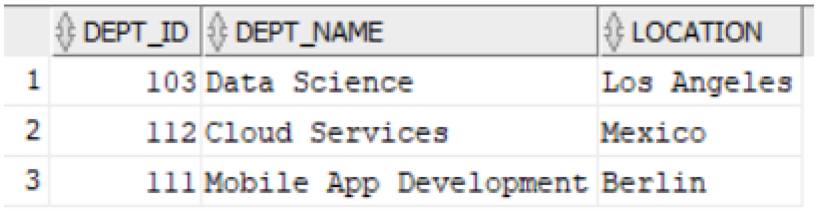
SQL Tasks

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;
```

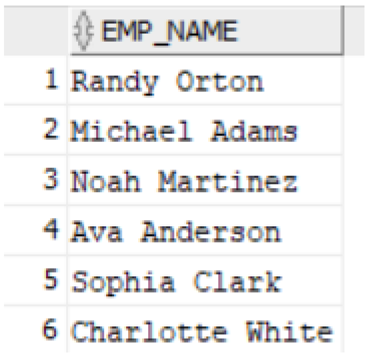


	DEPT_ID	DEPT_NAME	LOCATION
1	103	Data Science	Los Angeles
2	112	Cloud Services	Mexico
3	111	Mobile App Development	Berlin

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
  );
```

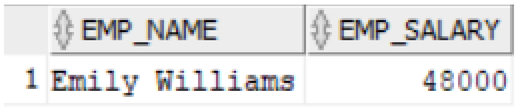


	EMP_NAME
1	Randy Orton
2	Michael Adams
3	Noah Martinez
4	Ava Anderson
5	Sophia Clark
6	Charlotte White

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;
```

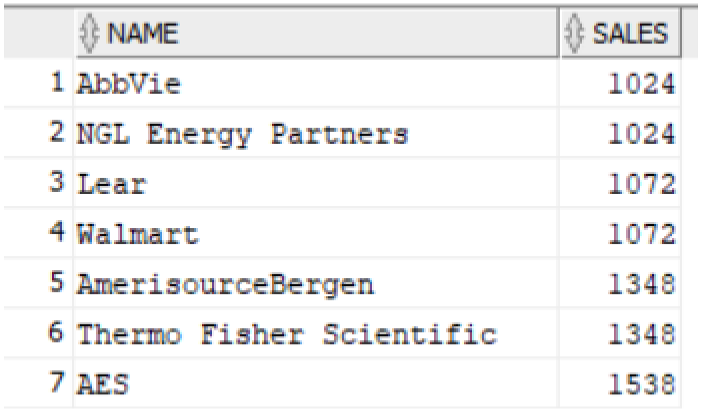


	EMP_NAME	EMP_SALARY
1	Emily Williams	48000

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);
```

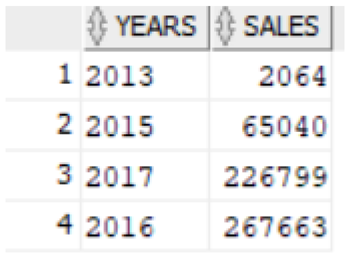


	NAME	SALES
1	AbbVie	1024
2	NGL Energy Partners	1024
3	Lear	1072
4	Walmart	1072
5	AmerisourceBergen	1348
6	Thermo Fisher Scientific	1348
7	AES	1538

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.

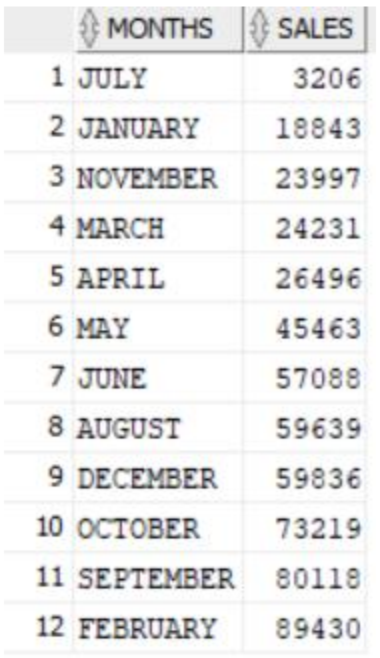
List Year wise, month wise Sales

```
SELECT
   to_char(order_date, 'YYYY') years,
   SUM(amount)              AS sales
FROM
   xx1554_orders
GROUP BY
   to_char(order_date, 'YYYY')
ORDER BY
   sales;
```



