```
## Sql subqueries
use internshala;

describe employees;

alter table employees
add date_hire date;

set sql_safe_updates = 0;
update employees
set date_hire = str_to_date(hire_date, '%Y-%m-%d');

alter table employees
drop hire_date;
select * from employees;
--------------------------

## single row subquery

/* In single-row subqueries, we mostly use aggregate functions like:

MAX() → maximum value

MIN() → minimum value

AVG() → average value

SUM() → total

COUNT() → row count

Since these aggregates return only one value, we can compare that value
with columns in the outer query using operators like:

= (equal to)

> (greater than)

< (less than)

>=, <=, <>

*/


# Task 1: Find employees who earn more than the average salary

select employee_id, concat(first_name, ' ', last_name) as joinednames,
salary from employees where salary >
(select avg(salary) as AvgSalary from employees);

# Task 2: Find employees who joined before the earliest hire date in
their company

select f.employee_id, f.date_hire, f.department_id
from employees f where f.date_hire =
(select min(e.date_hire)
from employees e
```

```
where e.department_id= f.department_id);

with mini as
(select department_id, min(date_hire) as m
from employees
group by department_id)
select e.employee_id, e.date_hire, e.department_id
from employees e join mini m
using(department_id)
where e.date_hire = m.m;


# Task 3 Find employees who work in the same department as Neena
Kochhar

select employee_id, department_id
from employees where department_id in
(select department_id
from employees
where concat(first_name, ' ',  last_name) = 'Neena Kochhar');

# Task 4: Find employees who earn more than the average salary of their
own department

select f.employee_id, f.salary, f.department_id from employees f
where f.salary >
(select avg(e.salary) as avgsalary
from employees e
where e.department_id = f.department_id);


# Task 5 Task: Find employees who earn more than the average salary of
their job role (job_id).

select f.employee_id, f.salary, f.job_id from employees f
where f.salary >
(select avg(e.salary) as avgsalary
from employees e
where e.job_id = f.job_id);

# Task 6 Find employees whose salary is higher than the salary of their
manager.

select e.employee_id, m.manager_id, e.salary, m.salary
from employees e join employees m
on e.manager_id = m.employee_id
where e.salary > m.salary;

# Task 7: Find employees who were hired before their manager.

select e.employee_id, e.date_hire
from employees e join employees m
on e.manager_id = m.employee_id
where e.date_hire < m.date_hire;


/* Task 8 Find the employees who earn more than the average salary of
their department,
```

```
 and display their name, salary, and department name. */

select f.employee_id, f.first_name, f.department_id
from employees f where f.salary >
(select avg(e.salary) as avgsalary
from employees e
where e.department_id = f.department_id);

# Task 9 Find the employees who have the highest salary in their
department.
select f.employee_id, f.salary from employees f where f.salary =
(select max(e.salary) as maxsalary
from employees e
where e.department_id = f.department_id);

 with high as
 (select employee_id, department_id, salary,
 rank () over(partition by department_id order by salary desc) rn
 from employees)
 select * from high
 where rn = 1;


# Task 10 : Find the second highest salary in the company in the same
department
select j.employee_id, j.salary, j.department_id
from employees j where j.salary =
(select max(f.salary) as second_high
from employees f where f.department_id = j.department_id
and f.salary <
(select max(e.salary) as first_high
from employees e
where e.department_id = f.department_id));


#Task 11 Find all passengers who paid a fare higher than the average
fare of their passenger class (pclass).

select * from titanic;

select e.passenger_no, e.fare
from titanic e where e.fare >
(select avg(fare) as avgfare
from titanic t
where t.pclass = e.pclass);

#Task 12 Find all pairs of passengers who belong to the same passenger
class (pclass) and also embarked from the same town

select t.passenger_no, e.passenger_no
from titanic t join titanic e
on t.pclass = e.pclass
and t.embark_town = e.embark_town
where t.passenger_no <> e.passenger_no;

#Task 13 Find the passengers whose fare is equal to the maximum fare
paid in each passenger class (pclass).
```

```
select e.passenger_no, e.fare
from titanic e where e.fare =
(select max(fare) as maxfare
from titanic t
where t.pclass = e.pclass);


/* Task 14 Find all passengers who traveled in the same passenger class
as passengers who survived.
 Display their names, class, and survival status */

select passenger_no, first_name, pclass, survived
from titanic where pclass in
(select pclass
from titanic
where survived = 1);

# Task 15 Find the passengers who paid a fare greater than the overall
average fare of passengers who survived.

