TÆKNISKÓLINN 16-11-2017

GSF2A3U

LOKA

RKEFNI



# TÆKNISKÓLINN

16-11-2017

**Note:**

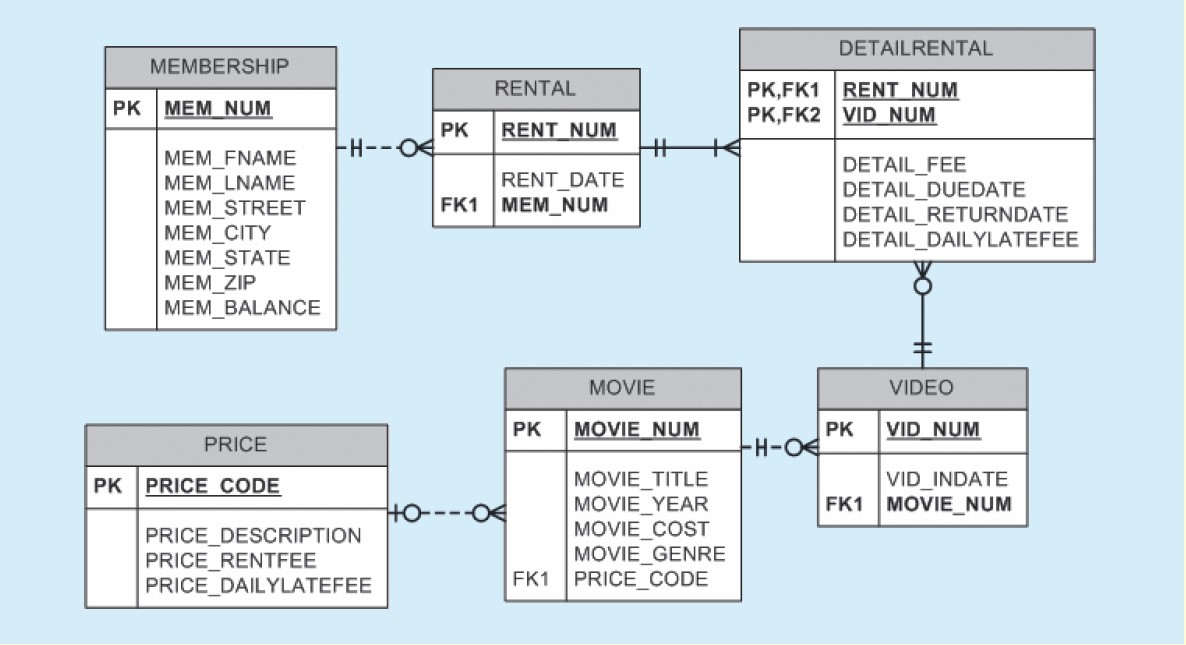
The project need to be submitted in GitHub, including diary, screen shots and SQL code. You can use phpmyadmin or workbench for database management. If you have an idea of a database project that you want to work on, or if you want to combine Database with another course you are free to do so. The goal is to practice what you have been learning in this semester in GSF2A3U course, including database design, advanced SQL queries and subqueries, SQL stored procedures, functions and triggers.

**Project Business Rules:**

TinyVideo is a small movie rental company with a single store. TinyVideo needs a database system to track the rental of movies to its members. TinyVideo can own several copies (VIDEO) of each movie (MOVIE). For example, the store may have 10 copies of the movie “Twist in the Wind.” “Twist in the Wind” would be one MOVIE, and each copy would be a VIDEO. A rental transaction (RENTAL) involves one or more videos being rented to a member (MEMBERSHIP).

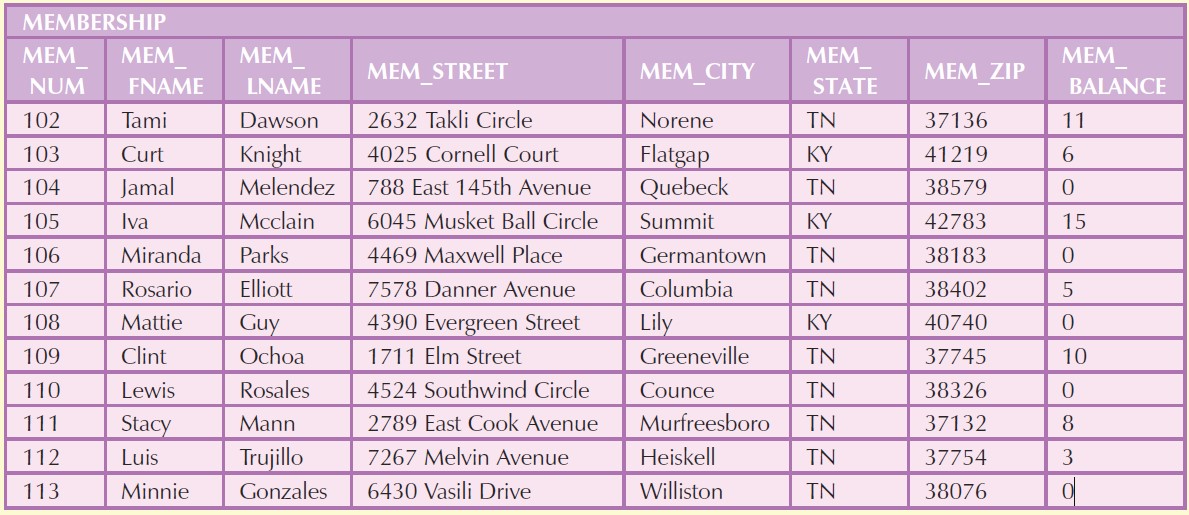
A video can be rented many times over its lifetime; therefore, there is a M:N relationship between RENTAL and VIDEO. DETAILRENTAL is the bridge table to resolve this relationship. The complete ERD is provided in **Figure P7.44**.