	YEARS	SALES
1	2013	2064
2	2015	65040
3	2017	226799
4	2016	267663

```
SELECT
   to_char(order_date, 'MONTH') months,
   SUM(amount)              AS sales
FROM
   xx1554_orders
GROUP BY
   to_char(order_date, 'MONTH')
ORDER BY
   sales;
```

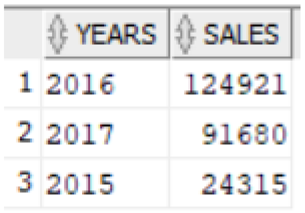


	MONTHS	SALES
1	JULY	3206
2	JANUARY	18843
3	NOVEMBER	23997
4	MARCH	24231
5	APRIL	26496
6	MAY	45463
7	JUNE	57088
8	AUGUST	59639
9	DECEMBER	59836
10	OCTOBER	73219
11	SEPTEMBER	80118
12	FEBRUARY	89430

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.

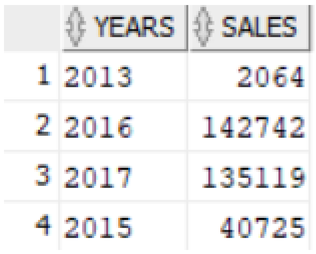
List Year wise, month wise Direct Sales, Online Sales separately

```
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');
```



	YEARS	SALES
1	2016	124921
2	2017	91680
3	2015	24315

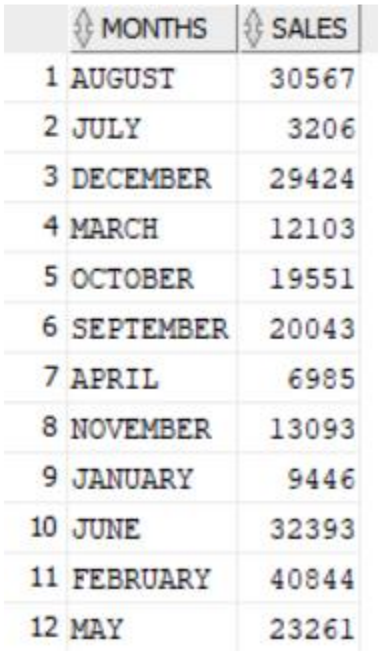
```
SELECT
   to_char(order_date, 'YYYY') years,
   SUM(amount)              AS sales
FROM
   xx1554_orders
WHERE
   order_mode = 'ONLINE'
GROUP BY
   to_char(order_date, 'YYYY');
```



	YEARS	SALES
1	2013	2064
2	2016	142742
3	2017	135119
4	2015	40725

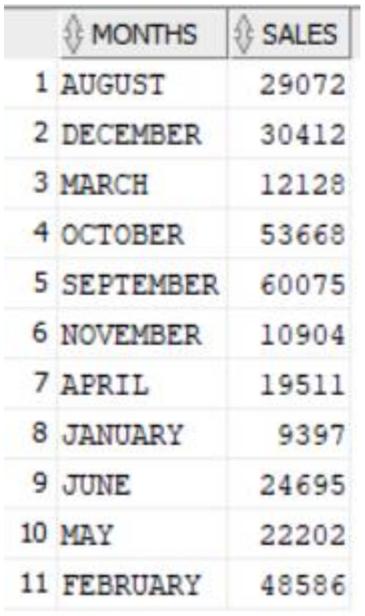
```
SELECT
   to_char(order_date, 'MONTH') months,
   SUM(amount)              AS sales
FROM
   xx1554_orders
WHERE
   order_mode = 'DIRECT'
GROUP BY
```

```
    to_char(order_date, 'MONTH');
```



