Instructions:

1. Attempt all the questions.
2. Write down all SQL answers in the given supplementary.

Q:1 Answer the following questions:

1. **When should you use the HAVING clause instead of WHERE?**
   Having should be used **with aggregate functions** if we want to filter data. Having should be used after Group by Clause. Where cannot be used with aggregate function.
   i.e.

```
Select Year, SUM(BillAmount)
From Sales
Group By Year
Having Sum(BillAmount) > 10000
```

2. **What is the primary purpose of Database Normalization?**
   The primary purpose is to **reduce data redundancy** and **prevent data anomalies.**

3. **What is the difference between Union and Union All?**
   UNION: Combines the results of two or more queries and **removes duplicate rows**.
   UNION ALL: Combines the results of two or more queries and **includes duplicates**.

4. **Why is indexing used in a database?**
   Indexing is used to **speed up the data retrieval** operations. It provides a quick lookup path to the data rows without scanning the entire table.

5. **What is a Database View?**
   A database view is a **virtual table** based on the result set of a query. It contains rows and columns just like a real table but data is not stored physically.

6. **In normalization, _____ dependency occurs when A → B and B → C, then A → C.**
   Transitive dependency.

7. **In the ACID properties, _____ ensures that a transaction executes either 0% or 100%.**
   Atomicity.

8. **Can unique key column store NULL?**
   Yes. A unique key column allows NULL values (only one NULL per column).

9. **What are the rules for defining a primary key?**
   UNIQUE + NOT NULL. PK cannot be null and it must contain unique values.

10. **What happens if you join two tables without specifying the ON condition?**
    It results in a **Cross Join.**

11. **What happens when you try to delete a record that is referenced by a foreign key?**

There are two scenarios based on **ON DELETE CACADE**.

1. If ON DELETE CASCADE is set, then rows from all the tables will be deleted which are referred by the row of a foreign key which is being deleted.
2. If ON DELETE CASCADE is not set, then it will throw a referential integrity error and block the deletion of a row which is referred in other tables.

12. What is the difference between CHAR(10) and VARCHAR(10)?

CHAR(10): Fixed-length; if you store 3 characters, it still stores 10 characters, 3 characters data and remaining 7 characters will be space.

VARCHAR(10): Variable-length; it only uses as much space as the characters are stored.

13. What is the difference between a full join and an inner join?

Full Join: Returns **all rows from both tables**, joining them where matches exist and filling with NULL where they don't.

Inner Join: Returns only those **rows where there is a match** in both tables based on ON COLUMNS.

14. What is the difference between clustered and non-clustered index?

Clustered Index defines the **physical order** of data in the table. only one clustered index allowed per table. Non-clustered Index creates a separate structure with indexed columns and pointers to the actual rows. Multiple non-clustered indexes can be created, improving performance for searches and joins.

15. List out the types of triggers. Common types include:

i. **DML** (Data Manipulation Language) triggers are fired in response to INSERT, UPDATE, or DELETE statements on a specific table or view.

ii. **DDL** (Data Definition Language) triggers are fired in response to DDL events such as CREATE, ALTER, or DROP statements on database objects.

iii. **Logon** triggers are fired when a user session is being established (i.e., a LOGON event).

16. How NULL values affect aggregate functions?

Aggregate functions, like SUM, AVG, MAX, COUNT (column_name), ignore NULL values in the columns they are processing; except for COUNT(*) which counts all rows including those with NULL.

17. Can you use = NULL or <> NULL to check for NULL values? Why?

No, we cannot use because NULL is not a value, so we can not use = or <> with NULL. We have to use IS NULL or IS NOT NULL.

18. Which window functions assign a rank or sequence number to rows within a result set?

ROW_NUMBER(), RANK(), and DENSE_RANK().

19. Can you use GROUP BY without aggregate functions?

Yes. It can be used to group identical values

20. What is the result of 100 + NULL?

ANS: NULL.

