IS 620 - Spring 2016
Deliverable 4
UMBC Event Management

Alan Blitstein
Krutika Karveershettar
Pooja Gond
Rashmi Fegade

Table Of Contents

1) Drop Table Statements

```
drop table waitlist;
drop table evntreg;
drop table message;
drop table event;
drop table organizers;
drop table locations;
drop table users;
```

2) Create Table Statements

```
create table users(
        user_id int,
        user_name varchar(30),
        user_phone varchar(15),
        user_email varchar(30),
        user_password varchar(30),
        user_type varchar(15),
        primary key (user_id)
        );

create table locations(
        loc_id int, -- location id
        loc_name varchar(30), -- location name
        loc_descr varchar(100), -- location description
        loc_capacity int, -- location capacity
        primary key (loc_id)
        );

create table organizers(
        orgn_id int, -- organizer id
        user_id int,
        orgn_name varchar(30), -- organizer name
        user_email varchar(30), -- organizer email id
        primary key(orgn_id),
        foreign key(user_id) references users(user_id)
        );

create table event(
        evnt_id int,
        loc_id int,
        orgn_id int, -- organizer id
        evnt_title varchar(30), -- event title
```

```
	evnt_descr varchar(100), -- event description
	evnt_strdatetime timestamp, -- event start date and time
	evnt_enddatetime timestamp, -- event end date and time
	evnt_url varchar(100), --event URL
	evnt_capacity int, -- event capacity
	evnt_wl_capacity int, -- event waitlist capacity
	evnt_active char, -- event status active or not
	event_full varchar(5), ---to check event is full or not
	primary key(evnt_id),
	foreign key(loc_id) references locations(loc_id),
	foreign key(orgn_id) references organizers(orgn_id)
	);

create table message(
	msg_id int, -- message id
	user_id int,
	msg_time timestamp, -- message time
	msg varchar(350), -- message body
	msg_type varchar(15), -- message type register or unregister
	primary key(msg_id),
	foreign key(user_id) references users(user_id)
	);

create table evntreg(
	evnt_id int,
	user_id int,
	reg_time timestamp, -- registeration date and time
	rating int, -- event rating
	commnt varchar(100), -- user comment
	foreign key(evnt_id) references event(evnt_id),
	foreign key(user_id) references users(user_id)
	);

create table waitlist(
	evnt_id int,
	wl_position int, -- waitlist position of the user
	user_id int,
	foreign key(evnt_id) references event(evnt_id),
	foreign key(user_id) references users(user_id)
	);
```

3) Insert Table Statements

```
insert into users
values (1,'Alan','4101233211','alan1@umbc.edu','alanpassword','student');
insert into users
values (2,'Pooja','4104258219','pooja1@umbc.edu','poojapassword','student');
insert into users
values (3,'Krutika','4107254322','krutika1@umbc.edu','krutikapassword','student');
insert into users
values (4,'Rashmi','4102876211','rashmi1@umbc.edu','rashmipassword','student');
insert into users
values (5,'Dr.Chen','4105285642','drchen1@umbc.edu','chenpassword','faculty');
insert into users
values (6,'Larry','4109993201','larry1@umbc.edu','larrypassword','staff');
insert into users
values (7,'Emily','4103323202','emily2@umbc.edu','emilypassword','staff');
insert into users
values (8,'Dr.Smith','4100983243','drsmith9@umbc.edu','smithpassword','faculty');
insert into users
values (9,'Mark','4108983218','mark1@umbc.edu','markpassword','student');
insert into users
values (10,'Dr.Jack','4103569087','drjack3@umbc.edu','jackpassword','faculty');
insert into users
values (11,'Grace','4100895101','grace3@umbc.edu','gracepassword','staff');
insert into users
values (12,'Mike','4105643201','mike5@umbc.edu','mikepassword','student');
insert into users
values (13,'Greg','4101292281','greg1@umbc.edu','jackpassword','student');
insert into users
values (14,'Jackie','4107743201','jackie1@umbc.edu','jackiepassword','student');
insert into users
values (15,'Mr.Garrison','4103276543','garrison5@umbc.edu','garrisonpassword','staff');
insert into users
values (16,'Dr.Jones','4103233201','drjones4@umbc.edu','cohenpassword','faculty');
insert into users
values (17,'Dr.Engle','4100090078','drengle3@umbc.edu','englepassword','faculty');
insert into users
values (18,'Harry','4105553435','harry7@umbc.edu','harrypassword','student');
insert into users
values (19,'Owen','4101233212','owen1@umbc.edu','owenpassword','student');
insert into users
values (20,'Barrry','4103651045','barry2@umbc.edu','barrypassword','student');
```

```
insert into locations
values (1, 'Sherman Hall', 'SH102', 10);
insert into locations
values (2, 'Library', 'LIB401', 15);
insert into locations
values (3, 'Fine Arts Bldg,', 'FA202', 8);
insert into locations
values (4, 'Sondheim Hall', 'SO201', 15);
insert into locations
values (5, 'IT/Eng Bldg.', 'ITE401', 12);
insert into locations
values (6, 'Math/Psych Bldg.', 'MP101', 20);
insert into locations
values (7, 'Lounge', 'LOU111', 6);

insert into organizers
values (1, 4, 'Rashmi','rashmi1@umbc.edu');
insert into organizers
values (2, 5, 'Dr.Chen','drchen1@umbc.edu');
insert into organizers
values (3, 7, 'Emily', 'emily2@umbc.edu');
insert into organizers
values (4, 9, 'Mark', 'mark1@umbc.edu');
insert into organizers
values (5, 10, 'Dr.Cohen','drcohen1@umbc.edu');

insert into event
values (1, 1, 1,'Chess Club', 'Weekly Chess Meeting', timestamp '2016-9-15 12:00:00.00',
timestamp '2016-9-15 13:00:00.00', 'www.umbc.edu/chess', 10, 5, 't','no');
insert into event
values (2, 7, 5,'Math Tutoring', 'Sign up for Math Help!', timestamp '2016-9-17 8:00:00.00',
timestamp '2016-9-17 10:00:00.00', 'www.umbc.edu/math', 10, 4, 't','yes');
insert into event
values (3, 6, 3,'Guest Speaker', 'Come listen to President Obama', timestamp '2016-10-01
17:00:00.00', timestamp '2016-10-01 18:00:00.00', 'www.umbc.edu/speaker', 15, 5, 't','no');
insert into event
values (4, 2, 2,'Book Club', 'Weekly Book Club Talk', timestamp '2016-10-07 12:00:00.00',
timestamp '2016-10-07 13:00:00.00', 'www.umbc.edu/bookclub', 25, 10, 't','no');
insert into event
values (5, 3, 4,'Drawing', 'Lets make art', timestamp '2016-10-09 14:00:00.00', timestamp '2016-
10-09 14:45:00.00', 'www.umbc.edu/draw', 7, 5, 't', 'yes');
insert into event
values (6, 7, 5,'Karoke', 'Welcome to the fun night', timestamp '2016-12-03 20:00:00.00',
timestamp '2016-12-03 22:00:00.00', 'www.umbc.edu/karoke', 20, 15, 't', 'no');
```

```
insert into message
values (1, 3, timestamp '2016-9-02 8:00:00.00', 'registering for math tutoring', 'register');
insert into message
values (2, 1, timestamp '2016-9-02 8:10:00.00', 'registering for math tutoring', 'register');
insert into message
values (3, 4, timestamp '2016-9-03 8:15:00.00', 'registering for math tutoring', 'register');
insert into message
values (4, 2, timestamp '2016-9-03 8:20:00.00', 'registering for math tutoring', 'register');
insert into message
values (5, 20, timestamp '2016-9-04 8:30:00.00', 'registering for math tutoring', 'register');
insert into message
values (6, 18, timestamp '2016-9-04 8:40:00.00', 'registering for math tutoring', 'register');
insert into message
values (7, 7, timestamp '2016-9-04 9:00:00.00', 'registering for math tutoring', 'register');
insert into message
values (8, 9, timestamp '2016-9-04 9:10:00.00', 'registering for math tutoring', 'register');



insert into evntreg
values (1, 1, timestamp '2016-9-04 20:10:00.00', 2, 'significant');
insert into evntreg
values (1, 3, timestamp '2016-9-05 16:10:00.00', 4, 'relishable');
insert into evntreg
values (1, 13, timestamp '2016-9-12 12:10:00.00', 5, 'preferable');
insert into evntreg
values (1, 5, timestamp '2016-9-08 08:10:00.00', 1, 'tiring');
insert into evntreg
values (1, 11, timestamp '2016-9-04 02:10:00.00', 2, 'sattisfactory');
insert into evntreg
values (1, 16, timestamp '2016-9-06 13:10:00.00', 5, 'likeable');
insert into evntreg
values (1, 10, timestamp '2016-9-03 10:10:00.00', 4,'pleasant');
insert into evntreg
values (1, 19, timestamp '2016-9-06 11:10:00.00', 3, 'forthcoming');
insert into evntreg
values (1, 17, timestamp '2016-9-04 20:20:00.00', 4, 'genial');
insert into evntreg
values (2, 3, timestamp '2016-9-02 8:00:00.00', 5, 'informative');
insert into evntreg
```

```
values (2, 1, timestamp '2016-9-02 8:10:00.00', 4, 'knowledge gained');
insert into evntreg
values (2, 4, timestamp '2016-9-03 8:15:00.00', 1, 'not good');
insert into evntreg
values (2, 2, timestamp '2016-9-03 8:20:00.00', 3, 'educational');
insert into evntreg
values (2, 20, timestamp '2016-9-04 8:30:00.00', 4, 'illuminating');
insert into evntreg
values (2, 18, timestamp '2016-9-04 8:40:00.00', 2, 'edifying');
insert into evntreg
values (2, 7, timestamp '2016-9-04 9:00:00.00', 5, 'newsy');
insert into evntreg
values (2, 9, timestamp '2016-9-04 9:10:00.00', 3, 'elucidative');
insert into evntreg
values (2, 12, timestamp '2016-9-04 20:10:00.00', 3, 'significant');
insert into evntreg
values (2, 14, timestamp '2016-9-04 17:10:00.00', 2, 'revelatory');
insert into evntreg
values (3, 3, timestamp '2016-9-30 17:10:00.00', 2, 'drudging');
insert into evntreg
values (3, 12, timestamp '2016-9-22 17:10:00.00', 1, 'arid');
insert into evntreg
values (3, 9, timestamp '2016-9-19 17:10:00.00', 3, 'uninteresting');
insert into evntreg
values (3, 6, timestamp '2016-9-14 17:10:00.00', 2, 'cloying');
insert into evntreg
values (3, 14, timestamp '2016-9-15 17:10:00.00', 5, 'revelatory');
insert into evntreg
values (5, 8, timestamp '2016-9-30 12:10:00.00', 5, 'good');
insert into evntreg
values (5, 13, timestamp '2016-10-05 23:10:00.00', 4, 'interesting');
insert into evntreg
values (5, 15, timestamp '2016-9-22 19:10:00.00', 5, 'favorable');
insert into evntreg
values (5, 17, timestamp '2016-10-08 07:10:00.00', 3, 'wonderful');
insert into evntreg
values (5, 18, timestamp '2016-10-02 17:10:00.00', 4, 'exiciting');
insert into evntreg
values (5, 2, timestamp '2016-10-05 15:10:00.00', 4, 'commendable');
insert into evntreg
values (5, 6, timestamp '2016-9-25 22:10:00.00', 5, 'superb');
insert into evntreg
values (6, 10, timestamp '2016-10-30 22:10:00.00', 4, 'super eminent');
insert into evntreg
```

```
values (6, 7, timestamp '2016-11-12 22:10:00.00', 3, 'admirable');
insert into evntreg
values (6, 15, timestamp '2016-11-14 22:10:00.00', 5, 'superb');
insert into evntreg
values (6, 3, timestamp '2016-11-05 22:10:00.00', 2, 'unskilled');
insert into evntreg
values (6, 1, timestamp '2016-11-25 22:10:00.00', 5, 'marvelous');


insert into waitlist values (2, 1, 5);
insert into waitlist values (2, 2, 13);
insert into waitlist values (2, 3, 16);
insert into waitlist values (2, 4, 19);
insert into waitlist values (5, 5, 9);
insert into waitlist values (5, 5, 11);

--this function checks if an event ID exists
--it returns 1 if the event exists and 0 if it does not

create or replace function evidcheck (ev_id number)
return int is
evrow event%rowtype;
begin
select e.* into evrow
from event e
where e.evnt_id = ev_id;
return (1);
exception when no_data_found then
return(0);
end;
```

4. Create procedure statements

Feature 1:

Register an account with the system. The user needs to provide name, phone#, email, password, and type of user (faculty, staff, or student). The procedure should check whether the email already exists in user table. If so, please print a message saying the account exists. Otherwise create an account with input values and return a new user ID.

Input: user_name, user_phone, user_email, user_password, user_type
Output: Message saying '*account exists*' if account already exist else print '*Account is created. Your user id is xyz*'.
Create Procedure Statements (with comments):

```
set serveroutput on;
CREATE OR REPLACE PROCEDURE RegisterAccount(u_name in varchar2, u_phone in
varchar2, u_email in varchar2, u_password in varchar2, u_type varchar2) IS
userCountExist number;
NewUserId number;
BEGIN
        SELECT COUNT(user_id)
        INTO userCountExist
        FROM users
        WHERE user_email = u_email; -- Check whether email id is already exist

        if (userCountExist >= 1 ) then
                dbms_output.put_line('Account Already Exist');
        else
                INSERT INTO users VALUES(userid_seq.nextval, u_name, u_phone, u_email,
u_password, u_type); -- creating new account with given values

                select userid_seq.currval into NewUserId from dual; -- fetching current user id if
from user table
                dbms_output.put_line('Account is created with new user id: ' || NewUserId);
        commit;
  END IF;
END;
```

Execute Statements:

```
exec RegisterAccount('Rashmi', 4102876211, 'rashmi1@umbc.edu', 'rashmipassword', 'student');
-- Example of already exist account
exec RegisterAccount('Susan', 4102349876, 'susan@umbc.edu', 'susanpassword', 'staff');
exec RegisterAccount('Robin', 4103249876, 'robin@umbc.edu', 'robinpassword', 'staff');
```

exec RegisterAccount('Tom', 4104329876, 'tom@umbc.edu', 'tompassword', 'staff'); -- Example of new user

Output:

PL/SQL procedure successfully completed.

Account Already Exist

PL/SQL procedure successfully completed.

Account Already Exist

PL/SQL procedure successfully completed.

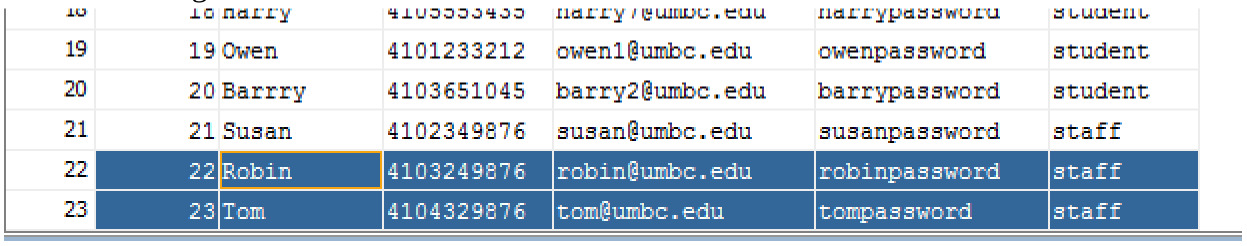
Account is created with new user id: 22

PL/SQL procedure successfully completed

Account is created with new user id: 23

Snapshot:

1. For user is already registered.
   In this case, as user is already exist in the table, procedure won't allow user to register again.
2. For user not registered.



18	18	Harry	4105553435	harry7@umbc.edu	harrypassword	student
19	19	Owen	4101233212	owen1@umbc.edu	owenpassword	student
20	20	Barrry	4103651045	barry2@umbc.edu	barrypassword	student
21	21	Susan	4102349876	susan@umbc.edu	susanpassword	staff
22	22	Robin	4103249876	robin@umbc.edu	robinpassword	staff
23	23	Tom	4104329876	tom@umbc.edu	tompassword	staff

In this case, user who are not registered before or not exist in the table are registered with new user id.