select passenger_no, fare from titanic where fare >
(select avg(fare) as avgfare
from titanic
where survived = 1);


/* Task 16 Display passengers who embarked from the same port as at
least one survivor.
Show passenger name, embark_town, and their survival status. */

select passenger_no, first_name, embark_town, survived from titanic
where embark_town in
(select distinct embark_town
from titanic
where survived = 1);

/* Task 17 Find passengers whose fare is higher than ANY fare paid by
first-class passengers. Display their name, class, and fare. */

select passenger_no, first_name, class, fare from titanic
where fare > Any
(select fare
from titanic
where pclass = 1);

/* Task 18 Identify passengers whose age is greater than ALL ages of
passengers who didn't survive.
Show their name, age, and survival status. */

select passenger_no, first_name, age, survived from titanic
where age > All
(select age
from titanic
where survived = 0);

# Task 19 Find passengers who have the same combination of class and
embark_town as any survivor. Display all their details.
```

```
select *
from titanic f where exists
(select t.passenger_no
from titanic t
where t.embark_town = f.embark_town
and t.pclass = f.pclass and survived = 1);



/* Task 20 List all passengers whose deck is NOT the same as any deck
occupied by survivors.
Show passenger name, deck, and survival status. */

select passenger_no, first_name, deck, survived from titanic
where deck not in
(select deck
from titanic
where survived = 1);


/* Task 21 Find passengers for whom there are NO other passengers with
the same sex and higher fare who survived.
Display name, sex, fare, and survival status. */

select * from titanic;

select e.passenger_no, e.first_name, e.sex, e.fare, e.survived
from titanic e where not exists
(select t.passenger_no
from titanic t
where t.sex = e.sex and t.fare > e.fare
and t.survived = 1);


/* Task 22 Create a query that shows passenger details only for those
classes where the average age of survivors is greater than 30.
 Use a subquery in the FROM clause. */

select * from titanic where pclass in
(select pclass
from titanic
where survived = 1
group by pclass
having avg(age) > 30);


/* Task 23 Find all Netflix originals whose IMDB score is higher than
the average IMDB score of all titles in their respective genre.
Display the title, genre, IMDB score, and the genre average. */

select * from netflix_originals;

select f.title, f.genreid, f.imdbscore, avg(f.imdbscore) over(partition
by f.genreid)
from netflix_originals f where f.imdbscore >
(select avg(imdbscore)
from netflix_originals n
where n.genreid = f.genreid);
```

```
/* task 24 Display titles where there EXISTS at least one other title
in
the same language that has both a higher IMDB score AND longer runtime.
Show title, language, runtime, and IMDB score. */

select f.title, f.language, f.runtime, f.imdbscore
from netflix_originals f where exists
(select title
from netflix_originals n
where n.language = f.language
and n.imdbscore > f.imdbscore
and n.runtime > f.runtime);

/* Task 25 Find the top 2 highest IMDB rated titles for each genre.
Display genre, title, IMDB score,
and rank within genre. Use only subqueries. */

with score as
(select title, genreid, imdbscore,
rank() over(partition by genreid order by imdbscore desc) as rn
from netflix_originals)
select * from score
where rn in (1,2);

/* Task 26 Find languages where there are NO titles
with IMDB score below 6.0 that also have runtime longer than the
average runtime of titles with IMDB score
 above 8.0 in the same language. Display language and count of titles.
*/

select e.language, count(title) as C
from netflix_originals e where not exists
(select f.imdbscore
from netflix_originals f where f.language = e.language and f.imdbscore
< 6 and f.runtime >
(select avg(n.runtime)
from netflix_originals n
where n.imdbscore > 8
and n.language = f.language))
group by e.language;


/* Task 27 Find the top 3 longest runtime titles in each language, only
among those with IMDB score ≥ 7.0.
Display Language, Title, Runtime, IMDBScore. */

with top3 as
(select title, language, runtime, imdbscore,
rank() over(partition by language order by runtime desc) as rn
from netflix_originals
where imdbscore >=7)
select * from top3
where rn in (1,2,3);

/* Task 28 Find all employees who work in the same department as their
manager. */
```

```
select e.employee_id, e.department_id, m.manager_id, m.department_id
from employees e join employees m
on e.department_id = m.department_id and
e.manager_id = m.employee_id
where e.employee_id <> m.employee_id;

# Task 29 Find all passenger pairs who have the same last name and the
same ticket class (pclass).

