VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY | KATEDRA INFORMATIKY

INTRODUCTION TO DATABASE SYSTEMS

Collection of tasks

Petr Lukáš, Peter Chovanec, Radim Bača

November 10, 2020

Contents

Introduction

This document is created for practicing of SQL language. This document is categorized into the 5 categories, where each of them represents the topic of one practice from the subject Introduction to Database Systems. Each category contains approximately 30 tasks to solve. The first practice is dedicated to base usage of command SELECT, the second practice is focused on the joins of the tables, the third practice is focused on aggregation functions, the fourth practice is focused on set operations and the last practice is about complex queries containing subqueries. This document is published in two versions: version without solutions and version with solutions. Students work on the practice with the version without solutions. The version with solutions will be published after the practice.

Sincerely ask students to report any mistakes (unclear tasks, mistakes in solutions, unclear description of solution and others) to one of the following email addresses : `petr.lukas@vsb.cz`, `peter.chovanec@vsb.cz` or `radim.baca@vsb.cz`. Your help can improve the practices in next academic years.

Sakila Database

We use database of artificial movie rental called Sakila for the practices of subject Introduction to Database Systems. The database is originally designed for demonstration of SQL queries in database system MySQL[1]. In the last years, the versions for another database systems[2] have been published, e.g. Microsoft SQL Server. In our case, the scripts for the Microsoft SQL Server will be used. These scripts are published on the website of the subject `dbedu.cs.vsb.cz`. The data in the database have been slightly modified for the better demonstration of some SQL possibilities, i.e. some data have to be added or modified to get satisfying results of some SQL queries.

Relational Data Model

The structure of relation database is visualised by so-called E-R (Entity-Relationship) diagram. The E-R diagram of database Sakila is presented in Figure 1. We recommend students to print out the Figure, because we will work with it very often.

In Figure, we can see table `film` containing a list of all movies and table `actor` containing a list of all actors. These tables are joined by association table `film_actor`, therefore we have information which actor acts in each movie. There is a relationship M:N between film and actor, that means one actor can act in many movie and one movie can be acted by many actors. Similar situation is presented in the case of table `category` containing a list of all movie categories; it is joined with the table `film` by association table `film_category`. Therefore, one movie can be marked by more categories (horror, comedy, etc.) and vice versa. Moreover, there are two relationships N:1 between tables `film` and table `language` containing a list of all languages. The first relationship describes the real language of the movie, the second relationship describes the original language of the movie (in the case that movie has been dubbed).

We can continue with the description of table `inventory` containing a list of all movie copies. There is a relationship 1:N between tables `film` and `inventory`, it means, the movie rental can own one movie in many copies. Table `rental` contains list of all movie rents. Each rent is associated to some specified movie copy, to some specified customer in table `customer` and it is processed by some specified employee in table `staff`. Therefore, there are relationships N:1 between table `rental` and tables `inventory`, `customer` and `staff`. Table `payment` contains a list of all payments for the rents. Each payment is done by some customer in table `customer` and processed by some employee in table `staff`. Let us note, that not all payments represent payments for the movie rents. Some of them represents e.g. payments for subscription.

The database contains also tables `country`, `city` and `address` which are joined by relationships 1:N, i.e. one country has many cities and one city has many addresses. Table `address` has relationship 1:1 to tables `customer`, `store` and `staff`, i.e. each customer/store/employee can have only one address.

