# SQL Assignment (Bansi Ghanva)

## 1) Get the total number of orders placed by each customer

```
SELECT customers.customer_id,
customers.company_name,
COUNT(orders.order_id) AS total_orders
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id
GROUP BY customers.customer_id, customers.company_name;
```

- (JOIN) Joins the customers and orders tables based on the condition customer_id
- (GROUP BY) Groups the result by customer_id and company_name
- (COUNT) Counts the no. of occurrences of the order_id column in the orders table

## 2) Find all suppliers who provide products in the ‘Seafood’ category

```
SELECT suppliers.supplier_id,
categories.category_name
FROM suppliers
JOIN products ON suppliers.supplier_id =
products.supplier_id
JOIN categories ON products.category_id =
categories.category_id
WHERE categories.category_name = "Seafood"
```

- (SELECT) Selects the supplier_id and category_name from thesuppliers table
- (JOIN) Joins the products table on the supplier_id column and the categorie table on the category_id column
- (WHERE) Filters the results to only include suppliers whose products are in the Seafood category

## 3) Get the total quantity of each product sold

```
SELECT p.product_id, p.product_name,
SUM(od.quantity) AS total_sales
FROM products p
JOIN order_details od ON p.product_id = od.product_id
GROUP BY p.product_id, p.product_name;
```

- SUM() Calculates the total quantity of each product sold by summing up the quantity column from order_details
- (GROUP BY) Results are grouped by product_id and product_name

## 4) Find the total sales (Quantity*Unit_Price) for each category of products

```
SELECT categories.category_name,
SUM(order_details.quantity * products.unit_price) AS
total_sales
FROM products
JOIN categories ON products.category_id =
categories.category_id
JOIN order_details ON products.product_id =
order_details.product_id
GROUP BY categories.category_name;
```

- (Total Sales) Calculates the total sales for each category by multiplying the quantity from the order_details table with the unit price from the products table.
- (GROUP BY) groups the result by the categories.category_name column to calculate the total sales

## 5) List the employees and the number of orders each employee has taken

```
SELECT e.employee_id, CONCAT(e.first_name, ' ',
e.last_name) AS employee_name, COUNT(o.order_id)
AS order_count
FROM employees e
LEFT JOIN orders o ON e.employee_id =
o.employee_id
GROUP BY e.employee_id, employee_name;
```

- (CONCAT)Groups the first name and last name together for employee_name
- (LEFT JOIN) Ensures that all employees, even those without any orders, are included in the result.

## 6) Get the customers who have placed more than 10 orders

```
SELECT customer_id, COUNT(order_id) AS
total_orders
FROM orders
GROUP BY customer_id
HAVING COUNT(order_id) > 10;
```

- COUNT() Counts the number of orders for each customer
- (GROUP BY) Results are grouped by customer_id
- (HAVING BY) Filters results and only includes customers with more than 10 orders

## 7) Get the top 5 most sold products

```
SELECT p.product_id, p.product_name,
SUM(od.quantity) AS total_quantity_sold
FROM products p
JOIN order_details od ON p.product_id = od.product_id
GROUP BY p.product_id, p.product_name
ORDER BY total_quantity_sold DESC
LIMIT 5;
```

- SUM() Calculates the total quantity of each product sold by summing up the quantity column from the order_details table
- (ORDER BY) Sorts the results in descending order based on the total quantity sold
- (LIMIT) Retrieve only the top 5 products with the highest quantity sold

## 8) Find the products that have never been ordered

```
SELECT product_id, product_name
FROM products
WHERE product_id NOT IN (
  SELECT DISTINCT product_id
  FROM order_details
);
```

- (DISTINCT) Subquery to select the distinct product_id values from order_details
- Main query filters the products by those product_id values that are not present in the subquery result, indicating they have never been ordered

## 9) Find the customers who have not placed any orders

```
SELECT c.customer_id, c.company_name
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
WHERE o.customer_id IS NULL;
```

- (LEFT JOIN) Performs left join with orders table based on customer_id column
- (WHERE) Filters the result to only include rows where the customer_id in the orders table is NULL, indicating that the customer has not placed any orders

## 10) List all ‘Orders’ with ‘Customer’ details and ‘Employee’ who processed it

```
SELECT o.order_id, o.order_date, c.customer_id,
c.company_name, e.employee_id,
CONCAT(e.first_name, ' ', e.last_name) AS
employee_name
FROM orders o
JOIN customers c ON o.customer_id = c.customer_id
JOIN employees e ON o.employee_id =
e.employee_id;
```

- (CONCAT)Groups the first name and last name together for employee_name
- This will give you the list of all orders with the associated customer details and the employee who processed each order.