	MONTHS	SALES
1	AUGUST	30567
2	JULY	3206
3	DECEMBER	29424
4	MARCH	12103
5	OCTOBER	19551
6	SEPTEMBER	20043
7	APRIL	6985
8	NOVEMBER	13093
9	JANUARY	9446
10	JUNE	32393
11	FEBRUARY	40844
12	MAY	23261

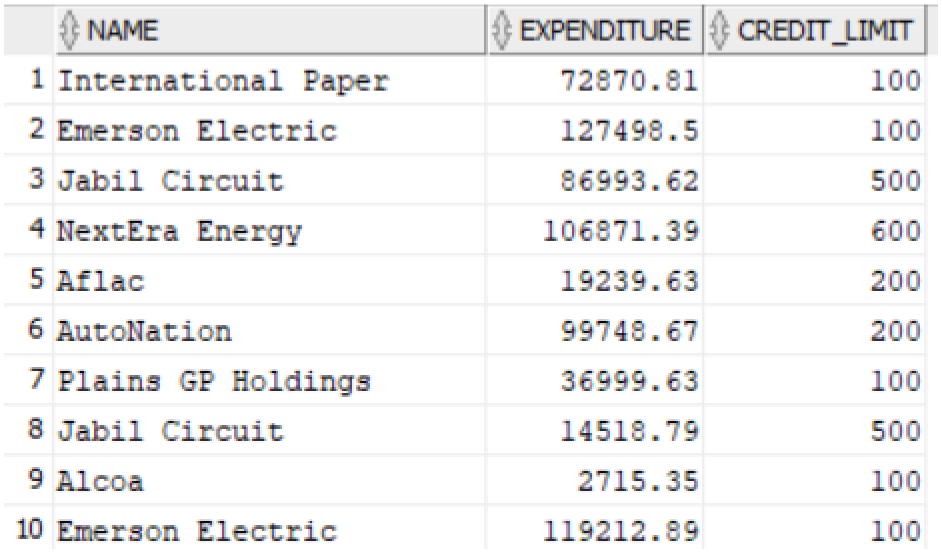
```
SELECT
    to_char(order_date, 'MONTH') months,
    SUM(amount)                AS sales
FROM
    xx1554_orders
WHERE
    order_mode = 'ONLINE'
GROUP BY
    to_char(order_date, 'MONTH');
```



	MONTHS	SALES
1	AUGUST	29072
2	DECEMBER	30412
3	MARCH	12128
4	OCTOBER	53668
5	SEPTEMBER	60075
6	NOVEMBER	10904
7	APRIL	19511
8	JANUARY	9397
9	JUNE	24695
10	MAY	22202
11	FEBRUARY	48586

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;
```

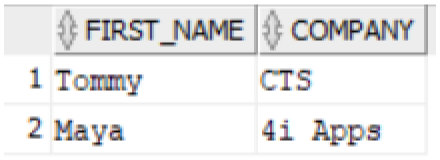


	NAME	EXPENDITURE	CREDIT_LIMIT
1	International Paper	72870.81	100
2	Emerson Electric	127498.5	100
3	Jabil Circuit	86993.62	500
4	NextEra Energy	106871.39	600
5	Aflac	19239.63	200
6	AutoNation	99748.67	200
7	Plains GP Holdings	36999.63	100
8	Jabil Circuit	14518.79	500
9	Alcoa	2715.35	100
10	Emerson Electric	119212.89	100

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;
```

