

Summary

RDBMS and its Features

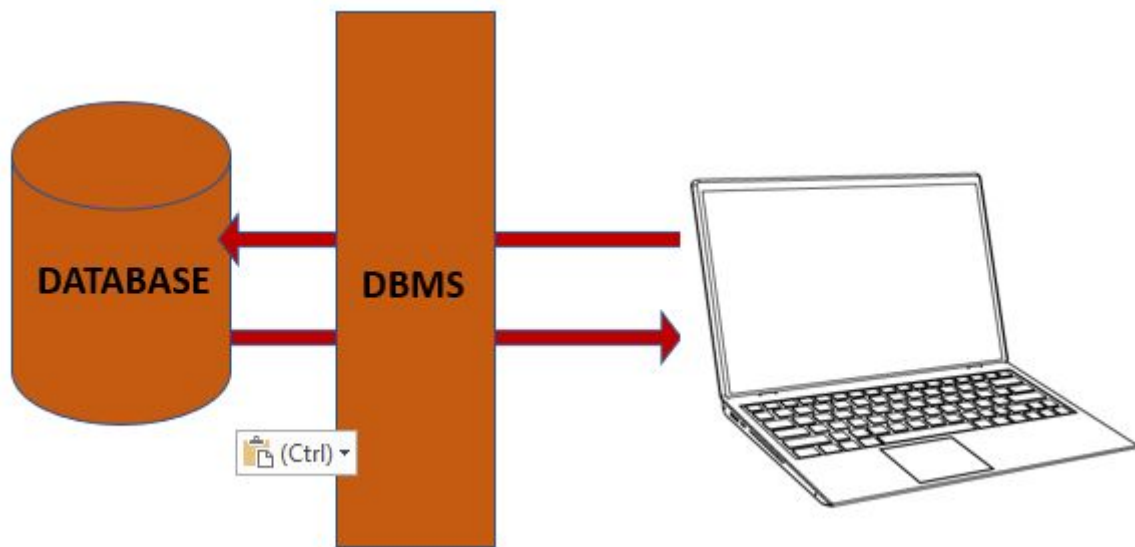
In this session, you understood the concept of RDBMS. You began by learning about the various drawbacks of the File-Based Storage System and saw how RDBMS can overcome these drawbacks. Further, you learnt about the features of RDBMS and the relational model. Finally, you learnt how to create an entity relationship diagram (ERD) to portray different interactions between the entities in the database.

Introduction to RDBMS

Before RDBMS was introduced, data was stored in a File-Based Storage System. However, this mode of storing data posed several challenges, which led to the popularity of relational database management systems. The key disadvantages of a File-Based Storage System are as follows:

- **Data redundancy (duplicate data):** When you are using the file system for data storage, you are likely to end up storing duplicate files in different folders. Duplicity of data creates data redundancy, making it difficult for users to update/delete data.
- **Data inconsistency:** If you want to update a file stored in multiple locations, then you will have to locate each copy of the file and update them one by one. In case you miss any copy of the file, it will give rise to data inconsistency on your machine.
- **Scattered data:** Data is mostly scattered across various files, and each file may be stored in a different format. As a result, developing new applications to retrieve this data becomes difficult.
- **No support for transactions:** Consider the example of a banking transaction. This process consists of multiple steps for transferring money from one account to another. Suppose an ongoing transaction fails in the middle, resulting in a partial success. In an ideal scenario, this money should be transferred back to its owner so that the transaction can be repeated. However, this is not possible in the case of a File Based Storage System. Examples of other transactions are communication channels, booking a ticket, etc.
- **Poor data security:** It is difficult to impose stringent security constraints in file processing systems.
- **Data integrity:** In order to ensure data quality, we need to impose some constraints on incoming data before storing it in the File Storage System. However, these constraints are not supported inherently in the File Storage System.

An RDBMS is a database management system that stores data in the form of tables, where each table is related to the other tables in the database. The key idea behind an RDBMS is that it stores your data while preserving the relationships within this data. This kind of model is called a relational model. Hence, it is said that an RDBMS follows a relational model.



Features of RDBMS

The first feature of an RDBMS is its tabular structure. Consider a company database consisting of the following two tables:

EMPLOYEE			
EMP_ID	NAME	D_ID	SALARY
1	VISHWA	D01	55000
2	MOHAN	D02	60000
3	SHIVA	D03	72000

DEPARTMENT	
D_ID	D_NAME
D01	ACCOUNT
D02	HR
D03	SALES

EMP_ID: Employee_ID

D_ID: Department ID

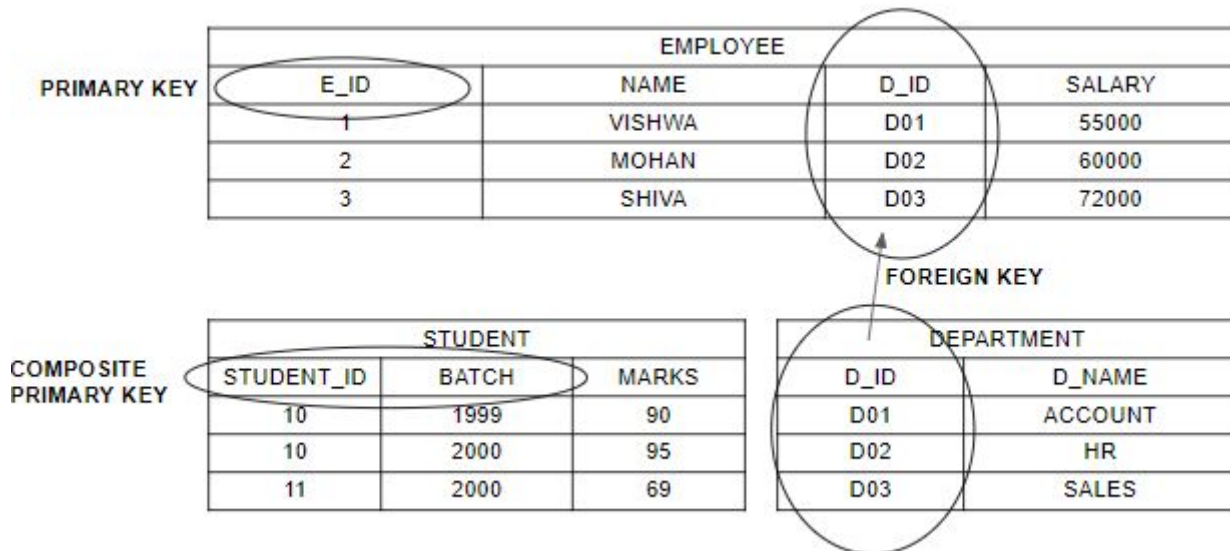
D_NAME: Department Name

The data is stored in the form of tables. Each of these tables is represented by its entities. In the table shown above, the entity would be 'EMPLOYEE'. Thus, entities can be used to identify a table.