## 11) Calculate the average product price by category

```
SELECT c.category_id, c.category_name,
AVG(p.unit_price) AS average_price
FROM products p
JOIN categories c ON p.category_id = c.category_id
GROUP BY c.category_id, c.category_name;
```

- AVG() Calculates the average product price by taking the average of the unit_price column from the products table
- (GROUP BY) Results are grouped by category_id'and category_name

## 12) Find the total revenue generated by each employee

```
SELECT  employees.employee_id,
CONCAT(employees.first_name, ' ', employees.last_name)
AS employee_name,
SUM(order_details.quantity * products.unit_price) AS
total_revenue
FROM employees
JOIN orders ON employees.employee_id =
orders.employee_id
JOIN order_details ON orders.order_id =
order_details.order_id
JOIN products ON order_details.product_id =
products.product_id
GROUP BY employees.employee_id,   employee_name;
```

- (Total Revenue) Calculates the total revenue generated by each employee by multiplying the quantity from the order_details table with the unit price from the products table.
- (GROUP BY) Ensures that the calculation is performed for each unique employee ID and name.

## 13) List all orders shipped to ‘Germany’

```
SELECT
o.order_id,o.order_date,o.ship_country,o.shipped_date
FROM orders o

WHERE ship_country = 'Germany';
```

- It retrieves all the orders that were shipped to Germany along with the order.id, order.date, order.ship_country, order.shipped_date

## 14) Find the most expensive product in each category

```
SELECT p.category_id, c.category_name,
p.product_id, p.product_name, p.unit_price
FROM products p
JOIN categories c ON p.category_id = c.category_id
WHERE (p.category_id, p.unit_price) IN (
  SELECT category_id, MAX(unit_price)
  FROM products
  GROUP BY category_id
```

- Main query filters the results by comparing the category and unit price to the results of the subquery
- The subquery retrieves the maximum unit price for each category using the MAX() function and grouping by category_id

## 15) Find the total revenue for the year 2016

```
SELECT
SUM(order_details.quantity * products.unit_price) AS
total_revenue
FROM orders
JOIN order_details ON orders.order_id =
order_details.order_id
JOIN products ON order_details.product_id =
products.product_id
WHERE YEAR(orders.order_date) = 2016;
```

- Select statement calculates the total_revenue by using the SUM function
- (WHERE) Filters the orders based on the year 2016 by extracting the year from the order_date column

## 16) List all products that are discontinued

```
SELECT p.product_id,p.product_name,p.discontinued
FROM products p
WHERE discontinued = 1;
```

- (WHERE) Filters the results and only includes products where the discontinued column is equal to 1, indicating that they are discontinued

17) List all the distinct countries to which orders have been shipped

```
SELECT DISTINCT ship_country

FROM orders;
```

- Query selects the ship_country column from the orders table and applies the DISTINCT keyword.
- (DISTINCT) Eliminates duplicate values,so result will contain only distinct countries

18) Find all employees who report to ‘Andrew Fuller’

```
SELECT e.employee_id, CONCAT(e.first_name, ' ',
e.last_name) AS employee_name
FROM employees e
JOIN employees f ON e.reports_to = f.employee_id
WHERE CONCAT(f.first_name, ' ', f.last_name) =
'Andrew Fuller';
```

- The letter ‘f’ is used as an alias for the second instance of the employees table (aliased as e)allows for differentiation between the two instances within the query.
- (WHERE) Filters the results to only include employees whose supervisor is 'Andrew Fuller'

19) Find the customers who have spent more than $5000 in total

```
SELECT customers.customer_id, customers.company_name,
SUM(order_details.quantity * products.unit_price) AS
total_spent
FROM customers
JOIN orders ON customers.customer_id = orders.customer_id
JOIN order_details ON orders.order_id = order_details.order_id
JOIN products ON order_details.product_id =
products.product_id
GROUP BY customers.customer_id, customers.company_name
HAVING SUM(order_details.quantity * products.unit_price) >
5000;
```