[1] `https://dev.mysql.com/doc/sakila/en/`

[2] `https://github.com/jOOQ/jOOQ/tree/master/jOOQ-examples/Sakila`

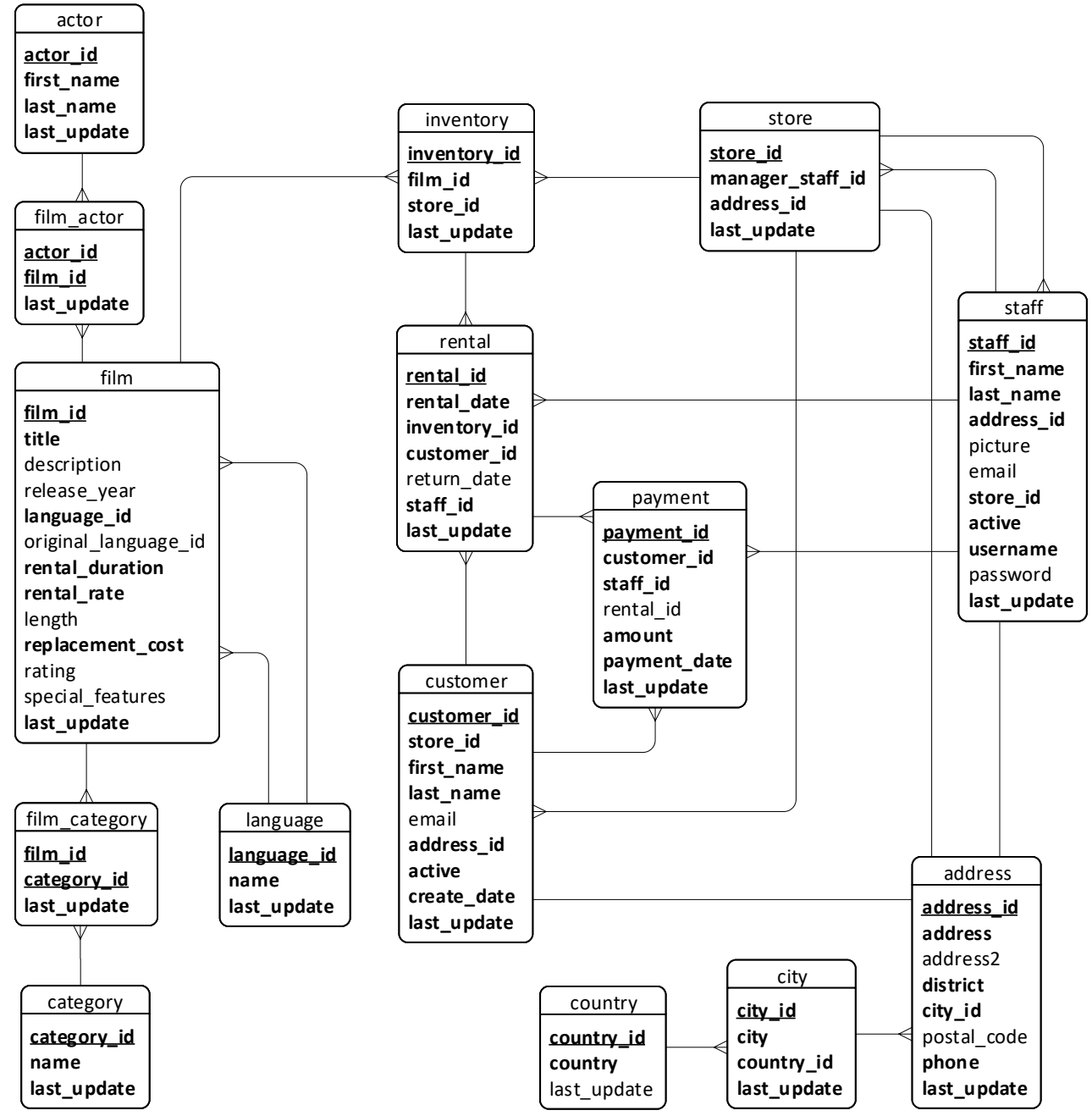

Figure 1: E-R diagram of database Sakila

Data Dictionary

Although, the name of the tables and attributes in database Sakila are mostly self-describing, we present their detail description in the form of data dictionary.

NULL an information whether the column is optional or not `NULL`
PK an information whether the column is a primary key
FK an information whether the column is a foreign key

RENTAL

the rental table contains one row for each rental of each inventory item with information about who rented what item, when it was rented, and when it was returned

column	data type	NULL	PK	FK	description
rental_id	integer number	no	yes	no	a surrogate primary key
rental_date	date and time	no	no	no	the date and time that the item was rented
inventory_id	integer number	no	no	yes	the item being rented
customer_id	integer number	no	no	yes	the customer renting the item
return_date	date and time	yes	no	no	the date and time the item was returned
staff_id	integer number	no	no	yes	the staff member who processed the rental
last_update	date and time	no	no	no	the time that the row was created or most recently updated

ACTOR

the actor table lists information for all actors

column	data type	NULL	PK	FK	description
actor_id	integer number	no	yes	no	a surrogate primary key
first_name	string, max. 45 chars.	no	no	no	the actor's first name
last_name	string, max. 45 chars.	no	no	no	the actor's last name
last_update	date and time	no	no	no	the time that the row was created or most recently updated

COUNTRY

the country table contains a list of countries

column	data type	NULL	PK	FK	description
country_id	integer number	no	yes	no	a surrogate primary key
country	string, max. 50 chars.	no	no	no	the name of the country
last_update	date and time	yes	no	no	the time that the row was created or most recently updated

CITY

the city table contains a list of cities

column	data type	NULL	PK	FK	description
city_id	integer number	no	yes	no	a surrogate primary key
city	string, max. 50 chars.	no	no	no	the name of the city
country_id	integer number	no	no	yes	a foreign key identifying the country that the city belongs to
last_update	date and time	no	no	no	the time that the row was created or most recently updated

