```sql
—----------
-- Create products table
CREATE TABLE products
(
   id           INTEGER PRIMARY KEY AUTOINCREMENT,
   name         VARCHAR(100),
   price        FLOAT,
   release_date  DATE
);

-- Insert data into products table
INSERT INTO products (name, price, release_date) VALUES ('iPhone 15', 800, '2023-08-22');
INSERT INTO products (name, price, release_date) VALUES ('Macbook Pro', 2100, '2022-10-12');
INSERT INTO products (name, price, release_date) VALUES ('Apple Watch 9', 550, '2022-09-04');
INSERT INTO products (name, price, release_date) VALUES ('iPad', 400, '2020-08-25');
INSERT INTO products (name, price, release_date) VALUES ('AirPods', 420, '2024-03-30');


-- Create customers table
CREATE TABLE customers
(
   id        INTEGER PRIMARY KEY AUTOINCREMENT,
   name      VARCHAR(100),
   email     VARCHAR(100)  -- Increased length for email to match standard
);

-- Insert data into customers table
INSERT INTO customers (name, email) VALUES ('Meghan Harley', 'mharley@demo.com');
INSERT INTO customers (name, email) VALUES ('Rosa Chan', 'rchan@demo.com');
INSERT INTO customers (name, email) VALUES ('Logan Short', 'lshort@demo.com');
INSERT INTO customers (name, email) VALUES ('Zaria Duke', 'zduke@demo.com');


-- Create employees table
CREATE TABLE employees
(
   id   INTEGER PRIMARY KEY AUTOINCREMENT,
   name VARCHAR(100)
);

-- Insert data into employees table
INSERT INTO employees (name) VALUES ('Nina Kumari');
INSERT INTO employees (name) VALUES ('Abrar Khan');
INSERT INTO employees (name) VALUES ('Irene Costa');


-- Create sales_order table
CREATE TABLE sales_order
(
   order_id      INTEGER PRIMARY KEY AUTOINCREMENT,
   order_date    DATE,
   quantity      INTEGER,
   prod_id       INTEGER,
   status        VARCHAR(20),
   customer_id   INTEGER,
   emp_id        INTEGER,
   FOREIGN KEY (prod_id) REFERENCES products(id),
   FOREIGN KEY (customer_id) REFERENCES customers(id),
   FOREIGN KEY (emp_id) REFERENCES employees(id)
);

-- Insert data into sales_order table
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-01', 2, 1, 'Completed', 1, 1);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-01', 3, 1, 'Pending', 2, 2);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-02', 3, 2, 'Completed', 3, 2);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-03', 3, 3, 'Completed', 3, 2);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-04', 1, 1, 'Completed', 3, 2);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-04', 1, 3, 'completed', 2, 1);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-04', 1, 2, 'On Hold', 2, 1);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-05', 4, 2, 'Rejected', 1, 2);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-06', 5, 5, 'Completed', 1, 2);
INSERT INTO sales_order (order_date, quantity, prod_id, status, customer_id, emp_id) VALUES
('2024-01-06', 1, 1, 'Cancelled', 1, 1);


-- Select data from all tables
-- SELECT * FROM products;
-- SELECT * FROM customers;
-- SELECT * FROM employees;
-- SELECT * FROM sales_order;

-- SELECT Upper(status) , COUNT(order_id)
-- FROM sales_order
-- GROUP BY 1

-- SELECT order_id
-- FROM sales_order

-- 6. For orders purchasing more than 1 item , How many are still not completed
-- SELECT COUNT(*) Not_completed_order FROM sales_order
-- WHERE lower(status) IS NOT "completed" AND quantity > 1;

-- SELECT * FROM sales_order
-- WHERE lower(status) IS NOT "completed" AND quantity > 1;

-- 7. Find the total number of orders corresponding to each delivery
-- status by ignoring the case in the delivery status.
-- The status with highest no of orders should be at the top.

-- Select status, COUNT(order_id) as Total_orders
-- FROM sales_order
-- GROUP BY Upper(status)
-- -- ORDER BY Total_orders DESC
-- Order BY 2 DESC

-- Select status,
-- CASE WHEN status = 'completed' THEN 'Completed'
-- else status
-- END as updated_status
-- From sales_order

-- Using Subquery

-- SELECT updated_status , COUNT (*) as Total_orders
-- FROM (
--    SELECT status,
--        CASE WHEN status = 'completed' THEN 'Completed'
--        ELSE status
--        END as updated_status
--        FROM sales_order
--    ) as sq
-- GROUP BY updated_status
-- ORDER BY Total_orders DESC


-- 8. Write a query to identify the total products purchased by each customer


-- SELECT c.name , SUM(so.quantity) as Total_products_sold
--    -- ,p.name
-- FROM sales_order as so
-- JOIN customers as c ON c.id = so.customer_id
-- -- JOIN products as p ON p.id = so.prod_id
-- GROUP BY c.name
-- ORDER BY 2 DESC

-- SELECT c.name as Customer_name , SUM(so.quantity) as Total_products_sold ,p.name
-- FROM sales_order as so
-- JOIN customers as c ON c.id = so.customer_id
-- JOIN products as p ON p.id = so.prod_id
-- GROUP BY c.name
-- ORDER BY 2 DESC

-- 9. Display the total sales and average sales done for each day.

-- SELECT so.order_date , p.name , SUM(so.quantity * p.price) as Total_sales, AVG(so.quantity * p.price)
as Average_sales
-- FROM sales_order as so
-- JOIN products as p ON p.id = so.prod_id
-- WHERE lower(so.status) IS "completed"
-- GROUP BY so.order_date
-- ORDER BY so.order_date DESC


-- Select *
-- FROM sales_order as so
-- JOIN products as p ON p.id = so.prod_id
-- GROUP BY order_date

-- 10. Display the customer name, employee name, and total sale amount of
-- all orders which are either on hold or pending.

-- SELECT p.name,c.name , e.name , so.status , SUM(p.price * so.quantity) as Total_sales
-- FROM sales_order as so
-- JOIN products as p ON p.id = so.prod_id
-- JOIN customers as c ON c.id = so.customer_id
-- JOIN employees as e ON e.id = so.emp_id
-- WHERE lower(so.status) IN ("on hold" , "pending")
-- GROUP BY p.name


-- SELECT p.name ,c.name as Customer , e.name as employe , so.status , SUM(price * quantity) as
Total_sales
-- FROM sales_order as so
-- JOIN products as p ON so.prod_id = p.id
-- JOIN customers as c ON so.customer_id = c.id
-- JOIN employees as e ON so.emp_id = e.id
-- WHERE lower(so.status) IN ("on hold" , "pending")
-- GROUP BY p.name


-- SELECT so.order_date,so.status,so.quantity , p.name ,p.price FROM sales_order as so
-- JOIN products as p ON p.id = so.prod_id
-- Where lower(so.status) IS NOT "completed"
-- 11. Fetch all the orders which were neither completed/pending
-- or were handled by the employee Abrar.
-- Display employee name and all details of order.


SELECT so.* ,e.name AS employe
FROM sales_order AS so
JOIN employees AS e ON so.emp_id = e.id
WHERE LOWER(so.status) NOT IN ('completed', 'pending')
   OR LOWER(e.name) LIKE '%Abrar%';
```