select * from titanic;

select distinct t.passenger_no, f.passenger_no
from titanic t join titanic f
on t.last_name = f.last_name
and t.pclass = f.pclass
where t.passenger_no <> f.passenger_no;

/* Task 30 Find all pairs of Netflix Original shows that have the same
IMDB Score and were released in the same year, but are different shows.
Display the titles of both shows, their shared IMDB score, and the
premiere year. */

select * from netflix_originals;

select n.title, m.title
from netflix_originals n join netflix_originals m
on n.imdbscore = m.imdbscore
where year(n.premiere_date) = year(m.premiere_date)
and n.title <> m.title;


/* Task 31 Find passengers who survived and were traveling in the same
 class as at least one passenger who didn't survive. Show their
details. */

 select passenger_no, pclass, survived from titanic
 where pclass in
 (select distinct pclass
 from titanic
 where survived = 0)
and survived = 1;

/* Task 32 Find passengers whose age is above the average age of all
passengers,
 but whose fare is below the maximum fare paid by passengers in their
embarkation town.
Display passenger details along with the relevant averages.
*/

select k.passenger_no
from titanic k where k.age >
(select avg(age)
from titanic ) and k.fare <
(select max(fare)
from titanic t
where t.embark_town = k.embark_town);
```

```
/* Task 33 Find passengers who are older than the youngest passenger in
First Class (pclass = 1),
but paid less fare than the average fare of passengers who didn't
survive.
Display their details along with the comparison values.
*/

select passenger_no, age
from titanic where age > (select min(age) from titanic where pclass =
1) and fare  <
(select avg(fare)
from titanic
where survived = 0);


/* Task 34 Find employees who earn more than the average salary of
their department,
but were hired earlier than the employee with the maximum salary in
their department.
Display employee details.
*/

select f.employee_id, f.salary, f.date_hire
from employees f where f.salary > (select avg(k.salary) from employees
k where k.department_id = f.department_id)
and f.date_hire <
(select max(t.date_hire) from employees t where t.salary =
(select max(e.salary)
from employees e
where e.department_id = f.department_id));

/* Task 35 Find passengers from the Titanic dataset who paid a fare
higher than the average fare of their embarkation town,
but are younger than the oldest passenger in their class.
Display passenger details. */

select f.passenger_no, f.fare, f.age
from titanic f where f.fare >
(select avg(k.fare) from titanic k
where k.embark_town = f.embark_town) and f.age <
(select max(t.age)
from titanic t
where t.pclass = f.pclass);

/* Task 36 Find all transactions where the total amount is greater than
the average total of all transactions. Include Invoice ID,
 Branch, Total, and show how much above the average each transaction
is. */

 select * from walmart_sales;

 select invoice_id, branch, total, total - (select avg(total) from
walmart_sales)
 from walmart_sales where total >
 (select avg(total) as totalsales
 from walmart_sales);
```

```
 /* Task 37
For each product line, find the branch that has the highest average
rating. Use a correlated subquery to identify
these top-performing branch-product combinations. */

select m.product_line, m.branch, avg(m.rating) A
from walmart_sales m group by m.product_line, m.branch having A =
(select max(A) from (select avg(rating) as A from walmart_sales w
where w.product_line = m.product_line group by w.branch) as Raj);

/* Task 38Identify product lines where the average unit price is higher
than the overall average unit price across all product lines.
 For these product lines,
also show how many transactions they have and their average rating. */

select product_line, avg(unit_price) as A, count(invoice_id),
avg(rating) as R
from walmart_sales group by product_line having A >
(select avg(unit_price)
from walmart_sales);


/* Task find customers who made purchases in branches that have above-
average gross income
AND bought products from product lines that have above-average ratings.
*/

select customer_type, branch, gross_income, rating from walmart_sales
where branch in
(select branch
from walmart_sales
group by branch having avg(gross_income) > (select avg(gross_income)
from walmart_sales))
and product_line in (select product_line from walmart_sales group by
product_line having avg(rating) >
(select avg(rating) from walmart_sales));


/*Find all transactions that occurred in months where the total monthly
sales revenue was above the
average monthly sales revenue across all months. For each qualifying
transaction, show the Invoice ID, Date,
Total, Branch, and include the monthly total for that transaction's
month. */

use internshala;

with Monthly_Total as
(select month(date_sale) as mon, sum(total) as total_sales
from walmart_sales
group by mon),
Monthly_Avg as
(select avg(total_sales) as avg_sales
from Monthly_Total)
select w.invoice_id, w.date_sale, w.branch, w.total
from walmart_sales w join Monthly_Total m
on month(w.date_sale) = m.mon
join Monthly_Avg n
```

```
on 1 = 1
where m.total_sales > n.avg_sales;
```