**SQL Practices**

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

**Solution**: Use xx1553\_departments and xx1553\_employees. Get the department\_id which are not in employees table. It results the departments which doesn’t have employees.

**Query**: SELECT

department\_id,

department\_name,

location\_id

FROM

xx1553\_departments

WHERE

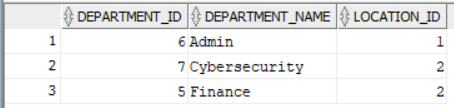
department\_id NOT IN (

SELECT

department\_id

FROM

xx1553\_employees);



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

**Solution**: Use xx1553\_employees table. First write sub-query for average salary of all employees and write the outer query to get salary of each employees and apply the condition where salary of each employees greater than the inner sub-query.

**Query**: SELECT

first\_name || last\_name AS emp\_name,

salary

FROM

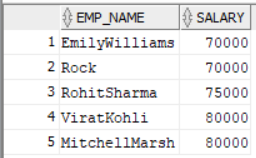
xx1553\_employees

WHERE

salary > (

SELECT

AVG(salary) FROM xx1553\_employee);



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

**Solution**: Use xx1553\_employees table. First write sub-query to lowest salary and write the outer query to get salary of each employees where salary is equal to inner sub-query.

**Query**: SELECT

first\_name || last\_name AS emp\_name,

salary

FROM

xx1553\_employees

WHERE

salary = (

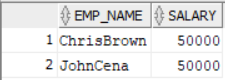
SELECT

MIN(salary)

FROM

xx1553\_employees

);



1. **List customer wise sales**

**Solution**: Here we should take two tables, xx1553\_customers & xx1553\_sales. Join two tables by equi join using common column in both tables called Customer\_id. Perform Group by function using Customer\_name and perform aggregation function(sum) on sale\_amount.

**Query**: SELECT

c.customer\_name,

SUM(s.sale\_amount) sales

FROM

xx1553\_customers c,

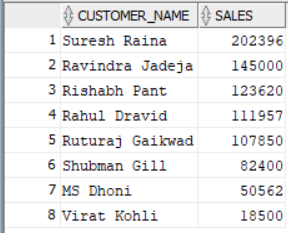
xx1553\_sales s

WHERE

c.customer\_id = s.customer\_id

GROUP BY

c.customer\_name;



1. **List Year wise, month wise Sales**

**Solution**: Use xx1553\_sales table. Extract year from sale\_date. Perform Group by function on year and aggregation (sum) on sale\_amount column. Hence get the required year wise sales.

**Query**: SELECT

to\_char(sale\_date, 'YYYY') AS year,

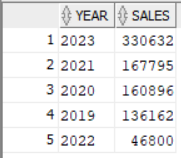
SUM(sale\_amount) AS sales

FROM

xx1553\_sales

GROUP BY

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



**Solution**: Use xx1553\_sales table. Extract month from sale\_date. Perform Group by function on month and aggregation (sum) on sale\_amount column. Hence get the required month wise sales.

**Query**: SELECT

to\_char(sale\_date, 'MON') AS month,

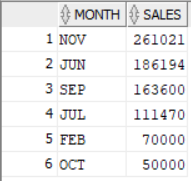
SUM(sale\_amount) AS sales

FROM

xx1553\_sales

GROUP BY

to\_char(sale\_date, 'MON');



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

**Solution**: Use xx1553\_sales table. Extract year from sale\_date. Perform Group by function on year and aggregation (count \*). In where condition give sale\_mode = ’Direct Sales’ . Hence get the required year wise Direct Sales count.

**Query**: SELECT

to\_char(sale\_date, 'YYYY') AS year,

COUNT(\*) AS sales\_count

FROM

xx1553\_sales

WHERE

sale\_mode = 'Direct Sales'

GROUP BY

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



**Solution**: Use xx1553\_sales table. Extract year from sale\_date. Perform Group by function on year and aggregation (count \*). In where condition give sale\_mode = 'Online Sales’ . Hence get the required year wise Online Sales count.

**Query**: SELECT

to\_char(sale\_date, 'YYYY') AS year,

COUNT(\*) AS sales\_count

FROM

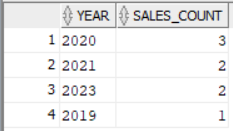
xx1553\_sales

WHERE

sale\_mode = 'Online Sales'

GROUP BY

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



**Solution**: Use xx1553\_sales table. Extract month from sale\_date. Perform Group by function on month and aggregation (count \*). In where condition give sale\_mode = ’Direct Sales’ . Hence get the required month wise Direct Sales count.

