```sql
-- Create Database
CREATE DATABASE SQL_Business;
USE SQL_Business;

-- View all data from tables
SELECT * FROM fact;
SELECT * FROM location;
SELECT * FROM product;

-- 1. Display the number of states present in the Location table
SELECT COUNT(DISTINCT state) AS number_of_states FROM location;
-- This counts the distinct states from the location table.

-- 2. How many products are of regular type?
SELECT COUNT(*) AS count_of_regular_type FROM product
WHERE type = 'regular';
-- This counts the number of products where type is 'regular'.

-- 3. How much spending has been done on marketing of product ID 1?
SELECT SUM(marketing) AS sum_of_marketing FROM fact
WHERE productid = 1;
-- This sums up the marketing expenses for product ID 1.

-- 4. What is the minimum sales of a product?
-- Using Joins
SELECT p.Product, MIN(f.sales) AS minimum_sales FROM fact AS f
JOIN product AS p
ON f.productid = p.productid
GROUP BY p.Product
ORDER BY minimum_sales LIMIT 1;
-- This joins the fact and product tables to find the product with the lowest sales.

-- Using Nested Sub-query
SELECT ProductId, `Product Type`, Product
FROM product
WHERE ProductId IN (
SELECT ProductId
FROM fact
WHERE sales = (SELECT MIN(sales) FROM fact)
);
-- The inner query finds the minimum sales, and the outer query retrieves the product
details.

-- 5. Display the max Cost of Goods Sold (COGS)
-- Using Joins
SELECT p.Product, p.`Product Type`, MAX(cogs) AS max_cogs FROM fact AS f
JOIN product AS p
ON f.productid = p.productid
GROUP BY p.Product, p.`Product Type`
ORDER BY max_cogs DESC LIMIT 1;
-- This joins tables to find the product with the highest COGS.

-- Using Nested Sub-query
SELECT Productid, Product, (SELECT MAX(cogs) FROM fact) AS maximum_sale
```

```
FROM product
WHERE ProductId IN (
SELECT ProductId
FROM fact
WHERE cogs = (SELECT MAX(cogs) FROM fact)
);
-- The subquery finds the highest COGS value, and the outer query fetches product details.

-- 6. Display the details of the product where product type is coffee.
SELECT * FROM product
WHERE `Product Type` = 'coffee';
-- This retrieves all products where the product type is 'coffee'.

-- 7. Display the details where total expenses are greater than 40.
SELECT * FROM fact
WHERE `Total Expenses` > 40;
-- This filters records where total expenses exceed 40.

-- 8. What is the average sales in area code 719?
SELECT AVG(sales) AS avg_sales FROM fact
WHERE `Area Code` = 719;
-- This calculates the average sales for Area Code 719.

-- 9. Find out the total profit generated by Colorado state.
SELECT SUM(profit) AS profit_sum FROM fact
WHERE `Area Code` IN (SELECT `Area Code` FROM location
WHERE state = 'Colorado');
-- This finds all area codes in Colorado and sums the profits for those areas.

-- 10. Display the average inventory for each product ID.
SELECT productid, AVG(inventory) AS avg_inventory FROM fact
GROUP BY productid
ORDER BY productid ASC;
-- Groups data by product ID and calculates the average inventory.

-- 11. Display the average budget of the Product where the average budget margin should be greater than 100.
SELECT productid, AVG(`Budget Margin`) AS avg_budget FROM fact
GROUP BY productid
HAVING AVG(`Budget Margin`) > 100;
-- Filters the results to only include products with an average budget margin above 100.

-- 12. Display the average total expense of each product ID on an individual date.
SELECT productid, date, AVG(`Total Expenses`) AS avg_total_expenses FROM fact
GROUP BY productid, date
ORDER BY productid, date;
-- Groups data by product ID and date, then calculates the average total expenses.

-- 13. Display the rank without any gap to show the sales-wise rank.
