1 a. Differences between UNION and UNION ALL UNION: Combines the result sets of two or more SELECT statements and removes duplicate rows.

UNION ALL: Combines the result sets of two or more SELECT statements and includes all rows, including duplicates.

1 b. Can we use aggregate functions in the WHERE clause? Justify your answer. No, aggregate functions cannot be used in the WHERE clause. The WHERE clause is used to filter rows before any groupings are made and before any aggregate functions are applied. Instead, aggregate functions should be used in the HAVING clause, which filters groups after the aggregation.

1c. Difference between TRUNCATE and ROUND functions TRUNCATE: Removes the decimal part of a number without rounding, effectively cutting off the digits after the decimal point.

ROUND: Rounds a number to a specified number of decimal places.

1d. Difference between ISNULL() and IFNULL() ISNULL(): Used to replace NULL with a specified replacement value.

IFNULL(): Similar to ISNULL(), but used in Mysql to return a specified value if the expression is NULL.

1e. What is referential integrity? Referential integrity ensures that relationships between tables remain consistent. It means that a foreign key in one table must always refer to a valid primary key in another table.

2a. Explain about EXISTS operator The EXISTS operator is used to test for the existence of any record in a subquery. It returns TRUE if the subquery returns one or more records.

2b. Difference between WHERE and HAVING WHERE: Filters rows before any groupings are made.

HAVING: Filters groups after the aggregation has been performed.

2c. Count total number of 'a' appearing in the phrase ‘GreatLearning’ You can use the following ```sql query to count the occurrences of 'a':

```
SELECT LENGTH('GreatLearning') - LENGTH(REPLACE('GreatLearning', 'a', '')) AS count_of_a;
```

2d. Adding and deleting a unique constraint to a column after creating a table Add Unique Constraint:

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE (column_name);
```

Delete Unique Constraint:

```
ALTER TABLE table_name DROP CONSTRAINT constraint_name;
```

2e. Applying a correlated sub-query in the HAVING clause A correlated sub-query is a sub-query that uses values from the outer query. Here’s an example:

```
SELECT department, COUNT(employee_id)
FROM employees
GROUP BY department
HAVING COUNT(employee_id) > (SELECT AVG(emp_count) FROM (SELECT COUNT(employee_id) AS emp_count FROM employees GROUP BY
```

3a. Display average rainfall and average evaporation for each location except records with evaporation as NULL

```
SELECT Location,
       AVG(Rainfall) AS AverageRainfall,
       AVG(Evaporation) AS AverageEvaporation
FROM AustraliaWeather
WHERE Evaporation IS NOT NULL
GROUP BY Location;
```

3b. Display the maximum temperature recorded in the morning and afternoon for each location in each month

```
SELECT Location,
       DATE_FORMAT(Date, '%Y-%m') AS Month,
       MAX(TempMorning) AS MaxTempMorning,
       MAX(TempAfternoon) AS MaxTempAfternoon
FROM AustraliaWeather
GROUP BY Location, DATE_FORMAT(Date, '%Y-%m');
```

3c. Add a new column to the existing table "australiaweather" with the name Pressure9pm

```
ALTER TABLE AustraliaWeather
ADD COLUMN Pressure9pm INT DEFAULT 1001 NOT NULL;
```

3d. Compare average rainfall for the month of January 2008 and July 2008

```
SELECT 'January_2008' AS Month, AVG(Rainfall) AS AverageRainfall
FROM AustraliaWeather
WHERE DATE_FORMAT(Date, '%Y-%m') = '2008-01'
UNION ALL
SELECT 'July_2008' AS Month, AVG(Rainfall) AS AverageRainfall
FROM AustraliaWeather
WHERE DATE_FORMAT(Date, '%Y-%m') = '2008-07';
```

3e. Compare the average humidity in January 2009 and February 2009

```
SELECT 'January 2009' AS Month, AVG(HumidityMorning + HumidityNoon) AS AverageHumidity
FROM AustraliaWeather
WHERE DATE_FORMAT(Date, '%Y-%m') = '2009-01'
UNION ALL
SELECT 'February 2009' AS Month, AVG(HumidityMorning + HumidityNoon) AS AverageHumidity
FROM AustraliaWeather
WHERE DATE_FORMAT(Date, '%Y-%m') = '2009-02';
```

3f. Display the highest recorded pressure in each location in every month, with specific conditions

```
SELECT Location,
       DATE_FORMAT(Date, '%Y-%m') AS Month,
       MAX(PressureMorning) AS MaxPressureMorning
FROM AustraliaWeather
WHERE Evaporation IS NOT NULL
  AND TempMorning BETWEEN 14 AND 30
  AND HumidityMorning <= 70
GROUP BY Location, DATE_FORMAT(Date, '%Y-%m');
```

4a. Identify circuits where no races have been held so far

```
SELECT CircuitName
FROM Circuits
WHERE CircuitID NOT IN (SELECT DISTINCT CircuitID FROM Races);
```

4b. Get details about the driver who holds the record for completing a lap in the shortest time duration

```
SELECT Drivers.DriverID, Drivers.DriverName, LapTimes.LapTime
FROM LapTimes
JOIN Drivers ON LapTimes.DriverID = Drivers.DriverID
ORDER BY LapTimes.LapTime ASC
LIMIT 1;
```

4c. Generate a report ranking race drivers based on their accumulated points

```
SELECT Drivers.DriverID, Drivers.DriverName, SUM(Results.Points) AS TotalPoints
FROM Results
JOIN Drivers ON Results.DriverID = Drivers.DriverID
GROUP BY Drivers.DriverID, Drivers.DriverName
ORDER BY TotalPoints DESC;
```

4d. Create a virtual table with details of all race drivers and the count of total number of races played, sorted by highest to lowest

```
CREATE VIEW DriverRaceCount AS
SELECT Drivers.DriverID, Drivers.DriverName, COUNT(Results.RaceID) AS TotalRaces
FROM Results
JOIN Drivers ON Results.DriverID = Drivers.DriverID
GROUP BY Drivers.DriverID, Drivers.DriverName
ORDER BY TotalRaces DESC;
```

4e. Generate a report displaying the ID and names of the races conducted, along with driver details, points scored, number of laps, and duration taken to complete the race

```
SELECT Races.RaceID, Races.RaceName, Drivers.DriverName, Results.Points, Results.Laps, Results.Duration
FROM Results
JOIN Races ON Results.RaceID = Races.RaceID
JOIN Drivers ON Results.DriverID = Drivers.DriverID
ORDER BY Races.RaceID, Results.Duration;
```