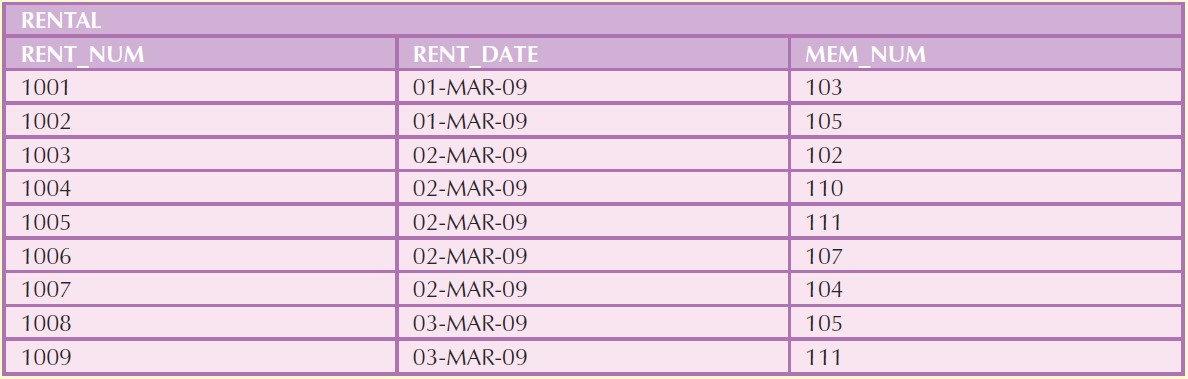
## Figure P7.44 MovieCo ERD

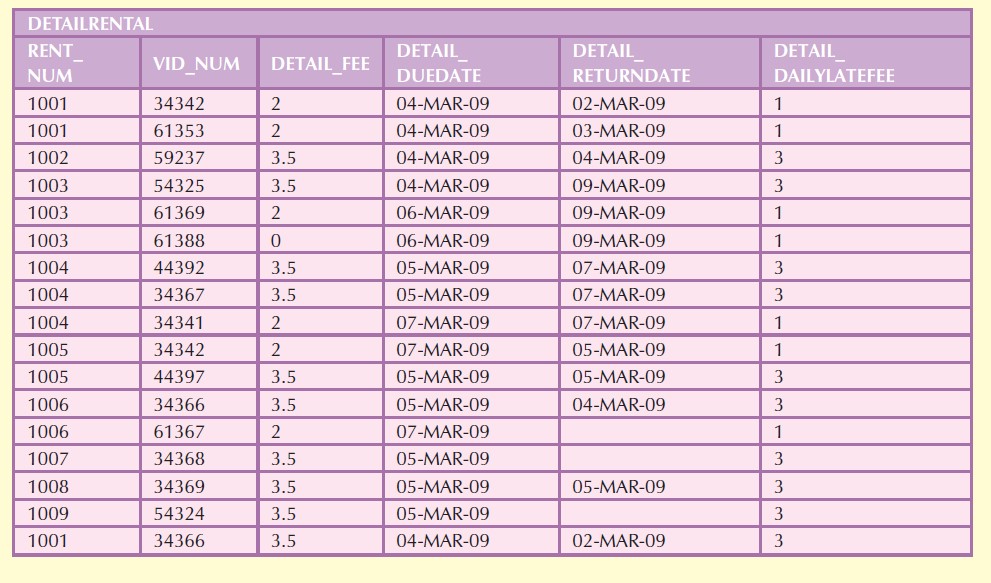


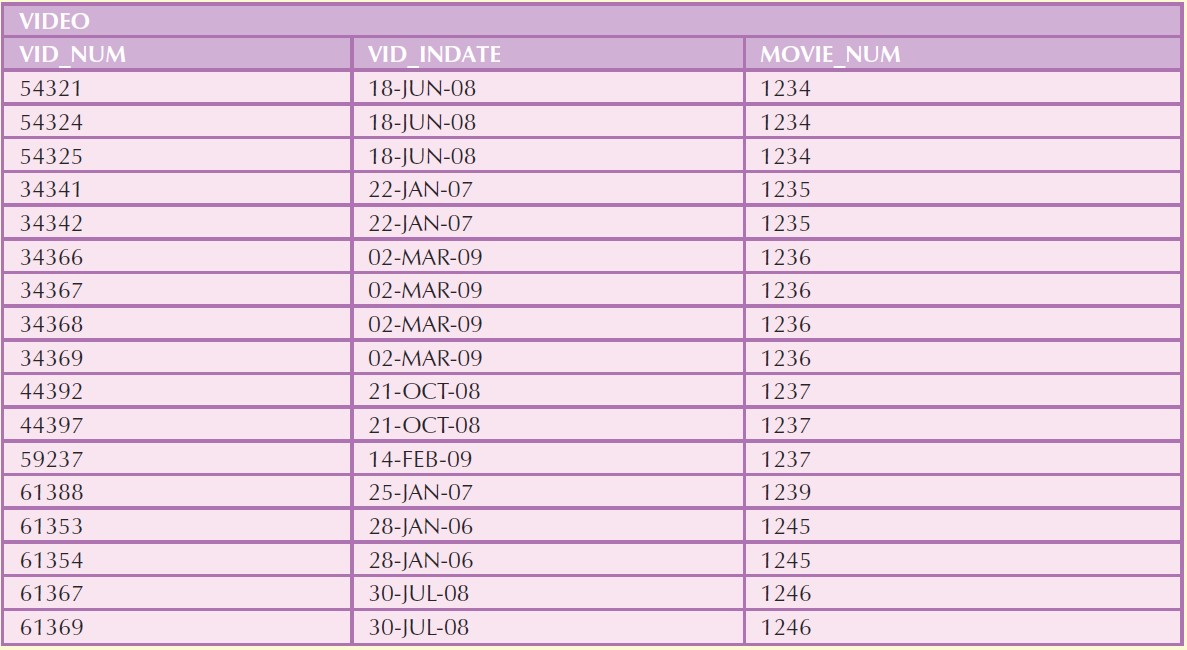
1. Write the SQL code to create the table structures for the entities shown in Figure P7.44. The structures should contain the attributes specified in the ERD. Use data types that are appropriate for the data that will need to be stored in each attribute. Enforce primary key and foreign key constraints as indicated by the ERD.

