Kristen Groom

CSC 452 online section

Database Programming

Homework 3

***Plsql Code:***

--CSC452 Summer Session II

--Homework3

--Kristen Groom

--7/30/18

-- Create HW3 text tables:

--Drop all tables:

set serveroutput on;

declare

sqlstring varchar2(100) :=

'drop table movies cascade constraints';

begin

execute immediate sqlstring;

--general exception to catch error if the table does not exist - repeated in below tables

exception

when others then

dbms\_output.put\_line('Table movies does not exist');

null;

end;

/

declare

sqlstring varchar2(100) :=

'drop table ages cascade constraints';

begin

execute immediate sqlstring;

exception

when others then

dbms\_output.put\_line('Table ages does not exist');

null;

end;

/

declare

sqlstring varchar2(100) :=

'drop table genres cascade constraints';

begin

execute immediate sqlstring;

exception

when others then

dbms\_output.put\_line('Table genres does not exist');

null;

end;

/

declare

sqlstring varchar2(100) :=

'drop table occupations cascade constraints';

begin

execute immediate sqlstring;

exception

when others then

dbms\_output.put\_line('Table occupations does not exist');

null;

end;

/

declare

sqlstring varchar2(100) :=

'drop table users cascade constraints';

begin

execute immediate sqlstring;

exception

when others then

dbms\_output.put\_line('Table users does not exist');

null;

end;

/

declare

sqlstring varchar2(100) :=

'drop table ratings cascade constraints';

begin

execute immediate sqlstring;

exception

when others then

dbms\_output.put\_line('Table ratings does not exist');

null;

end;

/

declare

sqlstring varchar2(100) :=

'drop table movie\_genres cascade constraints';

begin

execute immediate sqlstring;

exception

when others then

dbms\_output.put\_line('Table movie\_genres does not exist');

null;

end;

/

--table to hold genres temporarily in procedure below

declare

sqlstring varchar2(100) :=

'drop table movie\_genres\_temp cascade constraints';

begin

execute immediate sqlstring;

exception

when others then

dbms\_output.put\_line('Table movie\_genres\_temp does not exist');

null;

end;

/

--Create tables

--Ages table

declare

sqlstring varchar(500) :=

'create table ages (

ageid number(5),

agerange varchar2(10),

primary key (ageid)

)';

table\_count integer;

begin

execute immediate sqlstring;

dbms\_output.put\_line('Ages table created');

end;

/

-- Genres table

declare

sqlstring varchar2(500) :=

'create table genres (

genre varchar2(15),

primary key (genre)

)';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Genres table created');

end;

/

--Occupations table

declare

sqlstring varchar2(500) :=

'create table occupations (

occupationid number(5),

occupation varchar2(25),

primary key (occupationid)

)';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Occupations table created');

end;

/

-- Ages values

declare

sqlstring varchar2(500);

begin

sqlstring :=

'insert into Ages

values (:a, :b)';

execute immediate sqlstring

using 1, 'Under 18';

execute immediate sqlstring

using 18, '18-24';

execute immediate sqlstring

using 25, '25-34';

execute immediate sqlstring

using 35, '35-44';

execute immediate sqlstring

using 45, '45-49';

execute immediate sqlstring

using 50, '50-55';

execute immediate sqlstring

using 55, '55+';

dbms\_output.put\_line('Rows inserted into ages table');

end;

/

--Occupations values

declare

sqlstring varchar2(500);

begin

sqlstring :=

'insert into Occupations

values (:a, :b)';

execute immediate sqlstring

using 0, '"other" or not specified';

execute immediate sqlstring

using 1, 'academic/educator';

execute immediate sqlstring

using 2, 'artist';

execute immediate sqlstring

using 3, 'clerical/admin';

execute immediate sqlstring

using 4, 'college/grad student';

execute immediate sqlstring

using 5, 'customer service';

execute immediate sqlstring

using 6, 'doctor/health care';

execute immediate sqlstring

using 7, 'executive/managerial';

execute immediate sqlstring

using 8, 'farmer';

execute immediate sqlstring

using 9, 'homemaker';

execute immediate sqlstring

using 10, 'K-12 student';

execute immediate sqlstring

using 11, 'lawyer';

execute immediate sqlstring

using 12, 'programmer';

execute immediate sqlstring

using 13, 'retired';

execute immediate sqlstring

using 14, 'sales/marketing';

execute immediate sqlstring

using 15, 'scientist';

execute immediate sqlstring

using 16, 'self-employed';

execute immediate sqlstring

using 17, 'technician/engineer';

execute immediate sqlstring

using 18, 'tradesman/craftsman';

execute immediate sqlstring

using 19, 'unemployed';

execute immediate sqlstring

using 20, 'writer';

dbms\_output.put\_line('Rows inserted into occupations table');

end;

