**SQL Tasks**

**List Department details (ID, Name, Location) which does not have any employees.**

SELECT

dept\_id,

dept\_name,

location

FROM

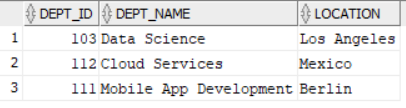
xx1554\_dept d,

xx1554\_emp e

WHERE

d.dept\_id = e.emp\_dept\_id (+)

AND e.emp\_id IS NULL;



Performed a left join on xx1554\_dept and xx1554\_emp. Retrieved the dept\_id values that were not present in the employees table. This resulted in the departments without any employees.

**List all employees whose salary is greater than average salary of all employees.**

SELECT

emp\_name

FROM

xx1554\_emp

WHERE

emp\_salary > (

SELECT

AVG(emp\_salary)

FROM

xx1554\_emp

);



Used the xx1554\_employees table. Initially crafted a sub-query to calculate the average salary of all employees. Then, implemented the outer query to obtain the salary of each employee, applying a condition where the salary of each employee exceeded the result from the inner sub-query.

**List all employees who are getting the lowest salary.**

SELECT

emp\_name,

emp\_salary

FROM

xx1554\_emp

ORDER BY

emp\_salary

FETCH FIRST 1 ROW ONLY;



Utilized the xx1554\_emp table to construct the query, arranged the table in ascending order based on salary, and fetched the first row to obtain the minimum salary.

**List customer wise sales**

SELECT

c.name,

SUM(o.amount) sales

FROM

xx1554\_customers c,

xx1554\_orders o

WHERE

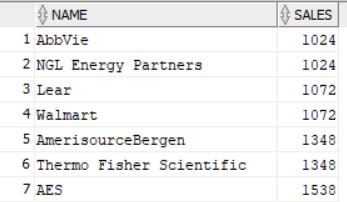
c.customer\_id = o.order\_id

GROUP BY

c.name

ORDER BY

SUM(o.amount);



The two tables were joined through an equi join using a common column called Customer\_id in both tables. Group by function was applied using Customer\_name, and an aggregation function (sum) was performed on sale\_amount.

**List Year wise, month wise Sales**

SELECT

to\_char(order\_date, 'YYYY') years,

SUM(amount) AS sales

FROM

xx1554\_orders

GROUP BY

to\_char(order\_date, 'YYYY')

ORDER BY

sales;



SELECT

to\_char(order\_date, 'MONTH') months,

SUM(amount) AS sales

FROM

xx1554\_orders

GROUP BY

to\_char(order\_date, 'MONTH')

ORDER BY

sales;



Utilizing the xx1554\_orders table, I extracted both year and month from sorder\_date. Following this, a Group by operation was applied to the combined year and month, and an aggregation (sum) was executed on the sale\_amount column, resulting in sales data categorized by each corresponding year and month.

**List Year wise, month wise Direct Sales, Online Sales separately**

SELECT

to\_char(order\_date, 'YYYY') years,

SUM(amount) AS sales

FROM

xx1554\_orders

WHERE

order\_mode = 'DIRECT'

GROUP BY

to\_char(order\_date, 'YYYY');



SELECT

to\_char(order\_date, 'YYYY') years,

SUM(amount) AS sales

FROM

xx1554\_orders

WHERE

order\_mode = 'ONLINE'

GROUP BY

to\_char(order\_date, 'YYYY');



SELECT

to\_char(order\_date, 'MONTH') months,

SUM(amount) AS sales

FROM

xx1554\_orders

WHERE

order\_mode = 'DIRECT'

GROUP BY

to\_char(order\_date, 'MONTH');



SELECT

to\_char(order\_date, 'MONTH') months,

SUM(amount) AS sales

FROM

xx1554\_orders

WHERE

order\_mode = 'ONLINE'

GROUP BY

to\_char(order\_date, 'MONTH');