—---------------

```
drop table if exists products;
create table products
(
        id                      int generated always as identity primary key,
        name            varchar(100),
        price           float,
        release_date   date

drop table if exists customers;
create table customers
(
   id      int generated always as identity primary key,
   name      varchar(100),
   email     varchar(30)
);


drop table if exists employees;
create table employees
(
   id      int generated always as identity primary key,
   name      varchar(100)
);
insert into employees values(default,'Irene Costa');



drop table if exists sales_order;
create table sales_order
```

```
(
	order_id		int generated always as identity primary key,
	order_date		date,
	quantity		int,
	prod_id			int references products(id),
	status			varchar(20),
	customer_id		int references customers(id),
	emp_id			int,
	constraint fk_so_emp foreign key (emp_id) references employees(id)
);
insert into sales_order values(default,to_date('01-01-2024','dd-mm-yyyy'),2,1,'Completed',1,1);
```

```
SELECT * FROM products;
SELECT * FROM customers;
SELECT * FROM employees;
SELECT * FROM sales_order;
```

1.Identify the total no of products sold

```
Select SUM(quantity) as Total_product_sold
From sales_order
```

2.Other than Completed, display the available delivery statuses

```
Select status
From sales_order
Where lower(status) IS NOT = “completed”  or <> !=
```

3. Display the order id, order_date and product_name for all the completed orders.

```
Select order_id , order_date , product_name
From sales_order so
Inner join products p
on p.id = so.prod_id
Where lower(so.status) = “completed”;
```