ADDRESS

the address table contains address information for customers, staff, and stores

column	data type	NULL	PK	FK	description
address_id	integer number	no	yes	no	a surrogate primary key
address	string, max. 50 chars.	no	no	no	the first line of an address
address2	string, max. 50 chars.	yes	no	no	an optional second line of an address
district	string, max. 20 chars.	no	no	no	the region of an address, this may be a state, province, prefecture, etc.
city_id	integer number	no	no	yes	a foreign key pointing to the city table
postal_code	string, max. 10 chars.	yes	no	no	the postal code or ZIP code of the address (where applicable)
phone	string, max. 20 chars.	no	no	no	the telephone number for the address
last_update	date and time	no	no	no	the time that the row was created or most recently updated

LANGUAGE

the language table is a lookup table listing the possible languages that films can have for their language and original language values

column	data type	NULL	PK	FK	description
language_id	integer number	no	yes	no	a surrogate primary key
name	string, max. 20 chars.	no	no	no	the English name of the language
last_update	date and time	no	no	no	the time that the row was created or most recently updated

CATEGORY

the category table lists the categories that can be assigned to a film

column	data type	NULL	PK	FK	description
category_id	integer number	no	yes	no	a surrogate primary key
name	string, max. 25 chars.	no	no	no	the name of the category
last_update	date and time	no	no	no	the time that the row was created or most recently updated

CUSTOMER

the customer table contains a list of all customers

column	data type	NULL	PK	FK	description
customer_id	integer number	no	yes	no	a surrogate primary key
store_id	integer number	no	no	yes	a foreign key identifying the customer's home store
first_name	string, max. 45 chars.	no	no	no	the customer's first name
last_name	string, max. 45 chars.	no	no	no	the customer's last name
email	string, max. 50 chars.	yes	no	no	the customer's email address
address_id	integer number	no	no	yes	a foreign key identifying the customer's address in the address table
active	string, max. 1 chars.	no	no	no	whether the customer is an active customer
create_date	date and time	no	no	no	the date the customer was added to the system
last_update	date and time	no	no	no	the time that the row was created or most recently updated

FILM

the film table is a list of all films potentially in stock in the stores

column	data type	NULL	PK	FK	description
film_id	integer number	no	yes	no	a surrogate primary key
title	string, max. 255 chars.	no	no	no	the title of the film
description	text	yes	no	no	a short description or plot summary of the film
release_year	string, max. 4 chars.	yes	no	no	the year in which the movie was released
language_id	integer number	no	no	yes	a foreign key pointing at the language table; identifies the language of the film
original_language_id	integer number	yes	no	yes	a foreign key pointing at the language table; identifies the original language of the film
rental_duration	integer number	no	no	no	the length of the rental period, in days
rental_rate	decimal number	no	no	no	the cost to rent the film for the period specified in the rental_duration column
length	integer number	yes	no	no	the duration of the film, in minutes
replacement_cost	decimal number	no	no	no	the amount charged to the customer if the film is not returned or is returned in a damaged state
rating	string, max. 10 chars.	yes	no	no	the MPAA rating assigned to the film
special_features	string, max. 255 chars.	yes	no	no	lists which common special features are included on the DVD
last_update	date and time	no	no	no	the time that the row was created or most recently updated

FILM_ACTOR

the film_actor table is used to support a many-to-many relationship between films and actors

column	data type	NULL	PK	FK	description
actor_id	integer number	no	yes	yes	the film_actor table is used to support a many-to-many relationship between films and actors
film_id	integer number	no	yes	yes	the film_actor table is used to support a many-to-many relationship between films and actors
last_update	date and time	no	no	no	the film_actor table is used to support a many-to-many relationship between films and actors

FILM_CATEGORY

the film_category table is used to support a many-to-many relationship between films and categories

column	data type	NULL	PK	FK	description
film_id	integer number	no	yes	yes	the film_category table is used to support a many-to-many relationship between films and categories
category_id	integer number	no	yes	yes	the film_category table is used to support a many-to-many relationship between films and categories
last_update	date and time	no	no	no	the film_category table is used to support a many-to-many relationship between films and categories

INVENTORY

the inventory table contains one row for each copy of a given film in a given store

column	data type	NULL	PK	FK	description
inventory_id	integer number	no	yes	no	a surrogate primary key
film_id	integer number	no	no	yes	a foreign key pointing to the film this item represents
store_id	integer number	no	no	yes	a foreign key pointing to the store stocking this item
last_update	date and time	no	no	no	the time that the row was created or most recently updated