- (JOIN) Products table with order_details based on product_id column
- (HAVING) Filters the results to include only the customers whose total spent amount is greater than $5000

20) List the top 5 employees who have processed the most orders

```
SELECT e.employee_id, CONCAT(e.first_name, ' ',
e.last_name) AS employee_name, COUNT(o.order_id)
AS total_orders
FROM employees e
JOIN orders o ON e.employee_id = o.employee_id
GROUP BY e.employee_id, employee_name
ORDER BY total_orders DESC
LIMIT 5;
```

- COUNT() Counts the number of orders processed by each employee
- (ORDER BY) Orders the results in descending order based on the total number of orders
- (LIMIT) Limits the result to the top 5 employees

## 21) Get the list of customers who have ordered ‘Chai’ product

```
SELECT c.customer_id, c.company_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
WHERE p.product_name = 'Chai';
```

- (WHERE) Filters the results to include only the orders that have the 'Chai' product based on the 'product_name' column from the 'products' table

## 22) Get the employees who have processed orders for ‘Chai’ product

```
SELECT e.employee_id, CONCAT(e.first_name, ' ',
e.last_name) AS employee_name
FROM employees e
JOIN orders o ON e.employee_id = o.employee_id
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
WHERE p.product_name = 'Chai';
```

- (WHERE) Filters the results to include only the orders that have the 'Chai' product based on the 'product_name' column from the 'products' table
- (CONCAT)Groups the first name and last name together for employee_name

## 23) Find the most common shipping country

```
SELECT ship_country, COUNT(*) AS shipping_count
FROM orders
GROUP BY ship_country
ORDER BY shipping_count DESC
LIMIT 1;
```

- COUNT() Counts the occurences of each unique shipping country
- (ORDER BY) sort the results in descending order based on the count of shipping occurrences
- (LIMIT) Used to retrieve only the top 1 row, which represents the most common shipping country

## 24) Find the order with the highest total cost

```
SELECT orders.order_id,
SUM(products.unit_price) AS total_cost
FROM orders
JOIN order_details ON orders.order_id =
order_details.order_id
JOIN products ON order_details.product_id =
products.product_id
GROUP BY orders.order_id
ORDER BY total_cost DESC
LIMIT 1;
```

- (GROUP BY) Groups the result by orders.order_id to calculate the total cost for each order.
- (ORDER BY) sort the results in descending order based on the total cost
- (LIMIT) Used to retrieve only the top 1 row, which represents the order with the highest total cost

## 25) Find the employees who have processed more than 100 orders

```
SELECT e.employee_id, CONCAT(e.first_name, ' ',
e.last_name) AS employee_name, COUNT(o.order_id)
AS total_orders
FROM employees e
JOIN orders o ON e.employee_id = o.employee_id
GROUP BY e.employee_id, employee_name
HAVING COUNT(o.order_id) > 100;
```

- COUNT() Counts the number of orders processed by each employee
- (HAVING) Filters the results to only include employees who have processed more than 100 orders

## 26) Find the customer who has ordered the most ‘Chai’ product

```
SELECT c.customer_id, c.company_name, COUNT(*)
AS total_orders
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
WHERE p.product_name = 'Chai'
GROUP BY c.customer_id, c.company_name
ORDER BY total_orders DESC
LIMIT 1;
```

- (COUNT) Counts the no. of orders
- (ORDER BY) sort the results in descending order based on the total number of orders
- (LIMIT) Retrieve only the top 1 row, which represents the customer who has ordered the most 'Chai' product

27)Find the average quantity of products ordered in each order

```
SELECT order_id, AVG(quantity) AS average_quantity

FROM order_details

GROUP BY order_id;
```

- (AVG) Find the average quantity of the products
- (GROUP BY) It groups the result with the help of the order_id

28) Find the top 3 most popular categories of products ordered

```
SELECT c.category_name, COUNT(*) AS order_count
FROM order_details od
JOIN products p ON od.product_id = p.product_id
JOIN categories c ON p.category_id = c.category_id
GROUP BY c.category_name
ORDER BY order_count DESC
LIMIT 3;
```