4.Sort the above query to show the earliest orders at the top. Also, display the customer who purchased these orders.

```
Select order_id , order_date , p.name as product_name , c.name as customer_name
```

```
From sales_order so
Inner join products p on p.id = so.prod_id
Inner join customer c on c.id = so.customer_id
Where lower(so.status) = “completed”;
ORDER BY order_date asc;
```

5. Display the total no of orders corresponding to each delivery status

```
Select lower(status) , COUNT(order_id) as Total_no_orders
From sales_order
Group BY lower(status)
```

6. For orders purchasing more than 1 item , How many are still not completed

```
-- SELECT COUNT(*) Not_completed_order FROM sales_order
-- WHERE lower(status) IS NOT "completed" AND quantity > 1;

--I was interested for all the details of the order as well
SELECT * FROM sales_order
WHERE lower(status) IS NOT "completed" AND quantity > 1;
```

7. Find the total number of orders corresponding to each delivery status by ignoring the case in the delivery status. The status with highest no of orders should be at the top.

```
Select status, COUNT(order_id) as Total_orders
FROM sales_order
GROUP BY Upper(status)
-- ORDER BY Total_orders DESC
Order BY 2 DESC
```

Code inside subquery

```
-- Select status,
-- CASE WHEN status = 'completed' THEN 'Completed'
-- else status
-- END as updated_status
-- From sales_order

-- Using Subquery

-- SELECT updated_status , COUNT (*) as Total_orders
-- FROM (
--     SELECT status,
--         CASE WHEN status = 'completed' THEN 'Completed'
--         ELSE status
--         END as updated_status
--         FROM sales_order
--     ) as sq
-- GROUP BY updated_status
-- ORDER BY Total_orders DESC


-- 8. Write a query to identify the total products purchased by each customer
SELECT c.name , SUM(so.quantity) as Total_products_sold
FROM sales_order as so
JOIN customers as c ON c.id = so.customer_id
GROUP BY c.name
ORDER BY 2 DESC

-- SELECT c.name , SUM(so.quantity) as Total_products_sold ,p.name
-- FROM sales_order as so
-- JOIN customers as c ON c.id = so.customer_id
-- JOIN products as p ON p.id = so.prod_id
-- GROUP BY c.name
-- ORDER BY 2 DESC


-- 9. Display the total sales and average sales done for each day.

SELECT so.order_date , p.name , SUM(so.quantity * p.price) as Total_sales, AVG(so.quantity *
p.price) as Average_sales
FROM sales_order as so
JOIN products as p ON p.id = so.prod_id
WHERE lower(so.status) IS "completed"
GROUP BY so.order_date
ORDER BY so.order_date DESC


-- Select *
-- FROM sales_order as so
-- JOIN products as p ON p.id = so.prod_id
-- GROUP BY order_date


-- 10. Display the customer name, employee name, and total sale amount of
-- all orders which are either on hold or pending.

-- SELECT p.name,c.name , e.name , so.status , SUM(p.price * so.quantity) as Total_sales
-- FROM sales_order as so
-- JOIN products as p ON p.id = so.prod_id
-- JOIN customers as c ON c.id = so.customer_id
-- JOIN employees as e ON e.id = so.emp_id
-- WHERE lower(so.status) IN ("on hold" , "pending")
-- GROUP BY p.name

-- SELECT so.order_date,so.status,so.quantity , p.name ,p.price FROM sales_order as so
-- JOIN products as p ON p.id = so.prod_id
-- Where lower(so.status) IS NOT "completed"
```

