

Basics of DBMS and SQL

1. Introduction to Databases and RDBMS

Databases

A database is a collection of information that can be stored, retrieved, modified, deleted, and processed.

Why Databases

The key disadvantages of a file-based storage system are as follows:

Data redundancy (duplicate data): While using the file system as data storage, you might end up storing duplicate files in different folders. The 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, you need to locate each copy of the file and update it individually. In case you miss out on 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 to transfer money from one account to another. Suppose an ongoing transaction fails in the middle, resulting in partial success. Ideally, this money should be transferred back to its owner so that the transaction can be repeated. However, this is not possible with 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 on file processing systems.

Data integrity: To ensure data quality, we need to impose some constraints on the incoming data before storing it into the file storage systems. However, these constraints are not inherently supported in the file storage system.

Databases are used to perform CRUD operations on data, where CRUD stands for: C: Create: Creating/adding/inserting a new record.

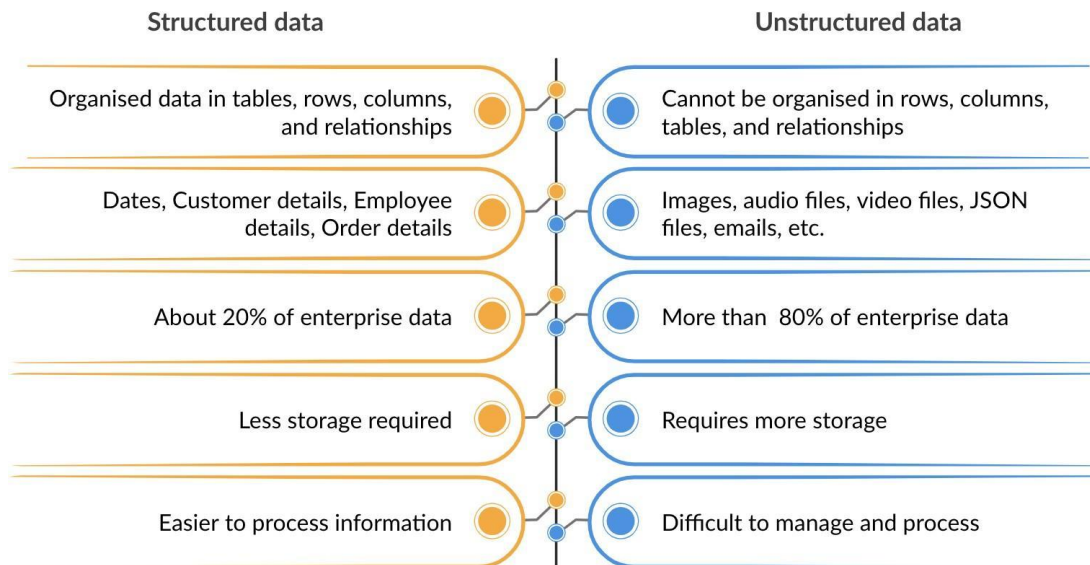
R: Read: Retrieving data.

U: Update: Changing an existing record.

D: Delete: Deleting an existing record.

Types of data

STRUCTURED vs UNSTRUCTURED DATA



Types of databases

Structured data can be stored in relational databases, while unstructured data can be stored in NoSQL databases.

Relational Database	NoSQL Database
<ul style="list-style-type: none">• Data items have pre-defined relationships between them.• Information is stored in structured tables with rows and columns.• Examples: MySQL, IBM DB2, Microsoft SQL Server	<ul style="list-style-type: none">• Uses a storage model optimised for specific requirements of the type of data being stored.• Document, key-value, columnar, graph databases• Examples: MongoDB, Apache Cassandra, Redis, Couchbase, and Apache HBase

Data	Database	Database Management System	Relational Data	Relational Database	Relational Database Management System
Information	Storage of information	Software that helps in performing CRUD operations to data in the database.	Data has relationships	Database that stores structured and relational data -Based on Relational Data Model	Software that helps in performing CRUD operations to structured data in relational databases.

RDBMS belongs to the database management system (DBMS) category. This kind of DBMS comprises a row-based table structure connecting related data elements and the essential functions to maintain data security, accuracy, consistency and integrity.