The first row in each table shows the different fields. In the table above, the fields are E_ID, NAME, D_ID and SALARY. The rows below the fields consist of the values under each field. These rows are also termed 'records'.

Within a table, you can have primary keys, composite primary keys and foreign keys.

1. **Primary keys:** A primary key is used to uniquely identify each row in a certain table.
2. **Composite primary keys:** This kind of key is used when a single key is not enough to uniquely identify each row in a table. In such a case, a combination of two or more keys can be used.
3. **Foreign keys:** A foreign key is a field in a table that acts as a primary key in another table, thus helping us to uniquely identify the rows in the latter table.

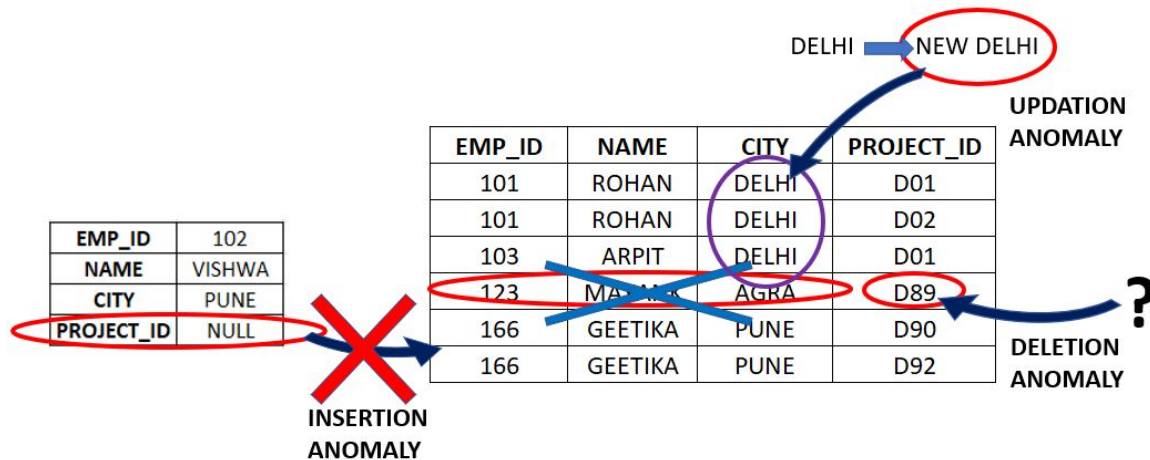


Normalisation: 1NF and 2NF

The second feature that you learnt about RDBMS is database normalisation.

You learnt how an RDBMS addresses the issue of data redundancy. Data redundancy occurs when the same piece of data is stored in multiple locations in a database. In such a case, the data could be prone to inconsistency; further, it becomes difficult for the user to make changes to the database.

A table in its denormalised form faces several drawbacks as shown below:



- **Insertion anomaly:** This kind of anomaly occurs when you cannot insert some attributes because certain attribute information is missing.

Consider the table shown above. Now, if a new employee is not assigned any project, then we cannot update the table with their information, as entering the NULL value in the project field is not a good decision. This is known as an insertion anomaly.

- **Update anomaly:** This kind of anomaly occurs in duplicated data when one copy is updated but the others are not, thus resulting in inconsistent data.

Now, let's revisit the example shown above. Consider a hypothetical situation wherein you need to change the city name 'Delhi' to 'New Delhi'. This needs to be updated in each row of the table that contains the name 'Delhi'. This kind of anomaly is termed update anomaly.

- **Deletion anomaly:** A deletion anomaly occurs when the deletion of certain attributes results in the deletion of some other attributes as well.

If you observe the table above, you will notice that only one Employee, Mayank, is assigned to Project ID D89. Suppose Mayank decides to leave the organisation; in this situation, there will be no Employee assigned to Project ID D89, and deleting Mayank's records will also result in the deletion of the record of Project ID D89. This is known as a deletion anomaly.

In order to solve these issues, the data can be converted to different levels of normalisation.

First normal form:

1. A cell in a table cannot have multiple values.
2. Each row must be uniquely identifiable.

PRIMARY KEY

PLAYER_ID	NAME	AGE	SPORT
P01	VISHWA	29	CRICKET, FOOTBALL
P02	ANKIT	34	CRICKET

1 NF

COMPOSITE PRIMARY KEY

PLAYER_ID	NAME	AGE	SPORT
P01	VISHWA	29	CRICKET
P01	VISHWA	29	FOOTBALL
P02	ANKIT	34	CRICKET

If you observe the first table, the cell under SPORT for PLAYER_ID, P01, contains two values, CRICKET and FOOTBALL. In order to avoid this situation, where a single cell has multiple values, you must split the record into two records so that each cell of the record contains a single value. Then choose the primary key so that each row can be identified uniquely. As shown in the example, we have split the record for P01 such that the new composite primary key consists of the PLAYER_ID and the SPORT column.

Second normal form:

1. It should be in the **first normal form**.
2. None of the attributes should be **partially dependent on the primary key**.

PARTIAL DEPENDENCY

ID	STU_ID	SUB_ID	MARKS	TEACHER
1	10	101	90	VISHWA
2	10	102	95	MOHAN
3	11	101	99	VISHWA

2 NF

COMPLETE DEPENDENCY

SUB_ID	TEACHER
101	VISHWA
102	MOHAN

+

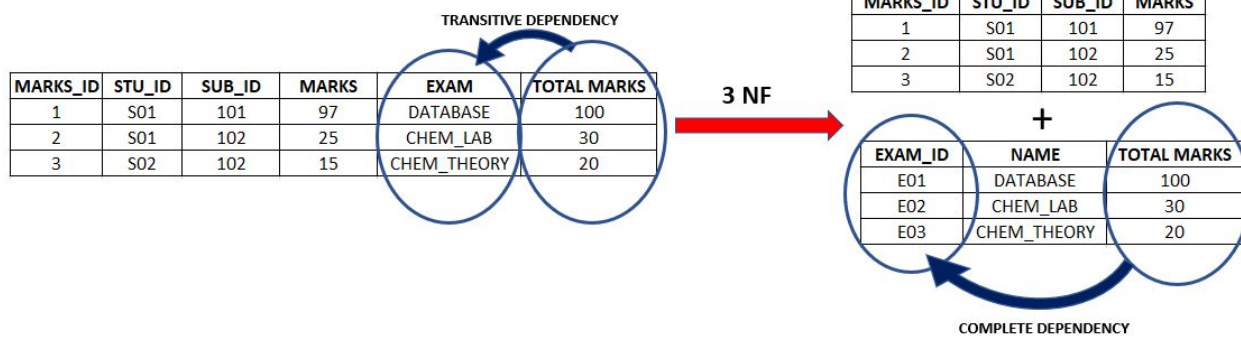
MARKS_ID	STU_ID	SUB_ID	MARKS
1	10	101	90
2	10	102	95
3	11	101	99

