ADVANCED SQL

QUESTION 1: What is a Common Table Expression (CTE), and how does it improve SQL query readability?

ANSWER: A Common Table Expression (CTE) is a temporary named result set defined using WITH that exists only for the duration of a query.

It improves readability by breaking complex queries into smaller, logical parts and avoids repeating subqueries.

QUESTION 2: Why are some views updateable while others are read-only? Explain with an example.

ANSWER: A view is updateable if it is based on a single table and does not use aggregates, joins, GROUP BY, DISTINCT, or subqueries. Views using joins or aggregate functions become read-only because the database cannot map changes back to a single row.

Example:

- Simple view on one table → updatable
- View with GROUP BY → read-only

QUESTION 3: What advantages do stored procedures offer compared to writing row SQL queries repeatedly?

ANSWER: Advantages of stored procedures

- Better performance (precompiled)
- Code reuse
- Improved security
- Easier maintenance
- Reduces network traffic

QUESTION 4: What is the purpose of triggers in a database? Mention one use case where a trigger is essential.

ANSWER: Triggers automatically execute when an event (INSERT, UPDATE, DELETE) occurs.

Use case:

Auditing deleted records by storing them in an archive table.

QUESTION 5: Explain the need for data modelling and normalization when designing a database.

ANSWER: Need for data modelling and normalization

- Reduces data redundancy
- Ensure data consistency
- Improves data integrity
- Makes database scalable and maintainable

QUESTION 6: Write a CTE to calculate the total revenue for each product

(Revenue = Price × Quantity), and return only products where revenue > 3000.

ANSWER: WITH RevenueCTE AS (

SELECT

p.ProductID,

p.ProductName,

p.Price * s.Quantity AS Revenue

FROM Products p

JOIN Sales s ON p.ProductID = s.ProductID)

SELECT ProductID, ProductName, Revenue

FROM RevenueCTE

WHERE Revenue > 3000;

QUESTION 7: Create a view named vw_CategorySummary that shows:

Category, TotalProducts, AveragePrice.

ANSWER: CREATE VIEW vw_CategorySummary AS

SELECT

Category,

COUNT(*) AS TotalProducts,

AVG(Price) AS AveragePrice

FROM Products

GROUP BY Category;

QUESTION 8: Create an updateable view containing ProductID, ProductName, and Price.

Then update the price of ProductID = 1 using the view.

ANSWER: CREATE VIEW vw_ProductDetails AS

SELECT ProductID, ProductName, Price

FROM Product;

UPDATE vw_ProductDetails

SET Price = 1500

WHERE ProductID = 1

QUESTION 9: Create a stored procedure that accepts a category name and returns all product belonging to that category.

ANSWER: DELIMITED //

CREATE PROCEDURE GetProductsByCategory(IN cat_name VARCHAR(50))

BEGIN

SELECT*

FROM Products

WHERE Category = cat_name;

END //

DELIMITER;

QUESTION 10: Create an AFTERDELETE trigger on the Products table that archives deleted product rows into a new table ProductArchive. The archive should store ProductID, ProductName, Category, Price, and DeletedAt timestamp.

ANSWER: CREATE TABLE ProductArchive (

ProductID INT,

ProductName VARCHAR(100),

Category VARCHAR(50),

Price DECIMAL(10,2),

DeletedAt TIMESTAMP);

DELIMITER //

CREATE TRIGGER after_product_delete

AFTER DELETE ON Products

FOR EACH ROW
BEGIN
INSERT INTO ProductArchive

VALUES (

OLD.ProductID,

OLD.ProductName,

OLD.Category,

OLD.Price,

NOW());

END //

DELIMITER;