/

-- Genres values

declare

sqlstring varchar2(500);

begin

sqlstring :=

'insert into genres

values (:a)';

execute immediate sqlstring

using 'Action';

execute immediate sqlstring

using 'Adventure';

execute immediate sqlstring

using 'Animation';

execute immediate sqlstring

using 'Children''s';

execute immediate sqlstring

using 'Comedy';

execute immediate sqlstring

using 'Crime';

execute immediate sqlstring

using 'Documentary';

execute immediate sqlstring

using 'Drama';

execute immediate sqlstring

using 'Fantasy';

execute immediate sqlstring

using 'Film-Noir';

execute immediate sqlstring

using 'Horror';

execute immediate sqlstring

using 'Musical';

execute immediate sqlstring

using 'Mystery';

execute immediate sqlstring

using 'Romance';

execute immediate sqlstring

using 'Sci-Fi';

execute immediate sqlstring

using 'Thriller';

execute immediate sqlstring

using 'War';

execute immediate sqlstring

using 'Western';

dbms\_output.put\_line('Rows inserted into genres table');

end;

/

--Movies table

declare

sqlstring varchar2(500) :=

'create table movies (

movieid number(5),

title varchar2(128),

genres varchar2(128),

primary key (movieid)

)';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Movies table created');

end;

/

-- insert data into movies table

begin

insert into movies

(select movieid, title, genres from movies\_temp);

dbms\_output.put\_line('Number of rows inserted into movies table: ' ||

to\_char(sql%rowcount));

end;

/

--Users table

declare

sqlstring varchar2(500) :=

'create table users (

userid number(5),

gender varchar2(26),

agecode number(5),

occupationid number(5),

zipcode varchar2(26),

primary key (userid),

foreign key (occupationid) references occupations(occupationid),

foreign key (agecode) references ages(ageid))';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Users table created');

end;

/

-- insert into users table

--change age value into agecode that references ageid from ages

begin

insert into users

(select userid, gender,

case

when age < 18 then 1

when age >= 18 and age < 25 then 18

when age >= 25 and age < 35 then 25

when age >= 35 and age < 45 then 35

when age >= 45 and age < 49 then 45

when age >= 50 and age < 55 then 50

when age >= 55 then 55

end as agecode,

occupationid, zipcode

from users\_temp);

dbms\_output.put\_line('Number of rows inserted into users: ' ||

to\_char(sql%rowcount));

end;

/

--Ratings table

declare

sqlstring varchar2(500) :=

'create table ratings (

userid number(5),

movieid number(5),

rating number(5),

timestamp number(11),

primary key (userid, movieid),

foreign key (userid) references users(userid),

foreign key (movieid) references movies(movieid))';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Ratings table created');

end;

/

--insert data into ratings table

begin

insert into ratings

(select userid, movieid, rating, timestamp from ratings\_temp);

dbms\_output.put\_line('Number of rows inserted into ratings: ' ||

to\_char(sql%rowcount));

end;

/

--movie\_genres table (connects movies and genres tables)

declare

sqlstring varchar2(200) :=

'create table movie\_genres (

movieid number(5),

genre varchar2(15),

primary key (movieid, genre),

foreign key (genre) references genres(genre))';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Ratings table created');

end;

/

--temporary table to hold genres from populate\_movie\_genres procedure below

create table movie\_genres\_temp (

movieid number(5),

genre1 varchar2(15),

genre2 varchar2(15),

genre3 varchar2(15),

genre4 varchar2(15),

genre5 varchar2(15),

genre6 varchar2(15));

--stored procedure to populate movie\_genres table

create or replace procedure populate\_movie\_genres as

begin

--the way I decided to handle this process is by creating a temporary table to hold each movie id and

--the separated genres with null in cols with no genre at that level

--I found the max pipes and decided to create 6 based on max pipes separating genres

insert into movie\_genres\_temp (movieid,

genre1 ,

genre2,

genre3,

genre4,

genre5,

genre6)

--get the movie ids

select movieid,

--this is the first case if no pipes are found this genre is the only genre - else it is the first genre

--using regular expressions - we're looking for chunks of text that end with pipe

case regexp\_count(genres, '[^|]+\|')

when 0 then genres

--genre1 is from the beginning of the string to the first pipe

else substr(genres, 1, instr(genres, '|', 1,1) - 1)

end as genre1,

--second condition

--extract value between pipe1 and pipe2

case

-- if it has only one pipe then the substring for second genre is between the first instance and the end of the entire string

--otherwise genre2 is between the first pipe and the second pipe

when regexp\_count(genres, '[^|]+\|') = 1 then substr(genres, instr(genres, '|', 1,1)+1, length(genres))

when regexp\_count(genres, '[^|]+\|') >= 2 then substr(genres, instr(genres, '|', 1,1)+1, instr(genres, '|', 1,2) - instr(genres, '|', 1,1) - 1)

end as genre2,

case

--same as above but now genre3 is between 2nd pipe and length of string

--or between 2nd pipe and 3rd pipe, below code continues this pattern up until 6 genres

when regexp\_count(genres, '[^|]+\|') = 2 then substr(genres, instr(genres, '|', 1,2)+1, length(genres))

when regexp\_count(genres, '[^|]+\|') >= 3 then substr(genres, instr(genres, '|', 1,2)+1, instr(genres, '|', 1,3) - instr(genres, '|', 1,2) - 1)