Feature 2:

Allow a user to login by providing user id and password. Please check whether user id exists and password matches. If not, please print a message to indicate the error. Otherwise print a message to indicate user has logged on. The procedure should return a value 1 for success login and 0 for unsuccessful log in.

Input: user_id, user_password
Output: 1 for successful login, 0 for unsuccessful login.

Create Procedure Statements (with comments)

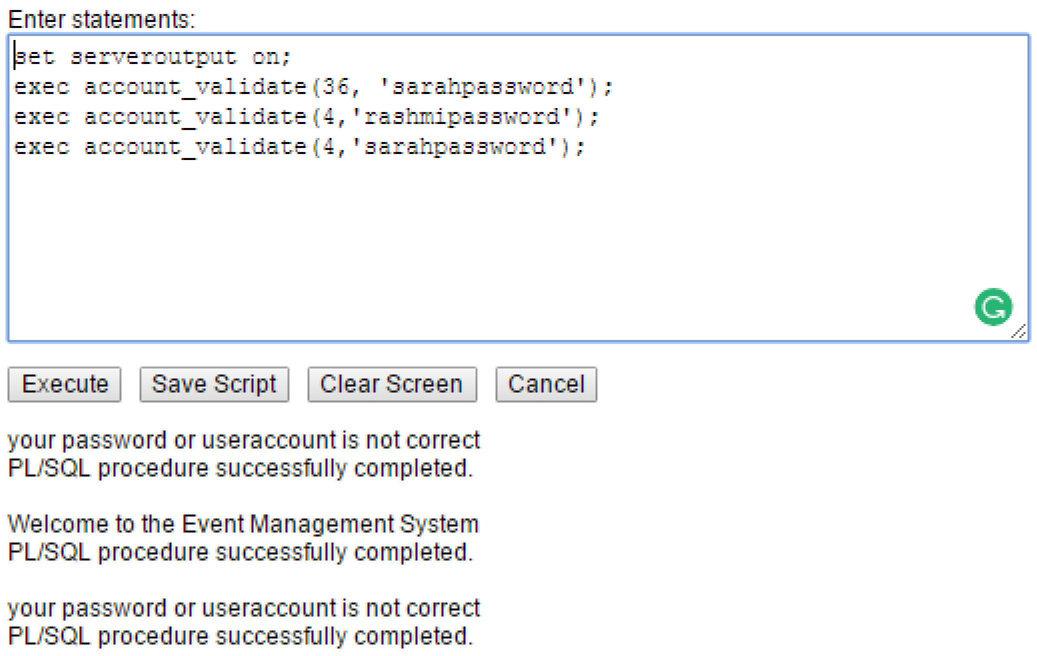
```
-- function to check login is successful or not. If it is successful it will return 1 else it will return 0.
set serveroutput on;
create or replace function account_validation (u_id number, u_password varchar) return number as
userid users.user_id%type;
userpass users.user_password%type;
success varchar2(20);
begin
        SELECT user_id, user_password  -- To check given user id and user passward in the user table
        into userid, userpass
        from users
        where user_id = u_id and user_password = u_password;

        if (userid != u_id and userpass != u_password)  then  -- Checking user is and user password matching or not. If it is not matches it will return 0 else it will return 1.
                return 0;
        else
                return 1;
        end if ;

exception
when no_data_found then
return -1;
end;

------------------------------------------------------------
-- Procedure to call the function account_validation
set serveroutput on;
create or replace procedure account_validate(u_id number, u_password varchar) as --this procedure is to Validate the identity of a user based on the input user-id and password
userid users.user_id%type;
userpass users.user_password%type;
```

```
success varchar2(20);
Begin
        success := account_validation(u_id, u_password); -- executing function

        if success = 1  then -- checking login is suceesful or not based on the result of function.
                dbms_output.put_line('Welcome to the Event Management System');


        else
                dbms_output.put_line('your password or useraccount is not correct ');


        end if ;


end;
```

Execute Statements:

```
exec account_validate(36, 'sarahpassword'); --For no data in table
exec account_validate(4,'rashmipassword'); -- For correct combination of userid and password
exec account_validate(4,'sarahpassword'); -- For incorrect combination of userid and password
```

Output:



```
set serveroutput on;
exec account_validate(36, 'sarahpassword');
exec account_validate(4,'rashmipassword');
exec account_validate(4,'sarahpassword');
```

```
your password or useraccount is not correct
PL/SQL procedure successfully completed.

Welcome to the Event Management System
PL/SQL procedure successfully completed.

your password or useraccount is not correct
PL/SQL procedure successfully completed.
```

Explanation of Output:

In the first and third statement, procedure gives output as your password and user account is not correct. When user Id and password does not matches it will not allow user to enter in the account.
In second statement, given user id and password matches and gives the output as ‘Welcome to the Event Management System’.

Feature 3:

Allow a user to read messages providing user id.

Input: user_id
Output: message
Exception:

- If no data found print ‘User is not registered.’

Create Procedure Statements (with comments)

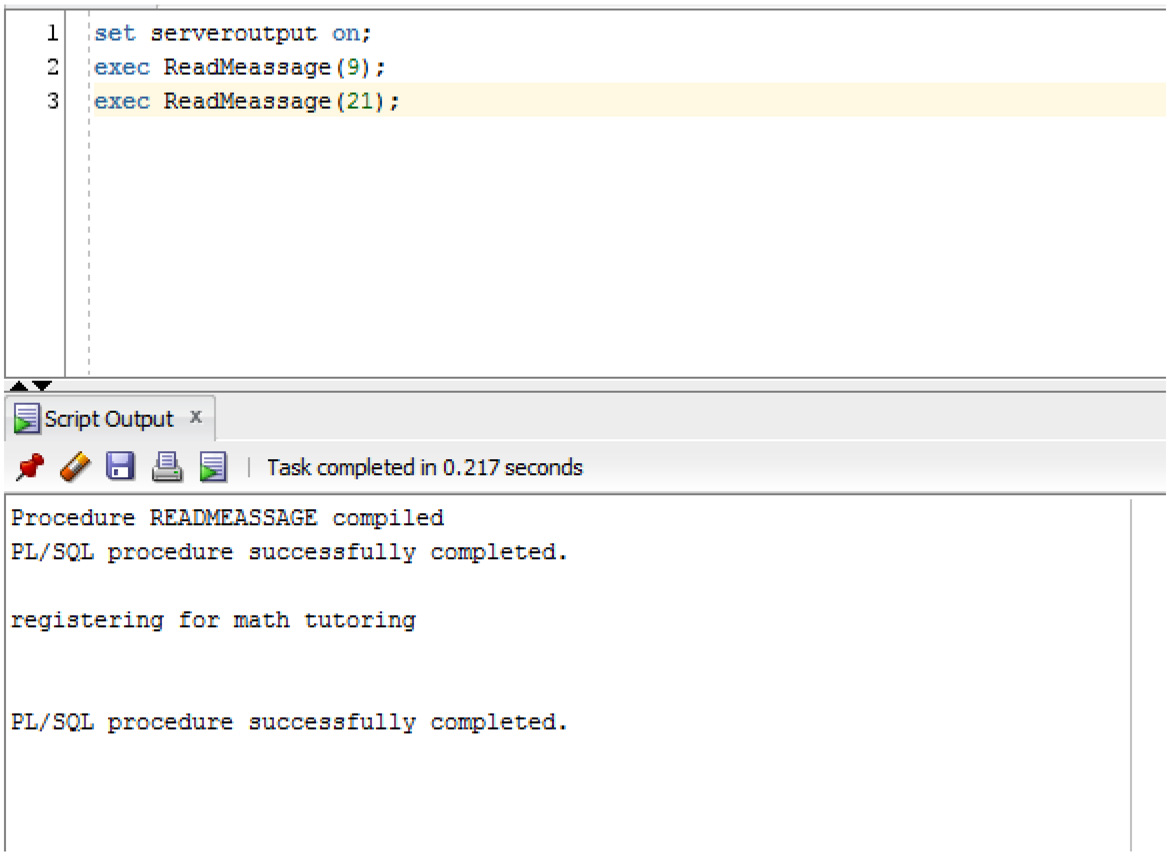
```
Set serveroutput on;
Create or replace
        PROCEDURE ReadMeassage(u_id in integer) IS --- This procedure allows to read
message by providing each user id
Cursor c1 is select msg from message where user_id = u_id; --- checks if the user id in the event
table matches the user id in message table
msg_g message.msg%type;
uid number;
BEGIN

SELECT user_id into uid from users where user_id = u_id;

If uid is not null then
        dbms_output.put_line('User is registered.');

        Open c1;
                Loop
                        fetch c1 into msg_g;
                        exit when c1%notfound;
                        dbms_output.put_line(msg_g);
                End loop;
        Close c1;
else
        dbms_output.put_line('User is not registered.');
```

```
end if;

exception
        when no_data_found then
        dbms_output.put_line('Wrong User Id.');

end;
```

Execute statement:

set serveroutput on;
exec ReadMeassage(9); -- for user id registered
exec ReadMeassage(21); -- for user id not registered

Output:



```
set serveroutput on;
exec ReadMeassage(9);
exec ReadMeassage(21);
```

```
Procedure READMEASSAGE compiled
PL/SQL procedure successfully completed.

registering for math tutoring


PL/SQL procedure successfully completed.
```

Explanation of output:

exec ReadMeassage(9); -- this is standard output as it prints the message for the user id registered
exec ReadMeassage(21); -- this is the output for the users who are not registered

Feature 4:

Create an event with title, description, start date and time, end date and time, location, an optional url, and organizer id. The procedure needs to check whether any other event at the same location has overlap duration with the new event. If so, print a message saying the event conflicts with an existing event. Otherwise, insert the event with input values into event table and print a message with new event ID.

Input: evnt_title, evnt_descr,evnt_strdatetime, evnt_enddatetime, evnt_url, loc_id, orgn_id
Output: 1 for successful login, 0 for unsuccessful login
Exception:

- If conflict occurs, print '*The event conflicts with an existing event*'.

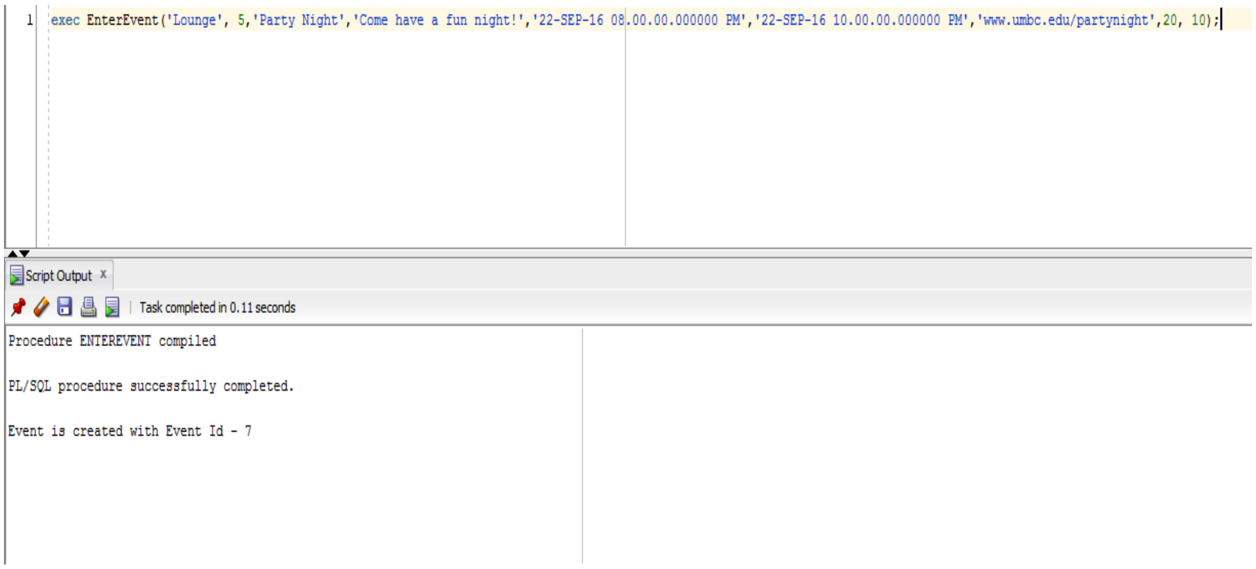
Create Procedure Statements (with comments):

```
set serveroutput on;
create or replace PROCEDURE EnterEvent(location in varchar, organizer_id in
INTEGER,event_title in VARCHAR,event_descr in VARCHAR,
event_strdatetime in timestamp,event_enddatetime in timestamp,event_url
varchar,event_capacity in INTEGER, event_wl_capacity in INTEGER) IS
ConflictEventCheck number;
location_id number;
x number;
BEGIN
select loc_id into location_id from locations where loc_name = location;

Select count(E2.evnt_id) into ConflictEventCheck  -- Checking whether any event having
conflict with another by comparing location , start time and end time.
from event E1, event E2
where E2.loc_id = location_id
and E2.evnt_strdatetime = event_strdatetime
and E2.evnt_id = E1.evnt_id
and E2.loc_id = E1.loc_id
and E2.evnt_strdatetime = E1.evnt_strdatetime;

-- if there are more records with same location , start time and end time then there is conflict. If
there is no conflict then it will create new event with new event Id.
if (ConflictEventCheck != 0) then
                        dbms_output.put_line('conflict occur');
else
   insert into Event(evnt_id, Loc_Id, orgn_id, evnt_title, evnt_descr, evnt_strdatetime,
evnt_enddatetime, evnt_url, evnt_capacity, evnt_wl_capacity) values
(evntid_seq.nextval,location_id, organizer_id, event_title, event_descr, event_strdatetime,
event_enddatetime, event_url, event_capacity, event_wl_capacity);
   select evntid_seq.currval into x from dual;  -- selecting current event id from event table.
   dbms_output.put_line('Event is created with Event Id - ' || x);
end if;

end;
```

Execute statement:

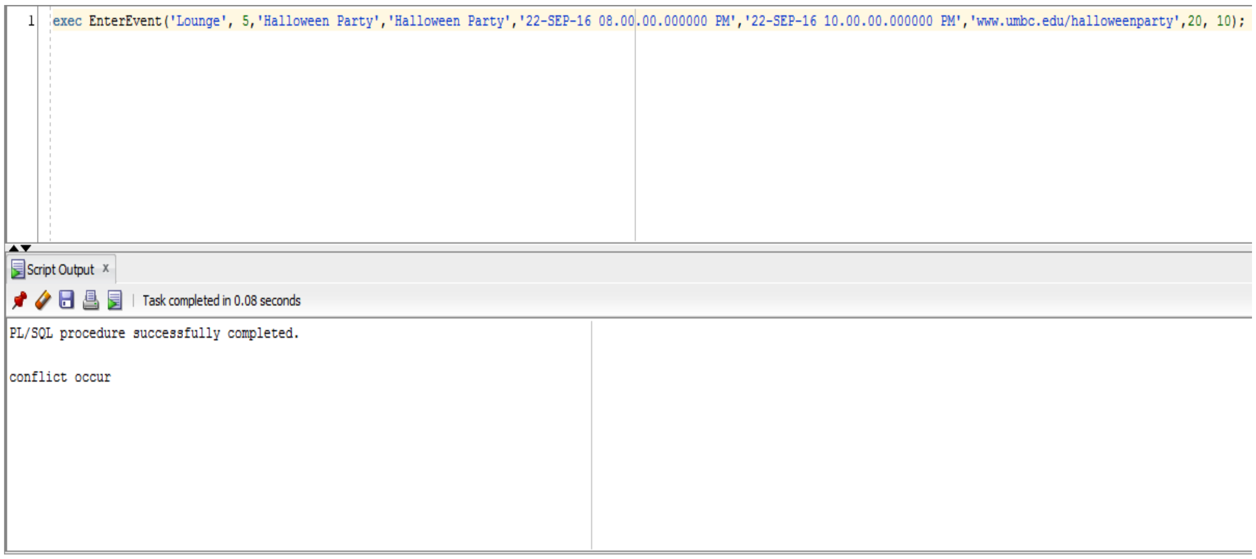
```
exec EnterEvent('Lounge', 5,'Party Night','Come have a fun night!','22-SEP-16 08.00.00.000000
PM','22-SEP-16 10.00.00.000000 PM','www.umbc.edu/partynight',20, 10); -- No conflict

exec EnterEvent('Lounge', 5,'Halloween Party','Halloween Party','22-SEP-16 08.00.00.000000
PM','22-SEP-16 10.00.00.000000 PM','www.umbc.edu/halloweenparty',20, 10); -- Conflict
```

Output:



```
exec EnterEvent('Lounge', 5,'Party Night','Come have a fun night!','22-SEP-16 08.00.00.000000 PM','22-SEP-16 10.00.00.000000 PM','www.umbc.edu/partynight',20, 10);
```

```
Procedure ENTEREVENT compiled

PL/SQL procedure successfully completed.

Event is created with Event Id - 7
```


```
exec EnterEvent('Lounge', 5,'Halloween Party','Halloween Party','22-SEP-16 08.00.00.000000 PM','22-SEP-16 10.00.00.000000 PM','www.umbc.edu/halloweenparty',20, 10);
```
Script Output

Task completed in 0.08 seconds
```
PL/SQL procedure successfully completed.

conflict occur
```

Explanation of output:

exec EnterEvent('Lounge', 5,'Party Night','Come have a fun night!','22-SEP-16 08.00.00.000000 PM','22-SEP-16 10.00.00.000000 PM','www.umbc.edu/partynight',20, 10); -- This is the standard output when there is no other event present which is conflicting with this one.

exec EnterEvent('Lounge', 5,'Halloween Party','Halloween Party','22-SEP-16 08.00.00.000000 PM','22-SEP-16 10.00.00.000000 PM','www.umbc.edu/halloweenparty',20, 10); -- This is the standard output when there is another event present which is conflicting with this one.

