SQL Basics

**1. Select all records from the actor table.**

SELECT \* FROM actor

**2. Write an SQL query to select the columns sex and body\_mass\_g from the table little\_penguins, sorted so that penguins with the highest body mass appear first.**

SELECT sex, body\_mass\_g FROM little\_penguins ORDER BY body\_mass\_g DESC

**3. Retrieve all records from the address table where the postal code is not specified. Sort the result by address\_id.**

SELECT \* FROM address WHERE postal\_code IS NULL ORDER BY address\_id ASC

**4. Retrieve the column name from the language table in alphabetical order.**

SELECT name FROM language ORDER BY name ASC

**5. Select all first and last names of actors from the actor table.**

SELECT first\_name, last\_name FROM actor

**6. Retrieve a list of values from the name column of the language table.**

SELECT name FROM language

**7. Select movie titles from the film table. Sort the resulting list alphabetically.**

SELECT title FROM film ORDER BY title

**8. From the customer table, select all records for last name - last\_name, first name - first\_name and email address email, sorting them by last name in alphabetical order.**

SELECT last\_name, first\_name, email FROM customer ORDER BY last\_name

**9. Write an SQL query that lists the unique rating values ​​from the film table in alphabetical order.**

SELECT DISTINCT rating FROM film ORDER BY rating

**10. Get the names of the five longest movies, sorted by duration in descending order.**

SELECT title FROM film ORDER BY length DESC LIMIT 5

**11. Select the title, description, and year of release of the films from the film table. Sort the resulting list by name in alphabetical order and print the first ten lines.**

SELECT title, description, release\_year FROM film ORDER BY title LIMIT 10

**12. For ease of display, we will divide the list of films into pages of ten entries each. To form the third page of the list, select the title, description and year of release of films from the film table.**

**Sort the resulting list by name in alphabetical order and print ten lines starting from the twenty-first.**

SELECT title, description, release\_year FROM film WHERE film\_id BETWEEN 21 and 30 ORDER BY title

SELECT title, description, release\_year FROM film WHERE film\_id BETWEEN 21 and 30 ORDER BY film\_id

**13. Select the title, rental price and duration of films from the film table. Sort the resulting list in descending order of cost; sort films with the same cost in ascending order of film duration.**

SELECT title, rental\_rate, length

FROM film

ORDER BY rental\_rate DESC, length ASC

**14. Find the longest movie in the film table. If multiple movies have the same length, choose the movie with the lowest replacement\_cost.**

**Write a query, without using aggregate functions, that returns two columns: title and release\_year.**

SELECT title, release\_year

FROM film

ORDER BY length DESC, replacement\_cost ASC

LIMIT 1

**15. Find all movies over three hours long. Write an SQL query that returns a result consisting of three columns: the title of the movie, its description and the duration in minutes, sorted by the length of the movie.**

SELECT title, description, length

FROM film

WHERE length > 180

ORDER BY length

**16. Find the employees working at store number 1 and get all their details.**

SELECT \*

FROM staff

WHERE store\_id = 1

**17. Find all currently active customers (active = 1) in the customer table. The results table should contain the following fields: customer\_id, first\_name and last\_name.**

SELECT customer\_id, first\_name, last\_name

FROM customer

WHERE active =1

**18. Find actors named Scarlett.**

SELECT \*

FROM actor

WHERE first\_name = 'Scarlett'

**19. Find all movies that have the word Student in their description. List the movie titles in alphabetical order.**

SELECT title

FROM film

WHERE description LIKE '%student%'

ORDER BY film\_id

**20. Find all movies over 3 hours long and get their title, year of release and duration, sorted by duration in ascending order.**

SELECT title, release\_year, length

FROM film

WHERE length > 180

ORDER BY length

**21. Find all comedies over three hours long. Write an SQL query that returns a result consisting of three columns: the title of the film, the year of release and the duration in minutes, sorted by the length of the film.**

SELECT title, release\_year, length

FROM film

WHERE film\_id IN (select film\_id FROM film\_category WHERE category\_id=5)

AND length>180

group by film\_id

order by length;