Since the 'TEACHER' column is partially dependent on the composite primary key, the table has to be split into two tables: the SUBJECT table and the MARKS table. After this, each column in the respective table becomes fully dependent on its corresponding primary key.

Third Normal Form and the ACID Model

Third normal form:

- It should already be in the second normal form.
- There should not be any transitive dependency in that particular table.



Since the 'TOTAL MARKS' column is connected to the primary key through the 'EXAM' column, this table exhibits transitive dependency. In order to avoid this kind of dependency, we must split the tables into two, as shown in the diagram above, so that each of the columns in the respective table is completely dependent only on the primary key.

Advantages of normalisation:

1. Using normalisation, you can split a table into multiple tables to avoid the occurrence of multiple rows. This solves the problem of **data duplicacy**.
2. Normalisation ensures **data integrity**, which means that the consistency and accuracy of the data are maintained. This suggests that you will not have to worry about having conflicting information in different locations in your database.
3. After normalisation, you will have to update the data only where it is necessary, rather than updating it in all its locations. This makes the process of **updating the data easy**.
4. Each table in a normalised database is well defined, and each row in a table is uniquely identifiable. This ensures that your database is much more **organised and flexible in design** as compared to the denormalised database.

The third feature of RDBMS is the ACID model.

A - Atomicity:

Atomicity means 'all or nothing'. Any operation on a database can be either a complete success or a complete failure. If an operation is partially successful and fails in between, then it will revert to its original state.

C - Consistency:

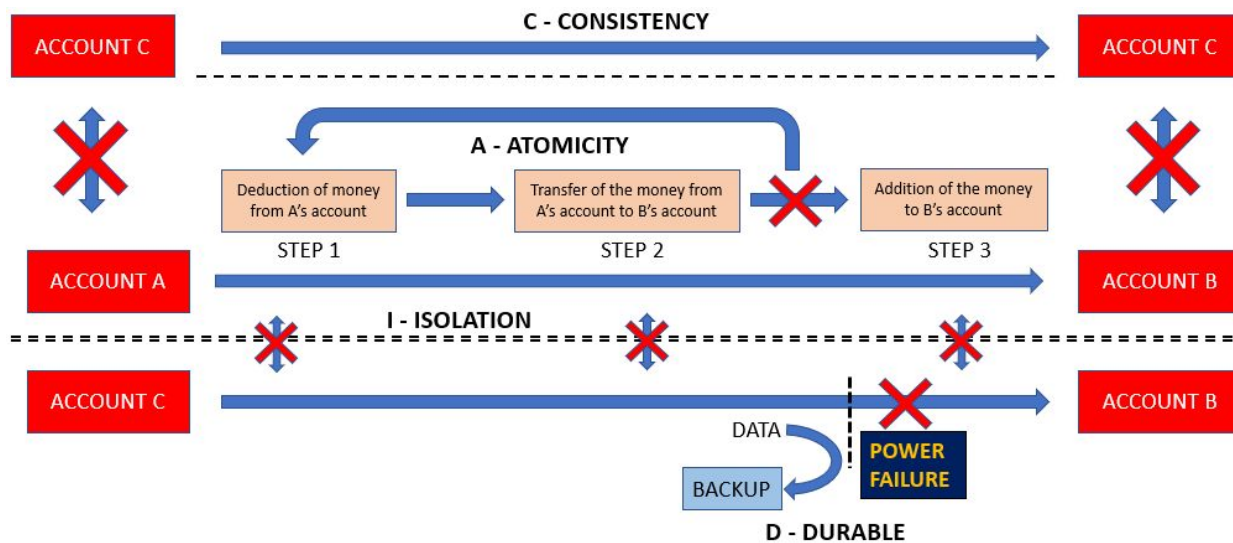
Once an operation has been performed, the state of the database must be consistent with that before the operation was performed. In other words, the execution of an operation cannot impact any other record or attribute apart from the ones it was set to execute on.

I - Isolation:

Any two operations are always isolated from or independent of each other. Their individual data will remain concealed from the data of other operations (transactions).

D - Durability:

In case of any failure (such as power failure), your data must never be lost. Hence, your RDBMS must ensure the durability of your data.



Relational Structure

The relational model is the base upon which an RDBMS is built. The relational model is comprised of three major components:

1. Data structure

- As the name suggests, this model deals with the structure of the data. Data is stored in the form of tables. Here, tables are also known as relations. A relation consists of two parts, which are detailed below.
- Relation heading:** The heading consists of different columns, and each heading value consists of a name, i.e., the identifier, and its data type.
- Relation body:** The relation body stores the different records within the table. Each cell within the record contains its individual name and data type.
- For an end user to access this data, they need to know only the attributes and their unique key identifiers; they need not know how the data is being stored physically.

2. Data integrity

It makes sure that the data stored is accurate at any point in time. Data integrity is ensured by three components:

- **Attribute integrity:** Attribute integrity ensures that each attribute in a relation heading consists of the relation name and its data type. Attribute integrity ensures that this rule is not violated.
- **Entity integrity:** This rule ensures that no two entities are the same, and each entity is uniquely identifiable. We achieve this integrity by introducing the following keys:
 - a. Primary keys, and
 - b. Composite primary keys.

Note: Primary keys and composite primary keys cannot be NULL.

- **Referential integrity:** This rule determines how different tables are related to each other. This is achieved with the help of foreign keys.

Some features associated with referential integrity are as follows:

- a. When a referenced row from a table where it acts as a primary key is deleted, its corresponding value in this table is set to NULL.
- b. In case you do not want your foreign key to be set to NULL, you can set it to the default value.

3. Data manipulation

This model mainly deals with how we can apply an operation between two relations to generate new relations.

