OLAP Queries

1.

```
SELECT
    category.category_name AS Category,
    YEAR(`order`.delivery_date) AS Year,
    MONTH(`order`.delivery_date) AS Month,
    SUM(my_orders.cost) AS Revenue
FROM
    category
    JOIN product_category_map ON category.categoryID =
product_category_map.categoryID
    JOIN product ON product_category_map.productID = product.productID
    JOIN my_orders ON product.productID = my_orders.productID
    JOIN `order` ON my_orders.orderID = `order`.orderID
GROUP BY
    category.category_name,
    YEAR(`order`.delivery_date),
    MONTH(`order`.delivery_date)
ORDER BY
    category.category_name ASC,
    YEAR(`order`.delivery_date) ASC,
    MONTH(`order`.delivery_date) ASC;
```

The category, product_category_map, product, my_orders, and order tables are joined in this query to extract revenue information by year and month for each category. The cost of all orders within each category, year, and month is totaled using the SUM function. The results are organized by category, year, and month, and then in ascending order by category, year, and month.

2.

```
SELECT
    category.category_name,
    SUM(product.price) AS total_sales,
    COUNT(DISTINCT `order`.userId) AS unique_customers
FROM
    product
    JOIN product_category_map ON product.productID =
product_category_map.productID
    JOIN category ON product_category_map.categoryID = category.categoryID
    JOIN my_orders ON product.productID = my_orders.productID
    JOIN `order` ON my_orders.orderID = `order`.orderID
GROUP BY
    category.category_name
ORDER BY
    total_sales DESC;
```

The total revenue and unique customers for each category are determined by this query. To obtain the required data, it combines the product, product_category_map, category, my_orders, and order tables. Each category's total sales are determined using the SUM function, and the number of distinct customers is determined using the COUNT DISTINCT function. The results are categorized and listed in descending order by total sales.

3.

```
SELECT
    YEAR(`order`.delivery_date) AS year,
    MONTH(`order`.delivery_date) AS month,
    COUNT(DISTINCT `order`.userId) AS unique_customers,
    SUM(`order`.order_value) AS total_sales
FROM
    `order`
WHERE
    YEAR(`order`.delivery_date) = 2022
GROUP BY
    YEAR(`order`.delivery_date),
    MONTH(`order`.delivery_date)
ORDER BY
    year ASC,
    month ASC;
```

This query calculates the total sales and unique customers for each month and year 2022.

4.

```
SELECT
    COALESCE(category.category_name, 'Total') AS category,
    SUM(product.quantity) AS total_quantity,
    SUM(product.price * product.quantity) AS total_sales
FROM
    product
    JOIN product_category_map ON product.productID =
product_category_map.productID
    JOIN category ON category.categoryID = product_category_map.categoryID
GROUP BY
    category.category_name WITH ROLLUP
ORDER BY
    total_sales DESC;
```

This search determines the total amount and total sales for each group of products, then sorts them according to total sales. Using the GROUP BY clause, it connects the product, product_category_map, and category tables and groups the outcomes by the category name. Based on the number and price of the products, the SUM function is used to determine the total quantity and total sales for each category. The ORDER BY clause is then used to order the results by total sales in descending order.

Triggers

1.

```
CREATE TRIGGER update_cart_total BEFORE UPDATE ON cart
FOR EACH ROW
BEGIN
IF NEW.userID IS NOT NULL AND NEW.productID IS NOT NULL THEN
SET NEW.total_cost = (
SELECT product.price * NEW.quantity
FROM product
WHERE product.productID = NEW.productID
);
END IF;
END $$
```

Update cart value after every update in cart

2.

```
DELIMITER $$
CREATE TRIGGER `useCoupon` BEFORE INSERT ON `order`
FOR EACH ROW BEGIN
    UPDATE coupons
    SET is_used = 1
    WHERE coupons.couponID = NEW.couponID;
END $$
DELIMITER ;
```

When a coupon has been used, a trigger is raised, and if it has, we set the is_used value for that row entry in the coupons table to 1.