**DATE : 11.06.2025**

**Section 1: Managing Databases**

1. **Which of the following is NOT a system database in SQL Server?**  
   a) master  
   b) model  
   c) tempdb  
   **d) userdb**
2. **Which system database stores all login accounts and configuration settings?**  
   a) tempdb  
   b) model  
   **c) master**  
   d) msdb
3. **What is the purpose of the model database in SQL Server?**  
   a) Backup  
   b) Log storage  
   **c) Template for new databases**  
   d) System configuration
4. **What are the two main types of database files in SQL Server?**  
   a) MDF and NDF  
   **b) LDF and MDF**c) NDF and BAK  
   d) BAK and TRN
5. **Which SQL command is used to create a new database?**  
   a) MAKE DATABASE  
   b) NEW DATABASE  
   c**) CREATE DATABASE**  
   d) INIT DATABASE
6. **What happens when you execute DROP DATABASE SalesDB?**  
   a) SalesDB is backed up  
   b) SalesDB is renamed  
   **c) SalesDB is deleted permanently**  
   d) SalesDB is restored
7. **Which command renames a database in SQL Server?**  
   a) RENAME DATABASE old\_name TO new\_name  
   **b) ALTER DATABASE old\_name MODIFY NAME = new\_name**  
   c) UPDATE DATABASE NAME  
   d) SET DATABASE NAME

**Section 2: Managing Tables**

1. **Which data type should be used to store a date of birth?**  
   a) VARCHAR  
   **b) DATE**  
   c) INT  
   d) TEXT
2. **What command is used to create a table?**  
   a) MAKE TABLE  
   b) INSERT TABLE  
   **c) CREATE TABLE**  
   d) DEFINE TABLE
3. **How do you add a new column to an existing table?**  
   **a) ALTER TABLE table\_name ADD column\_name datatype**  
   b) MODIFY TABLE table\_name ADD column\_name  
   c) UPDATE TABLE table\_name ADD column\_name  
   d) APPEND column\_name TO table\_name
4. **Which command is used to rename a table?**  
   a) RENAME TABLE old\_name TO new\_name  
   b) ALTER TABLE old\_name RENAME TO new\_name  
   **c) EXEC sp\_rename 'old\_name', 'new\_name'**  
   d) MODIFY TABLE RENAME
5. **What is the command to delete a table permanently?**  
   a) DELETE TABLE table\_name  
   b) ERASE TABLE table\_name  
   **c) DROP TABLE table\_name**  
   d) REMOVE TABLE table\_name

**Section 3: DML - Manipulating Data**

1. **Which command adds data into a table?**  
   **a) INSERT INTO**  
   b) ADD ROW  
   c) CREATE DATA  
   d) APPEND TO
2. **Which clause is used to update data in a table?**  
   a) MODIFY  
   **b) UPDATE**  
   c) CHANGE  
   d) SET TABLE
3. **What does the DELETE statement do?**  
   a) Removes a column  
   b) Removes all data from a table  
   **c) Removes specific rows**  
   d) Deletes the table schema
4. **Which clause is used to filter rows in a SELECT statement?**  
   a) HAVING  
   b) SELECT  
   **c) WHERE**  
   d) ORDER BY
5. **Which keyword ensures no duplicate records are returned?**  
   a) UNIQUE  
   b) NO\_REPEAT  
   **c) DISTINCT**  
   d) ONLY
6. **What does the LIKE keyword do in SQL?**  
   a) Finds exact matches  
   **b) Finds pattern-based matches**  
   c) Sorts records  
   d) Deletes matches
7. **Which operator is used to combine multiple conditions in a WHERE clause?**  
   a) TO  
   b) WITH  
   **c) AND / OR**d) IF / ELSE
8. **What does the BETWEEN operator do?**  
   a) Compares text fields  
   b) Finds rows outside a range  
   **c) Filters values within a range**  
   d) Joins tables

**DATE : 12.06.2025**

**Sql server practical assignment**

**Section a: managing databases**

1. list all system databases in sql server:  
select name from sys.databases where database\_id < 5;