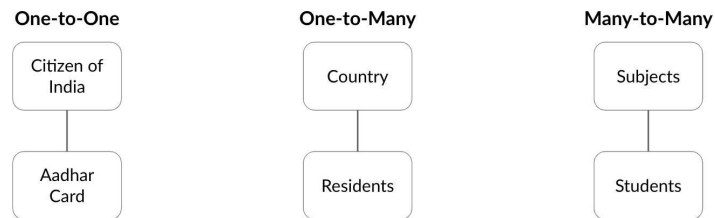
It stores data in tables, where each table is related to the other tables in the database. This kind of model is called a relational model. Hence, it is said that an **RDBMS follows a relational model.**

RDBMS:

- Proposed by EF Codd in 1969
- Model data in the form of relations
- Based on two concepts: Tables
Relations
- Relationships (1:1, 1;n, N:N)

Three main types of relations exist between tables in a relational database.

1. **One-to-One:** as in a citizen of India and his Aadhaar card.
Each citizen will have only one Aadhaar card, and one Aadhaar card cannot be assigned to more than one citizen.
2. **One-to-Many:** as in a country and its residents.
Each country will have many residents, but a resident can't be in more than one country at a time.
3. **Many-to-Many:** as in subjects and students in a class.
Each subject will be studied by many students, and each student will have many subjects to learn.



Scenarios for RDBMS:

- Strict schema
- Relational data
- Transactional requirements
- When data can be stored on a single server

Industrial use cases of RDBMS:

- Banking - Money transaction system
- Money Wallet - Google Pay, PhonePe
- E-commerce - Payments, users, etc.
- Restaurant Listing - Zomato
- Employee Management Software
- Twitter uses MySQL heavily for primary storage of Tweets and Users.
- YouTube uses MySQL

Features of RDBMS

To understand the features of RDBMS, let's consider the following example:

A company has employees and departments with their respective details. Employee details could be their id, name, department, salary, etc. Department details could be department id and department name.

To organise and relate this data, it is stored in a tabular structure, as shown below. A **table** is a collection of data elements organised in terms of rows and columns.

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

Entity: It is any real-world object that is represented in the form of tables in a database. In the table shown above, the entity for the table would be 'EMPLOYEE'. Thus, entities can be used to identify each table.

Relation Building: If you observe the tables, you will notice that the Employee table uses the information from the Department table to show you the department assigned to each employee (EMP_ID). In this way, the Employee and Department tables are related through the Department ID (D_ID).

Tables in a database are related to each other, which can be done using primary and foreign keys.

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

2. MySQL

[MySQL](#) is an open-source RDBMS. It is a very old and popular RDBMS used across industries.

- Free and open-source
- Relational Database Management System
- Interact directly with a MySQL database using SQL
- Wikipedia, Twitter, Flipkart - few of many who use MySQL
- Structured Query Language (SQL)

There are three types of commands within SQL:

1. Data Definition Language - DDL - which is used to work with the schema of the database.
 - CREATE - used to create the database or its objects
 - DROP - used to delete objects from the database.
 - ALTER - used to alter the structure of the database.
 - TRUNCATE - used to remove all records from a table, including all spaces allocated for the records are removed.
 - RENAME - used to rename an object existing in the database.
2. Data Manipulation Language - DML - which is used to modify the data in the database

INSERT – used to insert data into a table.

UPDATE – used to update existing data within a table.

DELETE – used to delete records from a database table.

3. Data Query Language - DQL - which is used to query the data in the database. SELECT – is used to retrieve data from a database.

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: 1. CREATE 2. ALTER 3. DROP	The most common DML commands are as follows: 1. INSERT 2. UPDATE 3. DELETE

There is one more type: Data Control Language DCL, which is used to control access to the data on the database. The commands are GRANT and REVOKE. However, for this course, it is out of scope.

The syntax is as follows:

CREATE	<pre>CREATE DATABASE databasename; CREATE TABLE table_name (column1 datatype, column2 datatype, column3 datatype, );</pre>
ALTER	<pre>ALTER TABLE table_name ADD column_name datatype;</pre>
DROP	<pre>DROP DATABASE databasename; DROP TABLE table_name;</pre>
INSERT	<pre>INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...);</pre>
UPDATE	<pre>UPDATE table name</pre>

	SET column1 = value1, column = value2, ... WHERE condition; 2
DELETE	DELETE FROM table name WHERE condition;

To create a primary key:

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

To create a Foreign Key:

```
CREATE TABLE Orders (
    OrderID int NOT NULL,
    OrderNumber int NOT NULL,
    PersonID int,
    PRIMARY KEY (OrderID),
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)
);
```

To learn more about these commands, please follow the links given below: [Data Definition Language](#)

[Data Manipulation Language](#)

SELECT: Used to read the data
The syntax is :

```
SELECT column1, column2, ...
FROM table_name;
```

The SELECT DISTINCT statement is used to return only distinct (different) values.