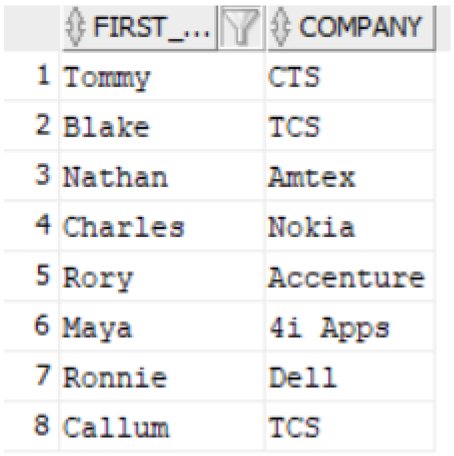


	FIRST_NAME	COMPANY
1	Tommy	CTS
2	Maya	4i Apps

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';
```

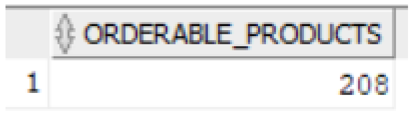


	FIRST_...	COMPANY
1	Tommy	CTS
2	Blake	TCS
3	Nathan	Amtex
4	Charles	Nokia
5	Rory	Accenture
6	Maya	4i Apps
7	Ronnie	Dell
8	Callum	TCS

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
   );
```



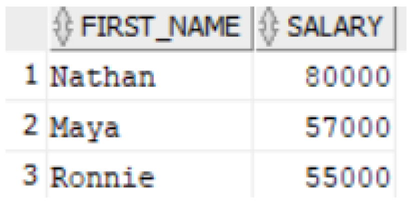
	ORDERABLE_PRODUCTS
1	208

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;
```

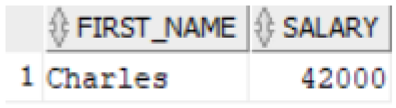


	FIRST_NAME	SALARY
1	Nathan	80000
2	Maya	57000
3	Ronnie	55000

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;
```

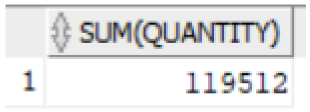


	FIRST_NAME	SALARY
1	Charles	42000

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;
```

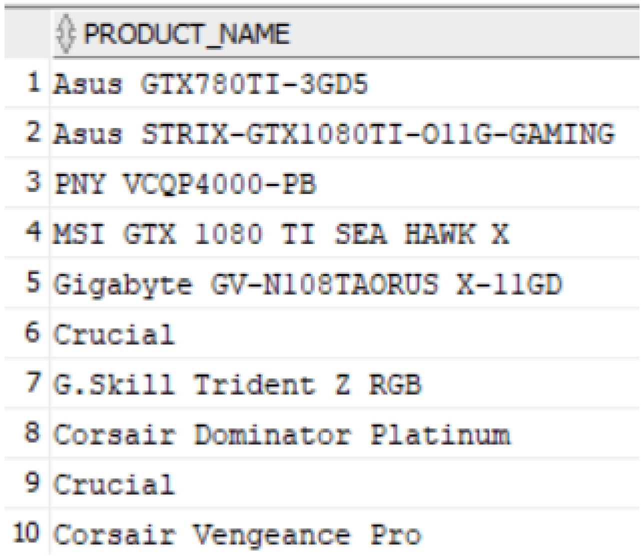


	SUM(QUANTITY)
1	119512

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
  );
```

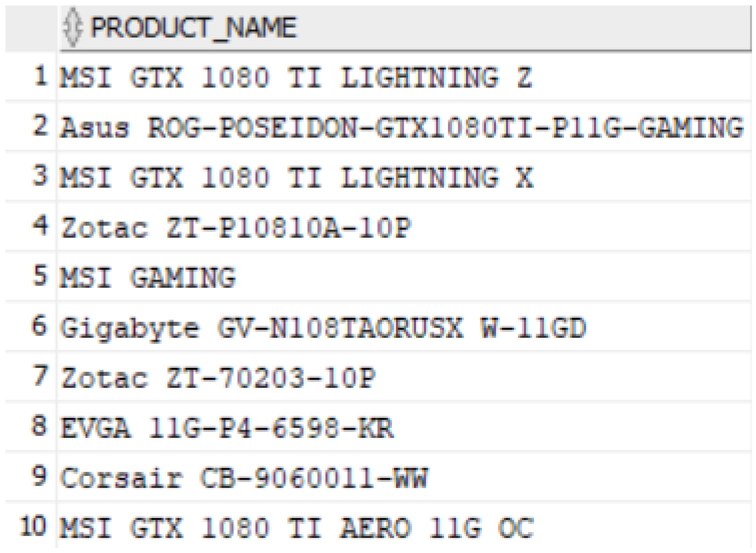


	PRODUCT_NAME
1	Asus GTX780TI-3GD5
2	Asus STRIX-GTX1080TI-O11G-GAMING
3	PNY VCQP4000-PB
4	MSI GTX 1080 TI SEA HAWK X
5	Gigabyte GV-N108TAORUS X-11GD
6	Crucial
7	G.Skill Trident Z RGB
8	Corsair Dominator Platinum
9	Crucial
10	Corsair Vengeance Pro