2. list physical file paths for all databases:  
select name, physical\_name from sys.master\_files;

3. create a new user-defined database named teamdb:  
create database teamdb;

4. rename the database teamdb to projectdb:  
alter database teamdb modify name = projectdb;

5. drop the projectdb database:  
drop database projectdb;

**Section b: managing tables**

1. create a table employees:  
create table employees (  
 empid int primary key,  
 name varchar(50),  
 department varchar(30),  
 joiningdate date,  
 isactive bit,  
 salary decimal(10,2)  
);

2. add a column salary (decimal) to the table:  
alter table employees add salary decimal(10,2);

3. rename table employees to teammembers:  
exec sp\_rename 'employees', 'teammembers';

4. drop the table teammembers:  
drop table teammembers;

**Section c: dml operations**

1. insert three rows into employees:  
insert into employees values  
 (1, 'amit', 'hr', '2022-01-01', 1, 50000),  
 (2, 'sneha', 'it', '2021-06-15', 1, 75000),  
 (3, 'john', 'finance', '2020-10-10', 0, 65000);

2. update salary of 'sneha' to 80000:  
update employees set salary = 80000 where name = 'sneha';

3. delete employee with isactive = 0:  
delete from employees where isactive = 0;

4. retrieve names and departments of all employees:  
select name, department from employees;

5. fetch employees from 'it' department with salary above 70000:  
select \* from employees where department = 'it' and salary > 70000;

6. apply filtering using like, between, and in:  
select \* from employees where name like 's%';  
select \* from employees where salary between 60000 and 80000;  
select \* from employees where department in ('it', 'finance');

# SQL Challenges and Solutions

## **1. Insert and Update with Integrity**

Create the 'students' table with constraints:

create table students (  
 student\_id int primary key,  
 name varchar(50) not null,  
 email varchar(100) unique not null,  
 marks int check (marks >= 0 and marks <= 100)  
);

Insert 5 records:

Insert into students values  
(1, 'john', 'john@example.com', 85),  
(2, 'alice', 'alice@example.com', 92),  
(3, 'rose', 'rose@example.com', 76),  
(4, 'priya', 'priya@example.com', 65),  
(5, 'ram', 'ram@example.com', 88);

Update a student's marks:

Update students  
set marks = 90  
where student\_id = 4;

## **2. String Function Challenge**

Split full name and display name lengths:

Select   
 full\_name,  
 substring\_index(full\_name, ' ', 1) as first\_name,  
 substring\_index(full\_name, ' ', -1) as last\_name,  
 length(substring\_index(full\_name, ' ', 1)) as firstname\_length,  
 length(substring\_index(full\_name, ' ', -1)) as lastname\_length  
from customers;

## **3. Date Function Usage**

Extract month, year and days ago from sale\_date:

Select   
 sale\_date,  
 monthname(sale\_date) as month\_name,  
 year(sale\_date) as sale\_year,  
 datediff(curdate(), sale\_date) as days\_ago  
from sales;

## **4. Mathematical Functions on Salary**

Calculate salary hike and round salary:

Select   
 salary,  
 salary \* 1.10 as salary\_after\_hike,  
 round(salary, -2) as rounded\_salary  
from employees;

## **5. System Function Check**

Retrieve system information:

Select   
 now() as current\_datetime,  
 database() as current\_database,  
 user() as login\_user;

## **6. Demo: Custom Result Set**

Display product name in uppercase and handle NULL prices:

Select   
 upper(product\_name) as product\_name\_upper,  
from products;

Update products

Set price = 'not available'

Where price is null;

# **SQL Joins and Aggregates Practice**

## **7. Aggregate Functions Practice**

From a 'transactions' table:

Select   
 sum(amount) as total\_sales,  
 avg(amount) as average\_sale,  
 max(amount) as max\_sale,  
 min(amount) as min\_sale  
from transactions;

