💡 Section 1: Database Fundamentals and Theory (Q1 & Q3)

Q1.1 & Q1.2: Definitions and Advantages (10 Marks)

Guideline	Principle	Perfect Answer Example
Data vs. Information	Use precise, contrasting terms. **Data is raw. Information is processed data** that provides meaning and context for decision-making.	**Data** is raw, unprocessed facts. **Information** is data that has been processed, organised, and structured to provide context and meaning.
DBMS Advantages	Memorise the five core benefits that solve traditional file system problems: **Sharing, Security, Integrity, Redundancy, and Access.**	1. Minimized Data Redundancy. 2. Increased Data Integrity. 3. Enhanced Data Security. 4. Improved Data Sharing. 5. Faster Data Access and Retrieval.
Access Language	Name the purpose and the standard language.	**Database Access Language** is code used to define, manipulate (insert, change, delete), and retrieve data. **Example: Structured Query Language (SQL).**

Q3: Keys and Data Analysis (10 Marks)

Guideline	Principle	Perfect Answer Example
Q3.1 Foreign Key (FK)	Identify the column in the child table that references the Primary Key (PK) of a parent table.	The Foreign Key in the Character table is **ClassID**.
Q3.3.1 Primary Key Suitability	PKs must be **unique** and **non-null**. If data shows a single column value repeats, it cannot be a PK.	**No,** the column is not suitable as a Primary Key. The value **Modise** is repeated for different clients, violating the uniqueness requirement.
Q3.3.2 Candidate Key Suitability	Candidate keys are potential Primary Keys, meaning they must also be **unique**. Use the same principle as the PK check.	**No,** the column is not suitable as a Candidate Key. The value **Johannesburg** is repeated for different clients, violating the uniqueness requirement.

📐 Section 2: ERD Interpretation (Q2 & Q3.2)

The most common error is misreading the multiplicities.

Guideline	Step-by-Step Method for Reading Business Rules	Perfect Answer Example (Q2.2.3 Country/Province)
1. Identify Cardinality	Look at the **OPPOSITE** end of the relationship line to the entity you are describing.	*To describe **Province**:* Look at the Country side: it shows 1..1.
2. Translate Notation	Translate the notation into mandatory (mandated by	1..1 means "must be in one

	the straight line) or optional (circle) and the quantity (1 or many).	and only one Country."
3. Write the Rule	Write the full sentence for the relationship: **Entity A (Cardinality) Action Entity B.**	**A Province** must belong to one and only one **Country**.
4. Reverse the Rule	Now describe the opposite direction.	*To describe **Country**:* Look at the Province side: it shows 1..*. **A Country** has one or more **Provinces**.

Q	Relationship Type Identification	Perfect Answer Example
Q2.2.1 (1..1 to 1..1)	If both sides are 1..1, it's **One-to-One (1:1)**.	**One-to-One (1:1):** One Person owns one Car, and one Car is owned by one Person.
Q2.2.2 (1..* to 1..*)	If both sides are 1..* or *, it's **Many-to-Many (M:N)**.	**Many-to-Many (M:N):** A Driver can drive many Trucks, and a Truck can be driven by many Drivers.

⚙️ Section 3: Normalisation (Q5)

Normalisation is a strict process. Follow the steps sequentially by identifying and resolving dependencies.

Q5.2: Identifying Normal Forms (9 Marks)

Guideline	Rule for Identification	Q5.2.2 Perfect Motivation (1NF)
1NF Check	Are all columns atomic? (Single values, no repeating groups). If Yes, it is at least 1NF.	Yes, all attributes are atomic.
2NF Check	Is there a **Partial Dependency (PD)**? (A non-key attribute depends on *part* of a composite key). If Yes, it is **NOT in 2NF**.	**NO**, it is not in 2NF because **Partial Dependencies** exist (e.g., Player Name -> Player Date Joined).
3NF Check	Is there a **Transitive Dependency (TD)**? (A non-key attribute depends on *another non-key attribute*). If Yes, it is **NOT in 3NF**.	**NO**, it is not in 3NF because **Transitive Dependencies** exist (e.g., Spaceship Name -> Spaceship Value).
Conclusion	State the highest form the relation *satisfies*.	**First Normal Form (1NF)** (because it violates both 2NF and 3NF).

Q5.3: Normalising to 2NF (Resolve Partial Dependencies) (10 Marks)

1NF Relation: R (<u>Player Name</u>, <u>Spaceship Registration</u>, Player Date Joined, Spaceship Name, Spaceship Value)

Step	Action	Resulting Relation
1. Find PDs	Identify attributes determined by only **part** of	

	the composite key: Player Name -> Player Date Joined and Spaceship Registration -> Spaceship Name, Spaceship Value.	
2. Decompose PDs	Create a new table for each partial key and its dependent non-key attributes.	**PLAYER** (<u>Player Name</u>, Player Date Joined)
3. Decompose PDs	Create a new table for the other partial key and its dependent attributes.	**SPACESHIP** (<u>Spaceship Registration</u>, Spaceship Name, Spaceship Value)
4. Maintain Link	Create a link table using the **full composite key** of the original 1NF relation to preserve the relationship.	**PLAYER_SPACESHIP** (<u>Player Name</u>, <u>Spaceship Registration</u>)