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
  );
```

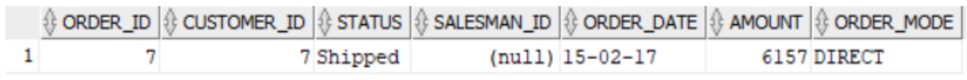


	PRODUCT_NAME
1	MSI GTX 1080 TI LIGHTNING Z
2	Asus ROG-POSEIDON-GTX1080TI-P11G-GAMING
3	MSI GTX 1080 TI LIGHTNING X
4	Zotac ZT-P10810A-10P
5	MSI GAMING
6	Gigabyte GV-N108TAORUSX W-11GD
7	Zotac ZT-70203-10P
8	EVGA 11G-P4-6598-KR
9	Corsair CB-9060011-WW
10	MSI GTX 1080 TI AERO 11G OC

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;
```

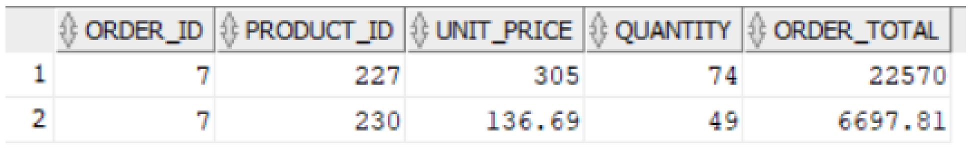


	ORDER_ID	CUSTOMER_ID	STATUS	SALESMAN_ID	ORDER_DATE	AMOUNT	ORDER_MODE
1	7	7	Shipped	(null)	15-02-17	6157	DIRECT

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;
```

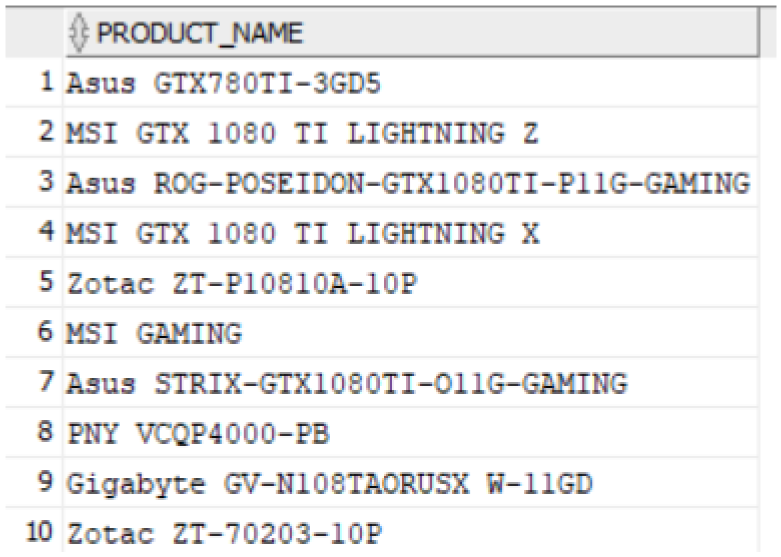


	ORDER_ID	PRODUCT_ID	UNIT_PRICE	QUANTITY	ORDER_TOTAL
1	7	227	305	74	22570
2	7	230	136.69	49	6697.81

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
    );
```