## **8. Grouping with Aggregation**

**Group by product category:**

Select   
 category,  
 sum(sale\_amount) as total\_sales,  
 count(\*) as transaction\_count  
from sales  
group by category;

## **9. Inner Join for Orders and Customers**

**Join 'orders' and 'customers' to show only customers with orders:**

Select   
 c.name as customer\_name,  
 o.amount as order\_amount  
from orders o  
inner join customers c on o.customer\_id = c.id;

## **10. Left Join for Products with or without Orders**

Show all products and their order details (if available):

Select   
 p.product\_name,  
 o.order\_id,  
 o.amount  
from products p  
left join orders o on p.product\_id = o.product\_id;

## **11. Right Join for Customer Contacts**

Show all customers, even if they don't have contact info:

Select   
 c.customer\_id,  
 c.name,  
 ct.phone\_number  
from contacts ct  
right join customers c on ct.customer\_id = c.customer\_id;

## **12. Full Outer Join for Suppliers and Products**

List all suppliers and products with matching or NULL values:

Select   
 s.supplier\_name,  
 p.product\_name  
from suppliers s  
full outer join products p on s.supplier\_id = p.supplier\_id;

**13. Cross Join for Offers**

Show all possible combinations of products and offers.

Select

p.product\_name,

o.offer\_name

From products p

Cross join offers o;

**14. Join with Aggregation**

Join orders and products, group by product category, and show total quantity sold and average price.

Select

p.category,

sum(o.quantity) as total\_quantity\_sold,

avg(p.price) as average\_price

From orders o

Join products p on o.product\_id = p.product\_id

Group by p.category;

**15. Join with Grouping and Filter**

Join students and marks, display student name and average marks, and filter to show only students with average marks > 75

Select

s.name as student\_name,

avg(m.marks) as average\_marks

From students s

Join marks m on s.student\_id = m.student\_id

Group by s.name

Having avg(m.marks) > 75;

**DATE : 13.06.2025**

SQL QUESTION PAPER - 1

**Section A: Basics & Data Definition (10 Marks)**

Q1. (3 marks) Differentiate between SQL and NoSQL. Provide two advantages and two disadvantages of each with real-world examples.

1. SQL (Structured Query Language):

- Relational database management system (RDBMS).

- Uses structured schema with tables, rows, and columns.

- Example: MySQL, PostgreSQL, Oracle.

2. NoSQL (Not Only SQL):

- Non-relational or distributed database system.

- Schema-less or dynamic schema; handles unstructured data.

- Example: MongoDB, Cassandra, Redis.

Advantages of SQL:

1. Strong ACID compliance (data integrity).

2. Powerful for complex queries using JOINs.

Disadvantages of SQL:

1. Not suitable for hierarchical or unstructured data.

2. Scaling is vertical and costly.

Advantages of NoSQL:

1. Easily handles large volumes of unstructured data.

2. Horizontal scalability (distributed systems).

Disadvantages of NoSQL:

1. Weaker consistency (eventual consistency in some models).

2. Limited support for complex queries or transactions.

Real-World Examples:

- SQL: Banking systems

-Mongodb – Social Media

Q2. Given the below unnormalized data, convert it to 1NF, 2NF, and 3NF: Student (StudentID, Name, CourseID, CourseName, InstructorName, InstructorPhone)

Q3.

a) Create a database named StudentDB.

Create database StudentDB;

b) Create a table Students with fields: StudentID, Name, DOB, Email.

Create table Students(

StudentID int primary key,

Name varchar(100),

DOB date,

Email varchar(250) );

c) Rename the table to Student\_Info.

Alter table Student RENAME to Student\_Info;

d) Add a column PhoneNumber.

Alter table Student\_Info

Add Column PhoneNumber varchar(15);

e) Drop the table.

Drop table Student\_Info;

**Section B: DML & Filtering Data (15 Marks)**

Q4. (5 marks)

a) Insert 3 student records into Student\_Info.