Feature 5:

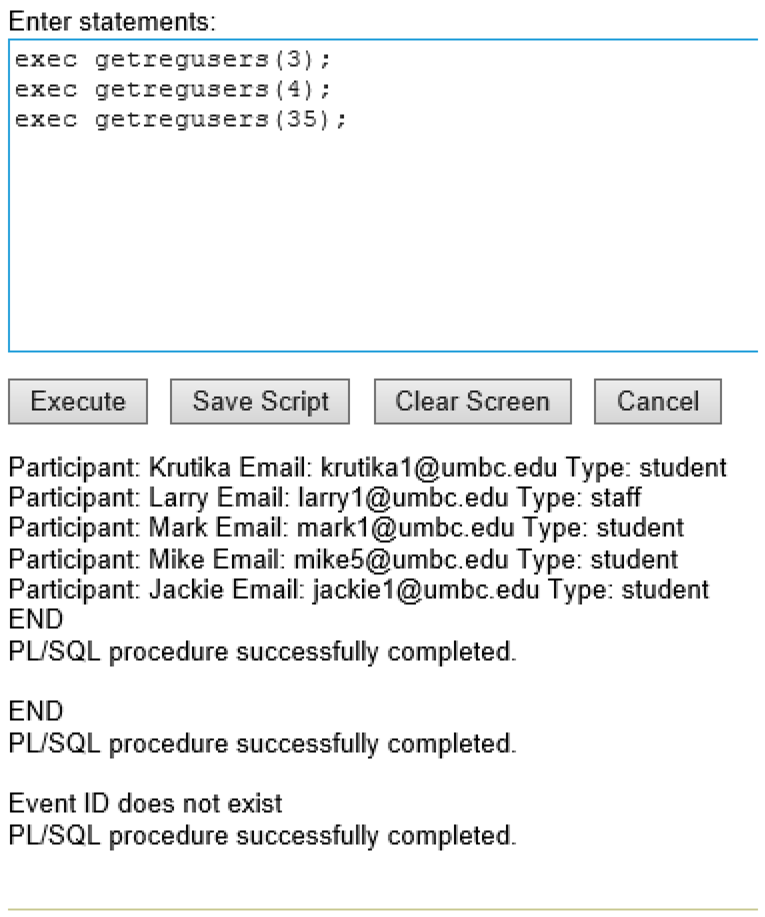
List people registered for an event by providing event id. The procedure prints out name of participants, their email addresses, and whether the participant is faculty, staff, or student.

Input: evnt_id
Output: user_id, user_email, user_type (will be multiple rows if more than 1 registered user)
Parameter Example: exec getregusers(3);
Exceptions:

- If evnt_id does not exist: print *Event ID does not exist*
- If nobody registered for event: END will appear alone

Create Procedure Statements (with comments):
create or replace PROCEDURE getregusers (ev_id int)
as cursor c1 is
select u.user_name as Participants, u.user_email as Email, u.user_type as u_Type

```
from users u, evntreg e --select participant, email, type of user
where e.evnt_id=ev_id and u.user_id=e.user_id; --when the desired event ID lookup matches
records in table
r c1%rowtype; --catch data into rowtype variable
begin
if evidcheck(ev_id) > 0 then
open c1;
loop
fetch c1 into r;
exit when c1%notfound; --exit when no more matches
dbms_output.put_line('Participant: '|| r.participants ||' Email: ' || r.Email ||' Type: '|| r.u_Type);
end loop;
dbms_output.put_line('END');--the tells user cursor is done parsing
close c1;
else
dbms_output.put_line('Event ID does not exist');
end if;
end;
```



```
exec getregusers(3);
exec getregusers(4);
exec getregusers(35);
```

```
Participant: Krutika Email: krutika1@umbc.edu Type: student
Participant: Larry Email: larry1@umbc.edu Type: staff
Participant: Mark Email: mark1@umbc.edu Type: student
Participant: Mike Email: mike5@umbc.edu Type: student
Participant: Jackie Email: jackie1@umbc.edu Type: student
END
PL/SQL procedure successfully completed.

END
PL/SQL procedure successfully completed.

Event ID does not exist
PL/SQL procedure successfully completed.
```

Explanation of output:
Exec getregusers(3): This is standard output as the event has participants registered
Exec getregusers(4): This event has no registrees - it simply prints out END signifying there are no users to print
Exec getregusers(35): This event does not exist. A statement is prined declaring that to be so.

Feature 6:

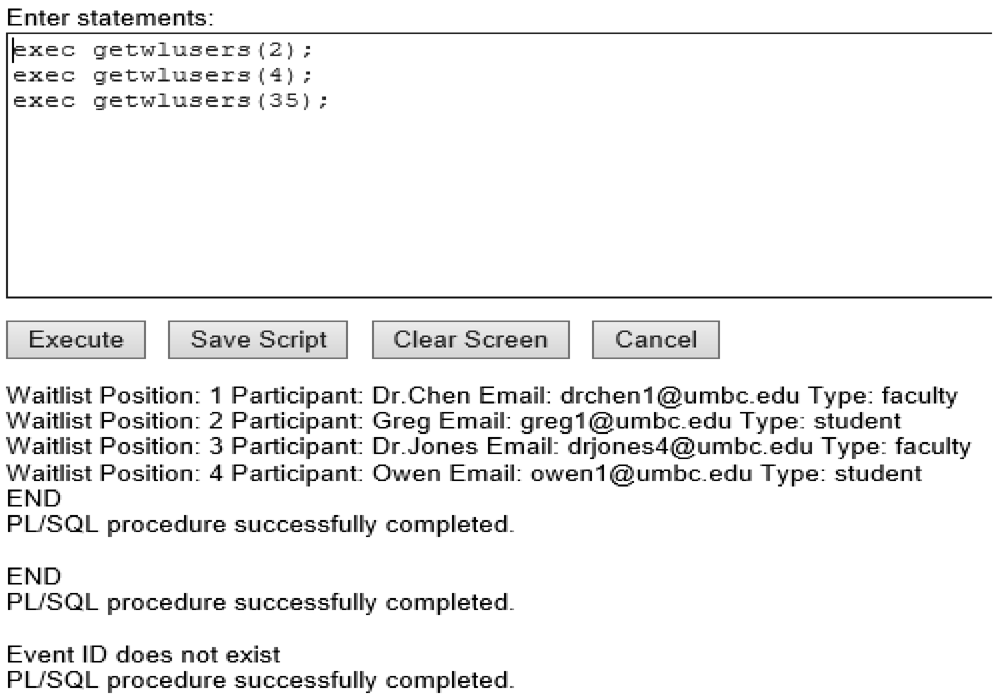
List people on wait list of an event by providing the event id. The procedure prints out names of users on the wait list, their email addresses, and whether they are faculty, staff, or students.

Input: evnt_id
Output: user_id, user_email, user_type (will be multiple rows if more than 1 waitlist user)
Parameter Example: exec getwlusers(2);
Exceptions:

- If evnt_id does not exist: print *Event ID does not exist*
- If nobody is on waitlist: END will appear alone

Create Procedure Statements (with comments):

```
create or replace PROCEDURE getwlusers (ev_id int)
as cursor c1 is
select u.user_name as Participants, u.user_email as Email, u.user_type as u_Type, w.wl_position
as WL_Position
from users u, waitlist w --select participant, email, type of user
where w.evnt_id=ev_id and u.user_id=w.user_id; --when the desired event ID lookup matches
records in table
r c1%rowtype; --catch data into rowtype variable
begin
if evidcheck(ev_id) > 0 then
open c1;
loop
fetch c1 into r;
exit when c1%notfound; --exit when no more matches
dbms_output.put_line('Waitlist Position: ' || r.wl_position || ' Participant: '|| r.participants ||' Email:
' || r.Email ||' Type: '|| r.u_Type);
end loop;
dbms_output.put_line('END');--the tells user cursor is done parsing
close c1;
else
dbms_output.put_line('Event ID does not exist');
end if;
End;
```



```
exec getwlusers(2);
exec getwlusers(4);
exec getwlusers(35);
```

```
Waitlist Position: 1 Participant: Dr.Chen Email: drchen1@umbc.edu Type: faculty
Waitlist Position: 2 Participant: Greg Email: greg1@umbc.edu Type: student
Waitlist Position: 3 Participant: Dr.Jones Email: drjones4@umbc.edu Type: faculty
Waitlist Position: 4 Participant: Owen Email: owen1@umbc.edu Type: student
END
PL/SQL procedure successfully completed.

END
PL/SQL procedure successfully completed.

Event ID does not exist
PL/SQL procedure successfully completed.
```

Explanation of output:
Exec getwlusers(2): This is standard output as the event has participants on the waitlist
Exec getwlusers(4): This event has nobody on the waitlist - it simply prints out END signifying there are no users to print
Exec getwlusers(35): This event does not exist. A statement is printed declaring that to be so.

Feature 7:

Return average rating of an event and total number of participants and number of people on wait list.

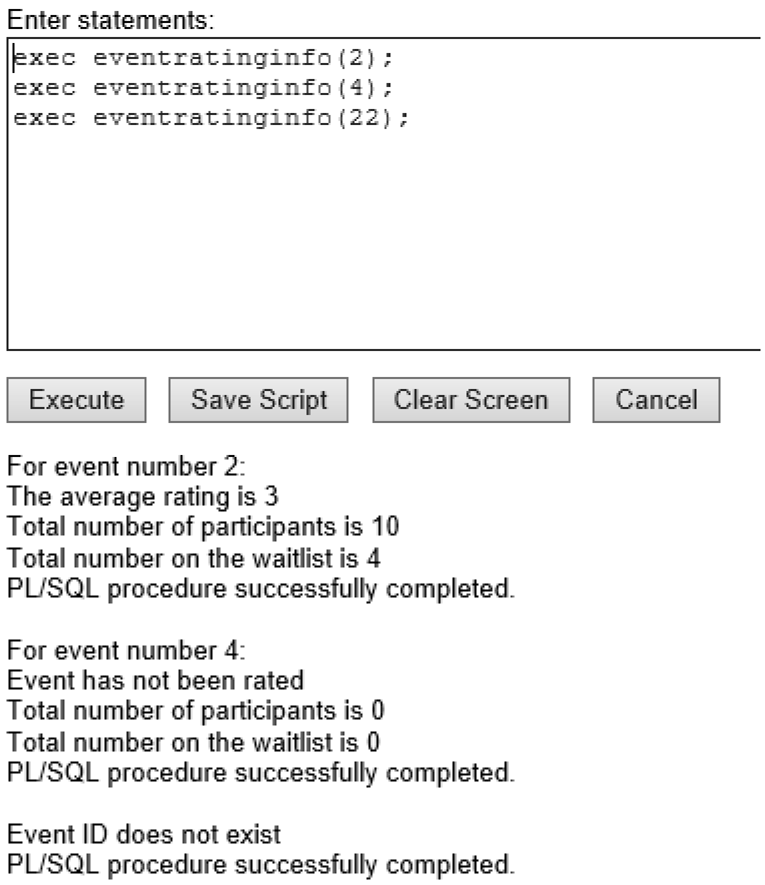
Input: evnt_id
Output: print *Average rating for event **x** is **y**. Total number of participants is **a** and there are* **b** people on the waitlist
Parameter Example: exec eventratinginfo(2);
Exceptions:

- If evnt_id does not exist: print *Event ID does not exist*
- If there are no ratings: print *There are currently not any ratings for this event*

Create Procedure Statements (with comments):

```
create or replace procedure eventratinginfo(ev_id in integer) is
avg_rating int;
totalpartic int;
totalwl int;
begin
if evidcheck(ev_id) > 0 then --first check if event exists
dbms_output.put_line('For event number ' || ev_id || ':');
select avg(e.rating) into avg_rating
from evntreg e
where e.evnt_id= ev_id;
if avg_rating > 0 then --check if event has been rated
dbms_output.put_line('The average rating is ' || avg_rating);
else
dbms_output.put_line('Event has not been rated');--message if the if statement isn't met
end if;
select count(*) into totalpartic
from evntreg e
where e.evnt_id=ev_id;
dbms_output.put_line('Total number of participants is ' || totalpartic);
select count(*) into totalwl
from waitlist w
```

```
where w.evnt_id=ev_id;
dbms_output.put_line('Total number on the waitlist is ' || totalwl);
else
dbms_output.put_line('Event ID does not exist');
end if;
end;
```



```
exec eventratinginfo(2);
exec eventratinginfo(4);
exec eventratinginfo(22);
```

```
For event number 2:
The average rating is 3
Total number of participants is 10
Total number on the waitlist is 4
PL/SQL procedure successfully completed.

For event number 4:
Event has not been rated
Total number of participants is 0
Total number on the waitlist is 0
PL/SQL procedure successfully completed.

Event ID does not exist
PL/SQL procedure successfully completed.
```

Explanation of output:

Exec eventratinginfo(2): This is standard output. All the information about the event is printed
Exec eventratinginfo(4): This is standard output except the user is informed that the event has not been rated
Exec eventratinginfo(22): This event does not exist, so it lets the user know that.

Feature 8:

Cancel an event. Please generate a message (by inserting into the message table) for each person who has registered or on wait list saying that the event has been canceled. You can use a flag in event table to indicate that the event has been canceled (so you don't need to delete the event and registration records).

Input: evnt_id
Output: generate a message for each person who has registered or on wait list saying that the event has been canceled. Also insert the message in message table.

Exception:

- If no events are registered for event: print "*Event not exists"*.

