**STORED PROCEDURES**

**STORED PROCEDURE TO INSERT INTO 'AVAILS' TABLE. AFTER PERFORMING CERTAIN CHECKS IT WILL INSERT APPROPRIATE DATA INTO THE TABLE.**

-- Function: project.avails\_insert(integer, integer, date, date, integer)

-- DROP FUNCTION project.avails\_insert(integer, integer, date, date, integer);

CREATE OR REPLACE FUNCTION project.avails\_insert(plid integer, pgid integer, ptodate date, pfromdate date, pnop integer)

RETURNS boolean AS

$BODY$

DECLARE

listID avails.listing\_id%TYPE;

to\_d listing.to\_date%TYPE;

from\_d listing.from\_date%TYPE;

a avails%ROWTYPE;

max\_guests home.maximum\_guests%TYPE;

group\_s experience.group\_size%TYPE;

max\_s home.max\_stay%TYPE;

min\_s home.min\_stay%TYPE;

BEGIN

--RAISE NOTICE 'Date omg';

listID = plID;

select to\_date into to\_d from listing where listing\_id = listID;

select from\_date into from\_d from listing where listing\_id = listID;

select maximum\_guests into max\_guests from home where listing\_id = listID;

select max\_stay into max\_s from home where listing\_id = listID;

select min\_stay into min\_s from home where listing\_id = listID;

select group\_size into group\_s from experience where listing\_id = listID;

if pfromdate < from\_d OR ptodate > to\_d then

RAISE NOTICE 'Date not in range';

RETURN FALSE;

else

--RAISE NOTICE 'Dates yeepiii';

for a in select \* from avails where listing\_id = listID

loop

if (a.from\_date = pfromdate OR a.from\_date = ptodate OR a.to\_date = pfromdate OR a.to\_date = ptodate) THEN

RAISE NOTICE 'Dates not available';

RETURN FALSE;

end if;

if (pfromdate < a.from\_date AND pfromdate< a.to\_date AND ptodate < a.from\_date AND ptodate < a.to\_date) OR (pfromdate > a.from\_date AND pfromdate > a.to\_date AND ptodate > a.from\_date AND ptodate > a.to\_date) THEN

if ((max\_guests is NULL AND pnop <= group\_s) OR (pnop <= max\_guests AND (ptodate-pfromdate >= min\_s AND ptodate-pfromdate <= max\_s))) THEN

INSERT INTO avails VALUES(plID,pgID,ptodate,pfromdate,pnop);

RAISE NOTICE 'Dates available';

RETURN TRUE;

end if;

else

RAISE NOTICE 'Dates not available';

RETURN FALSE;

end if;

end loop;

if ((max\_guests is NULL AND pnop <= group\_s) OR (pnop <= max\_guests AND (ptodate-pfromdate >= min\_s AND ptodate-pfromdate <= max\_s))) THEN

INSERT INTO avails VALUES(plID,pgID,ptodate,pfromdate,pnop);

RAISE NOTICE 'Dates available';

RETURN TRUE;

end if;

return FALSE;

end if;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION project.avails\_insert(integer, integer, date, date, integer)

OWNER TO "201501183"; OWNER TO "201501183";

**STORED PROCEDURE TO UPDATE AND INSERT INTO 'GUEST\_REVIEWS' TABLE. UPDATE IF THE TUPLE ALREADY EXISTS AND THE USER WISHES TO CHANGE THE VALUES ELSE IF THE TUPLE DOESN'T EXISTS THEN DIRECTLY INSERT INTO TABLE.**

-- Function: project.update\_greviews(integer, integer, integer, character varying)

-- DROP FUNCTION project.update\_greviews(integer, integer, integer, character varying);

CREATE OR REPLACE FUNCTION project.update\_greviews(plid integer, pgid integer, pr integer, pcom character varying)

RETURNS boolean AS

$BODY$

DECLARE

r guest\_reviews%rowtype;

BEGIN

FOR r IN SELECT \* FROM guest\_reviews

LOOP

IF (plID = r.listing\_id AND pgID = r.guest\_id) THEN

UPDATE guest\_reviews SET rating = pr, comments = pcom WHERE listing\_id = plid AND guest\_id = pgid;

RETURN TRUE;

END IF;

END LOOP;

INSERT INTO guest\_reviews VALUES(plID,pgID,pr,pcom);

RETURN TRUE;