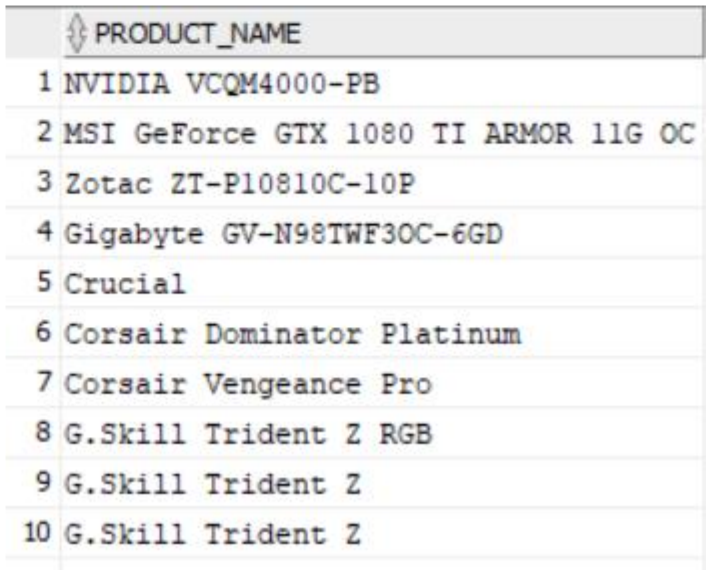


	PRODUCT_NAME
1	Asus GTX780TI-3GD5
2	MSI GTX 1080 TI LIGHTNING Z
3	Asus ROG-POSEIDON-GTX1080TI-P11G-GAMING
4	MSI GTX 1080 TI LIGHTNING X
5	Zotac ZT-P10810A-10P
6	MSI GAMING
7	Asus STRIX-GTX1080TI-O11G-GAMING
8	PNY VCQP4000-PB
9	Gigabyte GV-N108TAORUSX W-11GD
10	Zotac ZT-70203-10P

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
  );
```

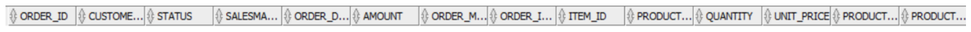


	PRODUCT_NAME
1	NVIDIA VCQM4000-PB
2	MSI GeForce GTX 1080 TI ARMOR 11G OC
3	Zotac ZT-P10810C-10P
4	Gigabyte GV-N98TWF3OC-6GD
5	Crucial
6	Corsair Dominator Platinum
7	Corsair Vengeance Pro
8	G.Skill Trident Z RGB
9	G.Skill Trident Z
10	G.Skill Trident Z

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;
```



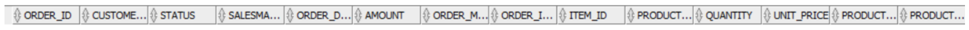
ORDER_ID	CUSTOME...	STATUS	SALESMA...	ORDER_D...	AMOUNT	ORDER_M...	ORDER_I...	ITEM_ID	PRODUCT...	QUANTITY	UNIT_PRICE	PRODUCT...	PRODUCT...

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;
```

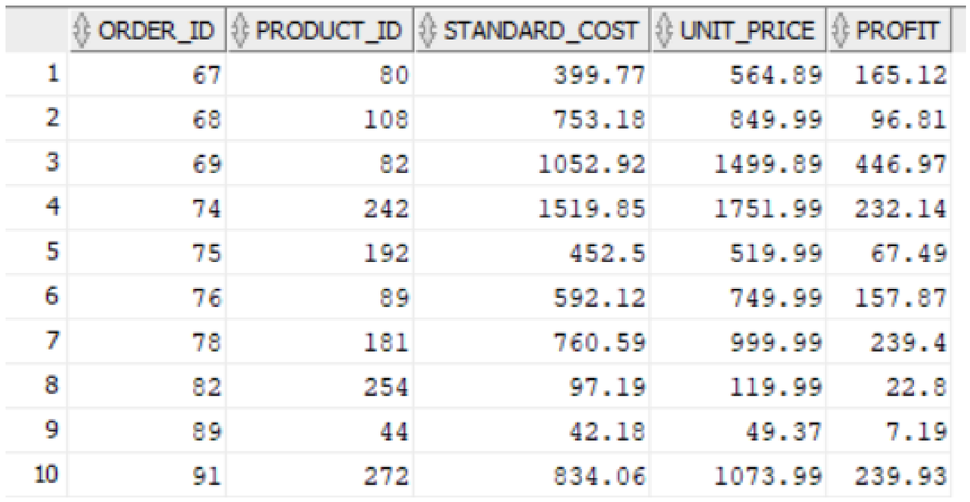


ORDER_ID	CUSTOME...	STATUS	SALESMA...	ORDER_D...	AMOUNT	ORDER_M...	ORDER_I...	ITEM_ID	PRODUCT...	QUANTITY	UNIT_PRICE	PRODUCT...	PRODUCT...

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;
```