Q.2 Zomato-like Food Delivery System (Database Design)

Customer

Column Name	Data Type	Description
CustomerId (PK)	INT	Unique Customer identifier
Name	VARCHAR(250)	Customer full name
Email	VARCHAR(250)	Unique email
Phone	VARCHAR(50)	Unique phone number
PasswordHash	VARCHAR(250)	Encrypted password
OTP	VARCHAR(10)	One Time Password for Verification
IsActive	BIT	Account active status
CreatedAt	DATETIME	Account creation time

CustomerAddresses

Column Name	Data Type	Description
AddressId (PK)	INT	Unique address id
CustomerId (FK)	INT	Reference to Customers
Label	VARCHAR(250)	Home / Office
Address	VARCHAR(250)	Address of the Customer
Latitude	DECIMAL	GPS latitude
Longitude	DECIMAL	GPS longitude
City	VARCHAR(50)	City name
Pincode	VARCHAR(6)	Postal code
IsDefault	BIT	Default address

Restaurant

Column Name	Data Type	Description
RestaurantId (PK)	INT	Restaurant identifier
OwnerId (FK)	INT	Restaurant owner
RestaurantName	VARCHAR(250)	Restaurant name
City	VARCHAR(50)	Restaurant city
IsVegOnly	BIT	Vegetarian only flag
AverageCostForTwo	DECIMAL	Cost estimate
Rating	DECIMAL	Average rating
IsOpen	BIT	Open/Close status

MenuItem

Column Name	Data Type	Description
MenuItemId (PK)	INT	Menu item id
RestaurantId (FK)	INT	Restaurant reference
ItemName	VARCHAR(250)	Food item name
Price	DECIMAL	Item price
IsVeg	BIT	Veg/Non-veg
IsAvailable	BIT	Availability status

Order

Column Name	Data Type	Description
OrderId (PK)	INT	Order identifier
CustomerId (FK)	INT	Customer
RestaurantId (FK)	INT	Restaurant
OrderStatus	ENUM	Placed / Preparing / Delivered
TotalAmount	DECIMAL	Final bill amount
PaymentStatus	ENUM	Paid / Pending
CreatedAt	DATETIME	Order time

OrderItem

Column Name	Data Type	Description
OrderItemId (PK)	INT	Unique order item identifier
OrderId (FK)	INT	Reference to Orders
MenuItemId (FK)	INT	Reference to Menu Items
Quantity	INT	Number of items ordered
UnitPrice	DECIMAL(10,2)	Price per item at order time
TotalPrice	DECIMAL(10,2)	Quantity × UnitPrice

Payment

Column Name	Data Type	Description
PaymentId (PK)	INT	Payment record
OrderId (FK)	INT	Order reference
PaymentMethod	ENUM	UPI / Card / Cash
Amount	DECIMAL	Paid amount
PaymentStatus	VARCHAR(50)	Payment result

DeliveryPartner

Column Name	Data Type	Description
PartnerId (PK)	INT	Delivery partner id
CustomerId (FK)	INT	Customer reference
VehicleType	VARCHAR(50)	Bike / Scooter
IsAvailable	BIT	Availability

OrderDelivery

Column Name	Data Type	Description
OrderDeliveryId (PK)	INT	Delivery record
OrderId (FK)	INT	Order reference
PartnerId (FK)	INT	Assigned partner
DeliveryStatus	ENUM	Assigned / Delivered

Reviews

Column Name	Data Type	Description
ReviewId (PK)	INT	Review identifier
OrderId (FK)	INT	Order reference
Rating	INT	1 to 5
Comments	VARCHAR	User feedback

Q.3 Basic SQL Solution Set

1. Calculate CGPA of all students

What we have assessed

```
SELECT s.StudentID, s.EnrollmentNo, s.Name, SUM(r.GradePoint * sub.Credits) / SUM(sub.Credits) AS CGPA
FROM Students s
INNER JOIN Results r on s.StudentID = r.StudentID
INNER JOIN Subjects sub on sub.SubjectID = r.SubjectID
GROUP BY s.StudentID, s.EnrollmentNo, s.Name;
```