STAFF

the staff table lists all staff members, including information on email address, login information, and picture

column	data type	NULL	PK	FK	description
staff_id	integer number	no	yes	no	a surrogate primary key
first_name	string, max. 45 chars.	no	no	no	the first name of the staff member
last_name	string, max. 45 chars.	no	no	no	the last name of the staff member
address_id	integer number	no	no	yes	a foreign key to the staff member's address in the address table
picture	image	yes	no	no	a BLOB containing a photograph of the employee
email	string, max. 50 chars.	yes	no	no	the staff member's email address
store_id	integer number	no	no	yes	the staff member's home store
active	bit	no	no	no	whether this is an active employee
username	string, max. 16 chars.	no	no	no	the user name used by the staff member to access the rental system
password	string, max. 40 chars.	yes	no	no	the SHA1 hashed password used by the staff member to access the rental system
last_update	date and time	no	no	no	the time that the row was created or most recently updated

STORE

the store table lists all stores in the system

column	data type	NULL	PK	FK	description
store_id	integer number	no	yes	no	a surrogate primary key
manager_staff_id	integer number	no	no	yes	a foreign key identifying the manager of this store
address_id	integer number	no	no	yes	a foreign key identifying the address of this store
last_update	date and time	no	no	no	the time that the row was created or most recently updated

PAYMENT

the payment table records each payment made by a customer, with information such as the amount and the rental being paid for (when applicable)

column	data type	NULL	PK	FK	description
payment_id	integer number	no	yes	no	a surrogate primary key
customer_id	integer number	no	no	yes	the customer whose balance the payment is being applied to
staff_id	integer number	no	no	yes	the staff member who processed the payment
rental_id	integer number	yes	no	yes	the rental that the payment is being applied to
amount	decimal number	no	no	no	the amount of the payment
payment_date	date and time	no	no	no	the date the payment was processed
last_update	date and time	no	no	no	the time that the row was created or most recently updated

1 SQL Basics, command SELECT

This practice will be about base syntax of the command SELECT. All queries will be processed over one table. Queries will be oriented on simple selection, projection, conditions, base date/-time/text functions and so-called aggregation functions.

1. Select email adresses of all inactive customers.
2. Select names and description of all movies with classification G (attribute `rating`). The result has to be ordered by the name of movie.
3. Select all information about payments since the year 2006 and payments with amount lower than 2.
4. Select all movies classified as G or PG.
5. Select all movies classified as G, PG or PG-13.
6. Select description of all movies not classified as G, PG and PG-13.
7. Select all information about movies longer that 50 minutes that have rental duration 3 or 5 days.
8. Select names of all movies longer than 70 minutes and names containing word 'RAIN-BOW' or beginning on word 'TEXAS'.
9. Select names of all movies which description contains word, their length is between 80 and 90 minutes and standard rental duration is odd number.
10. Select features (attribute `special_features`) of all movies where cost of replacement is between 14 and 16. Ensure that each feature occurs only once in the result and order the features alphabetically. Why is the result automatically ordered even if the ORDER BY is not used?
11. Select all information about movies with standard rental duration lower than 4 days or classified as PG. The result can not contain movies satisfying both condition.
12. Select all information about addresses with filled postal code.
13. Select IDs of all customers with some currently rented movie. Do you know how to count those customers?
14. Select year, month and day in separate columns of each payment in the database. Name the columns as pay_year, pay_month and pay_day.
15. Select movies with the length of their name not equal to 20 characters.
16. Select duration (in minutes) of each rent in the database. Name this column as duration [min.].
17. Select full name in one column for each active customer. Result has to contain two columns – `customer_id` and `full_name`.

18. Select zip code for each address in the database. In the case of null zip code print out text '(empty)'.
19. Select interval from – to (it means both dates in one column) for all closed rents (closed rent has filled return date).
20. Select interval from – to (it means both dates in one column) for all rents. If the rent is not closed yet, print only date of rent.
21. Select number of all movies in the database.
22. Select number of various movie classification (attribute `rating`).
23. Select number of all addresses, number of addresses with filled zip code and number of various zip codes using one query.
24. Select minimal, maximal and average length of all movies. Check if the average length is equal to ratio of summary length of all movies and total number of movies in the database.
25. Select number and sum of all payments of the year 2005.
26. Select total number of characters in names of all movies.

2 Table Joins

The first practice has been focused on queries over one table. However, more tables are usually needed in query to get a required result. In this practice, we show how to join tables in queries. We focus on inner joins and left outer joins. All tasks of this practice have to be solved without aggregation function, subqueries and constructions `IN/EXISTS`. All tasks have to be solved only by adequate join of several tables and restriction of redundant data in a result by code word `DISTINCT`.