Solution :

```
SELECT p.name ,c.name as Customer , e.name as employe , so.status , SUM(price * quantity)
as Total_sales
FROM sales_order as so
JOIN products as p ON so.prod_id = p.id
JOIN customers as c ON so.customer_id = c.id
JOIN employees as e ON so.emp_id = e.id
WHERE lower(so.status) IN ("on hold" , "pending")
GROUP BY p.name



-- 11. Fetch all the orders which were neither completed/pending
-- or were handled by the employee Abrar.
-- Display employee name and all details of order.


SELECT so.* ,e.name AS employe
FROM sales_order AS so
JOIN employees AS e ON so.emp_id = e.id
WHERE LOWER(so.status) NOT IN ('completed', 'pending')
  OR LOWER(e.name) LIKE '%Abrar%';
```

```
-- 12. Fetch the orders which cost more than 2000 but did not
-- include the MacBook Pro. Print the total sale amount as well.


SELECT so.*, price , p.name,(quantity * price) as total_cost
FROM sales_order as so
JOIN products as p ON p.id = so.prod_id
WHERE (quantity * price) > 2000
    AND lower(p.name) NOT LIKE '%macbook%'
    -- AND p.name IS NOT 'Macbook Pro'



-- 13. Identify the customers who have not purchased any product yet.

-- SELECT c.* , so.prod_id
-- FROM customers as c
-- LEFT JOIN sales_order as so ON so.customer_id = c.id
-- WHERE so.prod_id IS NULL

-- SELECT DISTINCT customer_id
-- FROM sales_order

Subquery solution
-- SELECT * FROM customers
-- WHERE id NOT IN (SELECT DISTINCT customer_id FROM sales_order)

-- 14. Write a query to identify the total products purchased
-- by each customer. Return all customers irrespective of
-- whether they have made a purchase or not.
-- Sort the result with the highest no of orders at the top.

-- Select c.name , COALESCE(SUM(quantity),0) as Tot_prod_purchased
-- FROM customers c
-- LEFT JOIN sales_order so ON c.id = so.customer_id
-- GROUP BY c.name
-- ORDER BY 2 DESC

-- COALESCE take the First argument and displaces
-- that or 2nd argument if 1st one is NULL


-- 15. Corresponding to each employee, display the total sales
-- they made of all the completed orders.
```

-- Display total sales as 0 if an employee made no sales yet.

SELECT e.name, COALESCE(SUM(so.quantity * p.price),0) as Tot_sales
FROM sales_order so
JOIN products p ON p.id = so.prod_id
RIGHT JOIN employees e ON e.id = so.emp_id AND lower(status) = 'completed'
GROUP BY e.name

-- HERE we have add the filter condition in JOIN function,
-- so we the use inside the AND as JOIN condition

-- 16. Re-write the above query to display the total sales made
-- by each employee corresponding to each customer.
-- If an employee has not served a customer yet
-- then display "-" under the customer.

SELECT e.name, COALESCE(c.name,'-'), COALESCE(SUM(so.quantity * p.price),0) as Tot_sales
FROM sales_order so
JOIN products p ON p.id = so.prod_id
JOIN customers c ON c.id = so.customer_id
RIGHT JOIN employees e ON e.id = so.emp_id AND lower(status) = 'completed'
GROUP BY e.name,c.name
ORDER BY 1,2;

-- ALWAYS use all the columns in GROUP BY , if they are inside aggregate then OK

-- 17. Re-write the above query to display only those
-- records where the total sales are above 1000

SELECT e.name, COALESCE(c.name,'-'), COALESCE(SUM(so.quantity * p.price),0) as Tot_sales
FROM sales_order so
JOIN products p ON p.id = so.prod_id
JOIN customers c ON c.id = so.customer_id
RIGHT JOIN employees e ON e.id = so.emp_id AND lower(status) = 'completed'
GROUP BY e.name,c.name
HAVING COALESCE(SUM(so.quantity * p.price),0) > 1000
ORDER BY 1,2;

-- Having clause should be used to filter the Grouped Data with Group By clause

```
-- 18. Identify employees who have served more than 2 customers.

SELECT e.name ,COUNT(DISTINCT(c.name))
FROM sales_order so
JOIN employees e ON e.id = so.emp_id
JOIN customers c ON c.id = so.customer_id
GROUP BY e.name
HAVING COUNT(DISTINCT(c.name)) > 2
ORDER BY 1

-- 19. Identify the customers who have purchased more than 5 products

Select c.name , SUM(so.quantity) as Tot_prod
FROM sales_order so
JOIN customers c ON c.id = so.customer_id
GROUP BY c.name
HAVING SUM(so.quantity) > 5
ORDER BY 2 DESC;

-- 20. Identify customers whose average purchase cost
-- exceeds the average sale of all the orders.



-- SELECT p.name ,AVG(quantity * price) as average_price
-- FROM sales_order so
-- JOIN products p ON p.id = so.prod_id
-- GROUP BY p.name


-- SELECT AVG(quantity * price) as average_price
-- FROM sales_order so
-- JOIN products p ON p.id = so.prod_id
```

Solution :-

```
SELECT c.name , AVG(quantity * price) as avg_purchase_cost
FROM sales_order so
JOIN customers c ON c.id = so.customer_id
JOIN products p ON p.id = so.prod_id
GROUP BY 1
HAVING AVG(quantity * price) > (SELECT AVG(quantity * price) as average_price
                    FROM sales_order so
                    JOIN products p ON p.id = so.prod_id)
ORDER BY 2 DESC;
```