**22. Select surnames, first names and email addresses of customers whose first and last names do not contain a single letter “A” (Latin letter). Sort the result by customer\_id.**

SELECT last\_name, first\_name, email

FROM customer

WHERE first\_name NOT LIKE '%a%' AND last\_name NOT LIKE '%a%'

ORDER BY customer\_id

**23. Find all movies rated NC-17 (Adults Only) that have the substring Database Administrator in their description. Print the title, description, year of release of these films in alphabetical order by title.SELECT title, description, release\_year.**

SELECT title, description, release\_year

FROM film

WHERE rating = 'NC-17' and description LIKE '%Database Administrator%'

ORDER BY film\_id

**24. Find all movies that have the words Dog or Cat in their description and are rated PG or PG-13 (for parental supervision). Display the titles, descriptions, and years of release of these films, sorted by title in alphabetical order.**

SELECT title, description, release\_year

FROM film

WHERE (description LIKE '%dog%' OR description LIKE '%cat%') AND rating IN ('PG', 'PG-13')

**25. Films rated R (Restricted) and NC-17 (Adults Only) may not be rented by young people. Get a list of these movies in two columns title and rating, sorted by movie title. To solve this problem, use a condition with the OR keyword.**

SELECT title, rating

FROM film

WHERE rating in('R', 'NC-17')

ORDER BY film\_id

**26. Films rated PG (Parental Guidance Suggested) and PG-13 (Parents Caution) may only be viewed by children under parental supervision. Get a list of these movies in two columns title, rating, sorted by title.**

SELECT title, rating

FROM film

WHERE rating in ('pg', 'pg-13')

ORDER BY film\_id

**27. Find all employees working on the Video Database project. Write a query that displays the employee number, first name, last name, hire date, and position code. Sort the results by last name in alphabetical order. If the last names are the same, sort by job code.**

SELECT e.EMP\_NO, e.FIRST\_NAME, e.LAST\_NAME, e.HIRE\_DATE, e.JOB\_CODE

FROM EMPLOYEE e

JOIN EMPLOYEE\_PROJECT ep

ON e.EMP\_NO = ep.EMP\_NO

WHERE ep.PROJ\_ID = 'VBASE'

ORDER BY e.LAST\_NAME, e.JOB\_CODE

**28. Write a query that retrieves a list of all employees working outside the US. The result must contain all columns of the EMPLOYEE table.**

SELECT \*

FROM EMPLOYEE

WHERE NOT JOB\_COUNTRY = 'USA'

**29. Write a query that retrieves a list of all employees hired in 1992. The result should contain the following columns FULL\_NAME - the full name of the employee and HIRE\_DATE - the date of hire. Sort the results in ascending order of appointment date.**

SELECT FULL\_NAME, HIRE\_DATE

FROM EMPLOYEE

WHERE HIRE\_DATE LIKE '1992%'

ORDER BY HIRE\_DATE

**30. Write an SQL query to get a list of films that are not available for rent (inventory table). Display the titles of these films in a column called film\_title in alphabetical order. Use a table join to solve the problem.**

SELECT f.title AS film\_title

FROM film f

LEFT JOIN inventory i ON f.film\_id = i.film\_id

WHERE i.inventory\_id IS NULL -- Фильмы, не арендованные

ORDER BY f.title ASC;

**31. Write an SQL query to get a list of languages ​​from the language table in which there are no movies available. Present the result in a table with one column - language, sorted alphabetically. Use a table join to solve the problem.**

SELECT name AS language

FROM language l

LEFT JOIN

film f ON l.language\_id = f.language\_id

WHERE f.film\_id IS NULL

ORDER BY l.name

**32. Write an SQL query that displays the titles of all movies and their categories from the Sakila database.**

SELECT f.title, c.name

FROM film f

JOIN

film\_category fc ON f.film\_id = fc.film\_id

JOIN

category c ON fc.category\_id = c.category\_id;

**33. Extract name and domain from customer email addresses in Sakila database. Write a query that returns three columns: email, address - the part of the email address before the "@" sign, and domain - the part after the "@". Sort the results by the email field.**