Create Procedure Statements (with comments):

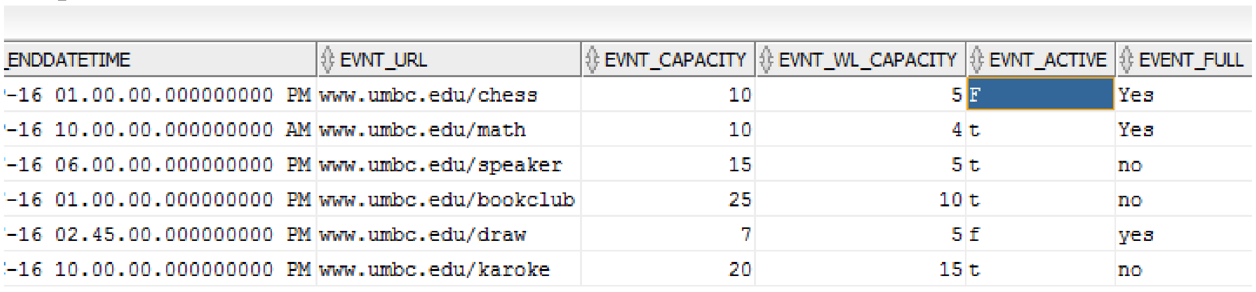
```
set serveroutput on;
create or replace procedure CancelEvent(event_id in number) IS
cursor c1 is select user_id from evntreg where evnt_id = event_id; -- to fetch regitered users from
eventreg table
cursor c2 is select user_id from waitlist where evnt_id = event_id; -- to fetch users on waitlist
from waitlist table
eventid event.evnt_id%type;
eventname event.evnt_title%type;
userid users.user_id%type;
msgs message.msg%type;
u_id users.user_id%type;
Begin

Select event_id, evnt_title into eventid, eventname from event where evnt_id = event_id; -- To
check event is exist in event table.
If eventid is not null then
        update event set evnt_active = 'F' where evnt_id = event_id; -- if event is cancelled then
flag event_active will be set as F(false) to indicate event is cancelled

-- generating message for users who are registered for that event.
  open c1;
        loop
                fetch c1 into userid;
                exit when c1%NOTFOUND;
                msgs := 'The Event ' || eventname || ' has been canceled ';
                insert into message(msg_id,user_id,msg_time,msg) values(msg_seq.nextval,
userid, systimestamp, msgs);
 END LOOP;
  Close c1;

-- generating message for users who are on waitlist.
  open c2;
```

```
            loop
                        fetch c2 into u_id;
                        exit when c2%NOTFOUND;
                        msgs := 'The Event ' || eventname || ' has been canceled ';
                        insert into message(msg_id,user_id,msg_time,msg) values(msg_seq.nextval, u_id,
systimestamp, msgs);
  END LOOP;
   Close c2;
else
   dbms_output.put_line('no such event');
end if;

exception
when no_data_found then
dbms_output.put_line('Event does not exist');
end;
```

Execute statement:

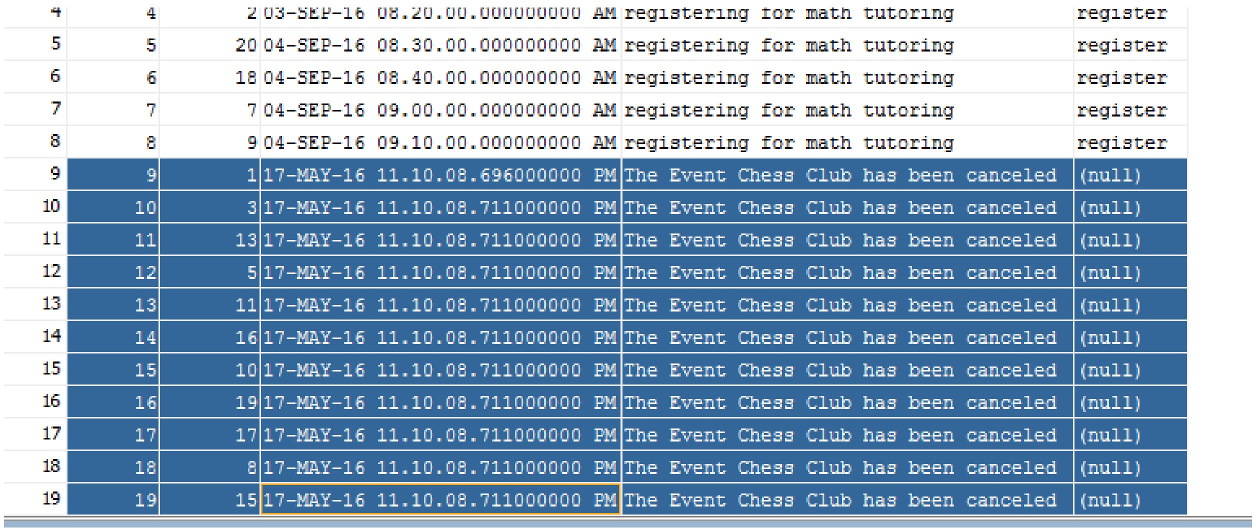
```
exec CancelEvent(1); -- Event exist
exec CancelEvent(10); -- Event not exist
```

Output:



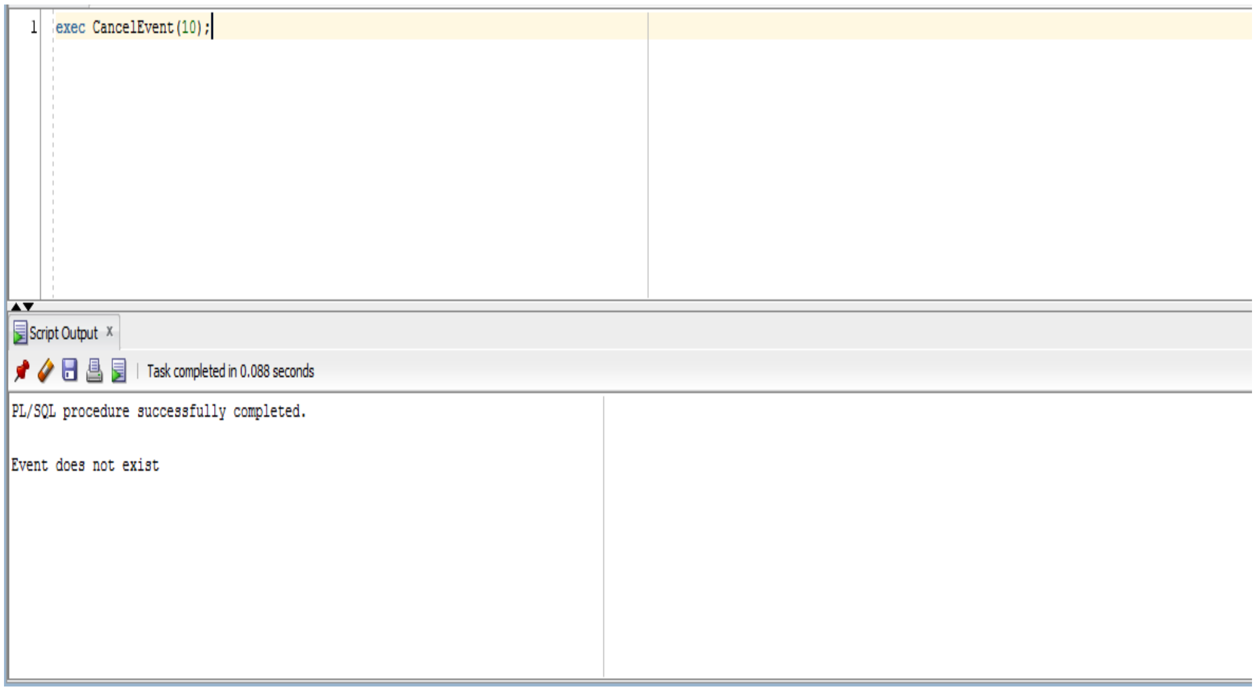
ENDDATETIME	EVNT_URL	EVNT_CAPACITY	EVNT_WL_CAPACITY	EVNT_ACTIVE	EVENT_FULL
-16 01.00.00.000000000 PM	www.umbc.edu/chess	10	5	F	Yes
-16 10.00.00.000000000 AM	www.umbc.edu/math	10	4	t	Yes
-16 06.00.00.000000000 PM	www.umbc.edu/speaker	15	5	t	no
-16 01.00.00.000000000 PM	www.umbc.edu/bookclub	25	10	t	no
-16 02.45.00.000000000 PM	www.umbc.edu/draw	7	5	f	yes
-16 10.00.00.000000000 PM	www.umbc.edu/karoke	20	15	t	no

Changes in the event table



```
 4   4   2 03-SEP-16 08.20.00.000000000 AM registering for math tutoring       register
 5   5  20 04-SEP-16 08.30.00.000000000 AM registering for math tutoring       register
 6   6  18 04-SEP-16 08.40.00.000000000 AM registering for math tutoring       register
 7   7   7 04-SEP-16 09.00.00.000000000 AM registering for math tutoring       register
 8   8   9 04-SEP-16 09.10.00.000000000 AM registering for math tutoring       register
 9   9   1 17-MAY-16 11.10.08.696000000 PM The Event Chess Club has been canceled (null)
10  10   3 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
11  11  13 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
12  12   5 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
13  13  11 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
14  14  16 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
15  15  10 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
16  16  19 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
17  17  17 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
18  18   8 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
19  19  15 17-MAY-16 11.10.08.711000000 PM The Event Chess Club has been canceled (null)
```

Message generated for user in the table



```
exec CancelEvent(10);
```

```
PL/SQL procedure successfully completed.

Event does not exist
```

Explanation of Output:

exec CancelEvent(1); -- This is the standard output when the event exists and is cancelled and generates a message for user in the message table.

CancelEvent(10); -- This is the standard output when the event does not exist

Feature 9:

Update the start and end date and time of an event by providing event ID and new start/end date and time. Update the event table if the event exists. If the event does not exist, print out a message saying wrong event ID. Please generate a message for each user who has registered or on wait list with the new date and time of the event.

Input: evnt_id, evnt_strdatetime, evnt_enddatettime
Output: message sent to users with the new evnt_strdatetime & new evnt_enddatettime

Exceptions:

- If evnt_id does not exist: print *"Wrong event id"*
- If input is null: print *null input, try again*

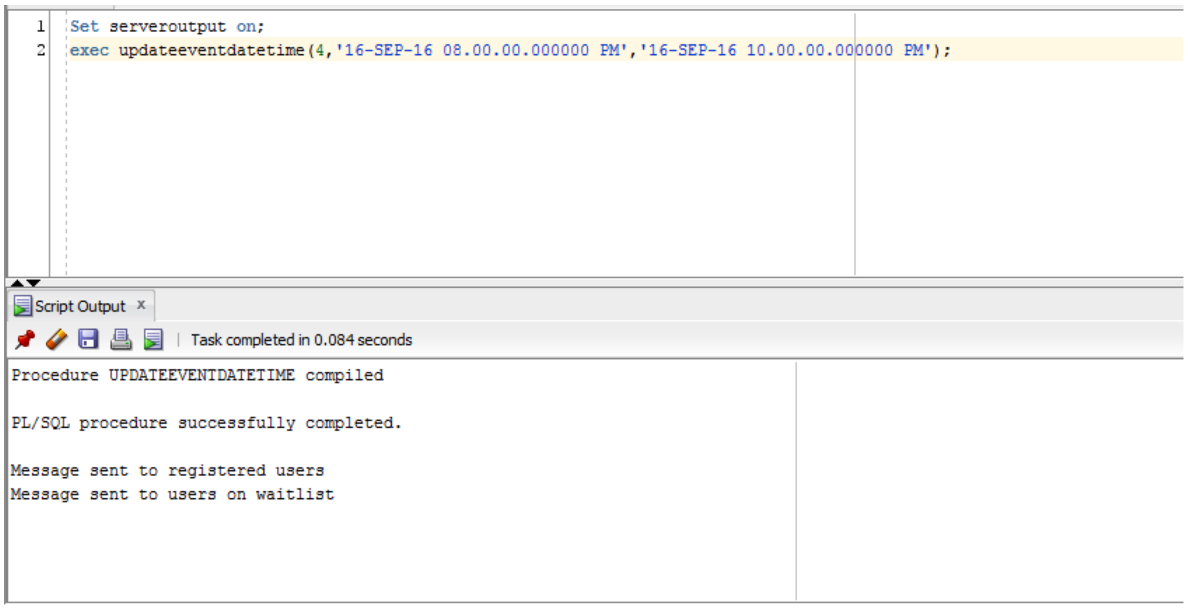
```
create or replace procedure Updateeventdatetime(event_id in number, strdatetime in timestamp,
enddatetime in timestamp) IS
cursor c1 is select user_id from evntreg where evnt_id = event_id;
 -- to fetch registered user id from eventreg table
cursor c2 is select user_id from waitlist where evnt_id = event_id;
eventid event.evnt_id%type;
eventname event.evnt_title%type;
userid users.user_id%type;
msgs message.msg%type;
u_id users.user_id%type;
Begin

Select event_id, evnt_title into eventid, eventname from event where evnt_id = event_id;
-- To check event is exist
If eventid is not null then
        Update event set evnt_strdatetime = strdatetime, evnt_enddatetime = enddatetime where
evnt_id = event_id; -- updating the start and end date and time to the new date and time
   open c1;
        Loop --  loop to update the message table for all the users who are registered about the
date and time changes.

                fetch c1 into userid;
                exit when c1%NOTFOUND;
                msgs := eventname||' event is now at: Start: '||strdatetime||' End: '||enddatetime;
                Insert into message(msg_id,user_id,msg_time,msg,msg_type)
values(msg_seq.nextval, userid, systimestamp, msgs,'Update');
  END LOOP;
  dbms_output.put_line('Message sent to registered users');
  Close c1;
```

```
   open c2;
            Loop
-- loop to update the message table for all the users who are on waitlist about the date and time
                        fetch c2 into u_id;
                        exit when c2%NOTFOUND;
                        msgs := eventname||' event is now at: Start: '||strdatetime||' End: '||enddatetime;
                        insert into message(msg_id,user_id,msg_time,msg,msg_type)
values(msg_seq.nextval, u_id, systimestamp, msgs,'Update');
  END LOOP;
  dbms_output.put_line('Message sent to users on waitlist');
   Close c2;
end if;
exception
when no_data_found then
   Dbms_output.put_line ('Wrong Event ID');
end;
```

Execute statement:

```
Set serveroutput on;
exec updateeventdatetime(4,'16-SEP-16 08.00.00.000000 PM','16-SEP-16 10.00.00.000000
PM'); -- Event exists

Set serveroutput on;
exec updateeventdatetime(40,'16-SEP-16 08.00.00.000000 PM','16-SEP-16 10.00.00.000000
PM'); -- Event does not exist
```

Output:


```
Set serveroutput on;
exec updateeventdatetime(4,'16-SEP-16 08.00.00.000000 PM','16-SEP-16 10.00.00.000000 PM');
```

```
Procedure UPDATEEVENTDATETIME compiled

PL/SQL procedure successfully completed.

Message sent to registered users
Message sent to users on waitlist
```
Explanation of Output:

For the given event id - if the event id exists the new start date/time and end date/time are updated. If not, the exception is handled and displays that Wrong Event ID.

Feature 10:

Search for events with a certain keyword in the title (you can use like), return start and end date and time, full event title, location, url, organizer name and email, and whether the event is full.

Input: keyword

Output: evnt_title, evnt_strdatetime, evnt_enddatettime, loc_name,evnt_url, orgn_name, user_email (organizer email), message whether the event is full. (will be multiple rows if more the keyword appears in more than 1 registered event)

Exceptions:

- If evnt_id does not exist: print *Event ID does not exist*
- If input is null: print *null input, try again*
- If nobody registered for event: print *No users have registered for this event*

Create Procedure Statements (with comments):

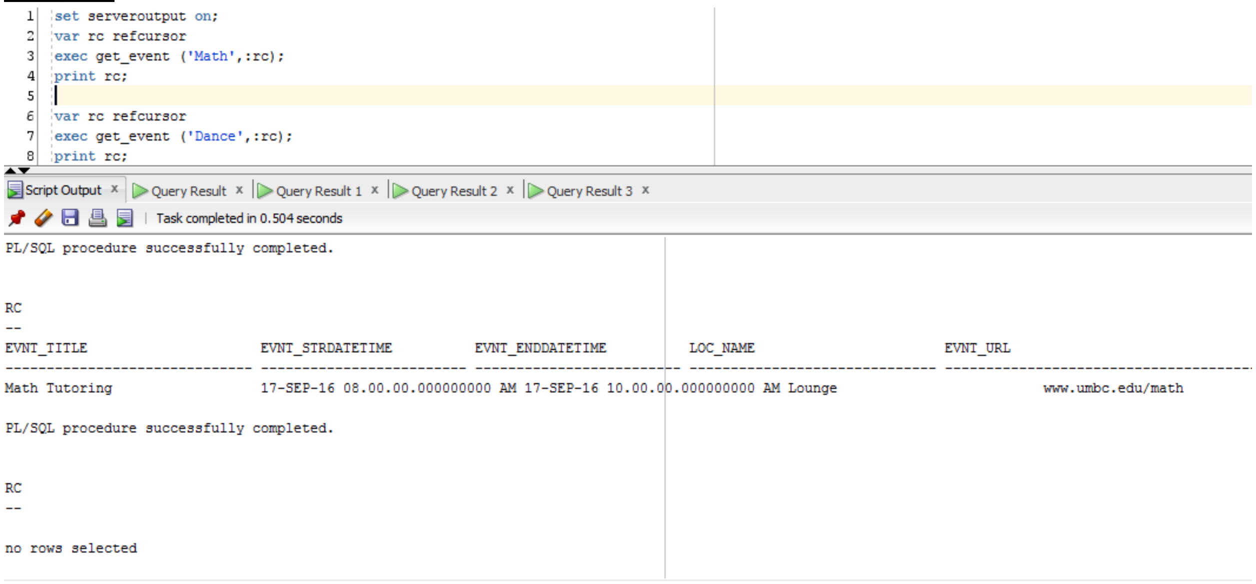
```
CREATE OR REPLACE PROCEDURE get_event(keyword in VARCHAR2, c1 IN OUT
sys_refcursor) AS
BEGIN
open c1 for
-- cursor to find events with the keyword
   Select evnt_title, evnt_strdatetime, evnt_enddatetime, loc_name, evnt_url, orgn_name,
user_email, event_full from event e, organizers o, locations l where evnt_title like '%' || keyword
```

```
|| '%' and e.orgn_id = o.orgn_id and e.loc_id = l.loc_id;
   exception
   -- if no data is found
  when no_data_found then
            dbms_output.put_line('No search results for the keyword');
END;
```

Execute statement:

```
Set serveroutput on;
var rc refcursor
exec get_event ('Math',:rc); -- valid keyword search
print rc;

Set serveroutput on;
var rc refcursor
exec get_event ('Dance',:rc); -- No data for keyword search
print rc;
```

Output:



```
set serveroutput on;
var rc refcursor
exec get_event ('Math',:rc);
print rc;

var rc refcursor
exec get_event ('Dance',:rc);
print rc;
```

```
PL/SQL procedure successfully completed.


RC
--
EVNT_TITLE                     EVNT_STRDATETIME          EVNT_ENDDATETIME          LOC_NAME                       EVNT_URL
------------------------------ ------------------------- ------------------------- ------------------------------ ----------------------------------------
Math Tutoring                  17-SEP-16 08.00.00.000000000 AM 17-SEP-16 10.00.00.000000000 AM Lounge                         www.umbc.edu/math

PL/SQL procedure successfully completed.


RC
--

no rows selected
```

Explanation of Output:

For the given keyword the procedure brings back all the events details for the event title having the keyword in it.