**List customers who are exceeding their credit limits**

SELECT

c.name,

( ot.quantity \* ot.unit\_price ) expenditure,

c.credit\_limit

FROM

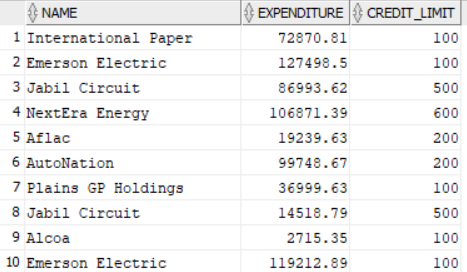
xx1554\_customers c

JOIN xx1554\_orders o ON c.customer\_id = o.customer\_id

JOIN xx1554\_order\_items ot ON o.order\_id = ot.order\_id

WHERE

( ot.quantity \* ot.unit\_price ) > c.credit\_limit;



Joined three tables, multiplied the quantity by the unit price to calculate expenditure, and then compared it with the credit limit to identify customers who exceed their credit limits.

**List all employees who were holding more than one Job in various periods in the company**

SELECT

e.first\_name,

j.company

FROM

xx1554\_employees e,

xx1554\_jobs j

WHERE

e.employee\_id = j.employee\_id

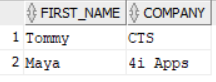
GROUP BY

e.first\_name,

j.company

HAVING

COUNT(\*) > 1;



Equi joined the 'employees' and 'jobs' tables using the employee ID, and then grouped them to count the rows. Rows with a count greater than one reveal employees who held more than one job.

**List all employees with their first job**

SELECT

e.first\_name,

j.company

FROM

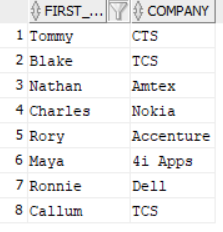
xx1554\_employees e,

xx1554\_jobs j

WHERE

e.employee\_id = j.employee\_id

AND j.experience = 'Fresher';



Equi joined the 'employees' and 'jobs' tables using the employee ID, and then filtered the 'experience' column for 'fresher' to identify employees with their first job.

**How any “orderable” products available**

SELECT

COUNT(product\_id) orderable\_products

FROM

xx1554\_products

WHERE

product\_id IN (

SELECT

product\_id

FROM

xx1554\_inventories

);



Used a subquery to check whether the product ID in the 'product' table is present in the 'inventory' table or not, and then counted the IDs to determine the count of orderable items.

**How to find the top three highest salary in emp table in oracle?**

SELECT

e.first\_name,

j.salary

FROM

xx1554\_employees e,

xx1554\_jobs j

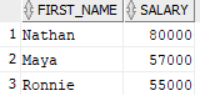
WHERE

e.employee\_id = j.employee\_id

ORDER BY

j.salary DESC

FETCH FIRST 3 ROWS ONLY;



Joined the 'jobs' table with the 'employees' table, arranged them based on the 'salary' column in descending order, and then fetched the top three to identify the employees with the highest salaries.

**SQL Query to find fifth highest salary with empno**

SELECT

first\_name,

salary

FROM

(

SELECT

e.first\_name,

j.salary,

DENSE\_RANK()

OVER(

ORDER BY

j.salary DESC

) "RANK"

FROM

xx1554\_employees e,

xx1554\_jobs j

WHERE

e.employee\_id = j.employee\_id

)

WHERE

"RANK" = 5;



Used a window function, specifically dense rank, to assign ranks to employees based on their salaries. Additionally, I utilized a subquery to extract the employee with a rank of 5.

**What is the total on-hand quantity of all products**

SELECT

SUM(quantity)

FROM

xx1554\_inventories;



Used xx1554\_inventories table. By aggregation function(sum), we can be able to add total on-hand quantity of all products.

**List the products does not have stock**

SELECT

product\_name

FROM

xx1554\_products

WHERE

product\_id NOT IN (

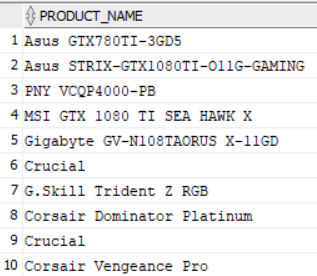
SELECT

product\_id

FROM

xx1554\_inventories

);



Used a subquery to check whether the product ID in the 'product' table is present in the 'inventory' table or not.