END; $BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION project.update\_greviews(integer, integer, integer, character varying)

OWNER TO "201501183";

**STORED PROCEDURE TO UPDATE AND INSERT INTO 'HOST\_REVIEWS' TABLE. UPDATE IF THE TUPLE ALREADY EXISTS AND THE USER WISHES TO CHANGE THE VALUES ELSE IF THE TUPLE DOESN'T EXISTS THEN DIRECTLY INSERT INTO TABLE.**

-- Function: project.update\_hreviews(integer, integer, integer, character varying)

-- DROP FUNCTION project.update\_hreviews(integer, integer, integer, character varying);

CREATE OR REPLACE FUNCTION project.update\_hreviews(plid integer, pgid integer, pr integer, pcom character varying)

RETURNS boolean AS

$BODY$

DECLARE

r host\_reviews%rowtype;

BEGIN

FOR r IN SELECT \* FROM host\_reviews

LOOP

IF (plID = r.listing\_id AND pgID = r.guest\_id) THEN

UPDATE host\_reviews SET rating = pr, comments = pcom where listing\_id = plid and guest\_id = pgid;

RETURN TRUE;

END IF;

END LOOP;

INSERT INTO host\_reviews VALUES(plID,pgID,pr,pcom);

RETURN TRUE;

END; $BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION project.update\_hreviews(integer, integer, integer, character varying)

OWNER TO "201501183";

**STORED PROCEDURE TO DELETE AN ENTRY FROM AVAILS WHEN THE USER DECIDES TO CANCEL THE AVAILED LISTING AND ACCORDINGLY CHANGE THE HOST EARNINGS .**

CREATE OR REPLACE FUNCTION project.delete\_avails(list integer, guest integer, to\_d date, from\_d date)

RETURNS boolean AS

$BODY$

DECLARE

cancel varchar(50);

cost\_m int;

hostID int;

BEGIN

select cancellation\_policy into cancel from listing where listing\_id = list;

select cost into cost\_m from listing where listing\_id = list;

select host\_id into hostID from listing where listing\_id = list;

if (cancel = '10 days Refund 25' AND from\_date - now()::date >= 10) then

DELETE FROM avails where listing\_id = list AND guest\_id = guest AND to\_date = to\_d AND from\_date = from\_d;

UPDATE host set earnings = (earnings - (cost\*0.25))::int where host\_id = hostID;

RETURN true;

elsif (cancel = '30 days Refund 75' AND from\_date - now()::date >= 30) then

DELETE FROM avails where listing\_id = list AND guest\_id = guest AND to\_date = to\_d AND from\_date = from\_d;

UPDATE host set earnings = (earnings - (cost\*0.75))::int where host\_id = hostID;

RETURN true;

elsif (cancel = '10 days Refund 20' AND from\_date - now()::date >= 10) then

DELETE FROM avails where listing\_id = list AND guest\_id = guest AND to\_date = to\_d AND from\_date = from\_d;

UPDATE host set earnings = (earnings - (cost\*0.2))::int where host\_id = hostID;

RETURN true;

elsif (cancel = '30 days Refund 50' AND from\_date - now()::date >= 30) then

DELETE FROM avails where listing\_id = list AND guest\_id = guest AND to\_date = to\_d AND from\_date = from\_d;

UPDATE host set earnings = (earnings - (cost\*0.50))::int where host\_id = hostID;

RETURN true;

elsif (cancel = '10 days Refund 30' AND from\_date - now()::date >= 10) then

DELETE FROM avails where listing\_id = list AND guest\_id = guest AND to\_date = to\_d AND from\_date = from\_d;

UPDATE host set earnings = (earnings - (cost\*0.3))::int where host\_id = hostID;

RETURN true;

elsif (cancel = '30 days Refund 60' AND from\_date - now()::date >= 30) then

DELETE FROM avails where listing\_id = list AND guest\_id = guest AND to\_date = to\_d AND from\_date = from\_d;

UPDATE host set earnings = (earnings - (cost\*0.6))::int where host\_id = hostID;

RETURN true;

else

RAISE NOTICE 'No Refund';

return false;

end if;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION project.delete\_avails(integer, integer, date, date)

OWNER TO "201501183";

**STORED PROCEDURE TO CALCULATE THE EARNINGS OF THE HOST IN 'HOST' TABLE.**

-- Function: project.host\_earnings()

-- DROP FUNCTION project.host\_earnings();