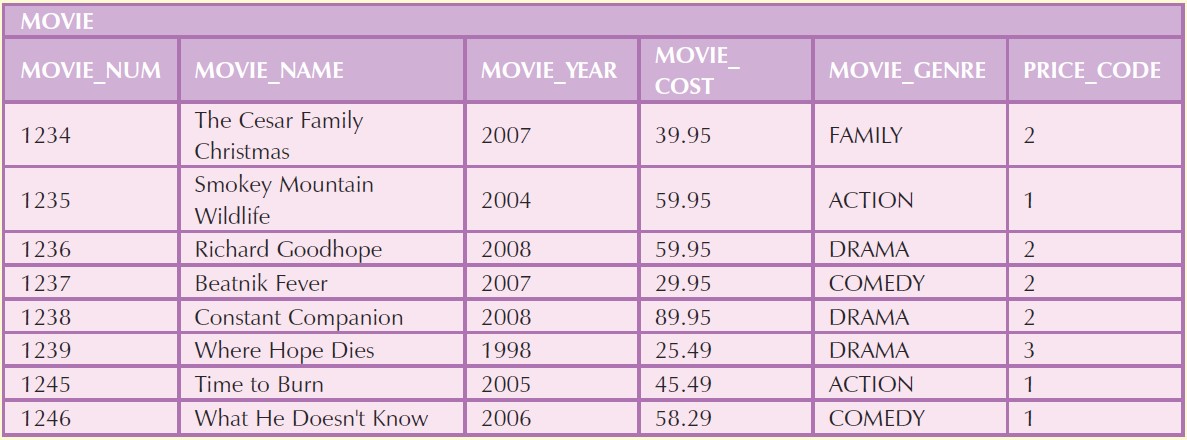
1. The following tables provide a very small portion of the data that will be kept in the database. This data needs to be inserted into the database for testing purposes. Write the INSERT commands necessary to place the following data in the tables that were created in Problem 1.

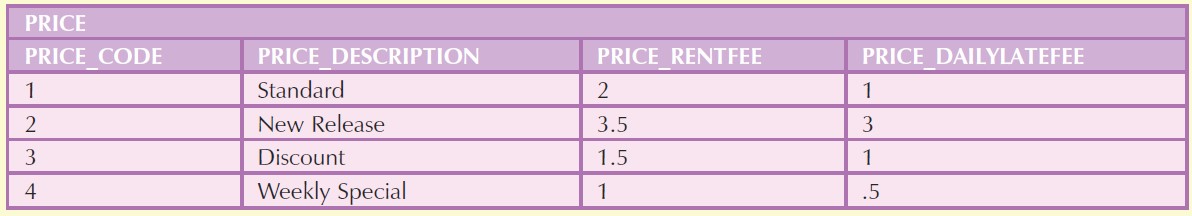












### MEMBERSHIP

For Questions 3–32, use the tables that were created in Problem 1 and the data that was loaded into those tables in Problem 2.

1. Write a query to display the movie title, movie year, and movie cost for all movies that contain the word “hope” anywhere in the title. Sort the results in ascending order by title. (The results are shown in figure P7.55.)

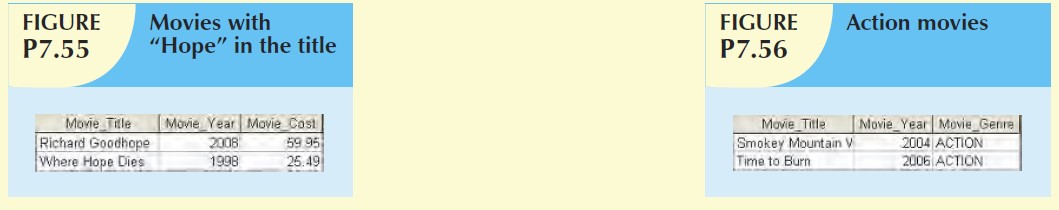
select movie\_title, movie\_year, movie\_cost from movie

where movie\_title like "%hope%“;

1. Write a query to display the movie title, movie year, and movie genre for all action movies. (The results are shown in Figure P7.56.)

select movie\_title, movie\_year, movie\_genre from movie

where movie\_genre = "action";



### LNAME MEM\_STREET MEM\_CITY

1. Write a query to display the movie number, movie title, and movie cost for all movies with a cost greater than $40. (The results are shown in Figure P7.57.)

select movie\_num, movie\_title, movie\_cost from movie

where movie\_cost > 40;

1. Write a query to display the movie number, movie title, movie cost, and movie genre for all movies that are either action or comedy movies and that have a cost that is less than $50. Sort the results in ascending order by genre. (The results are shown in Figure P7.58.)

select movie\_num, movie\_title, movie\_cost, movie\_genre from movie

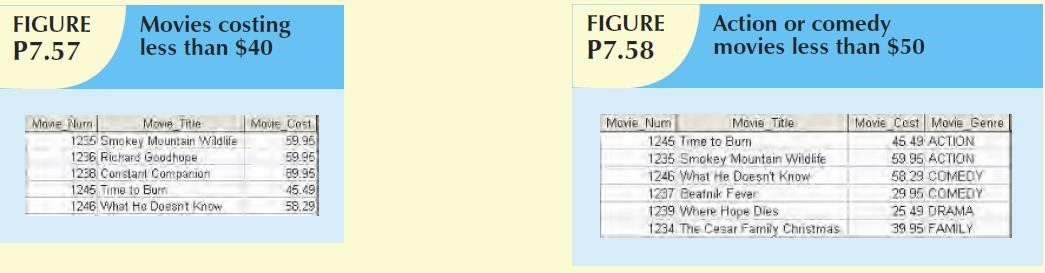
where movie\_genre = "action"

or movie\_genre = "comedy"

and movie\_cost < 50

order by movie\_genre;