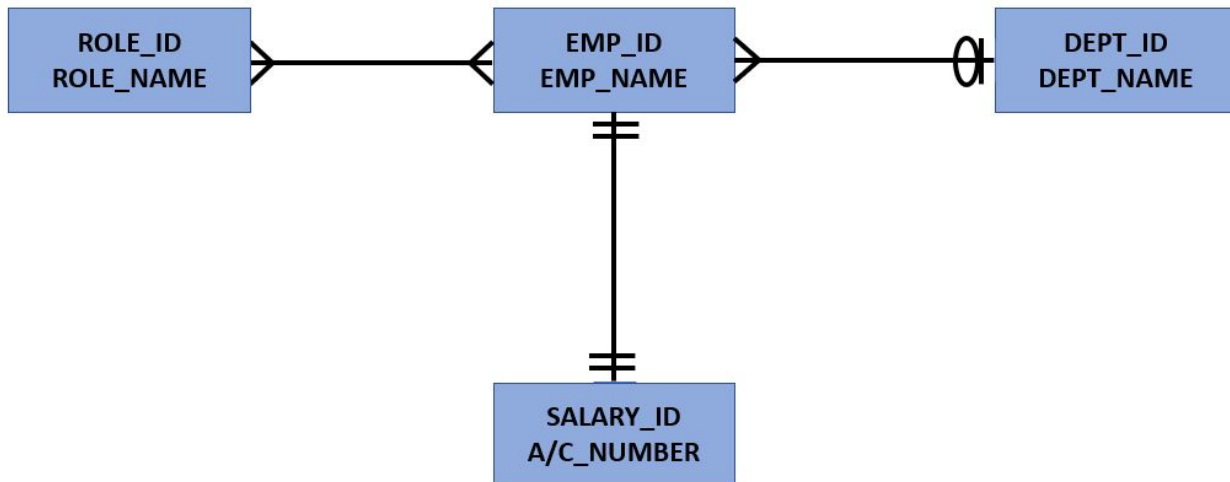
ERDs

Entity Relationship Diagrams

An entity relationship diagram, or ERD, is a visualisation of all the tables that represent how all the entities interact with each other.

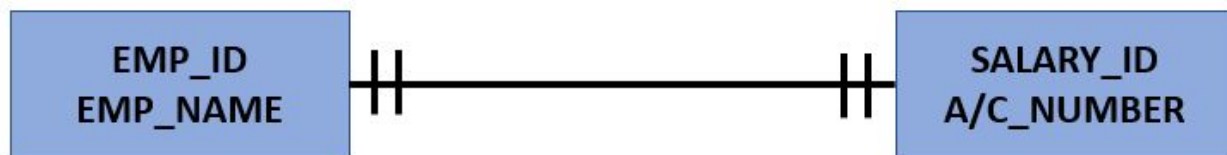
Consider a company database that consists of the following tables and the ERD:

1. The 'Employee' table
2. The 'Department' table
3. The table of the 'Account Details' of the employees
4. The 'Employee Roles' table



The different interactions can be explained as follows:

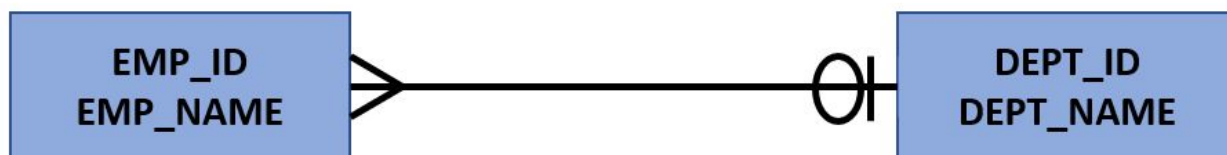
1. One-to-one relationship



This representation translates to the following statements:

'Only one employee corresponds to one salary account', and 'Only one salary account corresponds to one employee'.

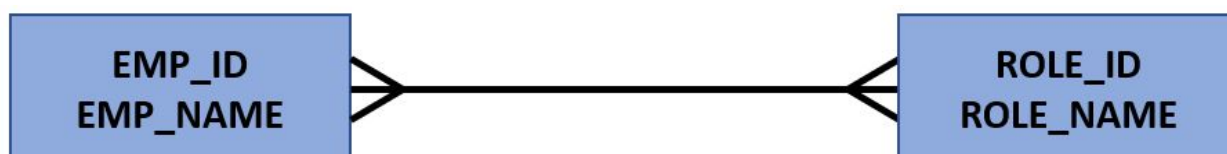
2. Many-to-zero/one relationship



This interaction can be translated to the following statements:

'Many employees may correspond to minimum zero or maximum one department', and 'Minimum zero or maximum one department corresponds to multiple employees'.

3. Many-to-many relationship



This representation can be translated to the following statements:

“Any employee in the ‘Employee’ table can have multiple roles”, and ‘Any role in the ‘Roles’ table can be assigned to multiple employees”.

Summary

Basic SQL Querying

In this session, you learnt how to write basic SQL queries. You began this session by understanding the use cases and features of SQL. After this, you learnt about DDL and DML statements as well as basic statements and operators, aggregate and inbuilt functions, string functions, datetime functions, nested queries and views.

Introduction to SQL

SQL stands for Structured Query Language. In some applications, it is also referred to as DSL, or Domain Specific Language. SQL is used for managing and manipulating data held in a relational database management system (RDBMS).

SQL helps you to:

1. Create data,
2. Manipulate data,
3. Delete data, and
4. Share data.

Why should you learn SQL?

1. SQL is language-agnostic, which makes it a language that can be easily learnt and comprehended.
2. It is also supported by all the modern database systems such as Oracle, MySQL and SQL Server.
3. In the last session, you learnt how SQL fills the gap between Hadoop and Hive. This is because Hive systems, which are used for big data processing, are also based on the SQL syntax, which is easy and user-friendly.