CREATE OR REPLACE FUNCTION project.host\_earnings()

RETURNS boolean AS

$BODY$

DECLARE

r record;

BEGIN

UPDATE host SET earnings = 0;

FOR r IN SELECT host\_id,sum(cost) FROM listing join avails on (listing.listing\_id = avails.listing\_id) group by host\_id order by host\_id

LOOP

UPDATE host SET earnings = r.sum WHERE host\_id=r.host\_id;

END LOOP;

RETURN TRUE;

END; $BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION project.host\_earnings()

OWNER TO "201501183";

**STORED PROCEDURE TO SOLVE THE FOLLOWING QUERRY: (For each Month find the country which saw the most number of listings(SP).)**

-- Function: project.most\_listing(integer, integer)

-- DROP FUNCTION project.most\_listing(integer, integer);

CREATE OR REPLACE FUNCTION project.most\_listing(pmonth integer, pyear integer)

RETURNS character varying AS

$BODY$

DECLARE

z integer;

c varchar(40);

BEGIN

SELECT max(count) into z FROM (SELECT count(listing\_id),country FROM list\_loc where extract(month from from\_date)=pmonth AND EXTRACT(YEAR FROM from\_date)=pyear group by country) AS R1;

SELECT country into c FROM (SELECT count(listing\_id),country FROM list\_loc where extract(month from from\_date)=pmonth AND EXTRACT(YEAR FROM from\_date)=pyear group by country) AS R1 WHERE count = z;

RETURN c;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION project.most\_listing(integer, integer)

OWNER TO "201501183";

**STORED PROCEDURE TO SOLVE THE FOLLOWING QUERRY: (Host who has earned the most till now in a particular country(SP).)**

-- Function: project.most\_earnings(character varying)

-- DROP FUNCTION project.most\_earnings(character varying);

CREATE OR REPLACE FUNCTION project.most\_earnings(pcountry character varying)

RETURNS integer AS

$BODY$

DECLARE

z integer;

c integer;

BEGIN

SELECT max(earnings) into z FROM HOST NATURAL JOIN LISTING NATURAL JOIN LOCATION WHERE COUNTRY = pcountry;

SELECT host\_id into c FROM host WHERE earnings = z;

RETURN c;

END;$BODY$

LANGUAGE plpgsql VOLATILE

COST 100;

ALTER FUNCTION project.most\_earnings(character varying)

OWNER TO "201501183";

**STORED PROCEDURE TO SOLVE THE FOLLOWING QUERRY:(Effective cost for a user given the listing id and number of people.(SP))**

-- Function: project.effective\_cost(integer, character varying)

-- DROP FUNCTION project.effective\_cost(integer, character varying);

CREATE OR REPLACE FUNCTION project.effective\_cost(n integer, ctry character varying)

RETURNS SETOF project.max\_list AS

$BODY$

DECLARE

r max\_list%rowtype;

BEGIN

FOR r in SELECT listing\_id, (1000\*n/(cost\*max))::real as effective from max\_list where country = ctry order by effective desc

LOOP

RETURN NEXT r;

END LOOP;

RETURN;

END;

$BODY$

LANGUAGE plpgsql VOLATILE

COST 100

ROWS 1000;

ALTER FUNCTION project.effective\_cost(integer, character varying)

OWNER TO "201501183";

**CONSOLE APPLICATION:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sqlda.h>

//EXEC SQL include sqla.h;

sqlda\_t \*sqlda1;

sqlda\_t \*sqlda2;

EXEC SQL BEGIN DECLARE SECTION;

char search[20000],query[20000],x[20000],username[200],name[100];

char c,\*ans;

const char \*password;

int i=0,intval;

long long int longlongval;

EXEC SQL END DECLARE SECTION;

EXEC SQL WHENEVER SQLERROR CALL print\_sqlca();

void print\_sqlca()