**List the items which can be ordered**

SELECT

product\_name

FROM

xx1554\_products

WHERE

product\_id IN (

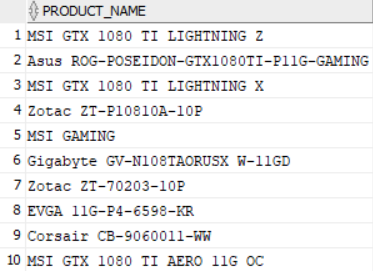
SELECT

product\_id

FROM

xx1554\_inventories

);



Used a subquery to check whether the product ID in the 'product' table is present in the 'inventory' table or not.

**Get the order details for one order**

SELECT

\*

FROM

xx1554\_orders

WHERE

order\_id = :order\_id;



Use xx1554\_orders table. Randomly select one order (order\_id=7) and retrieve details of that order using where clause.

**Verify whether the order\_total is calculated correctly or not**

SELECT

order\_id,

product\_id,

unit\_price,

quantity,

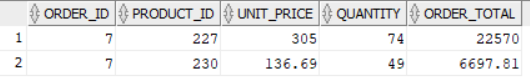
( quantity \* unit\_price ) AS order\_total

FROM

xx1554\_order\_items

WHERE

order\_id = :order\_id;



Used a bind parameter to check the order total by multiplying the unit price with the quantity in the 'order\_items' table for any orders.

**List the items which are ordered**

SELECT

product\_name

FROM

xx1554\_products

WHERE

product\_id IN (

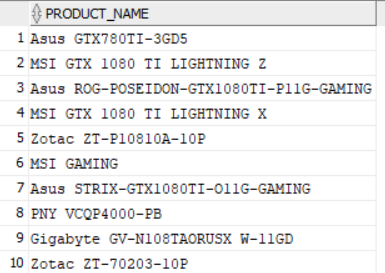
SELECT

product\_id

FROM

xx1554\_order\_items

);



Used a subquery to check whether the product ID in the 'product' table is present in the 'order items' table or not.

**List of items which are not yet ordered**

SELECT

product\_name

FROM

xx1554\_products

WHERE

product\_id NOT IN (

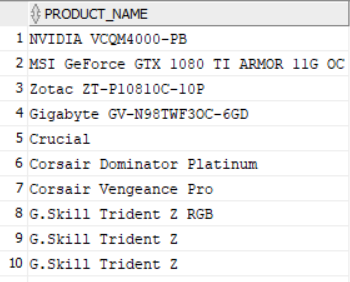
SELECT

product\_id

FROM

xx1554\_order\_items

);



Used a subquery to check whether the product ID in the 'product' table is present in the 'order items' table or not.

**List the Order details where items are ordered less than the list price**

SELECT

\*

FROM

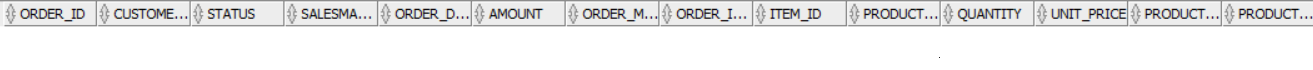
xx1554\_orders o

JOIN xx1554\_order\_items ot ON o.order\_id = ot.order\_id

JOIN xx1554\_products p ON ot.product\_id = p.product\_id

WHERE

ot.unit\_price < p.list\_price;



After joining the tables, I checked the 'unit price' and 'list price' columns. However, there were no orders with prices less than the list price.

**List the Order details where items are ordered less than the minimum price**

SELECT

\*

FROM

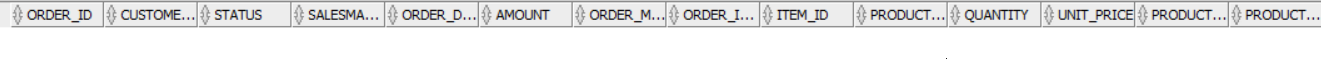
xx1554\_orders o

JOIN xx1554\_order\_items ot ON o.order\_id = ot.order\_id

JOIN xx1554\_products p ON ot.product\_id = p.product\_id

WHERE

ot.unit\_price < p.standard\_cost;



After joining the tables, I checked the 'unit price' and 'list price' columns. However, there were no orders with prices less than the standard cost.