### MEM\_ZIP ME



1. Write a query to display the movie number, and movie description for all movies where the movie description is a combination of the movie title, movie year, and movie genre with the movie year enclosed in parentheses. (The results are shown in Figure P7.59.)

select movie\_num as Movie\_Num, CONCAT(movie\_title, " (", movie\_year, ") ", movie\_genre) as Movie\_Description

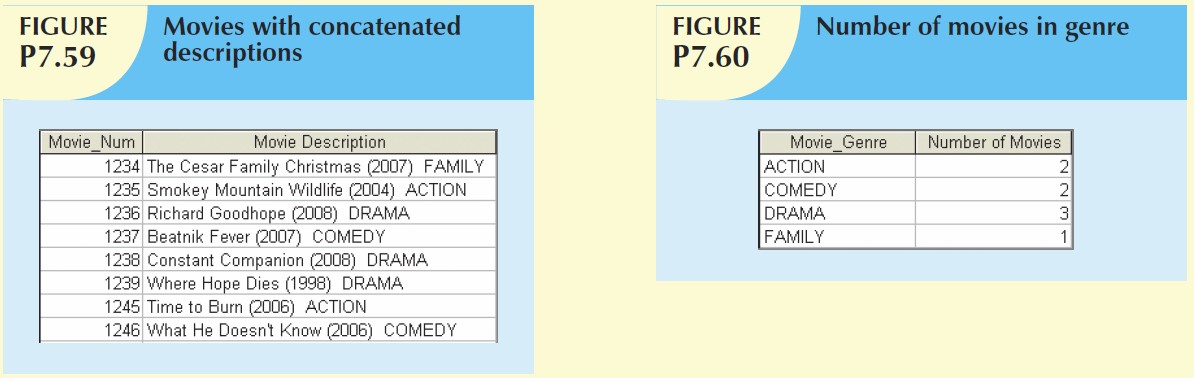
from movie;

1. Write a query to display the movie genre and the number of movies in each genre. (The results are shown in Figure P7.60.)

select movie\_genre as Movie\_Genre, count(movie\_title) as Number\_of\_Movies

from movie

group by movie\_genre;



1. Write a query to display the average cost of all of the movies. (The results are shown in Figure P7.61.)

select avg(movie\_cost) as Average\_movie\_cost

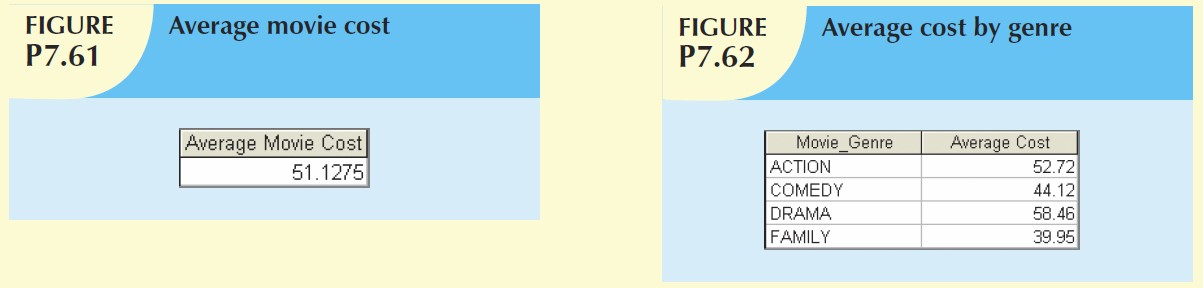
from movie;

1. Write a query to display the movie genre and average cost of movies in each genre. (The results are shown in Figure P7.62.)

select movie\_genre as Movie\_Genre, avg(movie\_cost) as Average\_Cost

from movie

group by movie\_genre;



1. Write a query to display the movie title, movie genre, price description, and price rental fee for all movies with a price code. (The results are shown in Figure P7.63.)

select m.movie\_title, m.movie\_genre, p.price\_description, p.price\_rentfee

from movie as m

natural join price as p

order by p.price\_description;

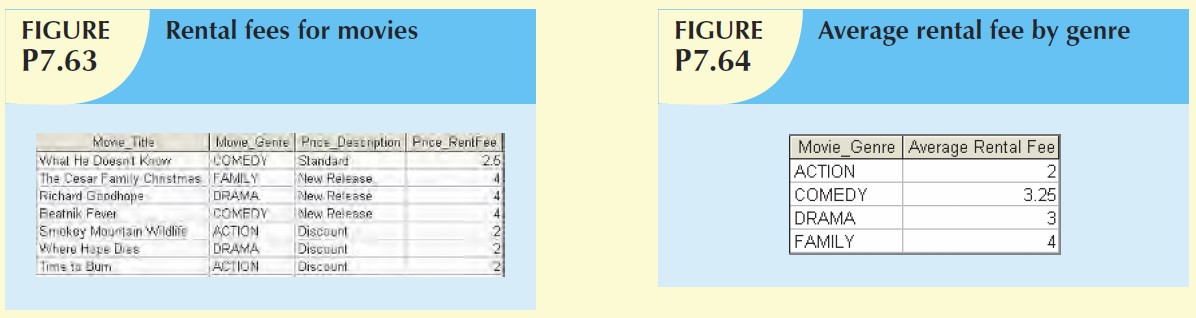
1. Write a query to display the movie genre and average price rental fee for movies in each genre that have a price. (The results are shown in Figure P7.64.)

select m.movie\_genre, avg(p.price\_rentfee)

from movie as m

natural join price as p

group by m.movie\_genre;



1. Write a query to display the movie title, movie year, and the movie cost divided by the price rental fee for each movie that has a price to determine the number of rentals it will take to break even on the purchase of the movie. (The results are shown in Figure P7.65.)