What industry expects

```
WITH LastAttempt AS (
   SELECT
      r.StudentID,  r.SubjectID,  r.Gradepoint,  sub.Credits,
      ROW_NUMBER() OVER ( PARTITION BY r.StudentID, r.SubjectID ORDER BY r.ExamDate DESC) AS rn
   FROM #Results r
   INNER JOIN #Subjects sub ON r.SubjectID = sub.SubjectID

)
SELECT
   s.StudentID,  s.Branch, s.EnrollmentNo, s.Name,
   CASE  WHEN SUM(la.Credits) > 0 THEN ROUND( SUM(la.Credits * la.Gradepoint) / SUM(la.Credits),  2)
         ELSE 0
   END AS CGPA
FROM LastAttempt la
INNER JOIN  #Students s ON la.StudentID = s.StudentID
WHERE la.rn = 1
GROUP BY s.StudentID,s.Branch, s.EnrollmentNo, s.Name
ORDER BY s.Branch,CGPA DESC,s.EnrollmentNo,s.Name
```

2. List students which are currently studying in 6th semester and having CGPA more than 7.5.

```
SELECT Branch,  EnrollmentNo, Name, CurrentSemester, CGPA
FROM  Students
WHERE CurrentSemester = 6 AND CGPA > 7.5
```

3. List branch wise Top 5 students based on CGPA. Same CGPA must have same rank.

```
WITH RankedStudents
AS (
        SELECT StudentID, EnrollmentNo, Name, Branch, CurrentSemester, CGPA, DENSE_RANK() OVER
        (PARTITION BY Branch ORDER BY CGPA DESC) AS RankNo
        FROM Students
)
SELECT Branch, RankNo, EnrollmentNo,Name,CGPA
FROM RankedStudents
WHERE RankNo <= 5
ORDER BY Branch, RankNo, EnrollmentNo,Name DESC;
```

4. List students with subjects which were Absent in the exam.

```
SELECT s.EnrollmentNo, s.Name, sub.SubjectName
FROM Students s
INNER JOIN Results r on s.StudentID = r.StudentID
INNER JOIN Subjects sub on sub.SubjectID r.SubjectID
WHERE r.ResultStatus = 'Absent';
```

5. List subjects which are never taken by any student.

```
SELECT SubjectName FROM Subjects WHERE SubjectID NOT IN (SELECT SubjectID FROM Results);
```

6. List students with subjects who failed the same subject more than once.

```
SELECT s.EnrollmentNo, s.Name, sub.SubjectName, COUNT(*) AS FailCount
FROM Students s
```

```
INNER JOIN Results r on s.StudentID = r.StudentID
INNER JOIN Subjects sub on sub.SubjectID = r.SubjectID
WHERE r.Grade = 'FF'
GROUP BY s.EnrollmentNo, s.Name, sub.SubjectName HAVING COUNT(*) > 1;
```

7. List students with subject & result whose first exam was within 6 months of the enrollment.

```
SELECT s.EnrollmentNo, s.Name, sub.SubjectName, r.ExamDate
FROM Students s
INNER JOIN Results r on s.StudentID = r.StudentID
INNER JOIN Subjects sub on sub.SubjectID = r.SubjectID
WHERE r.ExamDate = ( SELECT MIN(ExamDate) FROM Results WHERE StudentID = s.StudentID )
AND r.ExamDate <= DATEADD(MONTH, 6, s.EnrollmentDate);
```

8. List all subjects along with number of students enrolled.

```
SELECT sub.SubjectName, COUNT(DISTINCT r.StudentID) AS StudentCount
FROM Subjects sub
INNER JOIN Results r on sub. SubjectID = r. SubjectID
GROUP BY sub.SubjectName;
```

9. Create Histogram. Histogram means Subject wise Grade wise Student count.

```
SELECT sub.SubjectName, r.Grade, COUNT(*) AS GradeCount
FROM Subjects sub
INNER JOIN Results r on sub. SubjectID = r. SubjectID
GROUP BY sub.SubjectName, r.Grade;
```