Insert into Student\_Info values (1, ‘Alice’, ‘1999-05-08’, ‘alice@.com’,’9745643578’), (2, ‘Rose’, ‘2001-01-17’, ‘rose@yahoo.com’,’6379537547’), (3, ‘Charlie’, ‘2005-08-23’,’charlie@outlook.com’,’9237588669’);

b) Update one student's phone number.

Update Student\_Info

Set phoneNumber = ‘9873937453’

Where StudentID = 1;

c) Delete one student whose email ends with @gmail.com.

Delete from Student\_Info

Where Email Like ‘%@gmail.com’

Limit 1;

d) Retrieve only names and emails of students born after the year 2000.

Select Name,Email from Student\_Info

Where Year(DOB)>2000;

e) Retrieve distinct domain names from the email column.

Select distinct substring(email, charindex('@', email) + 1, len(email)) as domain

From student\_info;

Q5.

a) Retrieve students with names starting with 'A'.

Select Name from Student\_Info

Where Name Like ‘A%’;

b) Retrieve students with phone number between 9000000000 and 9999999999.

Select \* from Student\_Info

Where PhoneNumber Between 9000000000 AND 9999999999;

c) Retrieve students using IN operator on city names.

Select \* from Student\_Info

Where City IN (‘Chennai’,’Coimbatore’);

d) Use AND, OR to filter students based on age and email provider.

AND:

Select \* from Student\_Info

Where age>18 AND Email Like ‘%@gmail.com’;

OR:

Select \* from Student\_Info

Where age>18 OR Email Like ‘%@gmail.com’;

e) Use table and column aliasing in a query to get all student names and DOBs.

Select Stu.Name as Student\_Name, Stu.DOB as DateOfBirth

From Student\_Info as Stu;

Q6. Create a new table Marks(StudentID, Subject, Marks). Insert at least 3 rows.

Create table Marks(

StudentID int,

Subject varchar(50),

marks decimal(5,2),

Foreign key (StudentID) references Student\_Info(StudentID));

a) Display student IDs and their subjects where marks > 70.

Select StudentID, Subject from Marks

Where marks>70;

b) Display subjects with average marks.

Select Subject, AVG(marks) as Average\_Marks from Marks

Group By subject;

c) Filter subjects with average marks between 60 and 90.

Select Subject from Marks

Group By Subject

Where AVG(marks) between 60 and 90;

**Section C: Functions & Grouping (10 Marks)**

Q7.

a) Get the current date and format it as "YYYY-MM-DD".

Select CURDATE() as CurrentDate;

b) Extract month and year from a DOB column.

Select MONTH(DOB) as Month,

YEAR(DOB) as Year

From Student\_Info;

c) Convert a student's name to uppercase.

Select UPPER(Name) as UpperCaseName

From Student\_Info;

d) Round off marks to 2 decimal places.

SELECT Subject, ROUND(Marks, 2) AS RoundedMarks

FROM Marks;

e) Use system function to return user name or current database.

Select USER() as CurrentUser;

Select DATABASE() as CurrentDatabase;

Q8. (5 marks)

a) Display total marks of each student.

Select Subject, SUM(marks) as Total\_Marks

From Marks

Group By StudentID;

b) Display subject-wise highest mark.

Select Subject, MAX(marks) as Highest\_Mark

From Marks

Group By Subject;

c) Use GROUP BY and HAVING to display subjects with average marks > 75.

Select Subject, AVG(marks) as Average\_Marks

From Marks

Group By Subject

Having AVG(marks)>75;

**Section D: Joins and Subqueries (25 Marks)**

Q9. (5 marks)

a) Inner Join to retrieve students and their courses.

Select S.StudentID, S.Name, C.CourseName

From Student\_Info As S

Inner join Courses As C ON S.StudentID = C.StudentID;

b) Left Join to get all students even if not enrolled.

Select S.StudentID, S.Name, C.CourseName

From Student\_Info As S

Left join Courses As C ON S.StudentID = C.StudentID;

c) Right Join to get all courses even if no students.

Select S.StudentID, S.Name, C.CourseName

From Student\_Info As S

Right join Courses As C ON S.StudentID = C.StudentID;

d) Full Outer Join equivalent using UNION.