Among the existing RDBMS systems, you learnt how to work with MySQL. RDBMS commonly finds application in the following:

- The banking sector
- E-commerce websites

- Social networking sites

DDL Statements

SQL commands are mainly divided into two subcategories:

1. DDL (Data Definition Language), and
2. DML (Data Manipulation Language)

DDL (Data Definition Language)	DML (Data Manipulation Language)
DDL deals with defining the structure of the data. It creates and modifies database objects such as types and number of attributes, data types of columns and various keys such as primary and foreign keys in the tables.	DML commands are used to make modifications within the data present in the database.
The most common DDL commands are as follows: <ol style="list-style-type: none"> 1. CREATE 2. ALTER 3. DROP 	The most common DML commands are as follows: <ol style="list-style-type: none"> 1. INSERT 2. UPDATE 3. DELETE

In the following table, you can find the basic commands within the Data Definition Language:

DDL Syntaxes	
CREATE	Used to create databases, tables, primary keys, foreign keys, etc.
ALTER	Used to change the structure of the table ALTER TABLE - ADD To add additional columns to the table ALTER TABLE - DROP To remove columns from an existing table ALTER TABLE - MODIFY To modify an existing column in a table
DROP	Used to drop tables, columns or the entire database

DML Statements

In the following table, you can find the basic commands within the Data Manipulation Language:

DML Commands	
INSERT	This command is used to add data to existing columns in a table in RDBMS.
UPDATE	This command is used to update existing rows in a table in RDBMS.
DELETE	This command is used to delete specific rows from a table in RDBMS.

Note:

1. If you wish to view your table at any point in time, use this command:
`SELECT * FROM table_name`
2. The commands in SQL are **not case-sensitive**, i.e., you can write your SQL queries in uppercase or lowercase.

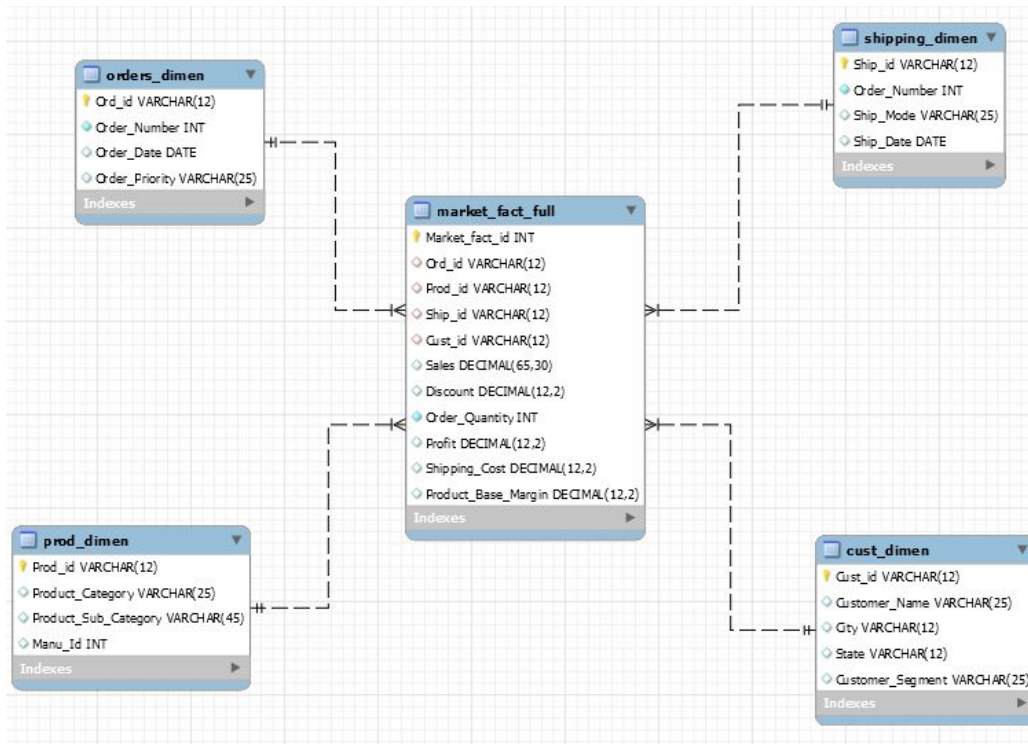
SQL Basic Statements and Operators

Here are some common data types supported in MySQL:

Numeric Data Types	INT, TINYINT, SMALLINT, BIGINT, DOUBLE, DECIMAL
Date and Time Data Types	TIMESTAMP, DATETIME, YEAR
String Data Types	CHAR, VARCHAR, ENUM

Additional links: [MySQL Data Types](#).

MySQL also enables you to create ERDs for your database as shown below:



Some of the important SQL commands are as follows:

- A. **SELECT**: This is used to read the data.
- B. **WHERE**: This is used as a conditional statement to filter out data.
- C. **OPERATORS**: Some examples of operators that can be used along with the WHERE clause are 'OR', 'AND', 'IN' and 'BETWEEN'.

```
SELECT customer_name ,customer_segment, City FROM cust_dimen WHERE customer_segment='Corporate' OR city='Mumbai'
```

Read data from columns customer_name ,customer_segment, City in the cust_dimen table

Filter out those rows where either customer_segment attribute is 'Corporate' or city attribute is 'Mumbai'

```
SELECT * FROM cust_dimen WHERE state IN ('Tamil Nadu' , 'Karnataka' , 'Telangana' , 'Kerala');
```

Read all the columns from cust_dimen table

Filter out those rows where the state attribute is one of 'Tamil Nadu' , 'Karnataka' , 'Telangana' , 'Kerala';

```
SELECT ord_id , shipping_cost FROM market_fact_full WHERE ord_id LIKE '%\_5%' AND shipping_cost BETWEEN 10 and 15
```

Read data from columns 'ord_id' , 'shipping_cost' in the market_fact_full table

Filter out those rows where:

1. ord_id contains the substring '_5' in their string
2. Value of the shipping_cost lies between 10 and 15

- D. The **LIKE operator** is a logical operator that checks whether a string contains a specified pattern or not. The following two wildcards are used to specify the pattern:
 1. '%' allows you to check with a string of any length.
 2. '_' allows you to check with a single character.

Here are some use cases:

Example	Description
<pre>SELECT Customer_name FROM cust_dimen WHERE customer_name LIKE 'CLA%';</pre>	This will return all the names of the customers whose names start with 'CLA'.
<pre>SELECT Customer_name FROM cust_dimen WHERE customer_name LIKE 'CL_I%RE GOOD';</pre>	<p>OUTPUT: CLAIRE GOOD</p> <p>This will return those strings that contain a single character between 'CL' and 'IRE GOOD'.</p>
<pre>SELECT Customer_name FROM cust_dimen WHERE customer_name LIKE '%IRE%';</pre>	This statement will return all the strings that contain the substring 'IRE' in the strings.

Note: In the example demonstrated above, the escape operator ('\') has been used to search for strings containing the substring '_5'. This is because the character '_' is in itself an operator. Hence, we need to specify the escape character so that MySQL interprets '_' as a literal character. To understand the LIKE operator in depth, you can refer to this [link](#).