SELECT

email,

SUBSTRING\_INDEX(email, '@', 1) AS address,

SUBSTRING\_INDEX(email, '@', -1) AS domain

FROM

customer

ORDER BY

email ASC;

**34. Get address table column definitions.**

DESCRIBE address;

**35. Get a list of film table indexes and their definitions.**

SHOW INDEX FROM film;

**36. Find movies from the Sakila database that have no actor records using a JOIN table join. Print the result with the fields title, release\_year sorted by movie title.**

SELECT f.title, f.release\_year

FROM film f

LEFT JOIN

film\_actor fa ON f.film\_id = fa.film\_id

WHERE fa.actor\_id IS NULL

**37. Find clients whose first name is the last name of another client. Print a table with the fields customer\_id, first\_name, last\_name for the first client and the same fields customer\_id, first\_name, last\_name for the second. Sort by customer\_id of the first customer.**

SELECT

c1.customer\_id

c1.first\_name,

c1.last\_name,

c2.customer\_id,

c2.first\_name,

c2.last\_name

FROM customer c1

JOIN

customer c2 ON c1.first\_name = c2.last\_name

**38. Find clients who met each other at one of the rental points. Display a table with the fields meet\_time - according to the rental time, store\_id, customers - a list of met clients in the format JOHN SHOW, DAENERYS TARGARYEN - in the order of their last names. Sort the resulting table by meeting time and rental point number (Clients met if they rented films from the same branch at the same time.)**

SELECT r.rental\_date AS meet\_time,

s.store\_id,

GROUP\_CONCAT(DISTINCT CONCAT(c.first\_name, ' ', c.last\_name) ORDER BY c.last\_name SEPARATOR ',') AS customers

FROM rental r

JOIN customer c ON r.customer\_id = c.customer\_id

JOIN staff s ON r.staff\_id = s.staff\_id

GROUP BY r.rental\_date, s.store\_id

HAVING COUNT(DISTINCT c.customer\_id) > 1

ORDER BY meet\_time, s.store\_id;

**39. Write an SQL query to search for movies in the Sakila database that are available (in the inventory table) but have never been rented out. List the names of these films in alphabetical order. To solve the problem, use a table join.**

SELECT f.title

FROM film f

JOIN inventory i ON f.film\_id = i.film\_id

LEFT JOIN rental r ON i.inventory\_id = r.inventory\_id

WHERE r.rental\_id IS NULL

ORDER BY f.film\_id;

**40. Get all the movies in the following categories: Comedy, Music and Travel. Print a table with columns film\_id, title and category, sorted by film\_id. Write a query without using the OR keyword in the condition.**

SELECT f.film\_id, f.title, c.name AS category

FROM film f

LEFT JOIN

film\_category fc ON f.film\_id = fc.film\_id

LEFT JOIN

category c ON fc.category\_id = c.category\_id

WHERE

c.name IN ('comedy', 'music', 'travel')

ORDER BY f.film\_id

**41. Select first and last names of customers whose first and last names begin with the same letter.**

**Sort the results by first and last name.**

SELECT first\_name, last\_name FROM customer

WHERE LEFT (first\_name, 1) = LEFT (last\_name, 1)

GROUP BY customer\_id

ORDER BY first\_name, last\_name ASC;

**42. Find all movies rented by KATIE ELLIOTT. Print the result in two columns title and rating. Sort the list so that "adult" films (rated R) come first, and then all the others in alphabetical order.**

JOIN

inventory i ON r.inventory\_id = i.inventory\_id

JOIN

film f ON i.film\_id = f.film\_id

JOIN

customer c ON r.customer\_id = c.customer\_id

WHERE

c.first\_name = 'KATIE' AND c.last\_name = 'ELLIOTT'

ORDER BY

CASE

WHEN f.rating = 'R' THEN 0

ELSE 1

END,

f.title;

Computations

**1. Write a query to calculate the circumference of a circle with a diameter of 7. Display the result in the circle\_perimeter column.**

SELECT

7 \* PI() AS circle\_perimeter;

**2. Write a query to calculate the area of ​​a circle with radius 12. Round the result to six decimal places and display it in the circle\_area column.**