SELECT Sales, DENSE_RANK() OVER (ORDER BY Sales DESC) AS ranks FROM fact;
-- Uses DENSE_RANK() to rank sales values without gaps.

-- 14. Find the state-wise profit and sales along with the product name.
```

```sql
SELECT l.State, p.Product, SUM(f.Profit) AS sum_profit, SUM(f.Sales) AS sum_sales
FROM fact AS f
JOIN location AS l ON f.`Area Code` = l.`Area Code`
JOIN product AS p ON f.productid = p.productid
GROUP BY l.State, p.Product;
-- Groups data by state and product, then calculates total profit and sales.

-- 15. If there is an increase in sales of 5%, calculate the increased sales.
SELECT sales, sales * 0.05 AS increased_sales, sales + (sales * 0.05) AS increment FROM
fact;
-- Calculates a 5% increase in sales and the new total after the increase.

-- 16. Find the maximum profit along with the product ID and product type.
SELECT p.productid, p.`Product Type`, MAX(f.profit) AS max_profit
FROM fact f
JOIN product p ON p.productid = f.productid
GROUP BY p.productid, p.`Product Type`;
-- Finds the highest profit for each product ID and product type.

-- 17. Create a stored procedure to fetch the result according to the product type.
DELIMITER //
CREATE PROCEDURE ptype(IN prod_type VARCHAR(20))
BEGIN
SELECT * FROM product WHERE `Product Type` = prod_type;
END //
DELIMITER ;
-- This stored procedure retrieves products based on the given product type.

-- 18. Create a condition where total expenses < 60 = profit, else loss.
SELECT `Total Expenses`,
CASE
WHEN `Total Expenses` < 60 THEN 'PROFIT'
ELSE 'LOSS' END AS RESULT
FROM fact;
-- Uses CASE WHEN to label rows as 'PROFIT' or 'LOSS' based on expenses.

-- 19. Give the total weekly sales value with the date and product ID details using
ROLLUP.
SELECT WEEK(date) AS WEEK, date, productid, SUM(sales) AS TOTAL_SALES
FROM fact
GROUP BY WEEK, date, productid WITH ROLLUP;
-- Uses ROLLUP to calculate total weekly sales with a breakdown by date and product.

-- 20. Apply UNION and INTERSECT operators on the tables with the attribute area_code.
-- Using UNION
SELECT `Area Code` FROM fact
UNION
SELECT `Area Code` FROM location;
-- UNION returns unique area codes from both tables.

-- Using INTERSECT
SELECT `Area Code` FROM fact
WHERE `Area Code` IN (SELECT `Area Code` FROM location);
-- INTERSECT-like query that finds common area codes between the two tables.
-- 21. Change the product type from coffee to tea where product ID is 1 and undo it.
SET SQL_SAFE_UPDATES = 0;
START TRANSACTION;
UPDATE product SET `Product Type` = 'tea'
WHERE ProductId = 1;
ROLLBACK;
-- Starts a transaction, updates the product type, then rolls back the change.

-- 22. Delete the records in the Product Table for regular type.
DELETE FROM product
WHERE type = 'regular';
-- Deletes all products where the type is 'regular'.

-- 23. Display the ASCII value of the fifth character from the column Product.
SELECT ASCII(SUBSTRING(product, 5, 1)) AS 5TH_CHAR_ASCII FROM product;
-- Retrieves the ASCII value of the fifth character in the product name.

-- 24. Find the top 3 states with the highest total sales.
SELECT l.State, SUM(f.Sales) AS TotalSales
FROM fact f
JOIN Location l ON f.`Area Code` = l.`Area Code`
GROUP BY l.State
ORDER BY TotalSales DESC LIMIT 3;
-- Groups sales by state and returns the top 3 states with the highest sales.

-- 25. Find the month-over-month sales growth for each product type.