Select S.StudentID, S.Name, C.CourseName

From Student\_Info AS S

Left join Courses AS C ON S.StudentID = C.StudentID

UNION

Select S.StudentID, S.Name, C.CourseName

From Student\_Info AS S

Right join Courses AS C ON S.StudentID = C.StudentID;

e) Cross Join to show all combinations.

Select S.StudentID, S.Name, C.CourseName

From Student\_Info AS S

Cross join Courses AS C;

SQL QUESTION PAPER - 2

## **Section a: advanced concepts & schema design** (10 marks)

Q1. (4 marks) Explain with examples the scenarios where NoSQL is preferred over SQL. Discuss types of NoSQL databases and suggest a real-time application for each.

Nosql is preferred when:  
- data is unstructured or semi-structured (e.g., json, xml).  
- schema flexibility is required.  
- horizontal scalability and high-speed access are essential.  
  
Types of nosql databases and examples:  
1. Document store – mongodb: used in content management systems.  
2. Key-value store – redis: used in session management.  
3. Column-family store – cassandra: used in time-series data like logs.  
4. Graph store – neo4j: used in social networks.

Q2. (6 marks) A retail store keeps the following unnormalized record: Customer (CustomerID, Name, Orders (OrderID, ProductID, Quantity, ProductName)) Normalize the data up to BCNF with appropriate table structures.

Unnormalized table: customer(customerid, name, orders(orderid, productid, quantity, productname))  
  
1nf:  
customer(customerid, name)  
orders(orderid, customerid, productid, quantity, productname)  
  
2nf:  
products(productid, productname)  
orders(orderid, customerid, productid, quantity)  
  
3nf and bcnf:  
no transitive dependencies; already in bcnf.

## Section b: complex ddl and dml (15 marks)

Q3. (5 marks)  
a) Create a database RetailDB and design a schema for Customers, Orders, and Products with primary and foreign keys.

create database retaildb;  
create table customers (customerid int primary key, name varchar(50));  
create table products (productid int primary key, name varchar(50), price decimal(8,2));  
create table orders (orderid int primary key, customerid int, productid int, quantity int,  
foreign key (customerid) references customers(customerid),  
foreign key (productid) references products(productid),  
check (quantity > 0));

b) ) Implement a check constraint on Quantity (>0) in Orders.

Handled above with check constraint.  
C) Alter the Products table to add 'Discount' column and update some values.

Alter table products add column discount decimal(5,2);  
update products set discount = 5.00 where productid = 1;

Q4. (5 marks)  
a) Insert 3 sample orders per customer.

Insert into orders values (1, 1, 1, 2), (2, 1, 2, 3), (3, 1, 3, 1);  
b) Update prices with 10% increase where quantity sold > 5.

Update products p join orders o on p.productid = o.productid  
set p.price = p.price \* 1.10  
where o.quantity > 5;  
c) Delete orders where the product has never been sold

Delete from orders where productid not in (select productid from orders);

Q5. (5 marks)  
a) Customers who ordered more than 3 different products.

Select customerid from orders group by customerid having count(distinct productid) > 3;  
b) Products not ordered by any customer.

Select \* from products where productid not in (select distinct productid from orders);  
c) Count of orders placed by each customer in the last 30 days.

Select customerid, count(\*) from orders where orderdate >= curdate() - interval 30 day group by customerid;

## Section c: advanced functions and aggregations (10 marks)

Q6. (5 marks)  
a) Use string functions to standardize and extract parts from customer email IDs.

Select lower(email), substring\_index(email, '@', -1) as domain from customers;  
b) Use date functions to compute days between order date and today.

Select datediff(curdate(), orderdate) as days\_passed from orders;  
c) Use system functions to return current user and host

Select user(), system\_user();  
d) Use nested functions to format a customer greeting string.