select m.movie\_title, m.movie\_year, (m.movie\_cost / p.price\_rentfee) as Breakeven\_Rentals

from movie as m

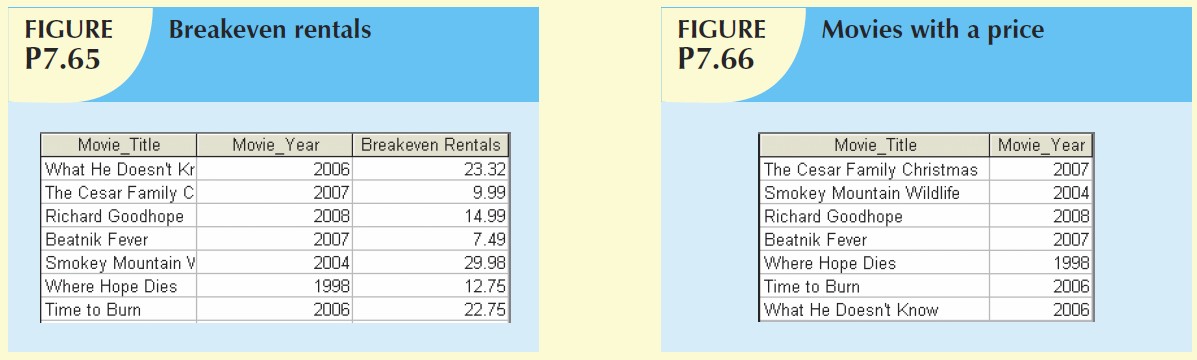
natural join price as p

group by m.movie\_title;

1. Write a query to display the movie title and movie year for all movies that have a price code. (The results are shown in Figure P7.66.)

select movie\_title, movie\_year from movie

where price\_code is not null;



1. Write a query to display the movie title, movie year, and movie cost for all movies that have a cost between $44.99 and $49.99. (The results are shown in Figure P7.67.)

select movie\_title, movie\_year, movie\_cost from movie

where 44.99 < movie\_cost < 49.99;

1. Write a query to display the movie title, movie year, price description, and price rental fee for all movies that are in the genres family, comedy, or drama. (The results are shown in Figure P7.68.)

select m.movie\_title, m.movie\_year, p.price\_description, p.price\_rentfee, m.movie\_genre

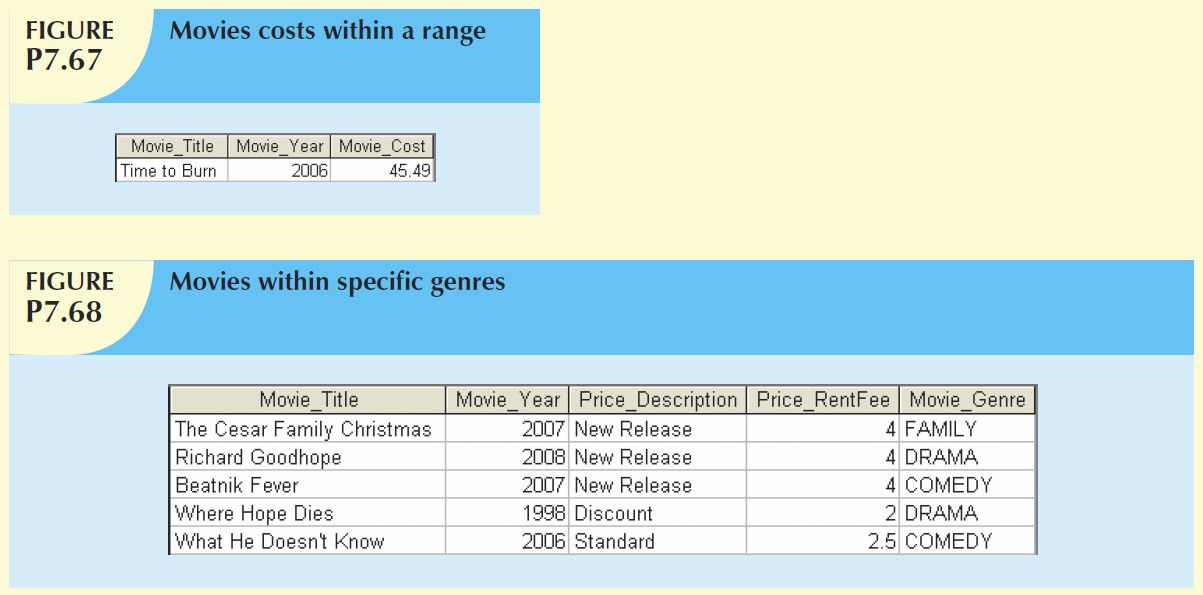
from movie as m

natural join price as p

where movie\_genre = "family"

or movie\_genre = "comedy"

or movie\_genre = "drama";



1. Write a query to display the minimum balance, maximum balance, and average balance for memberships that have a rental. (The results are shown in Figure P7.71.)

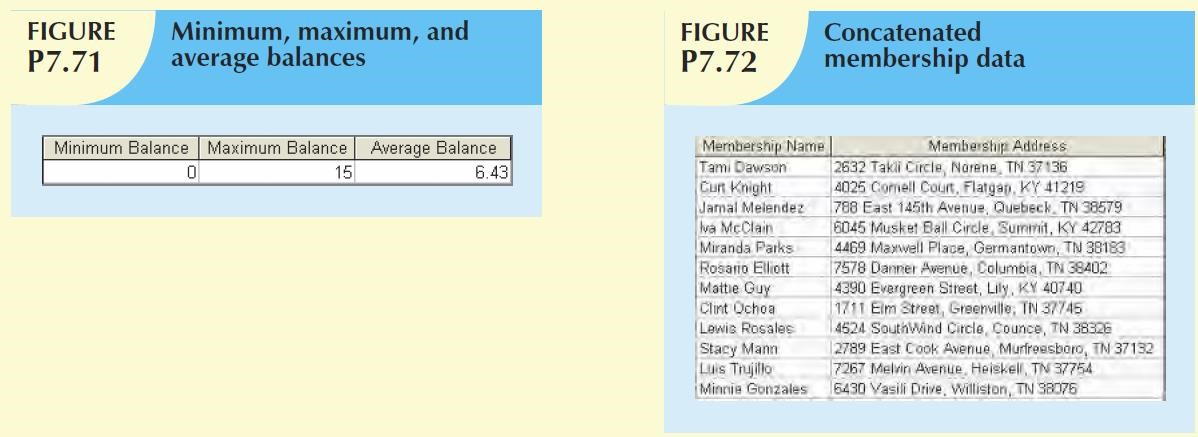
select min(mem\_balance), max(mem\_balance), avg(mem\_balance)

from membership;