**Find the profit of each order line (compare minimum price with order)**

SELECT

ot.order\_id,

ot.product\_id,

p.standard\_cost,

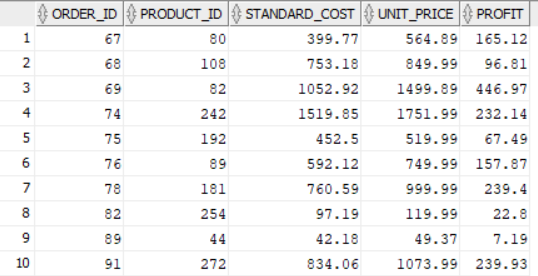
ot.unit\_price,

( ot.unit\_price - p.standard\_cost ) profit

FROM

xx1554\_order\_items ot

JOIN xx1554\_products p ON ot.product\_id = p.product\_id;



Calculated the profit by subtracting the standard cost from the unit price and performed an equi join based on the product ID.

**Find the profit of each order and its %**

SELECT

ot.order\_id,

SUM(ot.unit\_price - p.standard\_cost) profit,

round(SUM(ot.unit\_price - p.standard\_cost) / SUM(p.standard\_cost) \* 100) "PROFIT\_%"

FROM

xx1556\_order\_items ot

JOIN xx1556\_products p ON ot.product\_id = p.product\_id

GROUP BY

ot.order\_id;



Calculated the profit by subtracting the standard cost from the unit price, then computed the sum and its percentage. After performing an equi join based on the product ID, I grouped orders to obtain the profit for each order.

**Create table xx1554\_product by copying only orderable items from product master**

CREATE TABLE xx1554\_product

AS

SELECT

\*

FROM

xx1554\_inventories

WHERE

quantity > 0;

SELECT

\*

FROM

xx1554\_product;



Created a new table called xx1554\_product using create table syntax.

Write query to get orderable items by searching product\_id from xx1554\_inventories table.

**Take backup of employee master**

CREATE VIEW xx1554\_employees\_master AS

SELECT

\*

FROM

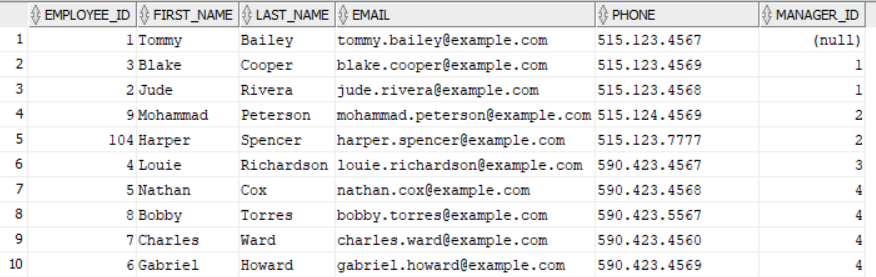
xx1554\_employees;

SELECT

\*

FROM

xx1554\_employees\_master;



To take backup, we can create view as backup\_employess for xx1554\_employees table.

**Create table xx1554\_employee with (id, full\_name, salary) and copy data from employee master**

CREATE TABLE xx1554\_employee

AS

(

SELECT

emp\_id AS id,

emp\_name AS full\_name,

emp\_salary AS salary

FROM

xx1554\_emp

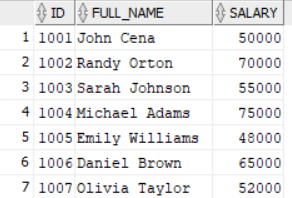
);

SELECT

\*

FROM

xx1554\_employee;



Created a table called xx1554\_employee with columns called id, full\_name, salary with create table syntax from xx1554\_emp table.

**In new table xx1554\_employee increment salary by 10%**

SELECT

id,

salary,

( salary \* ( 10 / 100 ) ) AS hike,

( salary + ( salary \* ( 10 / 100 ) ) ) AS new\_salary

FROM

xx1554\_employee;



Created a new table called xx1554\_employee. Created a new column called new\_salary in select statement by calculating 10% of salary and adding the value to salary. Results in salary increment by 10 percent.