Aggregate and Inbuilt Functions

Following are the aggregation functions in SQL:

1. **COUNT():** Counts the total number of records specified by the given condition
2. **SUM():** Returns the sum of all the non-NULL values
3. **AVG():** Returns the average of all the non-NULL values
4. **MIN():** Returns the minimum of the specified values (NULL is not included)
5. **MAX():** Returns the maximum of the specified values (NULL is not included)

The data is grouped using the GROUP BY clause. It groups rows that have the same values based on the specified condition and performs operations on the individual groups. The syntax is as follows:

```
SELECT column_names
FROM table_name
WHERE condition
GROUP BY column_names
HAVING condition
```

Print the total number of customers from each city within the 'city' column.	<pre>select count(Customer_Name) as City_Wise_Customers, City from cust_dimen group by City;</pre>
Print the total number of customers from Bihar in each segment.	<pre>select count(Customer_Name) as Segment_Wise_Customers, Customer_Segment from cust_dimen where State = 'Bihar' group by Customer_Segment;</pre>
Find the sum of the shipping cost for each customer whose total shipping cost is more than 50.	<pre>select Cust_id, sum(Shipping_Cost) as Customer_Wise_Shipping_Cost from market_fact_full group by Cust_id having Customer_Wise_Shipping_Cost > 50;</pre>

Basic SQL Statements and Operators

Ordering in SQL

The ORDER clause is used to sort the values in the columns in ascending or descending order. The order statement must be followed by 'asc' or 'desc' in order to specify an ascending or descending order, respectively. In case nothing is mentioned, the default sort is ascending.

The general syntax for the grouping and ordering functions is as follows:

```
SELECT column_names
FROM table
WHERE condition
GROUP BY column_names
HAVING condition
ORDER BY column_names
```


Some common examples are as follows:

List the customer names in alphabetical order.	<pre>select Customer_Name from cust_dimen order by Customer_Name;</pre>
<ol style="list-style-type: none"> 1. Group the products by the Product ID, apply the SUM operation to each group and alias it to product_wise_order_quantity. 2. Order the aliased output product_wise_order_quantity in descending order and print the top three product IDs. 	<pre>select Prod_id, sum(Order_Quantity) as Product_Wise_Order_Quantity from market_fact_full group by Prod_id order by Product_Wise_Order_Quantity limit 3;</pre>
Arrange the order IDs in the order of their recency.	<pre>select Ord_id, Order_Date from orders_dimen order by Order_Date desc;</pre>

String Functions

Function	Example	Description	Output
UPPER	SELECT UPPER('upgrad');	Converts to uppercase	UPGRAD
LOWER	SELECT LOWER('UPGRAD');	Convert to lower case	upgrad
SUBSTRING	SELECT SUBSTRING('upgradeability', 1, 6)	SUBSTRING(text, start_index, length of substring)	upgrad
SUBSTRING_INDEX	SELECT SUBSTRING_INDEX('www.up grad.com', '.', 1)	Returns the substring before the mentioned delimiter	www
	SUBSTRING_INDEX('www.up grad.com', '.', 2)	SUBSTRING_INDEX(text, delimiter, occurrence number of delimiter)	www.upgrad
LENGTH	SELECT LENGTH('upgrad');	Gives the number of characters in the input	6
REVERSE	SELECT REVERSE('upgrad');	Reverses the string	dargpu

Date Time Functions

Syntax	Description	My output
SELECT CURTIME()	Returns the current time	17:11:12
SELECT MONTH('2020-03-10')	Returns the month of the date	03
SELECT MONTHNAME('2020-03-10')	Returns the name of the month of the given date	March
SELECT STR_TO_DATE('10,03,2020','%d,%m,%Y')	Converts the string to date type	2020-03-10
SELECT YEAR('2020-03-10')	Extracts the year from the date	2020
SELECT DAYNAME('2020-03-10')	Returns the day of the week for the given date	Tuesday
SELECT DAYOFMONTH('2020-03-10')	Extracts the day of the month from the given date	10

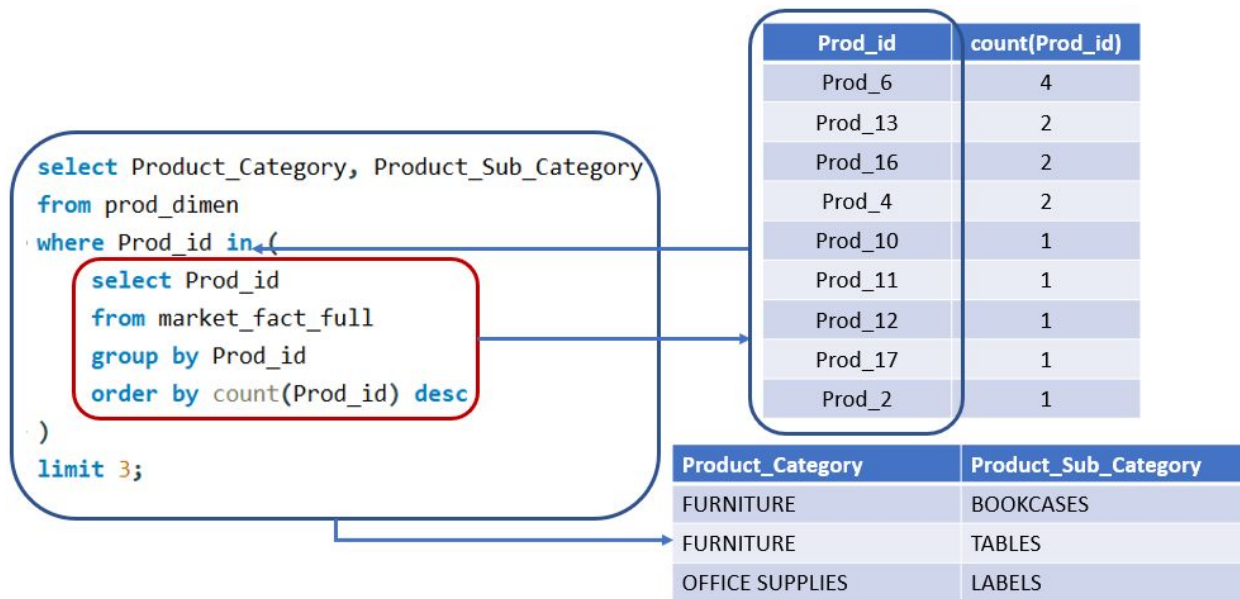
To learn more about string and datetime functions, you can visit these links:

[String functions](#)

[Date-time functions](#)

Nested Queries

In simple terms, a nested query is a query within another query. The inner query is known as the subquery; it performs an operation, the output of which is used by the main query to perform additional operations.