**Query**: SELECT

to\_char(sale\_date, 'MON') AS month,

COUNT(\*) AS sales\_count

FROM

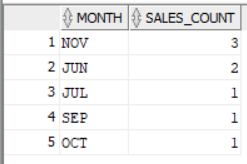
xx1553\_sales

WHERE

sale\_mode = 'Direct Sales'

GROUP BY

to\_char(sale\_date, 'MON');



**Solution**: Use xx1553\_sales table. Extract month from sale\_date. Perform Group by function on month and aggregation (count \*). In where condition give sale\_mode = 'Online Sales’ . Hence get the required month wise Online Sales count.

**Query**: SELECT

to\_char(sale\_date, 'MON') AS month,

COUNT(\*) AS sales\_count

FROM

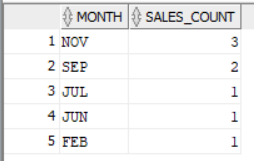
xx1553\_sales

WHERE

sale\_mode = 'Online Sales'

GROUP BY

to\_char(sale\_date, 'MON');



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

**Solution**: Use xx1553\_customers table. Check current\_balance with credit\_limit using where condition. If current\_balance is greater than credit\_limit, print respective customer\_name.

**Query**: SELECT

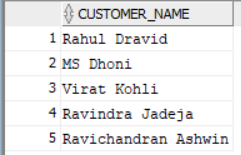
customer\_name

FROM

xx1553\_customers

WHERE

current\_balance > credit\_limit;



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

**Solution**: Use xx1553\_job\_history table and xx1553\_employees table. Join two tables by equi join using same column in both tables called employee\_id.

Perform Group by in employee\_id, company\_id, company\_name from xx1553\_job\_history table and first\_name and last\_name from xx1553\_employees table.

Perform having function and give condition count(\*) greater than 1.

In select statement, give employee\_id, company\_id, company\_name from xx1553\_job\_history table and concatinating first\_name and last\_name from xx1553\_employees table and perform aggregation function (count (\*)).

Hence, we get employees who were holding more than one Job in various periods in the company.

**Query**: SELECT

jh.employee\_id,

e.first\_name || e.last\_name AS name,

jh.company\_id,

jh.company\_name,

COUNT(\*) employed\_times\_in\_same\_company

FROM

xx1553\_job\_history jh,

xx1553\_employees e

WHERE

e.employee\_id = jh.employee\_id

GROUP BY

e.first\_name || e.last\_name,

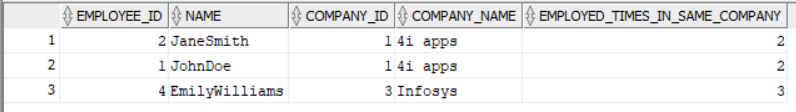
jh.employee\_id,

jh.company\_name,

jh.company\_id

HAVING

COUNT(\*) > 1;



1. **List all employees with their first job**

**Solution**: Use xx1553\_job\_history table. By using over partition by, group employee\_id.

Order by employee\_id and start\_date.

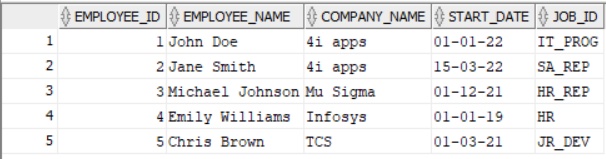
Use count as aggregation function. Naming alias as ct to count function. Store this in a with clause as temp.

Write another query using this with clause temp and xx1553\_employees table. Perform inner join to get name of the employees using common column called employee\_id.

After that in where clause give a condition called ct=1.

By then, able to get all employees with their first job.

**Query**: WITH temp AS (  
 SELECT  
 employee\_id,  
 start\_date,  
 company\_name,  
 job\_id,  
 COUNT(\*)  
 OVER(PARTITION BY employee\_id  
 ORDER BY  
 employee\_id, start\_date  
 ) AS ct  
 FROM  
 xx1553\_job\_history  
 )  
 SELECT  
 t.employee\_id, first\_name || ' ' || last\_name as employee\_name,  
 t.company\_name, start\_date, t.job\_id  
 FROM  
 temp t join xx1553\_employees e  
 on t.employee\_id=e.employee\_id  
 WHERE  
 ct = 1;



1. **How any “orderable” products available**

**Solution**: Use xx1553\_inventories table. Give condition as quantity is not null in where clause.

By then, able to get orderable products.

