1.

#include <stdio.h>

exec sql include sqlca;

exec sql begin declare section;

char sqlstmt[1024];

char \*MYID= "fedora/oracle";

exec sql end declare section;

void reportError(char\* succ, char \*fail) {

printf("%s\n", sqlca.sqlcode == 0 ? succ : fail);

}

void createTable(char\* command) {

strcpy(sqlstmt, command);

exec sql execute immediate :sqlstmt;

reportError("Table created", "Table not created");

}

void dropTable(char\* tableName) {

strcpy(sqlstmt, "drop table ");

strcat(sqlstmt, tableName);

exec sql execute immediate :sqlstmt;

reportError("Table dropped", "Table not dropped");

}

void connectDB() {

exec sql connect :MYID;

reportError("Connected to ORACLE", "Connect Failed");

}

int main() {

connectDB();

exec sql set transaction read write;

createTable("create table bank (b# char(2) primary key, name varchar2(10) not null, city varchar2(10) not null)");

createTable("create table customer (c# char(2) primary key, name varchar2(10) not null, age int not null, city varchar2(10) not null)");

createTable("create table account (c# char(2), b# char(2), balance int not null, primary key(b#, c#), foreign key(b#) references bank(b#) on delete cascade, foreign key(c#) references customer(c#) on delete cascade)");

// dropTable("account"); dropTable("bank"); dropTable("customer");

exec sql commit release;

exit(0);

}

2.

#include <stdio.h>

exec sql include sqlca;

exec sql begin declare section;

char sqlstmt[1024];

char bno[5];

char bname[10];

char bcity[10];

char cno[5];

char cname[10];

int cage;

char ccity[10];

int balance;

char \*username= "fedora";

char \*password= "oracle";

exec sql end declare section;

void reportError(char\* succ, char \*fail) {

printf("%s\n", sqlca.sqlcode == 0 ? succ : fail);

}

void connectDB() {

exec sql connect :username identified by :password;

reportError("Oracle connected", "Oracle not connected");

}

void insertCustomer(char\* custNum, char\* custName, int custAge, char\* custCity) {

printf("%s - %s - %d - %s\n", custNum, custName, custAge, custCity);

strcpy(cno, custNum);

strcpy(cname, custName);

cage = custAge;

strcpy(ccity, custCity);

exec sql execute s using :cno, :cname, :cage, :ccity;

reportError("Insert customer done", "Insert customer failed");

}

void insertBank(char\* bankNum, char\* bankName, char\* bankCity) {

printf("%s - %s - %s\n", bankNum, bankName, bankCity);

strcpy(bno, bankNum);

strcpy(bname, bankName);

strcpy(bcity, bankCity);

exec sql execute s using :bno, :bname, :bcity;

reportError("Insert bank done", "Insert bank failed");

}

void insertAccount(char\* custNum, char\* bankNum, int accBalance) {

printf("%s - %s - %d\n", custNum, bankNum, accBalance);

strcpy(cno, custNum);

strcpy(bno, bankNum);

balance = accBalance;

exec sql execute s using :cno, :bno, :balance;

reportError("Insert account done", "Insert account failed");

}

void prepareStatement(char\* prepareStmt) {

strcpy(sqlstmt, prepareStmt);

exec sql prepare s from :sqlstmt;

reportError("Statement prepared", "Statement not prepared");

}

int main() {

connectDB();

int choice;

int run = 1;

while(run) {

char bankNum[100], bankName[100], bankCity[100];

char custNum[100], custName[100], custCity[100];

int accBalance, custAge;

printf("Choose a table to insert to (-1 to exit):\n 1. Bank\n 2. Customer\n 3. Account\n > ");

scanf("%d", &choice);

switch(choice) {

case -1:

run = 0;

break;

case 1:

prepareStatement("insert into bank values (:v1, :v2, :v3)");

printf("Bank Insert: BankNumber BankName BankCity\n > ");

scanf("%s %s %s", bankNum, bankName, bankCity);

insertBank(bankNum, bankName, bankCity);

break;

case 2:

prepareStatement("insert into customer values (:v1, :v2, :v3, :v4)");

printf("Customer Insert: CustomerNumber CustomerName CustomerAge CustomerCity\n > ");

scanf("%s %s %d %s", custNum, custName, &custAge, custCity);

insertCustomer(custNum, custName, custAge, custCity);

break;

case 3:

prepareStatement("insert into account values (:v1, :v2, :v3)");

printf("Account Insert: CustomerNumber BankNumber AccontBalance\n > ");

scanf("%s %s %d", custNum, bankNum, &accBalance);

insertAccount(custNum, bankNum, accBalance);

break;

default:

printf("Invalid Option.\n");

break;

}

}

exec sql commit release;

exit(0);

}

3.