end as genre3,

case

when regexp\_count(genres, '[^|]+\|') = 3 then substr(genres, instr(genres, '|', 1,3)+1, length(genres))

when regexp\_count(genres, '[^|]+\|') >= 4 then substr(genres, instr(genres, '|', 1,3)+1, instr(genres, '|', 1,4) - instr(genres, '|', 1,3) - 1)

end as genre4,

case

when regexp\_count(genres, '[^|]+\|') = 4 then substr(genres, instr(genres, '|', 1,4)+1, length(genres))

when regexp\_count(genres, '[^|]+\|') >= 5 then substr(genres, instr(genres, '|', 1,4)+1, instr(genres, '|', 1,5) - instr(genres, '|', 1,4) - 1)

end as genre5,

case

when regexp\_count(genres, '[^|]+\|') = 5 then substr(genres, instr(genres, '|', 1,5)+1, length(genres))

when regexp\_count(genres, '[^|]+\|') >= 6 then substr(genres, instr(genres, '|', 1,5)+1, instr(genres, '|', 1,6) - instr(genres, '|', 1,5) - 1)

end as genre6

from movies;

-- now insert into the final movie\_genres tables

-- all genres unioned to get rid of null values

-- this separates movie ids and individual genres into rows

insert into movie\_genres (movieid, genre)

select movieid, genre1 genre

from movie\_genres\_temp

union

select movieid, genre2 genre

from movie\_genres\_temp

where genre2 is not null

union

select movieid, genre3 genre

from movie\_genres\_temp

where genre3 is not null

union

select movieid, genre4 genre

from movie\_genres\_temp

where genre4 is not null

union

select movieid, genre5 genre

from movie\_genres\_temp

where genre5 is not null

union

select movieid, genre6 genre

from movie\_genres\_temp

where genre6 is not null

order by movieid, genre;

end populate\_movie\_genres;

/

--execute populate\_movie\_genres procedure above

--probably do not need this as a named procedure

--but just trying it out

begin

populate\_movie\_genres;

dbms\_output.put\_line('Poulated movie\_genres table successfully');

end;

/

--drop genres column from movies

declare

sqlstring varchar2(200) :=

'alter table movies drop column genres';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Removed column genres from movies table successfully');

end;

/

-- now get year out of the title of movies:

--add column year to movies

--first add year

declare

sqlstring varchar2(200) :=

'alter table movies

add year number(5)';

begin

execute immediate sqlstring;

dbms\_output.put\_line('Added year column to movies table successfully');

end;

/

--get year into its own column

begin

update movies

set year = regexp\_substr(regexp\_substr( title, '\([^()]\d+\)' ), '\d+' );

dbms\_output.put\_line('Added years to year column for ' ||

to\_char(sql%rowcount) || ' rows');

end;

/

--remove year from title

begin

update movies

set title = regexp\_replace(title,'\([^()]\d+\)', '');

dbms\_output.put\_line('Removed year from title column for ' ||

to\_char(sql%rowcount) || ' rows');

end;

/

commit;

***Output:***

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

PL/SQL procedure successfully completed.

Ages table created

PL/SQL procedure successfully completed.

Genres table created

PL/SQL procedure successfully completed.

Occupations table created

PL/SQL procedure successfully completed.

Rows inserted into ages table

PL/SQL procedure successfully completed.

Rows inserted into occupations table

PL/SQL procedure successfully completed.

Rows inserted into genres table

PL/SQL procedure successfully completed.

Movies table created

PL/SQL procedure successfully completed.

Number of rows inserted into movies table: 3883

PL/SQL procedure successfully completed.

Users table created

PL/SQL procedure successfully completed.

Number of rows inserted into users: 6040

PL/SQL procedure successfully completed.

Ratings table created

PL/SQL procedure successfully completed.

Number of rows inserted into ratings: 201296

PL/SQL procedure successfully completed.

Ratings table created

PL/SQL procedure successfully completed.

Table MOVIE\_GENRES\_TEMP created.

Procedure POPULATE\_MOVIE\_GENRES compiled

Poulated movie\_genres table successfully

PL/SQL procedure successfully completed.

Removed column genres from movies table successfully

PL/SQL procedure successfully completed.

Added year column to movies table successfully

PL/SQL procedure successfully completed.

Added years to year column for 3883 rows

PL/SQL procedure successfully completed.

Removed year from title column for 3883 rows

PL/SQL procedure successfully completed.

Commit complete.

***Query:***

--Find all the movies from the most recent year in our database

--and their average ratings from the highest rated to lowest rated

select title, year,

round(avg(rating), 2) as ave\_ratings

from movies, ratings

where movies.movieid = ratings.movieid

and movies.year = (select max(year) from movies)

group by title, year

order by ave\_ratings desc;

***Analysis:***

I found 142 rows in my query from the year 2000. The highest rated movie from that year was titled *Requiem for a Dream,* with an average of 4.23 out of 5 and 80 ratings. The lowest rated movie from 200 was *Sunset Strip,* with an average rating of 1 but with only 1 rating.