If the keyword does not exist in any of the title no rows will be selected.

Feature 11:

Register for an event by providing user ID and event ID. First check if the event exists. If not print out a message saying wrong event ID. Then check whether the event is full. ***If so, put the user on the wait list with the next position and print out a message saying you are on waitlist and the waitlist position****. Otherwise register the user with the event and print a message you are registered.*

Input: user_id, evnt_id

Output: If event is full, print '*Event is full, you are on waitlist position xyz*' else print '*You are registered for an event xyz*'.

Exception:

- If no events are registered for event: print "*Event not exists".*

Create Procedure Statements (with comments):

```
set serveroutput on;
create or replace procedure EventRegistration( u_id in number, ev_id in number) IS
eventExist number;
Current_capacity number;
EventCapacity number;
WL_Capacity number;
Current_WL_capacity number;
new_wl_capacity number;
eventactive event.evnt_active%type;
userExist number;
wluserExist number;
Begin
   SELECT evnt_id, UPPER(Evnt_Active)   -- To check event is exist or not.
   INTO eventExist, eventactive
   FROM EVENT
   WHERE evnt_id = ev_id;

      if (eventExist != 0 and eventactive = 'T' ) then  -- If event id is not null and eventactive is
true then event is exist.
         dbms_output.put_line('Event is exist');

                  select count(user_id) into userExist from evntreg where  evnt_id = ev_id and
user_id = u_id; -- To check whether user is already exist on event registration table
```

```
    select count(user_id) into wluserExist from waitlist where  evnt_id = ev_id and user_id =
u_id; -- To check whether user is already exist on waitlist table

                        if (userExist != 0 or wluserExist != 0) then
                                    dbms_output.put_line('User ' ||u_id||': You are already registerd/on waitlist
for this event');

                        else

                        Select evnt_capacity into EventCapacity FROM event where evnt_id = ev_id; --
To find event capacity of particular event.
                                    Select COUNT(evnt_id) into Current_capacity FROM evntreg where
evnt_id = ev_id; -- To find how many users registered for that event.

                                    if (Current_capacity >= EventCapacity) then      -- If the current capacity
is equal to event capacity then event is full and it will check waitlist is full or not.

                                    UPDATE event set event_full = 'Yes' where evnt_id = ev_id;

                                                Select e.evnt_wl_capacity into WL_Capacity from event e where
evnt_id = ev_id; -- To find waitlist capacity of particular event.
                                                Select COUNT(wl_position) into Current_WL_capacity FROM
waitlist where evnt_id = ev_id; -- To find how many users are on waitlist for that event.

                                                if(Current_WL_capacity >= WL_Capacity) then     -- -- If the
current waitlist capacity is equal to waitlist capacity then waitlist of that event is full. If waitlist is
not full then user is added to the waitlist at next position.
                                                            dbms_output.put_line('Waitlist is full');
                                                else
                                                            new_wl_capacity := Current_WL_capacity + 1;
                                                            insert into waitlist(evnt_id, wl_position, user_id) values
(ev_id, new_wl_capacity, u_id);
                                                            dbms_output.put_line('You are on waitlist position ' ||
new_wl_capacity);
                                                end if;
                                    else
                        INSERT INTO evntreg(evnt_id, user_id, reg_time) VALUES (ev_id,
u_id, systimestamp);  -- If event registration is not full. User will be registered for that event.
                        dbms_output.put_line('User ' ||u_id||': You are registerd for this event');
                        end if;
                end if;

    else
        dbms_output.put_line('Event is not exist');
    end if;

exception
when no_data_found then
dbms_output.put_line('Wrong Event Id or User Id.');
end;
```

Execute Statement output:

```
set serveroutput on;
exec EventRegistration(8, 1); -- when event is not full
exec EventRegistration(11, 1); -- when user is already registered
exec EventRegistration(15, 1); -- Event is full, go to the waitlist (waitlist is not full)
exec EventRegistration(6, 2); -- when waitlist is full
exec EventRegistration(6, 8); -- when event is not exist or cancelled
```

Output:

```
PL/SQL procedure successfully completed.
Event is exist
User 8: You are registerd for this event

PL/SQL procedure successfully completed.
Event is exist
User 11: You are already registerd/on waitlist for this event

PL/SQL procedure successfully completed.
Event is exist
You are on waitlist position 1

PL/SQL procedure successfully completed.
```

```
Event is exist
Waitlist is full

PL/SQL procedure successfully completed.
Wrong Event Id or User Id.
```

Explanation of Output:

In first statement, when the event is exist and user is not already registered, procedure will register the user into that event.

In second statement, if the user is already registered or on waitlist then procedure will not allow the user to register again for that event.

In third statement, It check first whether the event is full or not. If event it full then it will put the user on waitlist number 1 and so on.

In the fourth statement, procedure will check whether the waitlist is full or not. If the waitlist is full. It will give output as 'waitlist is full' and will not allow user to register for the event.

In fifth statement, If the event is not exist then it will give output as 'Wrong Event Id or User Id'.

Feature 12:

Cancel an event registration by providing user ID and event ID. Please check whether the event exists and the user has indeed registered. If the user is registered, delete the registration and register the first person on the waitlist (if the wait list is not empty). Generate a message (insert into message table) for the current user saying event canceled. If a user on top of wait list is now registered, generate a message to that user saying that he or she is now registered with the event.

Input: user_id, evnt_id

Output: *If a user on top of wait list is now registered, print 'you are now registered with the event xyz'.*

Parameter Example: exec isEventExist(3);, cancel_evntreg(2,3);, Check isWaitlistEmpty();, if waitlist is not empty, register first person to the event.

Exception:

- If no events are registered for event: print "*Event not exists"*.
- If waitlist doesn't have entries: print 'Wait list is empty'.

Create Procedure Statements (with comments):

```
set serveroutput on;
Create or replace procedure CancelRegistration(u_id in number, ev_id in number) IS
Cursor c1 is select wl_position, user_id from waitlist where evnt_id = ev_id;
```

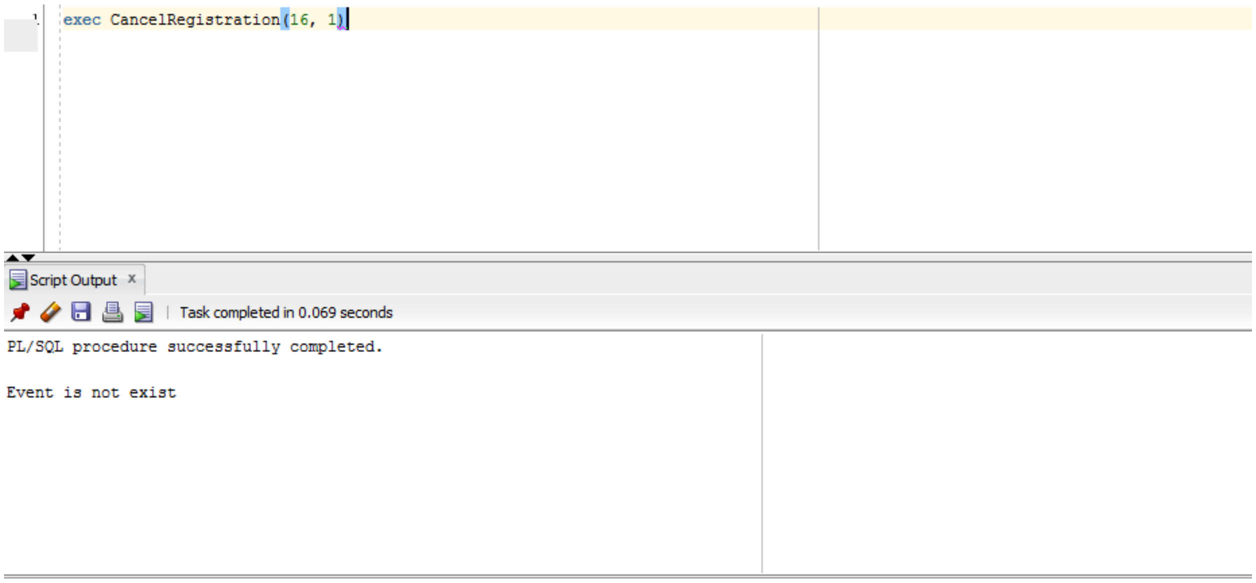
```
eventExist event.evnt_id%type;
eventactive event.evnt_active%type;
userexist number;
EventName event.evnt_title%type;
msgs message.msg%type;
wl_user number;
uid number;
messages message.msg%type;
waitlist_position waitlist.wl_position%type;

Begin
        SELECT e.evnt_id,e.evnt_title, upper(e.evnt_active), er.user_id INTO eventExist,
EventName, eventactive, userexist FROM event e, evntreg er WHERE e.evnt_id = ev_id and
er.evnt_id = e.evnt_id and er.user_id = u_id;
        if (eventExist = ev_id and userexist = u_id and eventactive = 'T') then -- checking event is
exist and user is registered for the event.
        dbms_output.put_line('Event is exist and user is registered.');

                -- Deleting the record with given event Id and User ID and inserting a message in
message table saying event registration has been cancelled.
                delete from evntreg where evnt_id = ev_id and user_id = u_id;
                msgs := 'Event '|| EventName ||' registration for User '|| u_id ||' has been canceled.';
                insert into message(msg_id,user_id,msg_time,msg) values(msg_seq.nextval, u_id,
systimestamp, msgs);

                Select user_id into uid from waitlist where evnt_id = ev_id and wl_position = 1;  -
- selecting user from waitlist with waitlist position 1.

                if uid is not null then

                        -- if waitlist is not null then it will register user with waitlist position 1 into
event registration table. Also, user with waitlist position 1 from waitlist table.
                        INSERT INTO evntreg(evnt_id, user_id, reg_time) VALUES (ev_id, uid,
systimestamp);
                        messages := 'You are registered for event '|| EventName;
                        insert into message(msg_id,user_id,msg_time,msg)
values(msg_seq.nextval, uid, systimestamp, messages);
                        delete from waitlist where evnt_id = ev_id and user_id = uid;

                                -- updating waitlist position in waitlist position from k to k-1.
```

```
                                        open c1;
                                         loop
                                                                fetch c1 into waitlist_position, wl_user;
                                                                exit when c1%NOTFOUND;
                                                                waitlist_position := waitlist_position - 1;
                                                                update waitlist set wl_position = waitlist_position
where evnt_id = ev_id and user_id = wl_user;
                                        END LOOP;
                                        Close c1;
                    else
                              dbms_output.put_line('Waitlist is empty');
                    end if;
          else
          dbms_output.put_line('Event is not exist');
          end if;

exception
when no_data_found then
dbms_output.put_line('Wrong User Id or Event Id ');
end;
```

Execution Statements:

```
exec CancelRegistration(16, 1); -- When waitlist is not empty
exec CancelRegistration(4, 2); -- When waitlist is not empty
exec CancelRegistration(10, 1);-- when waitlist is empty
exec CancelRegistration(10, 4); -- when event is not active
```

Output:



```
exec CancelRegistration(16, 1)
```

```
PL/SQL procedure successfully completed.

Event is not exist
```

Explanation of Output:

exec CancelRegistration(16, 1); -- It is the standard output when the waitlist is not empty and event does not exist
exec CancelRegistration(4, 2); -- This is the output when waitlist is not empty and the user registered for the event has his event cancelled.
exec CancelRegistration(10, 1);-- It is the output when waitlist is empty and users registered for the event has his event cancelled.
exec CancelRegistration(10, 4); -- It is the standard output when the event is no more active

Feature 13:

Enter a review for an event. The user provides a numerical rating (1 to 5) as well as some comment. To prevent abuse, the feature needs to check that the user has registered for the event and only registered users can enter reviews.

Input: evnt_id, user_id, rating (1-5), msg_type, message
Output: updated values for event rating and review comment for the event.

Create Procedure Statements (with comments):

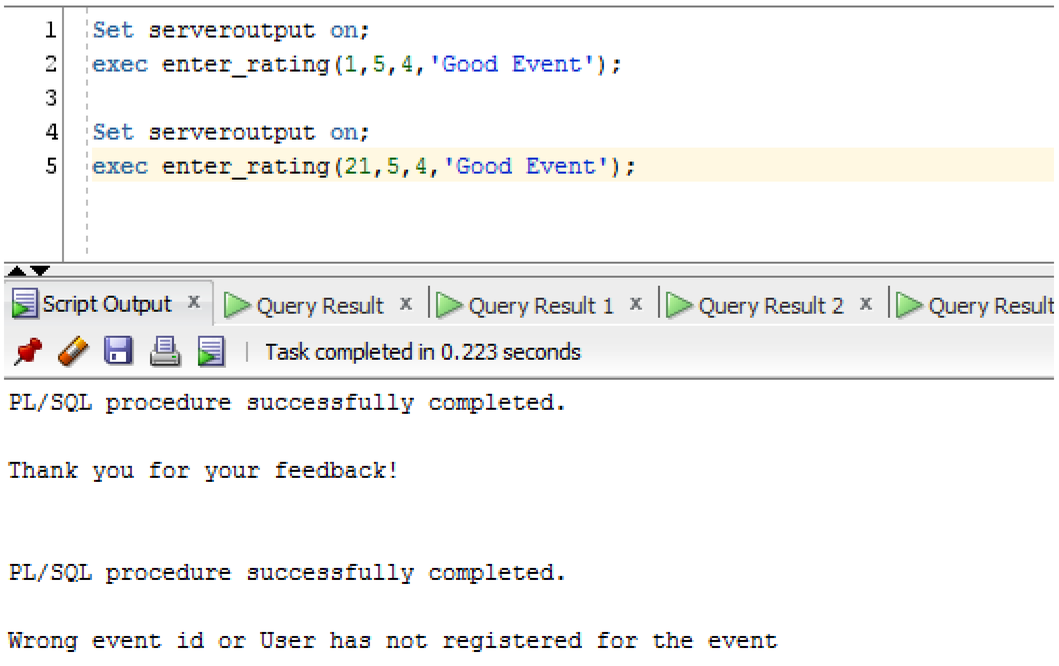
```
create or replace procedure enter_rating(event_id in number, u_id in number, rate in number,
cmmt in varchar
) IS
    eventid number;
    usr_id number;
```

```
      Begin
    select user_id into usr_id from evntreg where evnt_id = event_id and user_id = u_id; --
checking if user id exists
    select event_id into eventid from event where evnt_id = event_id;
-- check to see if event is registered

      if eventid is not null then
                     If (usr_id = u_id) then
                           Update evntreg set rating = rate, commnt = cmmt where evnt_id = event_id and
user_id = u_id; -- update rating for event
                           dbms_output.put_line('Thank you for your feedback!');
                           end if;
         end if;
     exception
     when no_data_found then
     dbms_output.put_line('Wrong event id or User has not registered for the event');
     end;
```

Execute Statement:

```
Set serveroutput on;
exec enter_rating(1,5,4,'Good Event'); -- valid data

Set serveroutput on;
exec enter_rating(21,5,4,'Good Event'); -- exception for wrong event id
```

Output:



```
Set serveroutput on;
exec enter_rating(1,5,4,'Good Event');

Set serveroutput on;
exec enter_rating(21,5,4,'Good Event');
```

Task completed in 0.223 seconds

```
PL/SQL procedure successfully completed.

Thank you for your feedback!


PL/SQL procedure successfully completed.

Wrong event id or User has not registered for the event
```

Explanation of Output:

If the user is a registered user for the event and if the event exists the user is allowed to update a rating and comments for the event.