1. Write a query to display the membership name (concatenate the first name and last name with a space between them into a single column), membership address (concatenate the street, city, state, and zip codes into a single column with spaces. (The results are shown in Figure P7.72.)

select concat(mem\_Fname, " ", mem\_Lname) as Membership\_Name, concat(mem\_street, ", ", mem\_city, ", ", mem\_state, ", ", mem\_zip) as Address

from membership;



1. Write a query to display the rental number, rental date, video number, movie title, due date, and return date for all videos that were returned after the due date. Sort the results by rental number and movie title. (The results are shown in Figure P7.73.)

select r.rent\_num, r.rent\_date, v.vid\_num, m.movie\_title, d.detail\_duedate, d.detail\_returndate

from rental as r

natural join video as v

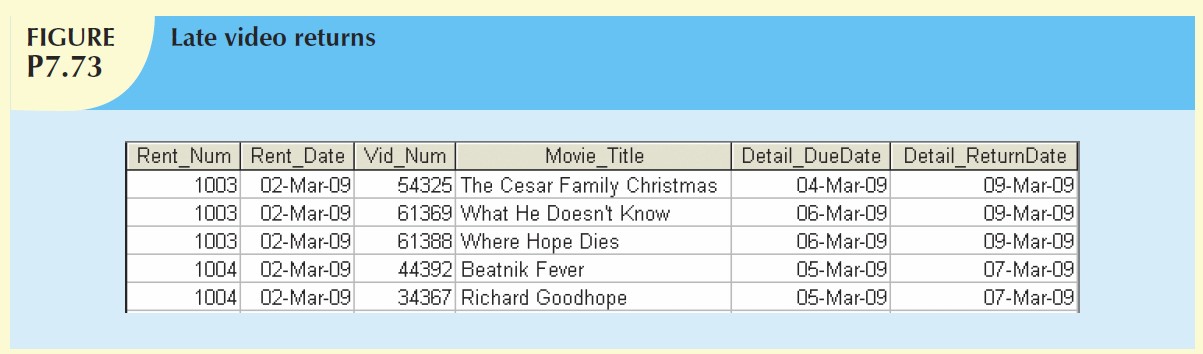
natural join movie as m

natural join detailrental as d

where d.detail\_duedate < d.detail\_returndate

group by m.movie\_title

order by r.rent\_num;



1. Write a query to display the rental number, rental date, video number, movie title, due date, return date, detail fee, and number of days past the due date that the video was returned for each video that was returned after the due date. Sort the results by rental number and movie title. (The results are shown in Figure P7.74.)

select r.rent\_num, r.rent\_date, v.vid\_num, m.movie\_title, d.detail\_duedate, d.detail\_returndate, d.detail\_fee, d.detail\_duedate - d.detail\_returndate as Days\_Past\_Due

from rental as r

natural join video as v

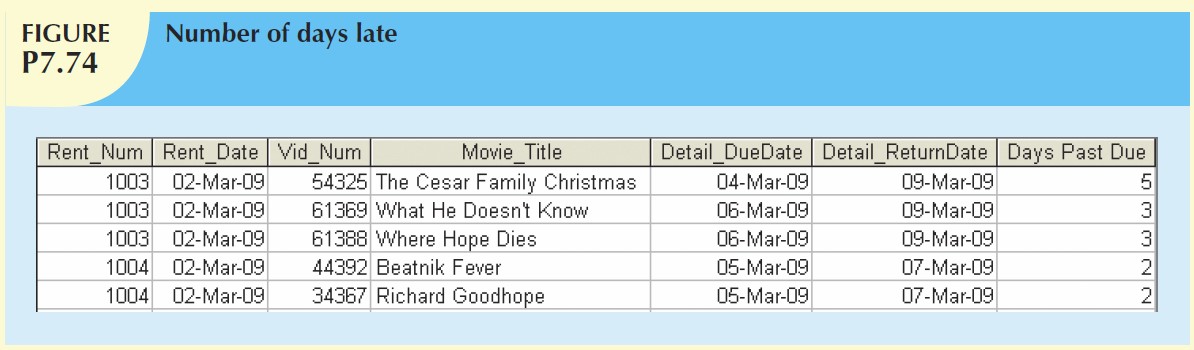
natural join movie as m

natural join detailrental as d

where d.detail\_duedate < d.detail\_returndate

group by m.movie\_title

order by r.rent\_num;



1. Write a query to display the rental number, rental date, movie title, and detail fee for each movie that was returned on or before the due date. (The results are shown in Figure P7.75.)