1. Select all information about cities including information about the countries, where are the cities located.
2. Select names of all movies including the names of their language.
3. Select IDs of all rents of customer with surname SIMPSON.
4. Select address (attribute `address` in table `address`) of customer with surname SIMPSON. Compare the number of records in the result with the previous task.
5. Select name and surname of all customers including their addresses, zip codes and cities.
6. Select name and surname of all customers including their cities.
7. Select IDs of all rents including name of the staff, name of the customer and title of the movie.
8. Select all movies (their titles) together with the actors playing in them (their names and surnames). How many records will be in the result of this query?
9. Select all actors (their names and surnames) together with their movies. What is the difference in comparison with previous query? What we can say about inner joins?
10. Select titles of all movies in the category 'Horror'.
11. Select all stores (their IDs) together with their managers (their names and surnames). Moreover, select addresses of stores and addresses of managers (attribute `address` in table `address`). As a last step, append the cities and countries of stores and managers to the result.
12. Select all movies (their IDs and titles) together with IDs of actors playing in them and IDs of categories belonging to. It means, a result of the query has to contain attributes `film_id`, `actor_id` and `category_id` and it has to be order by `film_id`.
13. Select all combinations of actors and categories (their IDs) where specified actors played in a movie of specified category. Result order by ID of actor. Consequently, extend result by the names and surnames of actors and by the names of categories.
14. Select names of movies that rental owns in at least one copy.
15. Select the actors playing in at least one comedy (category 'Comedy').

16. Select names of customers from Italy that borrowed movie with title MOTIONS DETAILS.

17. Select names of customers with the currently borrowed movie with actor SEAN GUINESS.

18. Select IDs and amounts of all payments together with the date of rental (attribute `rental_date` in table `rental`). In the case of payments not linked to any rent, print empty date of rental (`NULL`).

19. Select all languages together with the list of all movies catched in the specified language for each of them. Ensure that all languages are in the result.

20. Select all movies (their IDs and titles) together with their languages and original languages.

21. Select names of movies borrowed by customer TIM CARY and names of movies 48 minutes long.

22. Select names of movies that rental does not own (it means they are not in table `inventory`).

23. Select name of customer that did not pay for some rent.

24. Select all movies together with the name of language. The language has to be in result only, if it starts with letter 'I', otherwise print out value `NULL`.

25. Select all customers together with IDs of their payments higher than 9. In the case of customers without such payment, print out value `NULL`.

26. Select all rents (their IDs) together with the titles of movies (but only if they contain letter 'U') and with cities and countries of customer (but only if customer address contains letter 'A'). If the value does not satisfy the condition, print out value `NULL`.

27. Select all pairs movie title - customer surname where specified customer borrowed specified movie. In the case of rents after 01.01.2006, the customer surnamen has to be empty (it means `NULL`). Ensure that result do not contain redundant data.

3 Aggregate Functions and Group By

We already met with aggregate functions on the first practice, where we used them to get one row containing one or more calculated values. This practice will show use that aggregate functions can be used not only for complete data table but also for some groups of records in them. Consequently, the result will not be only one row, but more rows grouping records on the basis of some conditions. At the beginning, we will start with aggregate functions over one table, and then we will use your experiences from previous practice and we will use query over more tables.

1. Select the number of movies of particular classifications (attribute `rating`).
2. Select the number of surnames for particular customers (their IDs).
3. Select customer IDs ordered by the total amount of their payments. Customers without any payment will not be in the result.
4. Select number of actors with the specified name and surname of each actors name and surname. The result must be ordered by the number descendingly.
5. Select total amount of all payments for particular years and months. The result must be ordered by years and months.
6. Select stores (their IDs) with more than 2 300 movie copies.
7. Select the shortest movie per language ID and select only those language IDs where the shortest movie is longer than 46 minutes.
8. Select years and months when total amount of payments was higher than 20 000.
9. Let us consider just movies shorter than 50 minutes. We are interested in the total length per the rating, and we want only those ratings where the total length is higher than 250 minutes. The result must be ordered alphabetically.
10. Select the number of movies per language ID. The result *will not* contain languages without a movie.
11. Select the number of movies per language name. The result *will not* contain languages without any movie.
12. Select the number of movies per language name. The result *will* contain languages without any movie.
13. Select number of rentals per customer (print out his ID, first name and surname).
14. Select all customers (their IDs, first names and surnames) and how many *different* movies they rented.
15. Select names and surnames of actors acting in more than 20 movies.
16. Select all customers together with the informations: how much money they paid for rentals in total, how much money they paid for one rental maximally, minimally and in average.

17. Select average length of movie per movie category. Include *all* categories!

18. Select how much customers spent for rentals of particular movies. Select only movies with the total rental amount higher than 100.