**Query**: SELECT

product\_id,

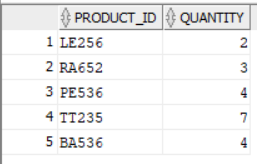
quantity

FROM

xx1553\_inventories

WHERE

quantity IS NOT NULL;



1. **[How to find top three highest salary in emp table in oracle?](http://stackoverflow.com/questions/2943429/how-to-find-top-three-highest-salary-in-emp-table-in-oracle)**

**Solution**: Use xx1553\_employees table. Get distinct salary. Because many employees may get different salaries. Order by salary in descending and used fetch condition to fetch first three rows.

**Query**: SELECT DISTINCT

salary

FROM

xx1553\_employees

ORDER BY

salary DESC

FETCH FIRST 3 ROWS ONLY;



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

**Solution**: Use xx1553\_employees table. Using dense\_rank() function, rank the employee\_salaries.

Why particularly dense\_rank means...?. Dense\_rank won’t skip the rank. If two or more employees have the same salary means, Rank will be same for those employees.

By then, we can be able to get the fifth highest salary with minimum one employee.

Created with clause as cte. By using cte, in where clause give rnk=5.

It fetches the fifth highest salary.

**Query**: WITH cte AS (

SELECT

employee\_id,

salary,

DENSE\_RANK()

OVER(

ORDER BY

salary DESC

) rnk

FROM

xx1553\_employees

)

SELECT

\*

FROM

cte

WHERE

rnk = 5;



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

**Solution**: Use xx1553\_inventories table. By aggregation function(sum), we can be able to add total on-hand quantity of all products.

**Query**: SELECT

SUM(quantity) total\_on\_hand\_quantity

FROM

xx1553\_inventories;



1. **List the products does not have stock**

**Solution**: Use xx1553\_inventories and xx1553\_products table. xx1553\_products table shows the name of all products.

xx1553\_inventories table provides stock of those products. Join two tables using equi join. Give condition quantity is not null. By then, we can get products that do not have stock.

**Query**: SELECT

product\_name

FROM

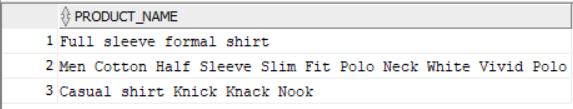
xx1553\_inventories i,

xx1553\_products p

WHERE

p.product\_id = i.product\_id

AND i.quantity IS NULL;



1. **List the items which can be ordered**

**Solution**: Use xx1553\_inventories table. In where clause, put quantity is not null. By then, we can be able to list the items which can be ordered.

**Query**: SELECT

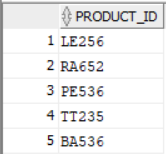
product\_id

FROM

xx1553\_inventories

WHERE

quantity IS NOT NULL;



1. **Get the order details for one order**

**Solution**: Use xx1553\_orders table. Randomly select one order (order\_id=2) and retrieve details of that order using where clause.

**Query**: SELECT

\*

FROM

xx1553\_orders

WHERE

order\_id = 2;



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

**Solution**: Use xx1553\_orders table. In this table quantity, unit price, total price are mentioned.

Using case when multiply quantity and unit price column, if result is equal to total price column, then print correct. vice versa.

**Query**: SELECT

quantity,

unit\_price,

total\_price,

CASE

WHEN ( quantity \* unit\_price ) = total\_price THEN

'CORRECT'

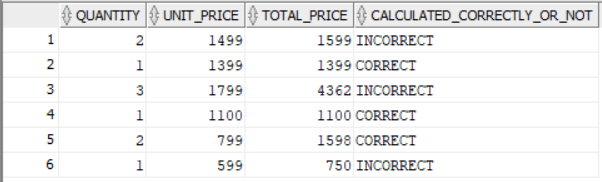
ELSE

'INCORRECT'

END calculated\_correctly\_or\_not

FROM

xx1553\_orders;



1. **List the items which are ordered**

**Solution**: Use xx1553\_products and xx1553\_orders table. Retrieve the product id which is present in xx1553\_orders table using in operator. Then able to get the items which are ordered.