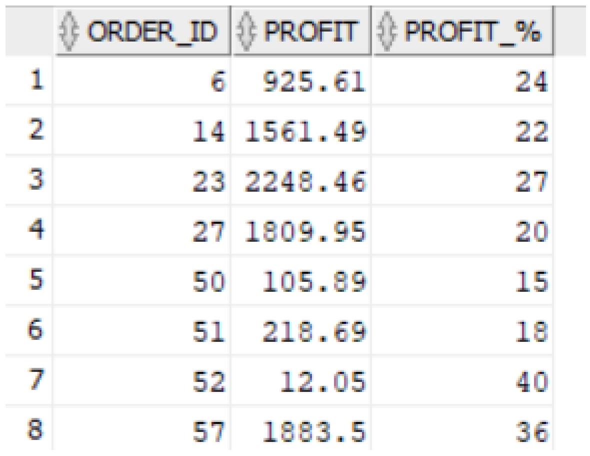


	ORDER_ID	PRODUCT_ID	STANDARD_COST	UNIT_PRICE	PROFIT
1	67	80	399.77	564.89	165.12
2	68	108	753.18	849.99	96.81
3	69	82	1052.92	1499.89	446.97
4	74	242	1519.85	1751.99	232.14
5	75	192	452.5	519.99	67.49
6	76	89	592.12	749.99	157.87
7	78	181	760.59	999.99	239.4
8	82	254	97.19	119.99	22.8
9	89	44	42.18	49.37	7.19
10	91	272	834.06	1073.99	239.93

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;
```

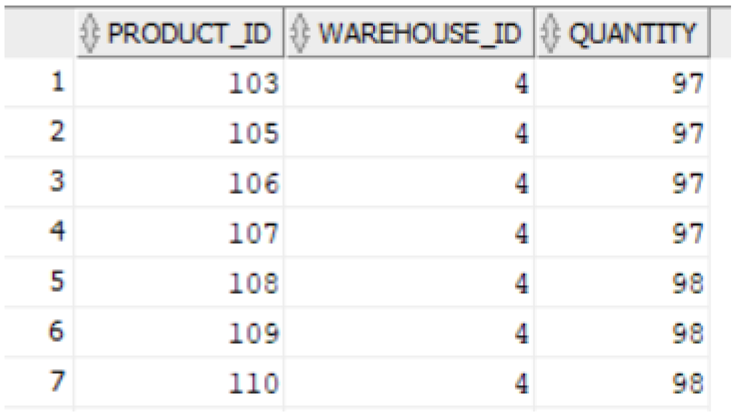


	ORDER_ID	PROFIT	PROFIT_%
1	6	925.61	24
2	14	1561.49	22
3	23	2248.46	27
4	27	1809.95	20
5	50	105.89	15
6	51	218.69	18
7	52	12.05	40
8	57	1883.5	36

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;
```