19. Select the number of *different* movie categories per actor. Select the actor ID, first name and last name.

20. Select addresses, cities and countries of customers which borrowed movies with together at least 40 different actors.

21. Select ID and title of all movies with category 'Horror' together with the number of different cities of customers that borrowed them.

22. Select all customers from Poland together with the number of different categories of the movies that they borrowed.

23. Select names of all languages together with the number of movies longer than 350 minutes catched in those languages.

24. Select all customers together with information how much they paid for rentals started in june.

25. Select names of all categories ordered by the number of movies catched in language starting with letter 'E'.

26. Select titles of movies shorter than 50 minutes which customers with surname BELL borrowed exactly 1x.

4 Set Operations and Quantifiers

Many tasks is possible to solve without so-called subqueries; it means clause `SELECT` is included in the query exactly once. This practice is focused on the constructions `IN`, `EXISTS`, `ANY` and `ALL` that require an application of subqueries. Although many of the following tasks is possible to solve also by aggregate functions, use mentioned constructions instead. All tasks in the practice is possible to solve without aggregate functions and data grouping. In the real world (and also on the SQL test) it will be up to you, if you will choose aggregate functions or subqueries to solve the tasks.

1. Select IDs and titles of the movies of actor with ID = 1. The query has to be solved without `JOIN`.
2. Select IDs of the movies of actor with ID = 1.
3. Select IDs and titles of the movies in which plays actor with ID = 1 as well as actor with ID = 10.
4. Select IDs and titles of the movies in which plays actor with ID = 1 or actor with ID = 10.
5. Select IDs of the movies in which did not play actor with ID = 1.
6. Select IDs and titles of the movies in which plays actor with ID = 1 or actor with ID = 10, but not both together.
7. Select IDs and titles of the movies in which plays actor PENELOPE GUINESS as well as actor CHRISTIAN GABLE.
8. Select IDs and titles of the movies in which did not play actor PENELOPE GUINESS.
9. Select customers (their IDs and names) which borrowed all movies from the following list: ENEMY ODDS, POLLOCK DELIVERANCE a FALCON VOLUME.
10. Select customers (their IDs and names) which borrowed movie GRIT CLOCKWORK in May as well as in June (of arbitrary year).
11. Select names and surnames of the customers which have the same surname as some actor.
12. Select titles of the movies with same length as another movies.
13. Select titles of the movies shorter than any movie of actor BURT POSEY.
14. Select names of the actors playing in any movie shorter than 50 minutes.
15. Select the movies rented at least twice.
16. Select the movies rented by at least two different customers.
17. Select the customers which borrowed at least two different movies at once (at the same moment).
18. Select customers (their names) which borrowed movie GRIT CLOCKWORK in May as well as in June of the same year.

19. Select the movies (their titles) shorter than all movies of actor BURT POSEY.

20. Select name of the actors which play only in movies shorter than 180 minutes.

21. Select the customers (their names) which never borrowed more than 3 movies in the same month. Use aggregate functions and group by to get number of rents.

22. Select the customers (their names) which borrowed movies only during summer (it means in the July and August).

23. Select the customers which have always returned the borrowed movies within 8 days. Ignore rentals that the customer has not returned yet.

24. Select the customers whose all rentals were longer than one day and they borrowed a movie starring DEBBIE AKROYD.

25. Select the names and surnames of customers who have made exactly one rent.

26. Select titles of the movies where only one actor plays.

27. Select customers who have always borrowed only the same movie.

28. Select the titles of movies that have ever been rented by customers which have never rented another movie.

29. Select all customers (names and surnames) and languages if the customer only rented movies in that language.

30. Select titles of the movies that have only been rented by customers who have never rented another movie.

31. Select names and surnames of the customers which have always borrowed only movies where the actor CHRISTIAN GABLE starred.

32. Select the actors which have always played only in a movie owned by rental in at least three copies. Use aggregate function to get the number of copies in the inventory. ční funkci.

33. Select the movies whose all copies have been rented at least 4x. Use aggregate function to get the number of copies in the inventory.

34. Select the actors (their names) whose all movies are longer than the movies where the actor CHRISTIAN GABLE starred.

35. Select the actors whose movies, longer than 180 minutes, have been borrowed by customers from the same country.

5 Subqueries

The last practice from SQL language is focused on the subqueries in general. The subqueries can solve many complex task in very elegant way.