WITH MonthlySales AS (
SELECT
DATE_FORMAT(STR_TO_DATE(f.Date, '%m/%d/%Y'), '%Y-%m') AS Month,
p.`Product Type`, SUM(f.Sales) AS TotalSales
FROM fact f
JOIN Product p ON f.ProductId = p.ProductId
GROUP BY Month, p.`Product Type`
)
SELECT Month, `Product Type`, TotalSales,
LAG(TotalSales) OVER (PARTITION BY `Product Type` ORDER BY Month) AS PreviousMonthSales,
ROUND((TotalSales - LAG(TotalSales) OVER (PARTITION BY `Product Type` ORDER BY Month)) /
NULLIF(LAG(TotalSales) OVER (PARTITION BY `Product Type` ORDER BY Month), 0) * 100, 2
) AS GrowthPercentage
FROM MonthlySales;
-- This query calculates month-over-month sales growth for each product type.
-- Uses the `LAG()` function to get previous months sales for comparison.
-- NULLIF prevents division by zero errors when computing the growth percentage.

-- 26. Identify the most profitable product type in each market region.
WITH ProfitByMarket AS (
SELECT
l.Market, p.`Product Type`, SUM(f.Profit) AS TotalProfit,
RANK() OVER (PARTITION BY l.Market ORDER BY SUM(f.Profit) DESC) AS Ranks
FROM fact f
JOIN Location l ON f.`Area Code` = l.`Area Code`
JOIN Product p ON f.ProductId = p.ProductId
GROUP BY l.Market, p.`Product Type`
)
SELECT Market, `Product Type`, TotalProfit
FROM ProfitByMarket
WHERE Ranks = 1;
-- This query ranks product types based on total profit within each market.
-- Uses `RANK()` to find the most profitable product type in each market.

-- 27. Compare actual sales vs. budgeted sales by product type.
SELECT
p.`Product Type`,
SUM(f.Sales) AS ActualSales,
SUM(f.`Budget Sales`) AS BudgetedSales,
SUM(f.Sales) - SUM(f.`Budget Sales`) AS SalesDifference
FROM fact f
JOIN Product p ON f.ProductId = p.ProductId
GROUP BY p.`Product Type`
ORDER BY SalesDifference DESC;
-- Compares actual sales with budgeted sales for each product type.
-- Calculates the difference to determine under-performance or over-performance.

-- 28. Find the most over-budget and under-budget states based on sales.
WITH StateSales AS (
SELECT
l.State,
SUM(f.Sales) AS ActualSales,
SUM(f.`Budget Sales`) AS BudgetedSales,
SUM(f.Sales) - SUM(f.`Budget Sales`) AS SalesDifference
FROM fact f
JOIN Location l ON f.`Area Code` = l.`Area Code`
GROUP BY l.State
)
SELECT State, ActualSales, BudgetedSales, SalesDifference
FROM StateSales
ORDER BY SalesDifference DESC;
-- Finds states where actual sales exceed or fall below budgeted sales.
-- Orders results to show the most over-budget and under-budget states.

-- 29. Find the cumulative sales for each product type over time.
SELECT
STR_TO_DATE(f.Date, '%m/%d/%Y') AS DateFormatted,
p.`Product Type`,
SUM(f.Sales) OVER (PARTITION BY p.`Product Type` ORDER BY f.Date ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS CumulativeSales
FROM fact f
JOIN Product p ON f.ProductId = p.ProductId;
-- Computes cumulative sales for each product type over time.
-- Uses `SUM() OVER (PARTITION BY)` to progressively sum sales data.

-- 30. Find the top-performing decaf products by profit margin.
SELECT p.Product, p.`Product Type`, AVG(f.Margin) AS AvgMargin
FROM fact f
JOIN Product p ON f.ProductId = p.ProductId
WHERE p.Product LIKE ('Decaf%')
GROUP BY p.Product, p.`Product Type`
```

```
ORDER BY AvgMargin DESC
LIMIT 5;
-- Identifies the top 5 decaf products based on average profit margin.
-- Filters product names starting with "Decaf" and sorts results in descending order.
```