In the image above, the inner query returns the Prod_id sorted by their count. Note that their corresponding count has only been shown for your understanding and is not returned as part of the inner query. In the outer query, we find the Product_Category and Product_Sub_Category for the corresponding Prod_id values and limit them to 3.

Views

A view is a virtual table created as a result of operations on a single table or multiple tables. A view can be treated like any other table and can be further updated or deleted.

One major advantage of creating a view is the data security. Suppose you have confidential data and you need to make specific columns available to a user who is denied access to the rest of the data. In such cases, views can be used to extract the specific data, which is then shared with the user while keeping the main table hidden. As a result, the end user can only see the view.

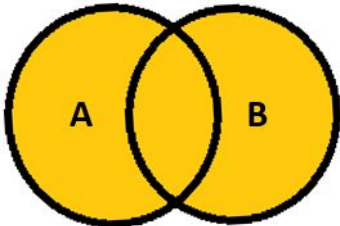
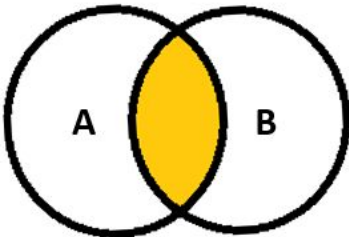
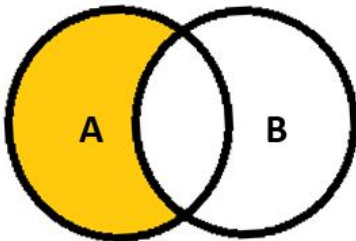
Summary

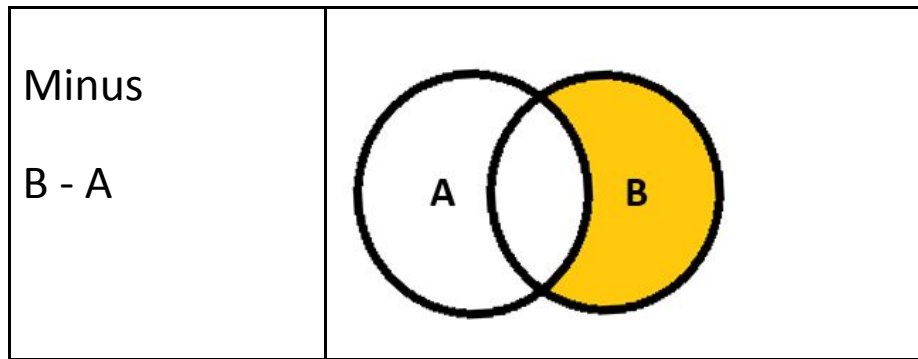
Advanced SQL Querying

In this session, you learnt about advanced queries such as joins, set operations and using views with joins.

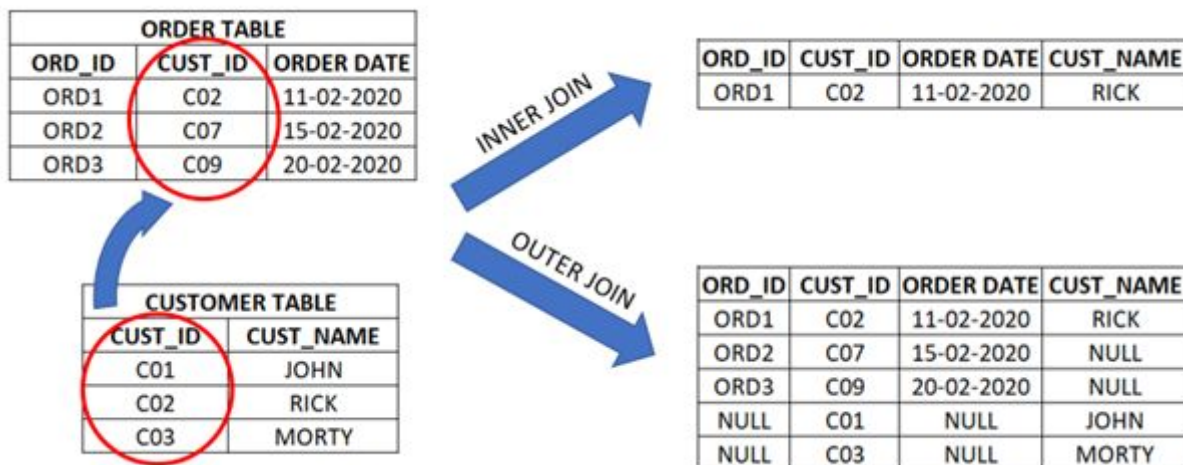
Venn Diagrams and Inner and Outer Joins

Venn diagrams help us to diagrammatically show the relationship between two or more sets. For any two given sets A and B, we can define the following relationships using Venn diagrams:

Union $A \cup B$	
Intersection $A \cap B$	
Minus $A - B$	



1. **Inner join:** It joins two tables along the rows wherever the matching condition is satisfied.
2. **Outer join:** It joins two tables along all the rows and returns a table containing the rows from both tables. In case the join condition fails, the missing values will be returned as NULL values in the table.

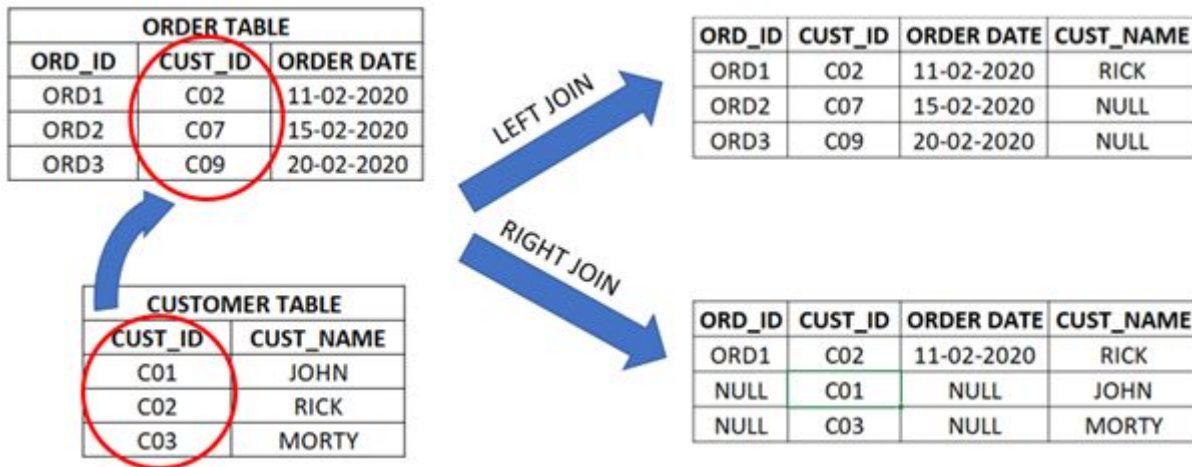


Left and Right Joins

1. **Left join:** It preserves the rows of the left table while attaching the rows of the right table with matching values in the corresponding column of the left table.