1. Select the number of actors in a movie and the number of categories of a movie for each movie in the database.
2. Select the number of borrowings lasting less than 5 days and the number of borrowings lasting less than 7 days for for each customer.
3. Select the number of copies (it means items in the store) of the English and French films for each store.
4. Select following information for each movie:
   (a) the number of actors in the movie,
   (b) the number of different customers who rented the movie in August,
   (c) the average amount of payment for your movie rental.
5. Select customers with more than 5 payments in June and the longest movie they rented has at least 185 minutes.
6. Select customers whose payments are with amount higher than 4 in the most cases.
7. Select actors playing in comedies two times more often than in horror movies.
8. Select the actors playing most often in movies longer than 150 minutes,it means they play more often in movies longer than 150 minutes than in other movies.
9. Select customers whose total payments are less than they should pay according to attributes the `film.rental_duration`, `film.rental_rate` and difference between attributes `rental_date` and `return_date`. You can ignore non-returned rents.
10. Select customers borrowing movies with actor TOM MCKELLEN more often than movies with actor GROUCHO SINATRA.
11. Select customers renting only movies in english language together with information how many rents they have.
12. Select customers who rented a movie with at least 15 actors together with the total amount of the payments they made.
13. Select the name of the longest movie(s).
14. Select the name of the longest movie(s) for each rating (attribute `film.rating`).
15. For each customer, find the last movie he rented. Sort the result alphabetically by last name and first name of the customers.
16. Select all actors (their name and surname) together with their longest movie.

17. Select all movies together with the customers who have borrowed them for the longest time (within one rent).
18. For each customer, select the last borrowed movie starring actor PENELOPE GUINESS. If the customer has never rented a movie with PENELOPE GUINESS, the customer will not be selected. Sort the result by customer ID.
19. List customers who have borrowed both the shortest and the longest movie.
20. Select the actors who played at least 2 times in the longest film.
21. Select movies that at least two customers rented for the last time.
22. Select all actors together with the average number of rents for the movies in which they play.
23. For each movie classification (attribute texttt film.rating), select the largest number of actors playing in the movie of that classification.
24. Select the most frequently cast actors, it means the actors who play in the largest number of movies. The number of movies the actor plays will be included in the result.
25. Select customers with the most rents.
26. Select the titles of movies that have been rented the most times. The number of rents will be included in the result.
27. Select the customers who made the most payments. The highest number of payments should be included in the result.
28. Select titles of the movies with number of rents higher than average number of rents.
29. Select the actors playing most often in moviess longer than 150 minutes, it means in movies with a length over 150 minutes, they are the most frequently cast actors.
30. Select the customers with the biggest difference between the minimum and maximum payment for a movie rent in June. The difference will be included in the result.
31. Select movies that have been rented by one customer the most times.
32. List customers borrowing the same movie the most times.
33. For each city, select the customer with the most rents.
34. Select all customers together with the title of the movie most often borrowed by him and the number of rents of this movie. Ignore customers without rents.
35. Select all categories together with their movies with the lowest number of rents.
36. Select all categories together with the most frequently cast actors in the movies of those categories.
37. Select all customers together with their favorite actor, it means the actor who played in the most different films the customer has borrowed. Ignore customers without rents.

6 Commands for data modification and definition

So far, we have not made any modifications to our database. We used only SELECT statements, whose possibilities are huge, but the data itself and the structure of the database remain intact. Today, on the contrary, we will show commands belonging to the category of DML (Data Manipulation Language) for editing the content of tables and commands belonging to the category of DDL (Data Definition Language) for editing the structure of tables.

This practise will differ slightly in structure from the previous ones for several reasons. Some tasks will consist of several points that need to be solved in the exact order. Furthermore, if it is not specified directly, you can solve tasks with multiple SQL statements, which you will run sequentially.

Before we start solving the tasks, note that while the syntax of the SELECT statement is almost the same across different DBMS (the ANSI SQL standard is followed), there are often slight differences between the commands in the DML and DDL categories. In this collection we will show the syntax for Microsoft SQL Server. Generally, the solution of problems for other relational DBMS will not differ.

1. (a) Insert a new actor named Arnold Schwarzenegger into the database. Leave the default value for the last record update (`last_update`) (i.e. do not set this value).
   (b) Insert the movie Terminator into the database. Find out the description and length of the film, for example, in the IMDB database[3]. Set the language of the movie to English, the standard rental period to 3 days and the price to 1.99. Other attributes will be left blank or set to the default value.
   (c) Update the database so that actor Arnold Schwarzenegger plays in the movie Terminator. Find out the actor ID and film ID in advance by suitable queries.
   (d) Set the Terminator movie in the 'Action' and 'Sci-Fi' categories. Find out the IDs of the relevant records in advance with suitable queries.
   (e) Put the Terminator movie in the 'Comedy' category. Solve the task using one command with subqueries. You have to avoid manually writing constants for IDs of movie and category. Find the required IDs using (`film.title` and `category.name`).
   (f) Set the rental price of the Terminator movie to 2.99. At the same time update the `last_update` attribute to the current timestamp.