Select concat('hello ', upper(name), '!') as greeting from customers;

Q7. (5 marks)  
a) Aggregate total revenue by product category

Select category, sum(price \* quantity) as revenue from products p join orders o on p.productid = o.productid group by category;  
b) Use GROUP BY with ROLLUP to compute subtotal and grand total sales.

Select category, sum(price \* quantity) from products p join orders o on p.productid = o.productid group by category with rollup;  
c) Use HAVING clause to filter categories with revenue > 100000  
Select category, sum(price \* quantity) as revenue from products p join orders o on p.productid = o.productid group by category having revenue > 100000;

## Section d: complex joins, subqueries, and set ops (25 marks)

Q8. (5 marks)  
a) Self join to list customers referred by other customers.

Select a.name as customer, b.name as referred\_by from customers a join customers b on a.referredby = b.customerid;  
b) Equi join across Orders and Products

Select \* from orders o join products p on o.productid = p.productid;  
c) Join Customers and Orders to display top 3 spenders using window function.

Select customerid, name, total\_spent from (  
Select o.customerid, c.name, sum(p.price \* o.quantity) as total\_spent,  
rank() over (order by sum(p.price \* o.quantity) desc) as rnk  
from orders o join products p on o.productid = p.productid join customers c on c.customerid = o.customerid  
group by o.customerid) where rnk <= 3;  
d) LEFT OUTER JOIN with WHERE NULL to identify inactive customers.

Select c.customerid, c.name from customers c left join orders o on c.customerid = o.customerid where o.orderid is null;  
e) Cross join for all product combinations in a bundle offer.

Select \* from products p1 cross join products p2 where p1.productid < p2.productid;

Q9. (5 marks)  
a) Correlated subquery to get customers whose order amount exceeds their average.

Select \* from customers c where exists (  
select 1 from orders o where o.customerid = c.customerid  
having sum(price \* quantity) > (select avg(price \* quantity) from orders where customerid = c.customerid));  
b) Subquery using EXISTS to find customers with at least 2 different products.

Select \* from customers c where exists (  
select productid from orders o where o.customerid = c.customerid group by productid having count(distinct productid) >= 2);  
c) Use ALL to find customers who ordered more than every other customer

Select \* from customers where customerid > all (select customerid from customers where customerid <> customers.customerid);  
d) Use ANY to find products costlier than some in category 'Electronics'.

Select \* from products where price > any (select price from products where category = 'electronics');  
e) Nested subquery to list top 3 best-selling products.

Select productid, sum(quantity) as total from orders group by productid order by total desc limit 3;

Q10. (5 marks)  
a) Simulate INTERSECT using INNER JOIN on two customer segments.

Select \* from segment1 inner join segment2 on segment1.customerid = segment2.customerid;  
b) Use EXCEPT to find products in inventory not yet ordered.

Select \* from products where productid not in (select productid from orders);  
c) Simulate MERGE: If customer exists, update; else insert.

Insert into customers (customerid, name) values (1, 'ram') on duplicate key update name = 'ram';  
d) Use UNION to combine two regional customer tables.

Select \* from region1 union select \* from region2;  
e) Write a WITH CTE that ranks customers by total spend and filters top 5.

With cte as (  
select customerid, sum(price \* quantity) as total\_spent from orders o join products p on o.productid = p.productid group by customerid)  
select \* from cte order by total\_spent desc limit 5;

**PRACTICE:**

1. Querying Data by Using Subqueries  
   2) Querying Data by Using Subqueries Using the EXISTS,  
   3) Querying Data by Using Subqueries using ANY,  
   4) Querying Data by Using Subqueries using ALL Keywords  
   5) Querying Data by Using Subqueries using Using Nested Subqueries  
   6) Querying Data by Using Subqueries  Using Correlated Subqueries  
   7) Querying Data by Using Subqueries Using UNION,  
   8) Querying Data by Using Subqueries using INTERSECT,  
   9) Querying Data by Using Subqueries using EXCEPT,  
   10)Querying Data by Using Subqueries using MERGE"

-- Alumni example  
CREATE TABLE Alumni (  
    AlumniID INT PRIMARY KEY,  
    Name VARCHAR(100),  
    BatchYear INT,  
    City VARCHAR(50)  
);