**Query**: SELECT

product\_id,

product\_name

FROM

xx1553\_products

WHERE

product\_id IN (

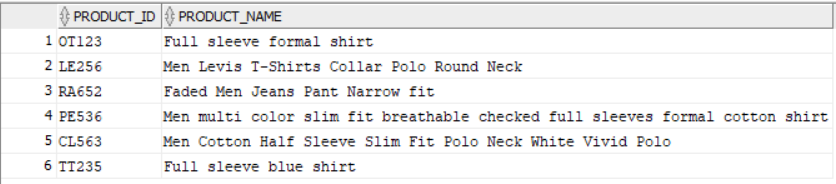
SELECT

product\_id

FROM

xx1553\_orders

);



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

**Solution**: Use xx1553\_products and xx1553\_orders table. Retrieve the product id which is not present in xx1553\_orders table using not in operator. Then able to get the items which are not yet ordered.

**Query**: SELECT

product\_id,

product\_name

FROM

xx1553\_products

WHERE

product\_id NOT IN (

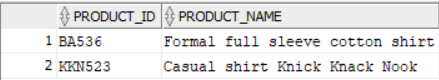
SELECT

product\_id

FROM

xx1553\_orders

);



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

**Solution**: Use xx1553\_orders and xx1553\_products table. Retrieve the order details where unit price which is mentioned in xx1553\_orders table is less than product price which is mentioned in xx1553\_products table.

**Query**: SELECT

\*

FROM

xx1553\_orders o

WHERE

unit\_price < (

SELECT

product\_price

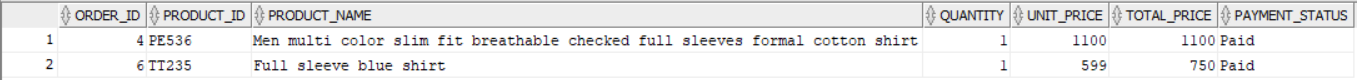
FROM

xx1553\_products p

WHERE

o.product\_id = p.product\_id

);



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

**Solution**: Use xx1553\_orders and xx1553\_products table. Retrieve the order details where unit price mentioned in xx1553\_orders table is less than the minimum product price of xx1553\_products table using where clause.

**Query**: SELECT

\*

FROM

xx1553\_orders

WHERE

unit\_price < (

SELECT

MIN(product\_price)

FROM

xx1553\_products

);



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

**Solution**: Use xx1553\_orders and xx1553\_products table. Join two tables by equi join using common column called product id. Calculate a new column called outcome\_price by subtracting product price of xx1553\_products table from unit price of xx1553\_ordres table.

Create another column called profit\_loss using case when.

In profit\_loss column, print profit if the value of outcome price is greater than 0 and loss if it is less than 0. if it is equal to 0 print No profit/loss.

**Query**: SELECT

p.product\_price,

unit\_price AS sale\_price,

( o.unit\_price - p.product\_price ) AS outcome\_price,

CASE

WHEN ( o.unit\_price - p.product\_price ) > 0 THEN

'profit'

WHEN ( o.unit\_price - p.product\_price ) < 0 THEN

'loss'

ELSE

'No profit/loss'

END profit\_loss

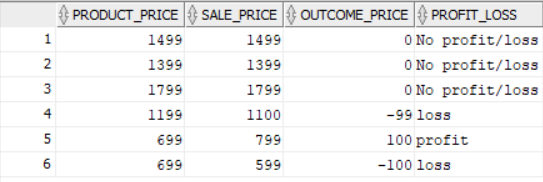
FROM

xx1553\_orders o,

xx1553\_products p

WHERE

o.product\_id = p.product\_id;



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

**Solution**: Use xx1553\_orders and xx1553\_products table. Join two tables by equi join using common column called product id. Calculate a new column called profit\_loss by subtracting product price of xx1553\_products table from unit price of xx1553\_ordres table.

Using case when, if the value of profit\_loss column is greater than 0 then divide the value by product\_price and multiply by 100. Otherwise print 0.

Results the profit percentage.

**Query**: SELECT

product\_price,

unit\_price - product\_price AS profit\_loss,

CASE

WHEN ( unit\_price - product\_price ) > 0 THEN

round(((unit\_price - product\_price) / product\_price) \* 100)

ELSE

0

END profit\_percent

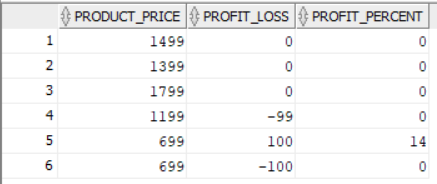
FROM

xx1553\_products p,

xx1553\_orders o

WHERE

p.product\_id = o.product\_id;



1. **Create table xx100\_product by copying only orderable items from product master**

**Solution**: Create a new table called xx100\_product using create table syntax.

Write query to get orderable items by searching product\_id from xx1553\_orders table in xx1553\_products.