2. (a) Create employees with your name and address (address information can of course be fictional). The username will be your login and you will be included in the warehouse with ID = 2. Find out the necessary constants for foreign keys in advance with suitable queries.
   (b) Create the address of our university in the database.
   (c) Create a new store at our university address. You will be the manager in the new store.
   (d) For each movie that the rental company owns in at least one copy, move its copy with the highest ID to the new store (see previous task).

[3] `https://www.imdb.com/title/tt0088247/`

3. Increase the rental price of all films with the actor ZERO CAGE by 10 %. Solve the task with one command without writing the constant for the actor ID (the actor will be identified by his name).
4. Set the original language to `NULL` to all movies whose original language (`original_language`) is Mandarin. Avoid select the ID for the language in separate query.
5. For each film with GROUCHO SINATRA, insert one new copy into the `inventory` table. All these new copies will be placed in the store with ID = 2. Leave the date of the last update of the record at the default value. Solve the task again with one command without writing the constant for the actor ID (the actor will be identified by his name).
6. Delete the Mandarin language from the database. Solve the task only after solving the task 4.
7. Delete the Terminator movie from the database (solve the task after solving the example 1). Is it possible to solve this task only by deleting the appropriate record from the `film` table?
8. Delete all inactive customers from the database.
9. Add the optional integer attribute `inventory_count` to the `movie` table. Set this attribute for all movies to the number of copies of that movie (the number of matching records in the `inventory` table).
10. Edit the `name` attribute in the `category` table to string of variable length 50 characters.
11. Add the mandatory text attribute `phone` with a maximum length of 20 characters to the `customer` table. Set the phone according to the `phone` attribute, which is part of the customer's address.
12. Add the mandatory attribute `create_date` to the `rental` table, whose default value will be the current timestamp.
13. Drop the attribute `rental.create_date` created in the previous task.
14. Add the optional attribute `creator_staff_id` to the table `film`, which will be a foreign key to the table `staff`. Name the foreign key `fk_film_staff`.
15. Set check of the attribute `staff.email` so that the email value always will contain the character '@' followed by the character '.'.
16. Drop the check constaint cread in the previous task.
17. Set the loan check so that the return date is always greater than the loan date.
18. Create a new table `reservation`, i.e. a table of reservations, with the automatically generated primary key `reservation_id` of the data type integer. The table will also contain the following attributes: mandatory reservation date `reservation_date` with a the current date as a default value, mandatory reservation end date `end_date`, mandatory customer ID `customer_id` as a foreign key to the table `customer`, mandatory movie ID `movie_id` as foreign key to table `film` and optional employee ID `staff_id` as foreign key to table `staff`.

19. Insert some two records in the table created in the previous task. Then delete the second record. What ID will be assigned to the next inserted record?

20. (a) Create a table `review` with the attributes `film_id` and `customer_id` representing foreign keys in the tables `film` and `customer`. Both of these attributes will together represent a composite primary key. The table will also contain mandatory attribute `stars`, which will take integers in the interval $\langle 1, 5 \rangle$, and optional attribute `actor_id`, which will be a foreign key to the table `actor`. Ensure that in the case you delete a customer or movie, all related records in the table `review` will be also automatically deleted. Also, make sure that when you delete an actor, for related records in table `review` will be `actor_id` set to `NULL`.

    (b) Insert two records in the table `review`:

    - Review of the movie ARMY FLINTSTONES by the customer BRIAN WYMAN - 4 stars, without mentioning the actor.
    - Review of the movie ARSENIC INDEPENDENCE by the customer CHERYL MURPHY – 5 stars mentioning actor EMILY DEE.

    (c) Delete customer BRIAN WYMAN and actor EMILY DEE from the database. Then look at the content of the table `review`.

21. Back up the content of the table `film` to the new table `film_backup`. The new table will be identical in structure with the table `film` but it will not contain primary or foreign key settings. In other words, attributes like `film_id`, `language_id` will be common integer (non-key) attributes.

22. Drop tables the `review` and `film_backup` created in the previous two tasks.

23. Create a table `rating` with attributes `rating_id` - integer automatically generated primary key, `name` - mandatory string with a maximum of 10 characters and `description` - an optional string with an unlimited number of characters. Select unique values from the attribute `rating` located in the table `film` and insert them into the new table. Create a mandatory attribute `rating_id` in the table `movie`, which will be a foreign key to the newly created table `rating`. The values in this attribute will be set according to the attribute `rating`. Finally, delete the original attribute `rating`.

24. Drop all tables from the database.