{

fprintf(stderr, "==== sqlca ====\n");

fprintf(stderr, "sqlcode: %ld\n", sqlca.sqlcode);

fprintf(stderr, "sqlerrm.sqlerrml: %d\n", sqlca.sqlerrm.sqlerrml);

fprintf(stderr, "sqlerrm.sqlerrmc: %s\n", sqlca.sqlerrm.sqlerrmc);

fprintf(stderr, "sqlerrd: %ld %ld %ld %ld %ld %ld\n", sqlca.sqlerrd[0],sqlca.sqlerrd[1],sqlca.sqlerrd[2],

sqlca.sqlerrd[3],sqlca.sqlerrd[4],sqlca.sqlerrd[5]);

fprintf(stderr, "sqlwarn: %d %d %d %d %d %d %d %d\n", sqlca.sqlwarn[0], sqlca.sqlwarn[1], sqlca.sqlwarn[2],

sqlca.sqlwarn[3], sqlca.sqlwarn[4], sqlca.sqlwarn[5],

sqlca.sqlwarn[6], sqlca.sqlwarn[7]);

fprintf(stderr, "sqlstate: %5s\n", sqlca.sqlstate);

fprintf(stderr, "===============\n");

}

void ExecuteQuerry()

{

printf("Input Your Query :\n");

scanf(" %[^\n]",query);

EXEC SQL PREPARE eq FROM :query;

EXEC SQL DECLARE cur1 CURSOR FOR eq;

EXEC SQL OPEN cur1;

while(sqlca.sqlcode==0)

{

EXEC SQL FETCH NEXT FROM cur1 INTO DESCRIPTOR sqlda1;

sqlda\_t \*cur\_sqlda;

for (cur\_sqlda = sqlda1;cur\_sqlda != NULL;cur\_sqlda = cur\_sqlda->desc\_next)

{

int i;

char name\_buf[1024];

char var\_buf[1024];

if(sqlca.sqlcode != 0)

break;

for (i = 0; i < cur\_sqlda->sqld; i++)

{

sqlvar\_t v = cur\_sqlda->sqlvar[i];

char \*sqldata = v.sqldata;

short sqllen = v.sqllen;

strncpy(name\_buf, v.sqlname.data, v.sqlname.length);

name\_buf[v.sqlname.length] = '\0';

switch (v.sqltype)

{

case ECPGt\_char:

memset(&var\_buf, 0, sizeof(var\_buf));

memcpy(&var\_buf, sqldata, (sizeof(var\_buf)<=sqllen ? sizeof(var\_buf)-1 : sqllen) );

break;

case ECPGt\_int: /\* integer \*/

memcpy(&intval, sqldata, sqllen);

snprintf(var\_buf, sizeof(var\_buf), "%d", intval);

break;

case ECPGt\_long\_long: /\* bigint \*/

memcpy(&longlongval, sqldata, sqllen);

snprintf(var\_buf, sizeof(var\_buf), "%lld", longlongval);

break;

default:

{

int i;

memset(var\_buf, 0, sizeof(var\_buf));

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

{

char tmpbuf[16];

snprintf(tmpbuf, sizeof(tmpbuf), "%02x ", (unsigned char) sqldata[i]);

strncat(var\_buf, tmpbuf, sizeof(var\_buf));

}

}

break;

}

printf("%s = %s\n", name\_buf, var\_buf);

}

printf("\n");

}

}

EXEC SQL CLOSE cur1;

EXEC SQL COMMIT;

}

void UpdateQuerry()

{

printf("Input Your Query : \n");

scanf(" %[^\n]",query);

//printf("%s\n",query);

EXEC SQL PREPARE uq FROM :query;

EXEC SQL EXECUTE uq;

if(sqlca.sqlcode==0)

{

fprintf(stderr, "Successful\n");

}

EXEC SQL COMMIT;

}

int main()

{

printf("USERNAME : ");

scanf(" %[^\n]",username);

getchar();

password=getpass("PASSWORD : ");

strcat(name,username);

strcat(name,"@10.100.71.21");

EXEC SQL CONNECT TO :name USER :username USING :password;

if(sqlca.sqlcode==0)

fprintf(stderr, "Connection Successful\n");

else

exit(1);

printf("Set your search path :\n");

scanf(" %[^\n]",search);

EXEC SQL PREPARE search\_path FROM :search;

EXEC SQL EXECUTE search\_path;

if(sqlca.sqlcode==0)

fprintf(stderr,"Search\_path Successfully set\n");

else

{

fprintf(stderr,"No such path available\n");

exit(1);

}

while(1)

{

printf("Press 1 for INSERT,UPDATE and DELETE Querry.\nPress 2 for SELECT querry.\nPress anything else to exit.\n");

scanf(" %c",&c);

if(c=='1')

{

UpdateQuerry();

}

else if(c=='2')

{

ExecuteQuerry();

}

else

break;

}

EXEC SQL DISCONNECT ALL;

return 0;

}