**Query**: CREATE TABLE xx100\_product

AS

(

SELECT

\*

FROM

xx1553\_orders

WHERE

product\_id IN (

SELECT

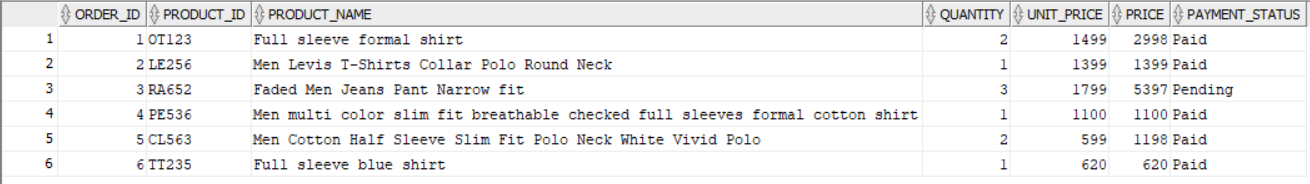
product\_id

FROM

xx1553\_products

)

);



1. **Take backup of employee master**

**Solution**: To take backup, we can create view as backup\_employess for xx1553\_employees table.

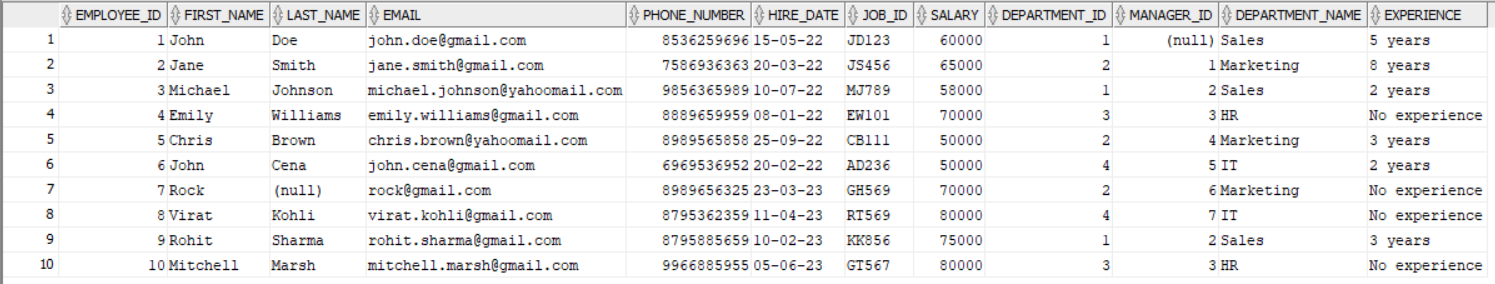
**Query**: CREATE OR REPLACE VIEW backup\_employees AS

SELECT

\*

FROM

xx1553\_employees;



1. **Create table xx100\_employee with (id, full\_name, salary) and copy data from employee master**

**Solution**: Create a table called xx100\_employee with columns called id, full\_name, salary with create table syntax from xx1553\_employees table.

**Query**: CREATE TABLE xx100\_employee

AS

(

SELECT

employee\_id AS id,

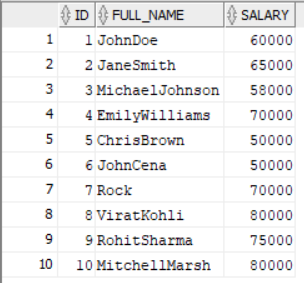
first\_name || last\_name AS full\_name,

salary

FROM

xx1553\_employees

);



1. **In new table xx100\_employee increment salary by 10%**

**Solution**: Create a new table called xx100\_employee.

Create a new column called salary\_increment\_by\_10\_percent in select statement by calculating 10% of salary and adding the value to salary.

Results in salary increment by 10 percent.

**Query**: SELECT

salary,

salary \* ( 10 / 100 ) AS ten\_percentage,

salary + ( salary \* ( 10 / 100 ) ) AS salary\_increment\_by\_10\_percent

FROM

xx100\_employee;