- (ORDER BY) Sorts the results in descending order based on order count
- (LIMIT) Retrieves only the top 3 rows, which represent the top 3 most popular categories of products ordered

## 29) Find the month in the year 2016 with the highest total sales

```
SELECT order_date,Total_sales
FROM (
SELECT orders.order_date,
    SUM(products.unit_price * order_details.Quantity
* (1 - order_details.Discount)) AS Total_sales
FROM Orders
     JOIN order_details ON orders.order_ID =
Order_Details.Order_ID
     JOIN products ON products.Product_ID =
order_details.Product_ID
WHERE YEAR(Order_Date) = 2016
GROUP BY orders.order_date) AS Monthly_sales
ORDER BY Total_sales DESC
LIMIT 1;
```

- WHERE Filters the orders based on the year 2016.
- GROUP BY Groups the orders by the order_date
- LIMIT 1 Ensures that only the highest total sales record is returned.

## 30) Find the employee who processed the most orders in August 2016

```
SELECT e.employee_id, CONCAT(e.first_name, ' ',
e.last_name) AS employee_name, COUNT(*) AS
order_count
FROM employees e
JOIN orders o ON e.employee_id = o.employee_id
WHERE o.order_date >= '2016-08-01' AND
o.order_date <= '2016-08-31'
GROUP BY e.employee_id, employee_name
ORDER BY order_count DESC
LIMIT 1;
```

- (ORDER BY) Sorts the results in descending order based on order count
- (LIMIT) Retrieves only the top 1 row, which represents the employee who processed the most orders in August 2016

## 31) Find the top 3 customers who have ordered the most products

```
SELECT c.customer_id, c.company_name,
COUNT(*) AS product_count
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
GROUP BY c.customer_id, c.company_name
ORDER BY product_count DESC
LIMIT 3;
```

- (ORDER BY) Sorts the results in descending order based on product count
- (LIMIT) Retrieves only the top 3 rows, which represent the top 3 customers who have ordered the most products

## 32) Find the employees who have not processed any orders

```
SELECT e.employee_id, CONCAT(e.first_name, ' ',
e.last_name) AS employee_name
FROM employees e
LEFT JOIN orders o ON e.employee_id =
o.employee_id
WHERE o.employee_id IS NULL;
```

- Query uses a LEFT JOIN between the 'employees' column
- WHERE filters the results to include only the rows where the employee_id in the orders table is NULL, indicating that there are no corresponding orders for that employee

## 33) Find the suppliers who supply the top 5 most sold products

```
SELECT s.supplier_id, s.company_name, COUNT(*)
AS product_count
FROM suppliers s
JOIN products p ON s.supplier_id = p.supplier_id
JOIN order_details od ON p.product_id =
od.product_id
GROUP BY s.supplier_id, s.company_name
ORDER BY product_count DESC
LIMIT 5;
```

- (ORDER BY) Sorts the results in descending order based on product count

## 34) Find the customers who have ordered products from all categories

```
SELECT c.customer_id, c.company_name
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
JOIN categories cat ON p.category_id =
cat.category_id
GROUP BY c.customer_id, c.company_name
HAVING COUNT(DISTINCT cat.category_id) =
(SELECT COUNT(*) FROM categories);
```

- HAVING Filters the results to include only the customers who have ordered products from all

## 35) Find the total sales for each year

```
SELECT YEAR(o.order_date) AS year,
SUM(od.quantity * p.unit_price) AS total_sales
FROM orders o
JOIN order_details od ON o.order_id = od.order_id
JOIN products p ON od.product_id = p.product_id
GROUP BY year
ORDER BY year;
```

- (ORDER BY) Sorts the result in ascending order based on the year
- (Total sales) multiplying the 'quantity' and 'unit_price' columns from the products table

