actual parameters are host arrays.

When the procedure finishes, it automatically assigns all row values in the index-by tables to corresponding elements in the host arrays. The program calls the procedure repeatedly, displaying each batch of employee data, until no more data is found.

Input Table

SQL> SELECT ename, empno, sal FROM emp ORDER BY sal DESC;

ENAME EMPNO SAL

---------- --------- --------

KING 7839 5000

SCOTT 7788 3000

FORD 7902 3000

JONES 7566 2975

BLAKE 7698 2850

CLARK 7782 2450

ALLEN 7499 1600

TURNER 7844 1500

MILLER 7934 1300

WARD 7521 1250

MARTIN 7654 1250

ADAMS 7876 1100

JAMES 7900 950

SMITH 7369 800

Stored Procedure

/\* available online in file 'sample6' \*/

#include <stdio.h>

#include <string.h>

typedef char asciz;

EXEC SQL BEGIN DECLARE SECTION;

/\* Define type for null-terminated strings. \*/

EXEC SQL TYPE asciz IS STRING(20);

asciz username[20];

asciz password[20];

int dept\_no; /\* which department to query \*/

char emp\_name[10][21];

char job[10][21];

EXEC SQL VAR emp\_name is STRING (21);

EXEC SQL VAR job is STRING (21);

float salary[10];

int done\_flag;

int array\_size;

int num\_ret; /\* number of rows returned \*/

int SQLCODE;

EXEC SQL END DECLARE SECTION;

EXEC SQL INCLUDE sqlca;

int print\_rows(); /\* produces program output \*/

int sqlerror(); /\* handles unrecoverable errors \*/

main()

{

int i;

/\* Connect to Oracle. \*/

strcpy(username, "SCOTT");

strcpy(password, "TIGER");

EXEC SQL WHENEVER SQLERROR DO sqlerror();

EXEC SQL CONNECT :username IDENTIFIED BY :password;

printf("\nConnected to Oracle as user: %s\n\n", username);

printf("Enter department number: ");

scanf("%d", &dept\_no);

fflush(stdin);

/\* Print column headers. \*/

printf("\n\n");

printf("%-10.10s%-10.10s%s\n", "Employee", "Job", "Salary");

printf("%-10.10s%-10.10s%s\n", "--------", "---", "------");

/\* Set the array size. \*/

array\_size = 10;

done\_flag = 0;

num\_ret = 0;

/\* Array fetch loop - ends when NOT FOUND becomes true. \*/

for (;;)

{

EXEC SQL EXECUTE

BEGIN personnel.get\_employees

(:dept\_no, :array\_size, :num\_ret, :done\_flag,

:emp\_name, :job, :salary);

END;

END-EXEC;

print\_rows(num\_ret);

if (done\_flag)

break;

}

/\* Disconnect from Oracle. \*/

EXEC SQL COMMIT WORK RELEASE;

exit(0);

}

print\_rows(n)

int n;

{

int i;

if (n == 0)

{

printf("No rows retrieved.\n");

return;

}

for (i = 0; i < n; i++)

printf("%10.10s%10.10s%6.2f\n",

emp\_name[i], job[i], salary[i]);

}

sqlerror()

{

EXEC SQL WHENEVER SQLERROR CONTINUE;

printf("\nOracle error detected:");

printf("\n% .70s \n", sqlca.sqlerrm.sqlerrmc);

EXEC SQL ROLLBACK WORK RELEASE;

exit(1);

}

Interactive Session

Connected to Oracle as user: SCOTT

Enter department number: 20

Employee Job Salary

-------- --- ------

SMITH CLERK 800.00

JONES MANAGER 2975.00

SCOTT ANALYST 3000.00

ADAMS CLERK 1100.00

FORD ANALYST 3000.00