select r.rent\_num, r.rent\_date, m.movie\_title, d.detail\_fee

from rental as r

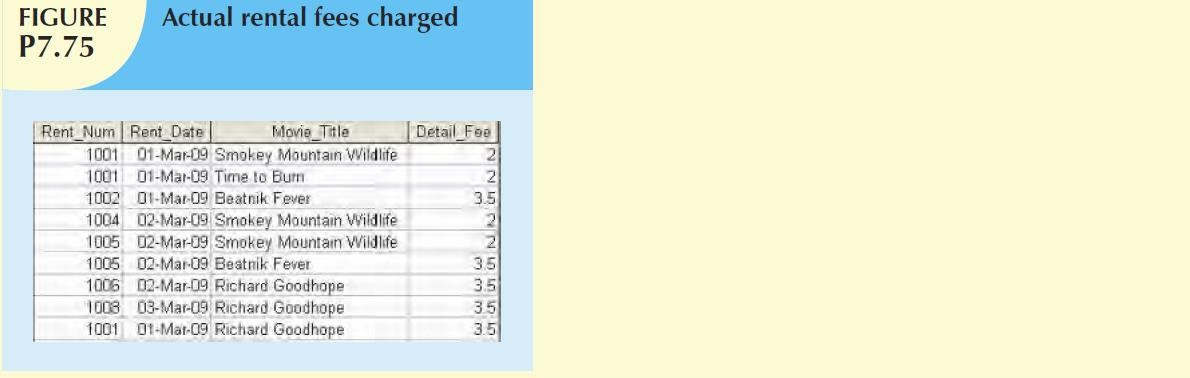
natural join movie as m

natural join detailrental as d

where d.detail\_duedate <= d.detail\_returndate

group by m.movie\_title

order by r.rent\_num;



1. Write a query to display the membership number, last name, first name, and total rental fees earned from that membership. (The results are shown in Figure P7.76.) The total rental fee is the sum of all of the detail fees (without the late fees) from all movies that the membership has rented.

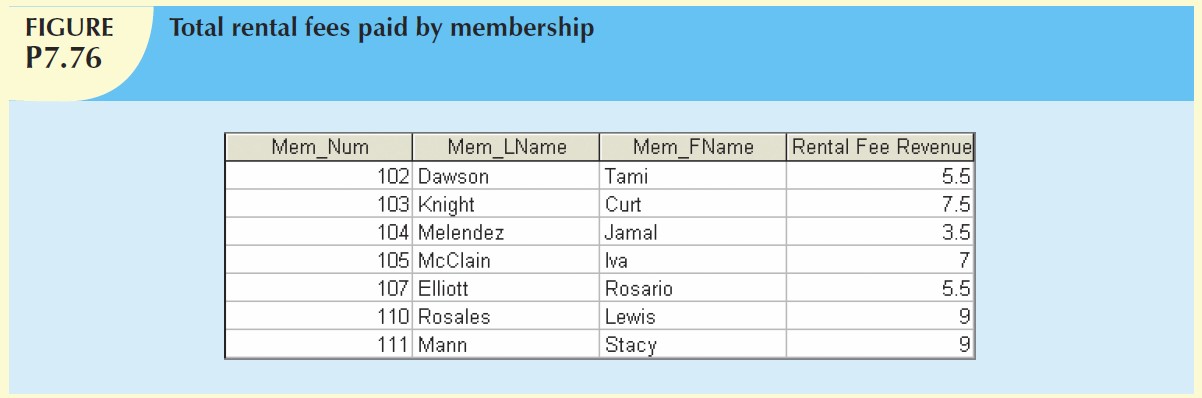
select m.mem\_num, m.mem\_Lname, m.mem\_Fname, sum(d.detail\_fee) as "Rental Fee Revenue"

from membership as m

natural join detailrental as d

natural join rental as r

group by m.mem\_num, m.mem\_Fname, m.mem\_Lname;

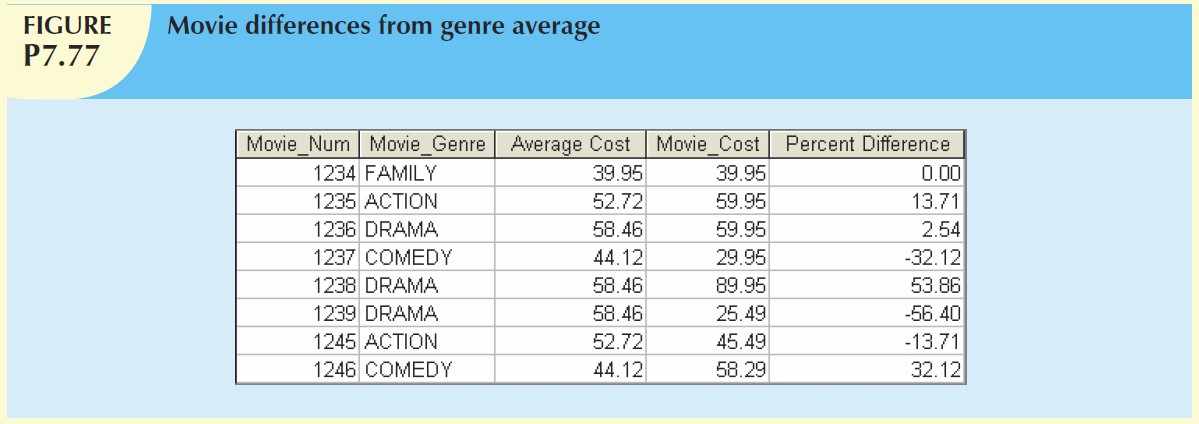


1. Write a query to display the movie number, movie genre, average movie cost of movies in that genre, movie cost of that individual movie, and the percentage difference between the average movie cost and the individual movie cost. (The results are shown in Figure P7.77.) (*Note:* The percentage difference is calculated as the cost of the individual movie minus the average cost of movies in that genre, divided by the average cost of movies in that genre multiplied by 100. For example, if the average cost of movies in the “family” genre is $25, if a given family movie cost $26, then the calculation would be ((26 – 25) / 25 \* 100), which would work out to be 4.00%. This indicates that this movie costs 4% more than the average family movie.)

select movie\_num, m.movie\_genre, avgcost as "average cost", movie\_cost, (movie\_cost - avgcost)/avgcost \* 100 as "percent difference"

from movie m, (select movie\_genre, avg(movie\_cost) as avgcost from movie group by movie\_genre) s

where m.movie\_genre = s.movie\_genre;



1. Alter the DETAILRENTAL table to include a derived attribute named DETAIL\_DAYSLATE to store integers up to 3 digits. The attribute should accept null values.