36) Classify customers based on their total order amounts such that total order amounts > 5000 should be classified as ‘High Value’, if > 1000 then as ‘Medium Value’ and otherwise as ‘Low Value’

| company_name | TotalOrderAmount | CustomerClass |
|---|---|---|
| Alfreds Futterkiste | 4596.2 | Medium Value |
| Ana Trujillo Emparedados y helados | 1402.95 | Medium Value |
| Antonio Moreno Taquería | 7515.349999999999 | High Value |
| Around the Horn | 13806.5 | High Value |
| B's Beverages | 6089.9 | High Value |

```
SELECT c.company_name,TotalOrderAmount,
 CASE
 WHEN TotalOrderAmount > 5000 THEN 'High Value'
 WHEN TotalOrderAmount > 1000 THEN 'Medium Value'
 ELSE 'Low Value'
 END AS CustomerClass
FROM (
SELECT o.customer_id,
 SUM(od.quantity) AS TotalOrderAmount
FROM orders o
JOIN order_details od ON o.order_id = od.order_id
GROUP BY o.customer_id
) AS customer_orders
JOIN customers c ON customer_orders.customer_id = c.customer_id;
```

- (CASE) statement evaluates the total_order_amount and assigns the appropriate classification
- If the total_order_amount is greater than 5000, it is classified as'High Value
- If the total_order_amount is greater than 1000, but not greater than 5000, it is classified as Medium Value
- If the total_order_amount does not meet the above conditions, it is classified as Low Value

37) products based on their sales volume such that TotalQuantity > 1000 Then Classify SalesCategory as ‘High Sales’. If TotalQuantity>500 Then ‘Medium Sales’ and else ‘Lower Sales’

| product_name | TotalQuantity | SalesCategory |
|---|---|---|
| Alice Mutton | 978 | Medium Sales |
| Aniseed Syrup | 328 | Low Sales |
| Boston Crab Meat | 1103 | High Sales |
| Camembert Pierrot | 1577 | High Sales |
| Carnarvon Tigers | 539 | Medium Sales |
| Chai | 828 | Medium Sales |

```
SELECT  p.product_name,
   product_sales.total_quantity,
   CASE
      WHEN product_sales.total_quantity > 1000 THEN 'High Sales'
      WHEN product_sales.total_quantity > 500 THEN 'Medium
Sales'
      ELSE 'Lower Sales'
   END AS sales_category
FROM products p
JOIN ( SELECT   product_id,
      SUM(quantity) AS total_quantity
   FROM  order_details
   GROUP BY  product_id  ) AS product_sales
ON
   p.product_id = product_sales.product_id;
```

- The CASE statement evaluates the total_quantity and assigns the appropriate sales_category
- If the total_quantity is greater than 1000, it is classified as High Sales
- If the total_quantity is greater than 500, but not greater than 1000, it is classified as Medium Sales
- If the total_quantity does not meet the above conditions, it is classified as 'Lower Sales

38) Classify employees based on the number of orders they have processed such that NumberOfOrders > 100 Then PerformanceCategory as ‘High Performing’. If NumberOfOrders>50 Then ‘Medium Performing’ and else ‘Lower Performing’

| first_name | last_name | NumberOfOrders | PerformanceCategory |
|---|---|---|---|
| Andrew | Fuller | 96 | Medium Performing |
| Anne | Dodsworth | 43 | Low Performing |
| Janet | Leverling | 127 | High Performing |
| Laura | Callahan | 104 | High Performing |
| Margaret | Peacock | 156 | High Performing |
| Michael | Suyama | 67 | Medium Performing |

```
SELECT
   e.first_name,
   e.last_name,
   order_counts.NumberOfOrders,
   CASE
      WHEN order_counts.NumberOfOrders > 100 THEN 'High
Performing'
      WHEN order_counts.NumberOfOrders > 50 THEN 'Medium
Performing'
      ELSE 'Lower Performing'
   END AS PerformanceCategory
FROM ( SELECT employee_id,
      COUNT(*) AS NumberOfOrders
   FROM orders
   GROUP BY employee_id
) AS order_counts
JOIN employees e ON e.employee_id = order_counts.employee_id;
```

- The CASE statement evaluates the NumberOfOrders and assigns the appropriate PerformanceCategory
- If the NumberOfOrders is greater than 100, it is classified as High Performing
- If the NumberOfOrders is greater than 50, but not greater than 100, it is classified as Medium Performing
- If the NumberOfOrders does not meet the above conditions, it is classified as Lower Performing