Q5.4: Normalising to 3NF (Resolve Transitive Dependencies) (9 Marks)

2NF Relations:

1. **PLAYER** (<u>Player Name</u>, Player Date Joined)
2. **SPACESHIP** (<u>Spaceship Registration</u>, Spaceship Name, Spaceship Value)
3. **PLAYER_SPACESHIP** (<u>Player Name</u>, <u>Spaceship Registration</u>)

Step	Action	Resulting Relation
1. Find TD	Check non-key to non-key dependencies. In SPACESHIP, assume Spaceship Name -> Spaceship Value (a non-key attribute determines another	

	non-key attribute).	
2. Decompose TD	Create a new table for the non-key determinant and its dependent attribute.	**SPACESHIP_VALUE_LOOKUP** (<u>Spaceship Name</u>, Spaceship Value)
3. Update Parent	Remove the transitively dependent non-key attribute from the original table, leaving the non-key determinant (which becomes a FK).	**SPACESHIP_DETAIL** (<u>Spaceship Registration</u>, Spaceship Name $\leftarrow$ *FK to Lookup*)
4. Final 3NF	All relations are now in 3NF.	**Final Relations: PLAYER, PLAYER_SPACESHIP, SPACESHIP_DETAIL, SPACESHIP_VALUE_LOOKUP**

💻 Section 4: SQL Application (Q6)

The key to perfect SQL is correct syntax and using the right clause for the job.

Q6.1.1: CREATE TABLE (5 Marks)

Guideline	Component	Perfect Code Snippet
Schema	Define all columns, data types, and NOT NULL constraints.	PresidentID INT AUTO_INCREMENT NOT NULL, Name VARCHAR(250) NOT NULL, Year YEAR NOT NULL, ...

Keys	Define the primary key (PK) and foreign key (FK) with references.	PRIMARY KEY (PresidentID), FOREIGN KEY (CountryID) REFERENCES Country (CountryID)

SQL

```sql
CREATE TABLE President (
  PresidentID INT AUTO_INCREMENT NOT NULL,
  CountryID INT NOT NULL,
  Name VARCHAR(250) NOT NULL,
  Surname VARCHAR(250) NOT NULL,
  Year YEAR NOT NULL,
  PRIMARY KEY (PresidentID),
  FOREIGN KEY (CountryID)
    REFERENCES Country (CountryID)
);
```

Q6.1.2 - Q6.1.5: SELECT, COUNT, INSERT, FILTER (4+4+3+3 Marks)

Question	SQL Command Guide	Perfect Code Snippet
Q6.1.2 (Count after 2009)	Use COUNT() and the WHERE clause for filtering individual rows.	SELECT COUNT(PresidentID) FROM President WHERE Year > 2009;
Q6.1.3 (INSERT)	Specify the table and list the columns and values in order.	INSERT INTO Country (CountryID, Name, Abbreviation, CallingCode) VALUES (5, 'Botswana', 'BW', '267');

Q6.1.4 (SELECT/ORDER)	Use SELECT * for all columns and ORDER BY for sorting.	SELECT * FROM Country ORDER BY Name ASC;
Q6.1.5 (LIKE Filter)	Use WHERE with the LIKE operator and the % wildcard for partial matches.	SELECT * FROM President WHERE Surname LIKE 'R%';

Q6.1.6: JOIN (5 Marks)

Guideline	**Principle**	**Perfect Code Snippet**
JOIN	Link tables using the **PK=FK** relationship. Use table aliases (P and C) for brevity.	FROM President P JOIN Country C ON P.CountryID = C.CountryID
SELECT	Select only the required columns, using AS to rename joined attributes (like Country Name).	SELECT P.Name, P.Surname, C.Name AS CountryName

```sql
SELECT
  P.Name,
  P.Surname,
  C.Name AS CountryName
FROM
  President P
JOIN
  Country C ON P.CountryID = C.CountryID;
```

Q6.2 & Q6.4: WHERE vs HAVING and Query Results (13 Marks)

Q	Guideline for Result/Explanation	Perfect Answer
Q6.2 WHERE vs HAVING	**WHERE** filters **rows** *before* grouping. **HAVING** filters **groups** *after* grouping (must be used with GROUP BY).	**WHERE** filters individual rows based on a condition applied to non-aggregated columns. **HAVING** filters groups of rows based on a condition applied to aggregate functions (e.g., COUNT(), SUM()).
Q6.4.2 (SELECT J.*, COUNT()... HAVING >= 2)	The GROUP BY clause groups all cases by judge. The COUNT() function counts the cases per group. The HAVING clause filters out groups (judges) with less than 2 cases.	The query returns **all columns for every Judge** who has handled **two or more** cases, along with the **total count of cases** they handled.
Q6.4.3 (DROP TABLE Judge;)	DROP TABLE removes the database object entirely, not just the data.	The entire Judge table, including its structure (schema) and all data, is **permanently deleted** from the database.