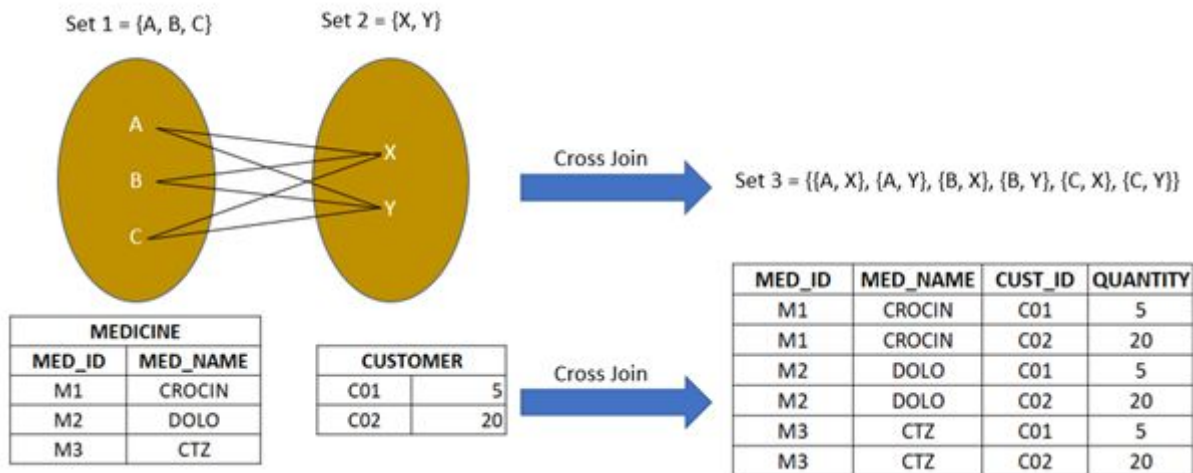
As you can see, we have preserved all the rows in the ORDER TABLE and attached only those rows from the CUSTOMER TABLE the CUST_ID values of which match those from the CUSTOMER TABLE.

2. **Right join:** It preserves the rows of the right table while attaching the rows of the left table with matching values in the corresponding column of the right table.



Cross Join

Using cross joins, you can merge two separate sets of m and n elements to form all the possible ways of combining them to form a single table of $m \times n$ elements.



Views with Joins

- `create view order_info`
`as select Ord_id, Sales, Order_Quantity, Profit, Shipping_Cost`
`from market_fact_full;`
- `create view market_facts_and_orders`
`as select *`
`from market_fact_full`
`inner join orders_dimen`
`using (Ord_id);`
- `select sum(Profit) as Year_Wise_Profit, year(Order_Date) as Order_Y`
`from market_facts_and_orders`
`group by Order_Year`
`order by Year_Wise_Profit desc`
`limit 1;`

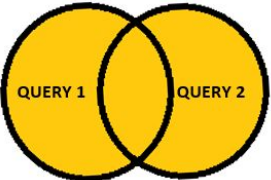
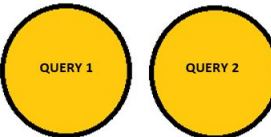
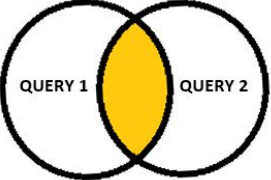
Creating a view called 'order_info' that consists of the following columns from the 'market_fact_full' table

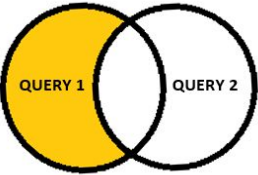
1. Ord_id
2. Sales
3. Order_Quantity
4. Profit
5. Shipping_Cost

Creating another view called 'market_facts_and_orders' that is made by the inner join of the 'market_fact_full' table with the 'orders_dimen' table on the 'Ord_id' column

Applying the GROUP BY function to find the total profits each year and sorting them in descending order. Finally the first row is returned which results in returning the year with highest profit

Intersect, Minus, Union and Union All

Operation	Condition	Diagrammatic Representation	Description	Syntax
UNION	1. Must be applied on the result of a SELECT Query		Returns the (unique) union of the select statements	SELECT * FROM table1 UNION SELECT * FROM table2
UNION ALL	2. The columns for both tables must be the same		Returns all the values from both the select statements	SELECT * FROM table1 UNION ALL SELECT * FROM table2
INTERSECT			Returns the intersection of the two select statements	SELECT * FROM table1 INTERSECT SELECT * FROM table2;

MINUS			Returns the values from the first select statement after removing the common elements in both the select statements	<pre>SELECT * FROM table1 MINUS SELECT * FROM table2;</pre>
--------------	--	---	---	---

Note: Although INTERSECT and MINUS are supported in other SQL-based RDBMS, these operators are not supported by MySQL. Hence, inner joins or nested queries can be used as an alternative to find the intersection or subtraction between two sets.

Disclaimer: All content and material on the upGrad website is copyrighted material, either belonging to upGrad or its bonafide contributors, and is purely for the dissemination of education. You are permitted to access print, and download extracts from this site purely for your own education only and on the following basis:

- You can download this document from the website for self-use only.
- Any copy of this document, in part or full, saved to disc or to any other storage medium, may only be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.
- Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of content for any other commercial/unauthorised purposes in any way which could infringe the intellectual property rights of upGrad or its contributors, is strictly prohibited.
- No graphics, images or photographs from any accompanying text in this document will be used separately for unauthorised purposes.
- No material in this document will be modified, adapted or altered in any way.
- No part of this document or upGrad content may be reproduced or stored in any other website or included in any public or private electronic retrieval system or service without upGrad's prior written permission.
- Any right not expressly granted in these terms are reserved.