	PRODUCT_ID	WAREHOUSE_ID	QUANTITY
1	103	4	97
2	105	4	97
3	106	4	97
4	107	4	97
5	108	4	98
6	109	4	98
7	110	4	98

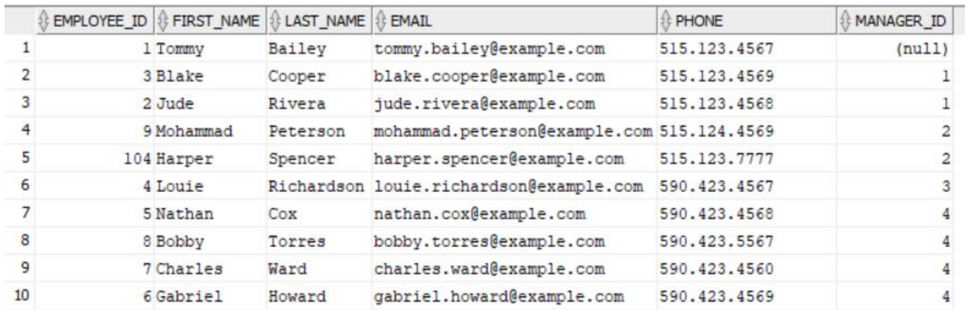
Created a new table called xx1554_product using create table syntax.

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;
```

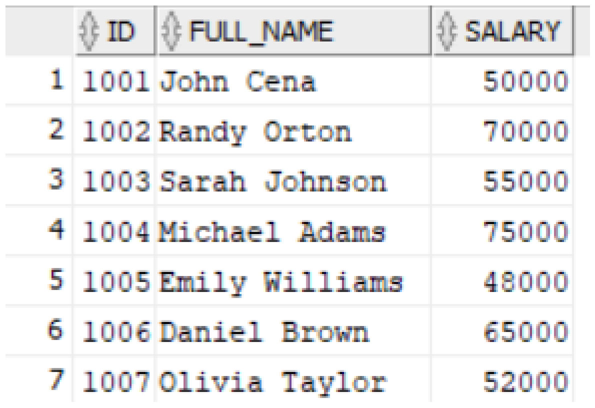


	EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE	MANAGER_ID
1	1	Tommy	Bailey	tommy.bailey@example.com	515.123.4567	(null)
2	3	Blake	Cooper	blake.cooper@example.com	515.123.4569	1
3	2	Jude	Rivera	jude.rivera@example.com	515.123.4568	1
4	9	Mohammad	Peterson	mohammad.peterson@example.com	515.124.4569	2
5	104	Harper	Spencer	harper.spencer@example.com	515.123.7777	2
6	4	Louie	Richardson	louie.richardson@example.com	590.423.4567	3
7	5	Nathan	Cox	nathan.cox@example.com	590.423.4568	4
8	8	Bobby	Torres	bobby.torres@example.com	590.423.5567	4
9	7	Charles	Ward	charles.ward@example.com	590.423.4560	4
10	6	Gabriel	Howard	gabriel.howard@example.com	590.423.4569	4

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;
```

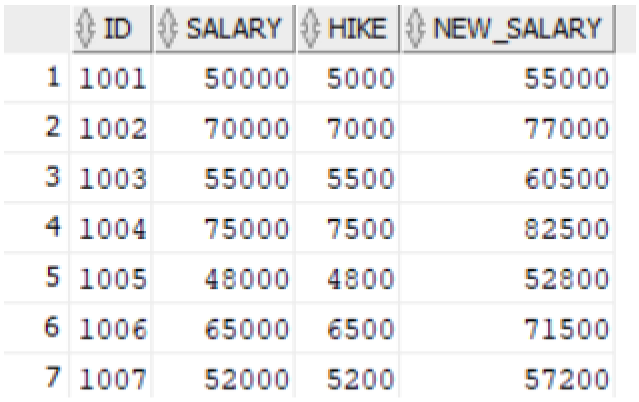


	ID	FULL_NAME	SALARY
1	1001	John Cena	50000
2	1002	Randy Orton	70000
3	1003	Sarah Johnson	55000
4	1004	Michael Adams	75000
5	1005	Emily Williams	48000
6	1006	Daniel Brown	65000
7	1007	Olivia Taylor	52000

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;
```



	ID	SALARY	HIKE	NEW_SALARY
1	1001	50000	5000	55000
2	1002	70000	7000	77000
3	1003	55000	5500	60500
4	1004	75000	7500	82500
5	1005	48000	4800	52800
6	1006	65000	6500	71500
7	1007	52000	5200	57200

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.