If the user is not registered or the event id entered is not valid then the error message "Wrong event id or User has not registered for the event" is displayed.

Feature 14:

Print out the total number of events, the average number of users registered per event, and the average number of users on wait list.

Input: N/A

Output: total number of events, the average number of users registered per event, and the average number of users on wait list.

Create Procedure Statements (with comments):

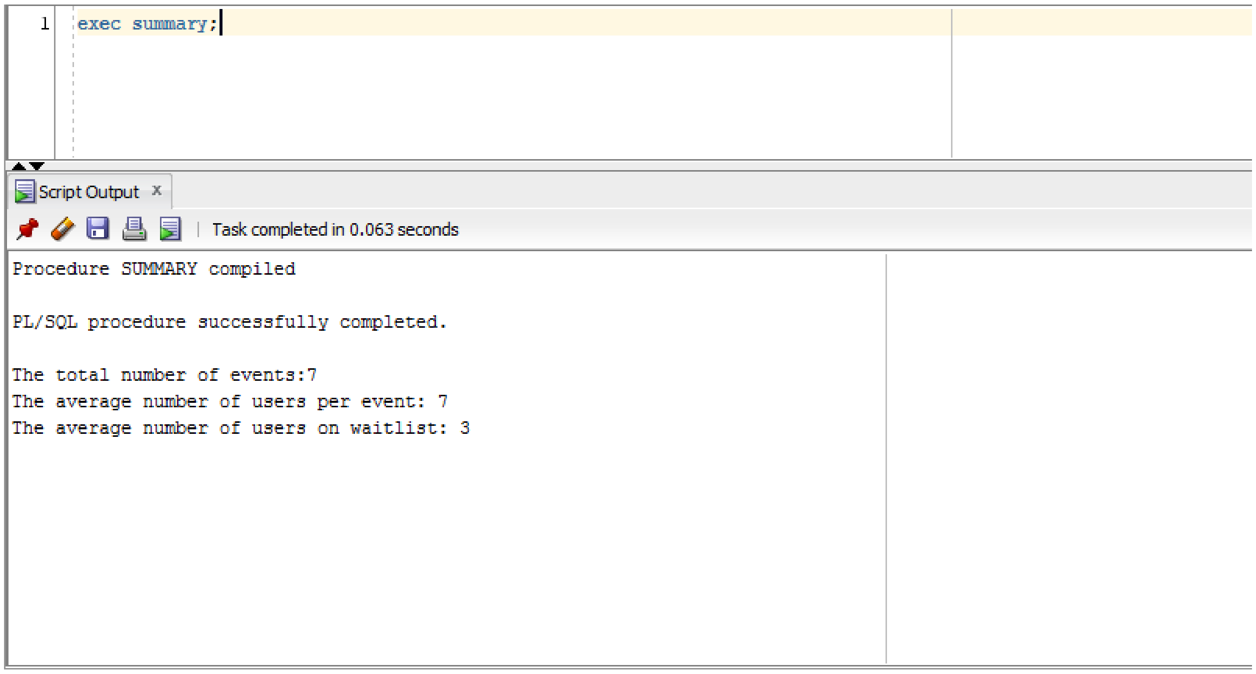
```
CREATE OR REPLACE PROCEDURE Summary AS
  CURSOR c1 IS SELECT count(evnt_id) FROM event; -- cursor to count total events
  total_evnts number;
  CURSOR c2 IS select count(user_id)/count(distinct evnt_id) from evntreg;
  avg_reg integer; -- count average users registered for events
  CURSOR c3 IS select count(user_id)/count(distinct evnt_id) from waitlist;
  avg_wl integer; -- counts average users on waitlist
```

```
BEGIN
  OPEN c1;
  fetch c1 into total_evnts;
  dbms_output.put_line('The total number of events:' || total_evnts);
  OPEN c2;
  FETCH c2 INTO avg_reg;
  dbms_output.put_line('The average number of users per event: ' || avg_reg);
  OPEN c3;
  FETCH c3 INTO avg_wl;
  DBMS_OUTPUT.PUT_LINE('The average number of users on waitlist: ' || avg_wl);
  CLOSE c3;
  CLOSE c2;
  Close c1;
END;
```

Execute Statement:

```
Set serveroutput on;
exec Summary;
```

Output:



```
exec summary;
```

Script Output | Task completed in 0.063 seconds

```
Procedure SUMMARY compiled

PL/SQL procedure successfully completed.

The total number of events:7
The average number of users per event: 7
The average number of users on waitlist: 3
```

Explanation of Output:

The procedure gives a count of the total number of events.

The average number of events registered per event and the average number of users on waitlist per event.

Feature 15:

Print out the top K events with the most participants (only counting those registered), the top K locations with most number of events, and top K events with the highest average ratings. K is an input parameter.

Input: K.
Output: top K events with the most participants, the top K locations with most number of events, and top K events with the highest average ratings.
Parameter Example: exec eventsummary(K); where K is input value.

Create Procedure Statements (with comments):

```
set serveroutput on;
Create or replace
PROCEDURE get_report (k in number) IS
Cursor c1 is select evnt_id, count(user_id) from evntreg group by evnt_id order by
count(user_id) desc ; -- To find top K events with the most participants
Cursor c2 is select loc_id, count(evnt_id) from event group by loc_id order by count(evnt_id)
desc; -- To find he top K locations with most number of events
Cursor c3 is select evnt_id, avg(rating) from evntreg group by evnt_id order by avg(rating) desc;
-- To find top K events with the highest average ratings
Event_with_most_participant number;
number_of_participants number;
Locations_with_most_event number;
number_of_events number;
highest_avgrating_events number;
average_rating number;
BEGIN
Open c1;
-- Here we are fetching events with most participants and number of participants registered in
variables. Loop will be repeted for k times.
Loop
        fetch c1 into Event_with_most_participant, number_of_participants;
        exit when c1%notfound or c1%rowcount > k;
        dbms_output.put_line('Events with most participant is ' || Event_with_most_participant);
End loop;
Open c2;
-- Here we are fetching locations with most events and number of locations in variables. Loop
will be repeted for k times.
Loop
        fetch c2 into Locations_with_most_event, number_of_events;
        exit when c2%notfound or c2%rowcount > k;
        dbms_output.put_line('Locations with most number of events ' ||
Locations_with_most_event);
End loop;

Open c3;
-- Here we are fetching events with highest average rating and average rating in variables. Loop
will be repeted for k times.
Loop
        fetch c3 into highest_avgrating_events, average_rating;
        exit when c3%notfound or c3%rowcount > k;
        dbms_output.put_line('Events with the highest average ratings ' ||
highest_avgrating_events);
End loop;
close c1;
close c2;
close c3;

END;
```

Execute Statement:

```
set serveroutput on;
exec get_report(2); -- Top 2 records
exec get_report(3); -- Top 3 records
exec get_report(4); -- Top 4 records
```

Output:

```
PL/SQL procedure successfully completed.

Events with most participant is 1
Events with most participant is 2
Locations with most number of events 7
Locations with most number of events 6
Events with the highest average ratings 5
Events with the highest average ratings 6
```

(Explanation: Statement 1 gives top two records.)

```
PL/SQL procedure successfully completed.

Events with most participant is 1
Events with most participant is 2
Events with most participant is 5
Locations with most number of events 7
Locations with most number of events 6
Locations with most number of events 3
Events with the highest average ratings 5
Events with the highest average ratings 6
Events with the highest average ratings 1
```
(Explanation: Statement 2 gives top three records.)

```
PL/SQL procedure successfully completed.
Events with most participant is 1
Events with most participant is 2
Events with most participant is 5
Events with most participant is 3
Locations with most number of events 7
Locations with most number of events 6
Locations with most number of events 3
Locations with most number of events 1
Events with the highest average ratings 5
Events with the highest average ratings 6
Events with the highest average ratings 1
Events with the highest average ratings 2
```
(Explanation: Statement 3 gives top four records.)

