

Perfect — let's tackle this systematically. Since your uploaded exam paper covers **SQL theory (Section A)**, **applied queries (Section B)**, and **advanced SQL with analytical functions/views (Section C)**, I'll answer each question clearly with examples and reasoning.

---

## SECTION A – 20 Marks (Conceptual + Basic SQL)

### 1(a) Unique Key vs Primary Key

- **Primary Key:**
  - Ensures uniqueness and **no NULLs**.
  - Only one per table.
  - Used to uniquely identify each row.
- **Unique Key:**
  - Ensures uniqueness but **allows one NULL**.
  - Multiple unique keys can exist in a table.
- **Example:**

```
CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE,
    Phone VARCHAR(15) UNIQUE
);
```

Here, `EmpID` is the primary key, while `Email` and `Phone` are unique keys.

---

### 1(b) Find Duplicate Emails

```
SELECT Email, COUNT(*) AS DuplicateCount
FROM Person
GROUP BY Email
HAVING COUNT(*) > 1;
```

---

### 1(c) Add Columns in MySQL

```
ALTER TABLE Employee ADD COLUMN Department VARCHAR(50);
ALTER TABLE Employee ADD COLUMN JoiningDate DATE;
```

---

### 1(d) Group By with NULLs

Yes, `GROUP BY` can be applied on a column with duplicates and NULLs.

- NULLs are treated as a single group.

- Example:

```
SELECT FirstName, COUNT(*)
FROM Employee
GROUP BY FirstName;
```

---

## 1(e) RANK() vs DENSE\_RANK()

- **RANK()**: Skips ranks if ties occur.
- **DENSE\_RANK()**: No gaps in ranking.
- Example:

```
SELECT Salary,
       RANK() OVER (ORDER BY Salary DESC) AS Rank,
       DENSE_RANK() OVER (ORDER BY Salary DESC) AS DenseRank
FROM Employee;
```

---

## 2(a) Control Child Table on Parent Change

Use **ON DELETE CASCADE / ON UPDATE CASCADE**.

```
CREATE TABLE Orders (
    OrderID INT PRIMARY KEY,
    CustID INT,
    FOREIGN KEY (CustID) REFERENCES Customer(CustID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

---

## 2(b) Foreign Key in MySQL

```
CREATE TABLE Department (
    DeptID INT PRIMARY KEY,
    DeptName VARCHAR(50)
);

CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    DeptID INT,
    FOREIGN KEY (DeptID) REFERENCES Department(DeptID)
);
```

---

## 2(c) Second Highest Salary

```
SELECT MAX(Salary) AS SecondHighestSalary
FROM Employee
WHERE Salary < (SELECT MAX(Salary) FROM Employee);
```

---

## 2(d) Drop Table Effects

- Dropping a table removes: **columns, constraints, indexes, defaults**.

- Does **not** drop: Views, Stored Procedures (they must be dropped separately).
  - **Constraints:** PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL.
- 

## 2(e) Multi-row Operators

- Operators: `IN`, `ANY`, `ALL`.
- Example:

```
SELECT EmpName
FROM Employee
WHERE Salary IN (SELECT Salary FROM Employee WHERE DeptID=10);
```

---

# ✓ SECTION B – 30 Marks (Applied Queries)

### 3(a) Credits vs Withdrawals

```
SELECT
    CASE WHEN studied_credits > 150 THEN 'More_Credits' ELSE
    'Fewer_Credits' END AS CreditCategory,
    COUNT(*) AS student_withdraw_count
FROM studentinfo
WHERE final_result = 'Withdrawn'
GROUP BY CreditCategory;
-- Inference: Students with >150 credits show higher withdrawal counts.
```

---

### 3(b) Least Enrollment Regions

```
SELECT region, COUNT(*) AS enrollments
FROM studentinfo
GROUP BY region
ORDER BY enrollments ASC
LIMIT 1;
```

---

### 3(c) Students with/without Assessments