alter table detailrental

add detail\_dayslate numeric(3,0);

1. Alter the VIDEO table to include an attribute named VID\_STATUS to store character data up to 4 characters long. The attribute should not

accept null values. The attribute should have a constraint to enforce the domain (“IN”, “OUT”, and “LOST”) and have a default value of “IN”.

alter table video

add vid\_status varchar(4)

default 'IN' not null

check (vid\_status in ('IN', 'OUT', 'LOST'));

1. Update the VID\_STATUS attribute of the VIDEO table using a subquery to set the VID\_STATUS to “OUT” for all videos that have a null value in the DETAIL\_RETURNDATE attribute of the DETAILRENTAL table.

update video

set vid\_status = 'OUT'

where vid\_num in (select vid\_num from detailrental where detail\_returndate is null);

1. Alter the PRICE table to include an attribute named PRICE\_RENTDAYS to store integers up to 2 digits. The attribute should not accept null values, and should have a default value of 3.

alter table price

add price\_rentdays int(2) not null

default 3;

1. Update the PRICE table to place the values shown in the following table in the PRICE\_RENTDAYS attribute.

update price

set price\_rentdays = 5

where price\_code = 1;

update price

set price\_rentdays = 3

where price\_code = 2;

update price

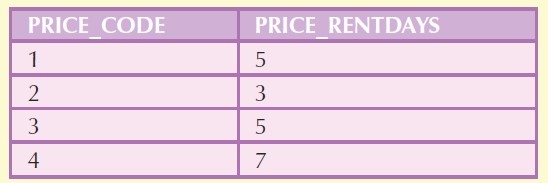
set price\_rentdays = 5

where price\_code = 3;

update price

set price\_rentdays = 7

where price\_code = 4;



1. Create a trigger named trg\_late\_return that will write the correct value to DETAIL\_DAYSLATE in the DETAILRENTAL table whenever a video is returned. The trigger should execute as a BEFORE trigger when the DETAIL\_RETURNDATE or DETAIL\_DUEDATE attributes are updated. The trigger should satisfy the following conditions.
   1. If the return date is null, then the days late should also be null.
   2. If the return date is not null, then the days late should determine if the video is returned late.
   3. If the return date is noon of the day after the due date or earlier, then the video is not considered late, and the days late should have a value of zero (0).
   4. If the return date is past noon of the day after the due date, then the video is considered late so the number of days late must be calculated and stored.

create trigger trg\_late\_returnbefore

update of detail\_returndate, detail\_duedate on detailrentalfor each row

begin

if :new.detail\_returndate is null

then :new.detail\_dayslate := null;

elseif trunc(:new.detail\_returndate) <= trunc(:new.detail\_duedate)

or (trunc(:new.detail\_returndate) = trunc(:new.detail\_duedate) + 1 and to\_char(:new.detail\_returndate, 'hh24:mi:ss') <= '12:00:00')

then :new.detail\_dayslate := 0;

else :new.detail\_dayslate := trunc(:new.detail\_returndate) -trunc(:new.detail\_duedate)

end if;

1. Create a trigger named trg\_mem\_balance that will maintain the correct value in the membership balance in the MEMBERSHIP table when videos are returned late. The trigger should execute as an AFTER trigger when the due date or return date attributes are updated in the DETAILRENTAL table. The trigger should satisfy the following conditions.
   1. Calculate the value of the late fee prior to the update that triggered this execution of the trigger. The value of the late fee is the days late times the daily late fee. If the previous value of the late fee was null, then treat it as zero (0).
   2. Calculate the value of the late fee after the update that triggered this execution of the trigger. If the value of the late fee is now null, then treat it as zero (0).
   3. Subtract the prior value of the late fee from the current value of the late fee to determine the change in late fee for this video rental.
   4. If the amount calculated in part c is not zero (0), then update the membership balance by the amount calculated for the membership associated the rental that this detail is a part of.

1. Create a stored procedure named prc\_new\_rental to insert new rows in the RENTAL table. The procedure should satisfy the following conditions.
   1. The membership number will be provided as a parameter.
   2. Use a Count () function to verify that the membership number exists in the MEMBERSHIP table. If it does not exist, then a message should be displayed stating that the membership does not exist and no data should be written to the database.
   3. If the membership does exist, then retrieve the membership balance and display a message stating the balance amount as the previous balance. (For example, if the membership has a balance of $5.00, then display “Previous balance: $5.00”.
   4. Insert a new row in the rental table using the sequence created in #42 above to generate the value for RENT\_NUM, the current system date for the value for RENT\_DATE, and the membership number provided as the value for MEM\_NUM.

1. Create a stored procedure named prc\_new\_detail to insert new rows in the DETAILRENTAL table. The procedure should satisfy the following requirements.
   1. The video number will be provided as a parameter.
   2. Verify that the video number exists in the VIDEO table. If it does not exist, then display a message that the video does not exist, and do not write any data to the database.
   3. If the video number does exist, then verify that the VID\_STATUS for that video is “IN”. If the status is not “IN”, then display a message that the return of the video must be entered before it can be rented again, and do not write any data to the database.
   4. If the status is “IN”, then retrieve the values of PRICE\_RENTFEE, PRICE\_DAILYLATEFEE, and PRICE\_RENTDAYS associated with the video from the PRICE table.
   5. Calculate the due date for the video rental by adding the number of days found in PRICE\_RENTDAYS above to 11:59:59PM (hours: minutes: seconds) on the current system date.
   6. Insert a new row in the DETAILRENTAL table using the previous value returned by rent\_num\_seq as the RENT\_NUM, the video number provided in the parameter as the VID\_NUM, the PRICE\_RENTFEE as the value for DETAIL\_FEE, the due date calculated

above for the DETAIL\_DUEDATE, PRICE\_DAILYLATEFEE as the value for DETAIL\_DAILYLATEFEE, and null for the DETAIL\_RETURNDATE.

*AGH TÆKNISKOLINN, 16-11-2017*