CREATE TABLE Events (  
    EventID INT PRIMARY KEY,  
    EventName VARCHAR(100),  
    EventType VARCHAR(50),   
    EventDate DATE  
);

CREATE TABLE Registrations (  
    RegistrationID INT PRIMARY KEY,  
    AlumniID INT,  
    EventID INT,  
    RegistrationDate DATE,  
    FOREIGN KEY (AlumniID) REFERENCES Alumni(AlumniID),  
    FOREIGN KEY (EventID) REFERENCES Events(EventID)  
);

INSERT INTO Alumni VALUES  
(1, 'Anusha', 2010, 'New delhi'),  
(2, 'Berlin', 2012, 'Chicago'),  
(3, 'Charu', 2011, 'Houston'),  
(4, 'Diva', 2013, 'Bhuhneshwar');

INSERT INTO Events VALUES  
(101, 'Annual Get-Together', 'GetTogether', '2025-01-20'),  
(102, 'Silver Jubilee Meet', 'AlumniMeet', '2025-02-10'),  
(103, 'Tech Talk Reunion', 'AlumniMeet', '2025-03-15');

INSERT INTO Registrations VALUES  
(1001, 1, 101, '2024-12-01'),  
(1002, 2, 101, '2024-12-05'),  
(1003, 1, 102, '2025-01-10'),  
(1004, 3, 102, '2025-01-11'),  
(1005, 2, 103, '2025-02-25');

-- 1 Subquery: List alumni who registered for more than one event  
SELECT Name   
FROM Alumni   
WHERE AlumniID IN (  
    SELECT AlumniID   
    FROM Registrations   
    GROUP BY AlumniID   
    HAVING COUNT(EventID) > 1  
);  
-- 2 Subquery with EXISTS: Find alumni who registered for the 'Silver Jubilee Meet'  
SELECT Name   
FROM Alumni a  
WHERE EXISTS (  
    SELECT 1   
    FROM Registrations r   
    JOIN Events e ON r.EventID = e.EventID  
    WHERE a.AlumniID = r.AlumniID AND e.EventName = 'Silver Jubilee Meet'  
);

-- 3 Subquery using ANY: Find events attended by alumni from any batch before 2012

SELECT EventName   
FROM Events   
WHERE EventID = ANY (  
    SELECT EventID   
    FROM Registrations   
    WHERE AlumniID IN (  
        SELECT AlumniID FROM Alumni WHERE BatchYear < 2012  
    )  
);

-- 4 Subquery using ALL: Events attended by alumni from all batches (2010, 2011, 2012, 2013)  
SELECT EventName   
FROM Events   
WHERE EventID IN (  
    SELECT r.EventID   
    FROM Registrations r   
    JOIN Alumni a ON r.AlumniID = a.AlumniID   
    GROUP BY r.EventID   
    HAVING COUNT(DISTINCT a.BatchYear) = 4  
);

-- 6 Nested Subqueries: Find event with the most registrations  
SELECT EventName   
FROM Events   
WHERE EventID = (  
    SELECT EventID   
    FROM (  
        SELECT EventID, COUNT(\*) AS RegCount   
        FROM Registrations   
        GROUP BY EventID   
        ORDER BY RegCount DESC   
        LIMIT 1  
    ) AS TopEvent  
);

-- 7 Correlated Subquery: Find alumni who registered for more than the average number of events  
SELECT Name   
FROM Alumni a  
WHERE (  
    SELECT COUNT(\*)   
    FROM Registrations r   
    WHERE r.AlumniID = a.AlumniID  
) > (  
    SELECT AVG(EventCount)   
    FROM (  
        SELECT COUNT(\*) AS EventCount   
        FROM Registrations   
        GROUP BY AlumniID  
    ) AS Sub  
);

-- 8 UNION: List all alumni and all event names in a single column  
SELECT Name AS Info FROM Alumni  
UNION  
SELECT EventName FROM Events;