SELECT

ROUND(PI() \* POW(12, 2), 6) AS circle\_area;

**3. Find the length of the hypotenuse of a right triangle with legs equal to 2 and 3. Print the result in the hypotenuse column. Round the result to three decimal places.**

SELECT

ROUND(CAST(SQRT(POW(2, 2) + POW(3, 2)) AS NUMERIC), 3) AS hypotenuse;

**4. Write a query that returns a table of factorial values ​​for integers from 0 to 10. The table must contain two columns n - a number from 0 to 10 and f the factorial value of this number.**

WITH RECURSIVE r AS (

SELECT 0 AS n, 1 AS f

UNION

SELECT n+1 AS f, f \* (n+1) as f

FROM r

WHERE n < 10)

SELECT \* FROM r;

**5. Generate a list of films in JSON format like {"id": 1, "title": "ACADEMY DINOSAUR", "category": "Documentary"} in a table with one column film sorted by movie ID**

SELECT

JSON\_OBJECT(

'id', f.film\_id,

'title', f.title,

'category', c.name

) AS film

FROM

film f

JOIN

film\_category fc ON f.film\_id = fc.film\_id

JOIN

category c ON fc.category\_id = c.category\_id

ORDER BY

f.film\_id;

**6. Get all records from the address table where the zip code is an even number. Output a table with two columns address\_id and postal\_code, sorted by address\_id.**

SELECT address\_id, postal\_code

FROM address

WHERE MOD(CAST(postal\_code AS UNSIGNED), 2) = 0

ORDER BY address\_id;

**7. Build a shared email list for clients and staff. Output a table with the following columns:**

**record\_type – customer or employee, last\_name, first\_name, email – personal data**

**Sort the table by last name and then by first name.**

SELECT

'customer' AS record\_type,

c.last\_name,

c.first\_name,

c.email

FROM

customer c

UNION ALL

SELECT

'employee' AS record\_type,

s.last\_name,

s.first\_name,

s.email

FROM

staff s

ORDER BY

last\_name, first\_name;

8.

SELECT

f.title,

r.rental\_date,

r.return\_date,

p.payment\_date,

f.rental\_rate,

ABS(f.rental\_rate - p.amount) AS lateness\_penalty, -- Расчёт положительного штрафа

p.amount

FROM

rental r

JOIN

customer c ON r.customer\_id = c.customer\_id

JOIN

payment p ON r.rental\_id = p.rental\_id

JOIN

inventory i ON r.inventory\_id = i.inventory\_id

JOIN

film f ON i.film\_id = f.film\_id

WHERE

c.first\_name = 'DOROTHY' AND c.last\_name = 'TAYLOR'

AND r.rental\_date BETWEEN '2005-08-01' AND '2005-08-31'

UNION ALL

SELECT

'Total' AS title,

NULL AS rental\_date,

NULL AS return\_date,

NULL AS payment\_date,

SUM(f.rental\_rate) AS rental\_rate, -- Сумма всех ставок аренды

SUM(ABS(f.rental\_rate - p.amount)) AS lateness\_penalty, -- Сумма всех штрафов

SUM(p.amount) AS amount -- Общая оплаченная сумма

FROM

rental r

JOIN

customer c ON r.customer\_id = c.customer\_id

JOIN

payment p ON r.rental\_id = p.rental\_id

JOIN

inventory i ON r.inventory\_id = i.inventory\_id

JOIN

film f ON i.film\_id = f.film\_id

WHERE

c.first\_name = 'DOROTHY' AND c.last\_name = 'TAYLOR'

AND r.rental\_date BETWEEN '2005-08-01' AND '2005-08-31';

**9. Make a list of surnames found both among users and among actors. Print a table with one column, last\_name, sorted alphabetically. Solve the problem using a table operator.**

SELECT last\_name

FROM customer

INTERSECT

SELECT last\_name

FROM actor

ORDER BY last\_name;

**10. Find palindromic names in the customer table. Sort the result by first\_name.**

SELECT first\_name

FROM customer

WHERE first\_name = REVERSE(first\_name)

ORDER BY first\_name;