10. List students with subject name which are not cleared by the student.

What we have assessed

```
SELECT DISTINCT s.EnrollmentNo, s.Name, sub.SubjectName
FROM Students s
INNER JOIN Results r on s.StudentID = r.StudentID
INNER JOIN Subjects sub on sub.SubjectID = r.SubjectID
WHERE r.Grade = 'FF';
```

What industry expects

```
WITH LastAttempt AS (
   SELECT r.StudentID, r.SubjectID, r.Grade, ROW_NUMBER() OVER (PARTITION BY r.StudentID, r.SubjectID ORDER BY
         r.ExamDate DESC) AS rn
   FROM #Results r
)
SELECT stu.Branch, stu.EnrollmentNo, stu.Name, sub.Semester, sub.SubjectCode, sub.SubjectName, LA.Grade
FROM LastAttempt as LA
INNER JOIN Subjects sub ON sub.SubjectID = LA.SubjectID AND LA.rn = 1 AND LA.Grade = 'FF'
INNER JOIN Students stu ON stu.StudentID = LA.StudentID
```

11. List students which never failed in any subject in any semester.

```
SELECT EnrollmentNo, Name FROM Students
WHERE StudentID NOT IN (SELECT StudentID FROM Results WHERE Grade = 'FF');
```

12. Write a query to compare two students' result. Display following columns Enrollment No, Semester, Subject Code, Grade, Grade Comparison Status (Matched, Not-Matched, Subject Not Found). Display all the subjects of both students.

```
SELECT
        s1.EnrollmentNo AS Student1_EnrollmentNo,
        s2.EnrollmentNo AS Student2_EnrollmentNo,
        sub.Semester AS SubjectSemester,
        sub.SubjectCode,
        r1.Grade AS [Student1_Grade],
        r2.Grade AS [Student2_Grade],
        CASE    WHEN r1.ResultID IS NULL OR r2.ResultID IS NULL THEN 'Subject Not Found'
                WHEN r1.Grade  = r2.Grade  THEN 'Matched'
```

```
                ELSE 'Not-Matched'
        END AS [GradeComparisonStatus]
FROM Students s1, Students s2, Results r1, Results r2, Subjects sub
WHERE   r1.SubjectID = sub.SubjectID AND r1.StudentID = s1.StudentID
        AND r2.SubjectID = sub.SubjectID AND r2.StudentID = s2.StudentID
        AND s1.StudentID = @Student1 AND s2.StudentID = @Student2;
```

13. List students who have cleared the subjects in more than 5 attempts.

What we have assessed

```
SELECT s.EnrollmentNo, s.Name, sub.SubjectName, COUNT(*) AS Attempts
FROM Students s
INNER JOIN Results r on s.StudentID = r.StudentID
INNER JOIN Subjects sub on sub.SubjectID = r.SubjectID
WHERE r.Grade <> 'FF'
GROUP BY s.EnrollmentNo, s.Name, sub.SubjectName
HAVING COUNT(*) > 5;
```

What industry expects

```
WITH MoreThen5AttemptStudent as(
        SELECT  StudentID, SubjectID, Semester
        FROM    Results AS r
        GROUP BY StudentID, SubjectID, Semester
        HAVING COUNT(1) > 5
),
StudentLastAttempt as (
        SELECT r.StudentID, r.SubjectID, r.Grade,
            ROW_NUMBER() OVER (PARTITION BY r.StudentID, r.SubjectID ORDER BY r.ExamDate DESC) AS rn
        FROM MoreThen5AttemptStudent as mr5
        INNER JOIN Results r ON r.StudentID = mr5.StudentID AND r.SubjectID = mr5.SubjectID AND r.Semester =
        mr5.Semester
)
SELECT
        stu.Branch, stu.EnrollmentNo, stu.Name, SubjectName
FROM    StudentLastAttempt AS StudentLastAttempt
INNER JOIN Students AS stu
ON stu.StudentID = StudentLastAttempt.StudentID AND rn = 1 AND StudentLastAttempt.Grade <> 'FF'
INNER JOIN Subjects sub on sub.SubjectID = StudentLastAttempt.SubjectID
```