```
SELECT si.id_student, a.code_module, a.code_presentation,
a.assessment_type, sa.date_submitted
FROM studentinfo si
LEFT JOIN studentassessment sa ON si.id_student = sa.id_student
LEFT JOIN assessments a ON sa.id_assessment = a.id_assessment;
```

---

### 3(d) Least Discharges per Provider

```
SELECT provider_id, provider_name, COUNT(*) AS total_discharges
FROM InpatientCharges
```

---

```
GROUP BY provider_id, provider_name
ORDER BY total_discharges ASC;
```

---

### 3(e) Insurance Coverage Ratio

```
SELECT provider_id, provider_name,
       (AVG(covered_charges) / AVG(total_payments)) AS
    Insurance_coverage_ratio
   FROM InpatientCharges
  GROUP BY provider_id, provider_name
 ORDER BY Insurance_coverage_ratio DESC;
```

---

## ✓ SECTION C – 30 Marks (Advanced SQL)

### 4(a)(i) Arsenal Wins (Home + Away)

```
SELECT COUNT(DISTINCT team) AS TotalWins
FROM (
    SELECT opponent AS team
      FROM home
     WHERE goals_scored > goals_conceded
     UNION
    SELECT opponent AS team
      FROM away
     WHERE goals_scored > goals_conceded
) AS Wins;
```

---

### 4(a)(ii) Longest Gap MP vs MLA

```
SELECT person_name,
       MAX(ABS(DATEDIFF(mp.date, mla.date))) AS GapDays
FROM MP mp
JOIN MLA mla ON mp.person_name = mla.person_name
GROUP BY person_name
ORDER BY GapDays DESC
LIMIT 1;
```

---

### 4(a)(iii) Longest Song Length

```
SELECT s.songName, a.artisteName, s.songLength, a.artisteID
FROM song s
JOIN song_artiste sa ON s.songID = sa.s_songID
JOIN artiste a ON sa.a_artisteID = a.artisteID
WHERE s.songLength = (
    SELECT MAX(songLength) FROM song
);
```

---

## 4(a)(iv) View + Highest Runtime per Genre

```
CREATE VIEW MovieGenreView AS
SELECT m.movieTitle, m.movieRating, g.genreType, m.movieRuntime
FROM movie m
JOIN movie_genre mg ON m.movieID = mg.m_movieID
JOIN genre g ON mg.g_genreID = g.genreID;

SELECT movieTitle, movieRating, genreType, movieRuntime
FROM (
    SELECT movieTitle, movieRating, genreType, movieRuntime,
           RANK() OVER (PARTITION BY genreType ORDER BY
    movieRuntime DESC) AS ranking
    FROM MovieGenreView
) AS RankedMovies
WHERE ranking = 1;
```

---

## 4(a)(v) Max Runtime Movies (Excluding One)

```
SELECT m.movieID, m.movieTitle, g.genreType, m.movieRuntime
FROM movie m
JOIN movie_genre mg ON m.movieID = mg.m_movieID
JOIN genre g ON mg.g_genreID = g.genreID
WHERE m.movieRuntime = (SELECT MAX(movieRuntime) FROM movie)
AND m.movieTitle <> 'Two imprisoned men';
```

---

## 4(a)(vi) Earliest Release Date per Genre

```
SELECT m.movieTitle, m.movieRating, g.genreType,
m.movieReleaseDate,
       MIN(m.movieReleaseDate) OVER (PARTITION BY g.genreType) AS
first_release_date_of_genre
FROM movie m
JOIN movie_genre mg ON m.movieID = mg.m_movieID
JOIN genre g ON mg.g_genreID = g.genreID;
```

---

💡 That's a **complete set of answers** for your exam paper — theory, queries, and advanced SQL.

Would you like me to also **annotate each query with business/data interpretation comments** (like exam-style insights), so your answers look richer and more professional?