DEMO SCRIPT

```
--Alan Blitstein
--Krutika Karveershettar
--Pooja Gond
--Rashmi Fegade
--Deliverable 4 Final Script Code
--GROUP 7, SPRING 2016, UMBC IS620

--DROP TABLES
drop table waitlist;
drop table evntreg;
drop table message;
drop table event;
drop table organizers;
drop table locations;
drop table users;

--CREATE TABLES
```

```
create table users(
      user_id int,
      user_name varchar(30),
      user_phone varchar(15),
      user_email varchar(30),
      user_password varchar(30),
      user_type varchar(15),
      primary key (user_id)
      );

create table locations(
      loc_id int, -- location id
      loc_name varchar(30), -- location name
      loc_descr varchar(100), -- location description
      loc_capacity int, -- location capacity
      primary key (loc_id)
      );

create table organizers(
      orgn_id int, -- organizer id
      user_id int,
      orgn_name varchar(30), -- organizer name
            user_email varchar(30), -- organizer email id
      primary key(orgn_id),
      foreign key(user_id) references users(user_id)
      );

create table event(
      evnt_id int,
      loc_id int,
      orgn_id int, -- organizer id
      evnt_title varchar(30), -- event title
      evnt_descr varchar(100), -- event description
      evnt_strdatetime timestamp, -- event start date and time
      evnt_enddatetime timestamp, -- event end date and time
      evnt_url varchar(100), --event URL
      evnt_capacity int, -- event capacity
      evnt_wl_capacity int, -- event waitlist capacity
      evnt_active char, -- event status active or not
      event_full varchar(5), ---to check event is full or not
      primary key(evnt_id),
      foreign key(loc_id) references locations(loc_id),
      foreign key(orgn_id) references organizers(orgn_id)
      );

create table message(
      msg_id int, -- message id
      user_id int,
      msg_time timestamp, -- message time
      msg varchar(350), -- message body
      msg_type varchar(15), -- message type register or unregister
      primary key(msg_id),
      foreign key(user_id) references users(user_id)
      );
```

```
create table evntreg(
      evnt_id int,
      user_id int,
      reg_time timestamp, -- registeration date and time
      rating int, -- event rating
      commnt varchar(100), -- user comment
      foreign key(evnt_id) references event(evnt_id),
      foreign key(user_id) references users(user_id)
      );

create table waitlist(
      evnt_id int,
      wl_position int, -- waitlist position of the user
      user_id int,
      foreign key(evnt_id) references event(evnt_id),
      foreign key(user_id) references users(user_id)
      );


--INSERT STATEMENTS

insert into users
values (1,'Alan','4101233211','alan1@umbc.edu','alanpassword','student');
insert into users
values
(2,'Pooja','4104258219','pooja1@umbc.edu','poojapassword','student');
insert into users
values
(3,'Krutika','4107254322','krutika1@umbc.edu','krutikapassword','student')
;
insert into users
values
(4,'Rashmi','4102876211','rashmi1@umbc.edu','rashmipassword','student');
insert into users
values
(5,'Dr.Chen','4105285642','drchen1@umbc.edu','chenpassword','faculty');
insert into users
values (6,'Larry','4109993201','larry1@umbc.edu','larrypassword','staff');
insert into users
values (7,'Emily','4103323202','emily2@umbc.edu','emilypassword','staff');
insert into users
values
(8,'Dr.Smith','4100983243','drsmith9@umbc.edu','smithpassword','faculty');
insert into users
values (9,'Mark','4108983218','mark1@umbc.edu','markpassword','student');
insert into users
values
(10,'Dr.Jack','4103569087','drjack3@umbc.edu','jackpassword','faculty');
insert into users
values
(11,'Grace','4100895101','grace3@umbc.edu','gracepassword','staff');
insert into users
values (12,'Mike','4105643201','mike5@umbc.edu','mikepassword','student');
```

```
insert into users
values (13,'Greg','4101292281','greg1@umbc.edu','jackpassword','student');
insert into users
values
(14,'Jackie','4107743201','jackie1@umbc.edu','jackiepassword','student');
insert into users
values
(15,'Mr.Garrison','4103276543','garrison5@umbc.edu','garrisonpassword','st
aff');
insert into users
values
(16,'Dr.Jones','4103233201','drjones4@umbc.edu','cohenpassword','faculty')
;
insert into users
values
(17,'Dr.Engle','4100090078','drengle3@umbc.edu','englepassword','faculty')
;
insert into users
values
(18,'Harry','4105553435','harry7@umbc.edu','harrypassword','student');
insert into users
values (19,'Owen','4101233212','owen1@umbc.edu','owenpassword','student');
insert into users
values
(20,'Barrry','4103651045','barry2@umbc.edu','barrypassword','student');


insert into locations
values (1, 'Sherman Hall', 'SH102', 10);
insert into locations
values (2, 'Library', 'LIB401', 15);
insert into locations
values (3, 'Fine Arts Bldg,', 'FA202', 8);
insert into locations
values (4, 'Sondheim Hall', 'SO201', 15);
insert into locations
values (5, 'IT/Eng Bldg.', 'ITE401', 12);
insert into locations
values (6, 'Math/Psych Bldg.', 'MP101', 20);
insert into locations
values (7, 'Lounge', 'LOU111', 6);

insert into organizers
values (1, 4, 'Rashmi','rashmi1@umbc.edu');
insert into organizers
values (2, 5, 'Dr.Chen','drchen1@umbc.edu');
insert into organizers
values (3, 7, 'Emily', 'emily2@umbc.edu');
insert into organizers
values (4, 9, 'Mark', 'mark1@umbc.edu');
insert into organizers
values (5, 10, 'Dr.Cohen','drcohen1@umbc.edu');

insert into event
```

```
values (1, 1, 1,'Chess Club', 'Weekly Chess Meeting', timestamp '2016-9-15
12:00:00.00', timestamp '2016-9-15 13:00:00.00', 'www.umbc.edu/chess', 10,
5, 't','no');
insert into event
values (2, 7, 5,'Math Tutoring', 'Sign up for Math Help!', timestamp
'2016-9-17 8:00:00.00', timestamp '2016-9-17 10:00:00.00',
'www.umbc.edu/math', 10, 4, 't','yes');
insert into event
values (3, 6, 3,'Guest Speaker', 'Come listen to President Obama',
timestamp '2016-10-01 17:00:00.00', timestamp '2016-10-01 18:00:00.00',
'www.umbc.edu/speaker', 15, 5, 't','no');
insert into event
values (4, 2, 2,'Book Club', 'Weekly Book Club Talk', timestamp '2016-10-
07 12:00:00.00', timestamp '2016-10-07 13:00:00.00',
'www.umbc.edu/bookclub', 25, 10, 't','no');
insert into event
values (5, 3, 4,'Drawing', 'Lets make art', timestamp '2016-10-09
14:00:00.00', timestamp '2016-10-09 14:45:00.00', 'www.umbc.edu/draw', 7,
5, 'f', 'yes');
insert into event
values (6, 7, 5,'Karoke', 'Welcome to the fun night', timestamp '2016-12-
03 20:00:00.00', timestamp '2016-12-03 22:00:00.00',
'www.umbc.edu/karoke', 20, 15, 't', 'no');



insert into message
values (1, 3, timestamp '2016-9-02 8:00:00.00', 'registering for math
tutoring', 'register');
insert into message
values (2, 1, timestamp '2016-9-02 8:10:00.00', 'registering for math
tutoring', 'register');
insert into message
values (3, 4, timestamp '2016-9-03 8:15:00.00', 'registering for math
tutoring', 'register');
insert into message
values (4, 2, timestamp '2016-9-03 8:20:00.00', 'registering for math
tutoring', 'register');
insert into message
values (5, 20, timestamp '2016-9-04 8:30:00.00', 'registering for math
tutoring', 'register');
insert into message
values (6, 18, timestamp '2016-9-04 8:40:00.00', 'registering for math
tutoring', 'register');
insert into message
values (7, 7, timestamp '2016-9-04 9:00:00.00', 'registering for math
tutoring', 'register');
insert into message
values (8, 9, timestamp '2016-9-04 9:10:00.00', 'registering for math
tutoring', 'register');



insert into evntreg
```

```
values (1, 1, timestamp '2016-9-04 20:10:00.00', 2, 'significant');
insert into evntreg
values (1, 3, timestamp '2016-9-05 16:10:00.00', 4, 'relishable');
insert into evntreg
values (1, 13, timestamp '2016-9-12 12:10:00.00', 5, 'preferable');
insert into evntreg
values (1, 5, timestamp '2016-9-08 08:10:00.00', 1, 'tiring');
insert into evntreg
values (1, 11, timestamp '2016-9-04 02:10:00.00', 2, 'sattisfactory');
insert into evntreg
values (1, 16, timestamp '2016-9-06 13:10:00.00', 5, 'likeable');
insert into evntreg
values (1, 10, timestamp '2016-9-03 10:10:00.00', 4,'pleasant');
insert into evntreg
values (1, 19, timestamp '2016-9-06 11:10:00.00', 3, 'forthcoming');
insert into evntreg
values (1, 17, timestamp '2016-9-04 20:20:00.00', 4, 'genial');
insert into evntreg
values (2, 3, timestamp '2016-9-02 8:00:00.00', 5, 'informative');
insert into evntreg
values (2, 1, timestamp '2016-9-02 8:10:00.00', 4, 'knowledge gained');
insert into evntreg
values (2, 4, timestamp '2016-9-03 8:15:00.00', 1, 'not good');
insert into evntreg
values (2, 2, timestamp '2016-9-03 8:20:00.00', 3, 'educational');
insert into evntreg
values (2, 20, timestamp '2016-9-04 8:30:00.00', 4, 'illuminating');
insert into evntreg
values (2, 18, timestamp '2016-9-04 8:40:00.00', 2, 'edifying');
insert into evntreg
values (2, 7, timestamp '2016-9-04 9:00:00.00', 5, 'newsy');
insert into evntreg
values (2, 9, timestamp '2016-9-04 9:10:00.00', 3, 'elucidative');
insert into evntreg
values (2, 12, timestamp '2016-9-04 20:10:00.00', 3, 'significant');
insert into evntreg
values (2, 14, timestamp '2016-9-04 17:10:00.00', 2, 'revelatory');
insert into evntreg
values (3, 3, timestamp '2016-9-30 17:10:00.00', 2, 'drudging');
insert into evntreg
values (3, 12, timestamp '2016-9-22 17:10:00.00', 1, 'arid');
insert into evntreg
values (3, 9, timestamp '2016-9-19 17:10:00.00', 3, 'uninteresting');
insert into evntreg
values (3, 6, timestamp '2016-9-14 17:10:00.00', 2, 'cloying');
insert into evntreg
values (3, 14, timestamp '2016-9-15 17:10:00.00', 5, 'revelatory');
insert into evntreg
values (5, 8, timestamp '2016-9-30 12:10:00.00', 5, 'good');
insert into evntreg
values (5, 13, timestamp '2016-10-05 23:10:00.00', 4, 'interesting');
insert into evntreg
values (5, 15, timestamp '2016-9-22 19:10:00.00', 5, 'favorable');
insert into evntreg
```

```
values (5, 17, timestamp '2016-10-08 07:10:00.00', 3, 'wonderful');
insert into evntreg
values (5, 18, timestamp '2016-10-02 17:10:00.00', 4, 'exiciting');
insert into evntreg
values (5, 2, timestamp '2016-10-05 15:10:00.00', 4, 'commendable');
insert into evntreg
values (5, 6, timestamp '2016-9-25 22:10:00.00', 5, 'superb');
insert into evntreg
values (6, 10, timestamp '2016-10-30 22:10:00.00', 4, 'super eminent');
insert into evntreg
values (6, 7, timestamp '2016-11-12 22:10:00.00', 3, 'admirable');
insert into evntreg
values (6, 15, timestamp '2016-11-14 22:10:00.00', 5, 'superb');
insert into evntreg
values (6, 3, timestamp '2016-11-05 22:10:00.00', 2, 'unskilled');
insert into evntreg
values (6, 1, timestamp '2016-11-25 22:10:00.00', 5, 'marvelous');


insert into waitlist values (2, 1, 5);
insert into waitlist values (2, 2, 13);
insert into waitlist values (2, 3, 16);
insert into waitlist values (2, 4, 19);
insert into waitlist values (5, 1, 9);
insert into waitlist values (5, 2, 11);

--EXTRA CODE/FUNCTIONS:

--Code for Sequences
CREATE SEQUENCE userid_seq START WITH 21; ---- sequence to generate new
user id
CREATE SEQUENCE evntid_seq START WITH 7; -- sequence to generate new event
id
CREATE SEQUENCE msg_seq START WITH 9; -- sequence to generate new message
id

--this function checks if an event ID exists
--it returns 1 if the event exists and 0 if it does not
create or replace function evidcheck (ev_id number)
return int is
evrow event%rowtype;
begin
select e.* into evrow
from event e
where e.evnt_id = ev_id;
return (1);
exception when no_data_found then
return(0);
end;
--FEATURES:
---------------------------------------------------------------------------
--------------------------------------------------
--Feature 1
```

```
---------------------------------------------------------------------------
--------------------------------------------------
-- Procedure to register an account
set serveroutput on;
CREATE OR REPLACE PROCEDURE RegisterAccount(u_name in varchar2, u_phone in
varchar2, u_email in varchar2, u_password in varchar2, u_type varchar2) IS
userCountExist number;
NewUserId number;
BEGIN
      SELECT COUNT(user_id)
      INTO userCountExist
      FROM users
      WHERE user_email = u_email; -- Check whether email id is already
exist

      if (userCountExist >= 1 ) then
            dbms_output.put_line('Account Already Exist');
      else
            INSERT INTO users VALUES(userid_seq.nextval, u_name, u_phone,
u_email, u_password, u_type); -- creating new account with given values

            select userid_seq.currval into NewUserId from dual; --
fetching current user id if from user table
            dbms_output.put_line('Account is created with new user id: '
|| NewUserId);
      commit;
  END IF;
END;
---------------------------------------------------------------------------
--------------------------------------------------
--Feature 2, user login by providing user id and password
---------------------------------------------------------------------------
--------------------------------------------------
-- function to check login is successful or not. If it is successful it
will return 1 else it will return 0.
set serveroutput on;
create or replace function account_validation (u_id number, u_password
varchar) return number as
userid users.user_id%type;
userpass users.user_password%type;
success varchar2(20);
begin
      SELECT user_id, user_password  -- To check given user id and user
passward in the user table
      into userid, userpass
      from users
      where user_id = u_id and user_password = u_password;

      if (userid != u_id and userpass != u_password)  then  -- Checking
user is and user password matching or not. If it is not matches it will
return 0 else it will return 1.
            return 0;
      else
            return 1;
      end if ;

exception
when no_data_found then
return -1;
end;

----------------------------------------------------------------
-- Procedure to call the function account_validation
set serveroutput on;
create or replace procedure account_validate(u_id number, u_password
varchar) as --this procedure is to Validate the identity of a user based
on the input user-id and password
userid users.user_id%type;
userpass users.user_password%type;
success varchar2(20);
Begin
      success := account_validation(u_id, u_password); -- executing
function

      if success = 1  then -- checking login is suceesful or not based on
the result of function.
            dbms_output.put_line('Welcome to the Event Management
System');

      else
            dbms_output.put_line('your password or useraccount is not
correct ');

      end if ;

end;

---------------------------------------------------------------------------
---------------------------------------------------
--Feature 3, allow user to read messages providing user id
---------------------------------------------------------------------------
---------------------------------------------------

Set serveroutput on;
Create or replace
      PROCEDURE ReadMeassage(u_id in integer) IS --- This procedure allows
to read message by providing each user id
Cursor c1 is select msg from message where user_id = u_id; --- checks if
the user id in the event table matches the user id in message table
msg_g message.msg%type;
uid number;
BEGIN

SELECT user_id into uid from users where user_id = u_id;

If uid is not null then
      dbms_output.put_line('User is registered.');
      Open c1;
            Loop
                  fetch c1 into msg_g;
                  exit when c1%notfound;
                  dbms_output.put_line(msg_g);
            End loop;
      Close c1;
else
      dbms_output.put_line('User is not registered.');
end if;

exception
      when no_data_found then
      dbms_output.put_line('Wrong User Id.');

end;

 -------------------------------------------------------------------------
---------------------------------------------------
--Feature 4, create event
--------------------------------------------------------------------------
--------------------------------------------------
set serveroutput on;
create or replace PROCEDURE EnterEvent(location in varchar, organizer_id
in INTEGER,event_title in VARCHAR,event_descr in VARCHAR,
event_strdatetime in timestamp,event_enddatetime in timestamp,event_url
varchar,event_capacity in INTEGER, event_wl_capacity in INTEGER) IS
ConflictEventCheck number;
location_id number;
x number;
BEGIN
      select loc_id into location_id from locations where loc_name =
location;

      Select count(E2.evnt_id) into ConflictEventCheck  -- Checking
whether any event having conflict with another by comparing location ,
start time and end time.
      from event E1, event E2
      where E2.loc_id = location_id
      and E2.evnt_strdatetime = event_strdatetime
      and E2.evnt_id = E1.evnt_id
      and E2.loc_id = E1.loc_id
      and E2.evnt_strdatetime = E1.evnt_strdatetime;

-- if there are more records with same location , start time and end time
then there is conflict. If there is no conflict then it will create new
event with new event Id.
      if (ConflictEventCheck != 0) then
            dbms_output.put_line('conflict occur');
      else
            insert into Event(evnt_id, Loc_Id, orgn_id, evnt_title,
evnt_descr, evnt_strdatetime, evnt_enddatetime, evnt_url, evnt_capacity,
evnt_wl_capacity) values (evntid_seq.nextval,location_id, organizer_id,
event_title, event_descr, event_strdatetime, event_enddatetime, event_url,
event_capacity, event_wl_capacity);
            select evntid_seq.currval into x from dual;  -- selecting
current event id from event table.
            dbms_output.put_line('Event is created with Event Id - ' ||
x);
      end if;

end;
----------------------------------------------------------------------------
---------------------------------------------------
--Feature 5, list people registered for event
----------------------------------------------------------------------------
---------------------------------------------------
create or replace PROCEDURE getregusers (ev_id int)
as cursor c1 is
select u.user_name as Participants, u.user_email as Email, u.user_type as
u_Type
from users u, evntreg e --select participant, email, type of user
where e.evnt_id=ev_id and u.user_id=e.user_id; --when the desired event ID
lookup matches records in table
r c1%rowtype; --catch data into rowtype variable
begin
if evidcheck(ev_id) > 0 then
      open c1;
            loop
                  fetch c1 into r;
                  exit when c1%notfound; --exit when no more matches
                  dbms_output.put_line('Participant: '|| r.participants ||'
Email: ' || r.Email ||' Type: '|| r.u_Type);
            end loop;
            dbms_output.put_line('END');--the tells user cursor is done
parsing
      close c1;
else
dbms_output.put_line('Event ID does not exist');
end if;
end;
----------------------------------------------------------------------------
--------------------------------------------------
--Feature 6, list people on waitlist of event
----------------------------------------------------------------------------
---------------------------------------------------
create or replace PROCEDURE getwlusers (ev_id int)
as cursor c1 is
select u.user_name as Participants, u.user_email as Email, u.user_type as
u_Type, w.wl_position as WL_Position
from users u, waitlist w --select participant, email, type of user
where w.evnt_id=ev_id and u.user_id=w.user_id; --when the desired event ID
lookup matches records in table
r c1%rowtype; --catch data into rowtype variable
begin
if evidcheck(ev_id) > 0 then
      open c1;
```

```
          loop
                fetch c1 into r;
                exit when c1%notfound; --exit when no more matches
                dbms_output.put_line('Waitlist Position: ' ||
r.wl_position || ' Participant: '|| r.participants ||' Email: ' || r.Email
||' Type: '|| r.u_Type);
          end loop;
          dbms_output.put_line('END');--the tells user cursor is done
parsing
      close c1;
else
dbms_output.put_line('Event ID does not exist');
end if;
end;
---------------------------------------------------------------------------
---------------------------------------------------
--Feature 7, return avg rating of an event, total # of partic, total # on
waitlist
---------------------------------------------------------------------------
---------------------------------------------------
create or replace procedure eventratinginfo(ev_id in integer) is
avg_rating int;
totalpartic int;
totalwl int;
begin
if evidcheck(ev_id) > 0 then --first check if event exists
      dbms_output.put_line('For event number ' || ev_id || ':');
      select avg(e.rating) into avg_rating
      from evntreg e
      where e.evnt_id= ev_id;
            if avg_rating > 0 then --check if event has been rated
                  dbms_output.put_line('The average rating is ' ||
avg_rating);
            else
                  dbms_output.put_line('Event has not been rated');--
message if the if statement isn't met
            end if;
      select count(*) into totalpartic
      from evntreg e
      where e.evnt_id=ev_id;
      dbms_output.put_line('Total number of participants is ' ||
totalpartic);

      select count(*) into totalwl
      from waitlist w
      where w.evnt_id=ev_id;

      dbms_output.put_line('Total number on the waitlist is ' || totalwl);
else
dbms_output.put_line('Event ID does not exist');
end if;
end;
---------------------------------------------------------------------------
---------------------------------------------------