#include <stdio.h>

exec sql include sqlca;

exec sql begin declare section;

char sqlstmt[1024];

char bno[5];

char bname[10];

char bcity[10];

char cno[5];

char cname[10];

int cage;

char ccity[10];

char \*username= "fedora";

char \*password= "oracle";

exec sql end declare section;

void reportError(char\* succ, char \*fail) {

printf("%s\n", sqlca.sqlcode == 0 ? succ : fail);

}

void connectDB() {

exec sql connect :username identified by :password;

reportError("Oracle connected", "Oracle not connected");

}

void prepareStatement(char\* prepareStmt) {

strcpy(sqlstmt, prepareStmt);

exec sql prepare t from :sqlstmt;

reportError("Statement prepared", "Statement not prepared");

}

void prepareStatementBank(char\* prepareStmt) {

strcpy(sqlstmt, prepareStmt);

exec sql prepare r from :sqlstmt;

//reportError("Statement prepared", "Statement not prepared");

}

int main() {

connectDB();

prepareStatement("select \* from customer order by c#");

exec sql declare a\_cursor cursor for t;

exec sql open a\_cursor;

reportError("Cursor opened - customer", "Cursor error - customer");

printf("C# Name Age City \n");

printf("------------------------------\n");

exec sql fetch a\_cursor into :cno, :cname, :cage, :ccity;

while(sqlca.sqlcode == 0) {

printf("%s %s %-5d %s\n", cno, cname, cage, ccity);

// START BANK

prepareStatementBank("select B.b#, B.name, B.city from bank B left join account A on B.b# = A.b# where A.c# = :v1 order by b#");

exec sql declare b\_cursor cursor for r;

exec sql open b\_cursor using :cno;

//reportError("Cursor opened - bank", "Cursor error - bank");

printf(" B# Name City \n");

printf(" ------------------------\n");

exec sql fetch b\_cursor into :bno, :bname, :bcity;

while(sqlca.sqlcode == 0) {

printf(" %s %s %s\n", bno, bname, bcity);

exec sql fetch b\_cursor into :bno, :bname, :bcity;

}

printf(" ------------------------\n");

exec sql close b\_cursor;

// END BANK

exec sql fetch a\_cursor into :cno, :cname, :cage, :ccity;

}

printf("------------------------------\n");

exec sql close a\_cursor;

exec sql commit release;

exit(0);

}

4.

set serveroutput on;

declare

cno customer.c#%type;

cname customer.name%type;

cage customer.age%type;

ccity customer.city%type;

bno bank.b#%type;

bname bank.name%type;

bcity bank.city%type;

cursor cc is select \* from customer;

cursor bc (num customer.c#%type) is

select B.b#, B.name, B.city from bank B left join account A on B.b# = A.b# where A.c# = num order by b#;

begin

open cc;

loop

fetch cc into cno, cname, cage, ccity;

exit when cc%NOTFOUND;

dbms\_output.put\_line(cno || ' ' || cname || ' ' || cage || ' ' || ccity);

open bc (cno);

loop

fetch bc into bno, bname, bcity;

exit when bc%NOTFOUND;

dbms\_output.put\_line('- ' || bno || ' ' || bname || ' ' || bcity);

end loop;

close bc;

end loop;

close cc;

end;

/

5.

create type Hobby\_v as varray(5) of varchar(10);

/

create type Student\_t as object (

Name varchar(10),

Hobbies Hobby\_v

);

/

create type Student\_v as varray(5) of Student\_t;

/

create type Supervise\_t as object (

Kind varchar(10),

Students Student\_v

);

/

create type Supervise\_v as varray(5) of Supervise\_t;

/

create table professor (

Name varchar(10),

Hobbies Hobby\_v,

Supervision Supervise\_v

);

insert into professor values (

'Henry',

Hobby\_v('Chess', 'Skiing'),

Supervise\_v(

Supervise\_t('Ph.D', Student\_v(

Student\_t('Young', Hobby\_v('Skiing', 'Soccer'))

)),

Supervise\_t('M.Sc', Student\_v(

Student\_t('James', Hobby\_v('Boxing')),

Student\_t('Adams', Hobby\_v('Chess', 'Skiing'))

))

)

);

insert into professor values (

'David',

Hobby\_v('Hiking', 'Travel'),

Supervise\_v(

Supervise\_t('M.Sc', Student\_v(

Student\_t('Scott', Hobby\_v('Hiking', 'Travel'))

))

)

);

6.

-- 1. List all students (both master and PhD) as a single set of names.

select S.name

from professor P,

table(P.Supervision) V,

table(V.Students) S;

Result: Young, James, Adams, Scott

-- 2. List professor and their students in a nested relation.

-- does this need cast multiset???

select P.name, S.name

from professor P,

table(P.Supervision) V,

table(V.Students) S;

Result: <Henry, Young>, <Henry, James>, <Henry, Adams>, <David, Scott>

-- 3. List professor and students in a nested relation such that they have the same hobby

select P.name, S.name

from professor P,

table(P.Supervision) V,

table(V.Students) S

where P.Hobbies && S.Hobbies;

-- && is supposed to be true if there is an overlap(have elements in common)

-- but it is deprecated in the current version of oracle..

Result: <Henry, Young>, <Henry, Adams>, <David, Scott>

-- 4. List the hobbies that Henry’s students have but Henry does not have.

select H.column\_value as DiffHobby

from professor P,

table(P.Supervision) V,

table(V.Students) S,

table(S.Hobbies) H

where not exists(

select \*

from table(P.Hobbies) K

where H.column\_value = K.column\_value

) and P.name = 'Henry';

Result: Soccer, Boxing