```

```
--Feature 8, cancel an event
---------------------------------------------------------------------------
-------------------------------------------------
set serveroutput on;
create or replace procedure CancelEvent(event_id in number) IS
cursor c1 is select user_id from evntreg where evnt_id = event_id; -- to
fetch regitered users from eventreg table
cursor c2 is select user_id from waitlist where evnt_id = event_id; -- to
fetch users on waitlist from waitlist table
eventid event.evnt_id%type;
eventname event.evnt_title%type;
userid users.user_id%type;
msgs message.msg%type;
u_id users.user_id%type;
Begin

Select event_id, evnt_title into eventid, eventname from event where
evnt_id = event_id; -- To check event is exist in event table.
If eventid is not null then
      update event set evnt_active = 'F' where evnt_id = event_id; -- if
event is cancelled then flag event_active will be set as F(false) to
indicate event is cancelled

-- generating message for users who are registered for that event.
   open c1;
      loop
            fetch c1 into userid;
            exit when c1%NOTFOUND;
            msgs := 'The Event ' || eventname || ' has been canceled ';
            insert into message(msg_id,user_id,msg_time,msg)
values(msg_seq.nextval, userid, systimestamp, msgs);
  END LOOP;
   Close c1;

-- generating message for users who are on waitlist.
   open c2;
      loop
            fetch c2 into u_id;
            exit when c2%NOTFOUND;
            msgs := 'The Event ' || eventname || ' has been canceled ';
            insert into message(msg_id,user_id,msg_time,msg)
values(msg_seq.nextval, u_id, systimestamp, msgs);
  END LOOP;
   Close c2;
else
   dbms_output.put_line('no such event');
end if;

exception
when no_data_found then
dbms_output.put_line('Event does not exist');
end;
```

```
--------------------------------------------------------------------------
------------------------------------------------
--Feature 9, update start/end time of event
--------------------------------------------------------------------------
------------------------------------------------
create or replace procedure Updateeventdatetime(event_id in number,
strdatetime in timestamp, enddatetime in timestamp) IS
cursor c1 is select user_id from evntreg where evnt_id = event_id;
 -- to fetch regitered user id from eventreg table
cursor c2 is select user_id from waitlist where evnt_id = event_id;
eventid event.evnt_id%type;
eventname event.evnt_title%type;
userid users.user_id%type;
msgs message.msg%type;
u_id users.user_id%type;
Begin

Select event_id, evnt_title into eventid, eventname from event where
evnt_id = event_id;
-- To check event is exist
If eventid is not null then
      Update event set evnt_strdatetime = strdatetime, evnt_enddatetime =
enddatetime where evnt_id = event_id; -- updating the start and end date
and time to the new date and time
    open c1;
      loop
            --  loop to update the message table for all the users who are
registered about the date and time changes.

            fetch c1 into userid;
            exit when c1%NOTFOUND;
            msgs := eventname||' event is now at: Start: '||strdatetime||'
End: '||enddatetime;
            insert into message(msg_id,user_id,msg_time,msg,msg_type)
values(msg_seq.nextval, userid, systimestamp, msgs,'Update');
  END LOOP;
   dbms_output.put_line('Message sent to registered users');
   Close c1;
   open c2;
      Loop
-- loop to update the message table for all the users who are on waitlist
about the date and time.

            fetch c2 into u_id;
            exit when c2%NOTFOUND;
            msgs := eventname||' event is now at: Start: '||strdatetime||'
End: '||enddatetime;
            insert into message(msg_id,user_id,msg_time,msg,msg_type)
values(msg_seq.nextval, u_id, systimestamp, msgs,'Update');
  END LOOP;
  dbms_output.put_line('Message sent to users on waitlist');
   Close c2;
end if;
exception
```

```
when no_data_found then
   dbms_output.put_line('Wrong Event ID');
end;
--------------------------------------------------------------------------
-------------------------------------------------
--Feature 10, search for events by keyword
--------------------------------------------------------------------------
--------------------------------------------------

CREATE OR REPLACE PROCEDURE get_event(keyword in VARCHAR2, c1 IN OUT
sys_refcursor) AS
BEGIN
open c1 for
-- cursor to find events with the keyword
    Select evnt_title, evnt_strdatetime, evnt_enddatetime, loc_name,
evnt_url, orgn_name, user_email, event_full from event e, organizers o,
locations l where evnt_title like '%' || keyword || '%' and e.orgn_id =
o.orgn_id and e.loc_id = l.loc_id;

   exception
   -- if no data is found
  when no_data_found then
      dbms_output.put_line('No search results for the keyword');
END;


--------------------------------------------------------------------------
----------------------------------------------------
Feature 11, register user for an event
--------------------------------------------------------------------------
--------------------------------------------------
set serveroutput on;
create or replace procedure EventRegistration( u_id in number, ev_id in
number) IS
eventExist number;
Current_capacity number;
EventCapacity number;
WL_Capacity number;
Current_WL_capacity number;
new_wl_capacity number;
eventactive event.evnt_active%type;
userExist number;
wluserExist number;
Begin
    SELECT evnt_id, UPPER(Evnt_Active)   -- To check event is exist or
not.
    INTO eventExist, eventactive
    FROM EVENT
    WHERE evnt_id = ev_id;

        if (eventExist != 0 and eventactive = 'T' ) then  -- If event id
is not null and eventactive is true then event is exist.
            dbms_output.put_line('Event is exist');
            select count(user_id) into userExist from evntreg where
evnt_id = ev_id and user_id = u_id; -- To check whether user is already
exist on event registration table
    select count(user_id) into wluserExist from waitlist where  evnt_id =
ev_id and user_id = u_id; -- To check whether user is already exist on
waitlist table

            if (userExist != 0 or wluserExist != 0) then
                  dbms_output.put_line('User ' ||u_id||': You are already
registerd/on waitlist for this event');

            else

                  Select evnt_capacity into EventCapacity FROM event where
evnt_id = ev_id; -- To find event capacity of particular event.
                  Select COUNT(evnt_id) into Current_capacity FROM evntreg
where evnt_id = ev_id; -- To find how many users registered for that
event.

                        if (Current_capacity >= EventCapacity) then      --
If the current capacity is equal to event capacity then event is full and
it will check waitlist is full or not.

                        UPDATE event set event_full = 'Yes' where evnt_id =
ev_id;

                        Select e.evnt_wl_capacity into WL_Capacity from
event e where evnt_id = ev_id; -- To find waitlist capacity of particular
event.
                        Select COUNT(wl_position) into Current_WL_capacity
FROM waitlist where evnt_id = ev_id; -- To find how many users are on
waitlist for that event.

                                    if(Current_WL_capacity >= WL_Capacity)
then     -- -- If the current waitlist capacity is equal to waitlist
capacity then waitlist of that event is full. If waitlist is not full then
user is added to the waitlist at next position.
                                          dbms_output.put_line('Waitlist is
full');
                                    else
                                          new_wl_capacity :=
Current_WL_capacity + 1;
                                          insert into waitlist(evnt_id,
wl_position, user_id) values (ev_id, new_wl_capacity, u_id);
                                          dbms_output.put_line('You are on
waitlist position ' || new_wl_capacity);
                                    end if;
                        else
                        INSERT INTO evntreg(evnt_id, user_id, reg_time)
VALUES (ev_id, u_id, systimestamp);  -- If event registration is not full.
User will be registered for that event.
                        dbms_output.put_line('User ' ||u_id||': You are
registerd for this event');
                        end if;
            end if;

      else
            dbms_output.put_line('Event is not exist');
      end if;

exception
when no_data_found then
dbms_output.put_line('Wrong Event Id or User Id.');
end;



---------------------------------------------------------------------------
--------------------------------------------------
--feature 12, cancel a users event registration
---------------------------------------------------------------------------
--------------------------------------------------
set serveroutput on;
Create or replace procedure CancelRegistration(u_id in number, ev_id in
number) IS
Cursor c1 is select wl_position, user_id from waitlist where evnt_id =
ev_id;
eventExist event.evnt_id%type;
eventactive event.evnt_active%type;
userexist number;
EventName event.evnt_title%type;
msgs message.msg%type;
wl_user number;
uid number;
messages message.msg%type;
waitlist_position waitlist.wl_position%type;

Begin
      SELECT e.evnt_id,e.evnt_title, upper(e.evnt_active), er.user_id INTO
eventExist, EventName, eventactive, userexist FROM event e, evntreg er
WHERE e.evnt_id = ev_id and er.evnt_id = e.evnt_id and er.user_id = u_id;
      if (eventExist = ev_id and userexist = u_id and eventactive = 'T')
then -- checking event is exist and user is registered for the event.
            dbms_output.put_line('Event is exist and user is
registered.');

            -- Deleting the record with given event Id and User ID and
inserting a message in message table saying event registration has been
cancelled.
            delete from evntreg where evnt_id = ev_id and user_id = u_id;
            msgs := 'Event '|| EventName ||' registration for User '||
u_id ||' has been canceled.';
            insert into message(msg_id,user_id,msg_time,msg)
values(msg_seq.nextval, u_id, systimestamp, msgs);
            dbms_output.put_line(msgs);
            Select count(user_id) into uid from waitlist where evnt_id =
ev_id and wl_position = 1;  -- selecting user from waitlist with waitlist
position 1.

            if (uid != 0) then

                  -- if waitlist is not null then it will register user
with waitlist position 1 into event registration table. Also, user with
waitlist position 1 from waitlist table.
                  INSERT INTO evntreg(evnt_id, user_id, reg_time) VALUES
(ev_id, uid, systimestamp);
                  messages := 'User '||uid||' are registered for event '||
EventName;
                  insert into message(msg_id,user_id,msg_time,msg)
values(msg_seq.nextval, uid, systimestamp, messages);
                  delete from waitlist where evnt_id = ev_id and user_id =
uid;
                        dbms_output.put_line(messages);

                        -- updating waitlist position in waitlist position
from k to k-1.
                              open c1;
                               loop
                                          fetch c1 into waitlist_position,
wl_user;
                                          exit when c1%NOTFOUND;
                                          waitlist_position :=
waitlist_position - 1;
                                          update waitlist set wl_position =
waitlist_position where evnt_id = ev_id and user_id = wl_user;
                              END LOOP;
                              Close c1;
            else
                  dbms_output.put_line('Waitlist is empty');
            end if;
      else
            dbms_output.put_line('Event is not exist');
      end if;

exception
when no_data_found then
dbms_output.put_line('Wrong User Id or Event Id ');
end;

--------------------------------------------------------------------------
--------------------------------------------------
--Feature 13, enter a review for event
--------------------------------------------------------------------------
--------------------------------------------------

set serveroutput on;

create or replace procedure enter_rating(event_id in number, u_id in
number, rate in number, cmmt in varchar
) IS

eventid number;

usr_id number;

Begin

      select user_id into usr_id from evntreg where evnt_id = event_id and
user_id = u_id; -- To check whether the user for that particular event is
registerd

      select event_id into eventid from event where evnt_id = event_id;
 -- To check whether event is exist.

      if eventid is not null then

            If (usr_id = u_id) then

                  Update evntreg set rating = rate, commnt = cmmt where
evnt_id = event_id and user_id = u_id;

                  dbms_output.put_line('Thank you for your feedback!');

            end if;

      end if;

exception

when no_data_found then

dbms_output.put_line('Wrong event id or User has not registered for the
event');

end;
---------------------------------------------------------------------------
--------------------------------------------------
--Feature 14, print total number of events, etc
---------------------------------------------------------------------------
--------------------------------------------------

CREATE OR REPLACE PROCEDURE Summary AS
  CURSOR c1 IS SELECT count(evnt_id) FROM event;
  total_evnts number;
  CURSOR c2 IS select count(user_id)/count(distinct evnt_id) from evntreg;
  avg_reg integer;
  CURSOR c3 IS select count(user_id)/count(distinct evnt_id) from
waitlist;
  avg_wl integer;
BEGIN
-- Cursor to give total number of events created.
  OPEN c1;
  fetch c1 into total_evnts;
  dbms_output.put_line('The total number of events:' || total_evnts);
-- Cursor to give average number of users per event
  OPEN c2;
  FETCH c2 INTO avg_reg;
  dbms_output.put_line('The average number of users per event: ' ||
avg_reg);
-- Cursor to give average number of users on waitlist
  OPEN c3;
  FETCH c3 INTO avg_wl;
  DBMS_OUTPUT.PUT_LINE('The average number of users on waitlist: ' ||
avg_wl);
  CLOSE c3;
  CLOSE c2;
  Close c1;
END;


--------------------------------------------------------------------------
---------------------------------------------------
--Feature 15, print out top K events with most participants
--------------------------------------------------------------------------
---------------------------------------------------
set serveroutput on;
Create or replace
PROCEDURE get_report (k in number) IS
Cursor c1 is select evnt_id, count(user_id) from evntreg group by evnt_id
order by count(user_id) desc ; -- To find top K events with the most
participants
Cursor c2 is select loc_id, count(evnt_id) from event group by loc_id
order by count(evnt_id) desc; -- To find he top K locations with most
number of events
Cursor c3 is select evnt_id, avg(rating) from evntreg group by evnt_id
order by avg(rating) desc; -- To find top K events with the highest
average ratings
Event_with_most_participant number;
number_of_participants number;
Locations_with_most_event number;
number_of_events number;
highest_avgrating_events number;
average_rating number;
BEGIN
Open c1;
-- Here we are fetching events with most participants and number of
participants registered in variables. Loop will be repeted for k times.
Loop
      fetch c1 into Event_with_most_participant, number_of_participants;
      exit when c1%notfound or c1%rowcount > k;
      dbms_output.put_line('Events with most participant is ' ||
Event_with_most_participant);
End loop;
Open c2;
-- Here we are fetching locations with most events and number of locations
in variables. Loop will be repeted for k times.
Loop
```

```
      fetch c2 into Locations_with_most_event, number_of_events;
      exit when c2%notfound or c2%rowcount > k;
      dbms_output.put_line('Locations with most number of events ' ||
Locations_with_most_event);
End loop;

Open c3;
-- Here we are fetching events with highest average rating and average
rating in variables. Loop will be repeted for k times.
Loop
      fetch c3 into highest_avgrating_events, average_rating;
      exit when c3%notfound or c3%rowcount > k;
      dbms_output.put_line('Events with the highest average ratings ' ||
highest_avgrating_events);
End loop;
close c1;
close c2;
close c3;

END;

---------------------------------------------------------------------------
--------------------------------------------------
--EXAMPLES AND TEST CASES
---------------------------------------------------------------------------
--------------------------------------------------
--test feature 1

exec RegisterAccount('Rashmi', 4102876211, 'rashmi1@umbc.edu',
'rashmipassword', 'student'); -- Example of already exist account
exec RegisterAccount('Susan', 4102349876, 'susan@umbc.edu',
'susanpassword', 'staff');
exec RegisterAccount('Robin', 4103249876, 'robin@umbc.edu',
'robinpassword', 'staff');
exec RegisterAccount('Tom', 4104329876, 'tom@umbc.edu', 'tompassword',
'staff'); -- Example of new user

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 2:

exec account_validate(36, 'sarahpassword'); --For no data in table
exec account_validate(4,'rashmipassword'); -- For correct combination of
userid and password
exec account_validate(4,'sarahpassword'); -- For incorrect combination of
userid and password

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 3:

exec ReadMeassage(9); -- for user id registered
exec ReadMeassage(21); -- for user id not registered
```

```
---------------------------------------------------------------------------
---------------------------------------------------
--test feature 4:

exec EnterEvent('Lounge', 5,'Party Night','Come have a fun night!','22-
SEP-16 08.00.00.000000 PM','22-SEP-16 10.00.00.000000
PM','www.umbc.edu/partynight',20, 10); -- No conflict
exec EnterEvent('Lounge', 5,'Halloween Party','Halloween Party','22-SEP-16
08.00.00.000000 PM','22-SEP-16 10.00.00.000000
PM','www.umbc.edu/halloweenparty',20, 10); -- Conflict

---------------------------------------------------------------------------
---------------------------------------------------
--test feature 5:
--event exists, expected output is participants name, email, and type from
the given event
Select * from users;
Select * from evntreg;
exec getregusers(3);
--event doesn't exist, expected output is a message that event does't
exist
Select * from event;
exec getregusers(35);

---------------------------------------------------------------------------
---------------------------------------------------
--test feature 6:
--event exists, expected output is waitlist position, name, email, and
type from given event
Select * from users;
Select * from waitlist;
exec getwlusers(2);
--event doesn't exists, expected output is message that event doesn't
exist
Select * from event;
exec getwlusers(35);

---------------------------------------------------------------------------
---------------------------------------------------
--test feature 7:
--event exists, expected output is average rating for given event, its
total # of participants and --total # on the waitlist
Select * from evntreg;
Select * from waitlist;
exec eventratinginfo(2);
--event doesn't exist
Select * from event;
exec eventratinginfo(21);

---------------------------------------------------------------------------
---------------------------------------------------
--test feature 8:

exec CancelEvent(1); -- Event exist
exec CancelEvent(10); -- Event not exist

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 9:
-- event exists
Set serveroutput on;
exec updateeventdatetime(4,'16-SEP-16 08.00.00.000000 PM','16-SEP-16
10.00.00.000000 PM');
-- event does not exist
exec updateeventdatetime(40,'16-SEP-16 08.00.00.000000 PM','16-SEP-16
10.00.00.000000 PM');

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 10:
--Execute statement:
var rc refcursor
exec get_event ('Math',:rc); -- valid keyword search
print rc;

var rc refcursor
exec get_event ('Dance',:rc); -- No data for keyword search
print rc;


---------------------------------------------------------------------------
--------------------------------------------------
--test feature 11:

set serveroutput on;
exec EventRegistration(8, 1); -- when event is not full
exec EventRegistration(11, 1); -- when user is already registered
exec EventRegistration(15, 1); -- Event is full, go to the waitlist
(waitlist is not full)
exec EventRegistration(6, 2); -- when waitlist is full
exec EventRegistration(6, 8); -- when event is not exist or cancelled

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 12:

exec CancelRegistration(16, 1); -- When waitlist is not empty
exec CancelRegistration(4, 2); -- When waitlist is not empty
exec CancelRegistration(10, 1);-- when waitlist is empty
exec CancelRegistration(10, 4); -- when event is not active

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 13:
Set serveroutput on;
exec enter_rating(1,5,4,'Good Event'); -- valid data

Set serveroutput on;
```

```
exec enter_rating(21,5,4,'Good Event'); -- exception for wrong event id

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 14:
exec summary;

---------------------------------------------------------------------------
--------------------------------------------------
--test feature 15:
exec get_report(2);
exec get_report(3);
exec get_report(4);
```