WHERE: Used as a conditional statement to filter out data

The syntax is:

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

OPERATORS: Some examples of operators used along with the WHERE clause are 'OR', 'AND', 'IN', 'BETWEEN'.

The following syntax is used :

OR	SELECT column1, column2, ... FROM table_name WHERE condition1 OR condition2 OR condition3 ...;
AND	SELECT column1, column2, ... FROM table_name WHERE condition1 AND condition2 AND condition3 ...;
BETWEEN	SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;
IN	SELECT column_name(s) FROM table_name WHERE column_name IN (value1, value2, ...);

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:

- '%' allows you to check with a string of any length.
- '_' allows you to check with a single character.

Consider the examples shown in the table below.

Example	Description
<pre>SELECT Customer_name FROM cust_dimen WHERE customer_name LIKE 'CL%';</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_IRE GOOD';</pre>	OUTPUT: CLAIRE GOOD Returns those strings that contain a single character between 'CL' and 'IRE GOOD'
<pre>SELECT Customer_name FROM cust_dimen WHERE customer_name LIKE '%IRE%';</pre>	This statement will return all the strings which contain the substring 'IRE'.

To learn more about this, follow the link given below.

[Select Statements](#)

The **aggregate and inbuilt functions** play a crucial role in analysing the data or extracting relevant insights.

The aggregation functions are mentioned below.

COUNT()	Counts the total number of records specified by the given condition
SUM()	Returns the sum of all the non-NULL values
AVG()	Returns the average of all the non-NULL values
MIN()	Returns the minimum of the specified values (NULL is not included)
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 for GROUP BY and HAVING is as follows:

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

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' to specify an ascending order or descending order, respectively. In case nothing is mentioned, the default sort is ascending.

The general syntax for the grouping and ordering functions that you have learned so far is as follows:

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

STRING FUNCTIONS			
Function	Example	Description	Output
UPPER	SELECT UPPER('upgrad')	Converts to uppercase	UPGRAD
LOWER	SELECT LOWER('UPGRAD'),	Convert to lower case	upgrad
SUBSTRNG	SELECT SUBSTRING('upgradeability', 1 6)	SUBSTRING(text, start i ndex. length of substri ng)	upgrad
SUBSTRING_I NDEX	SELECT SUBSTRING_INDEX('www upgr ad com', ' ' 1)	Returns the substring before the mentioned delimiter	www
	SUBSTRING INDEX('www.upgr ad.com', ' ' 2)	SUBSTRING_INDEX(te xt. delimiter. occurence number of delimiter)	www.upgrad
LENGTH	SELECT LENGTH('upgrad'),	Gives the number of characters in the input	5
REVERSE	SELECT REVERSE('upgrad'):	Reverses the string	dargpu

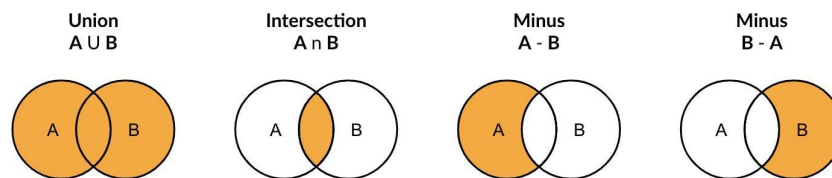
DATE -TIME FUNCTIONS		
Syntax	Description	My output
SELECT C URTI f'zIE()	Returns the current time	17: 11: 12
SELECT L\ONTH('2020 —03—1 D')	Returns the nJ onth of the date	03
SELECT MONTH NAF'zIE('2020-03-10')	Returns the nance of the ns onth of the given date	f'z1arch
SELECT STR TO DATE('1 0.04.2020'. ' o d.âSm . %£Y')	Converts the string to date type	2020-03- 10
SELECT YEAR('2020-03- 10')	Extracts the year from the date	2020
SELECT DAYNAL1E('2020-D3-10')	Returns the day of the v/EEK for the g iven date	Tuesday
SELECT DAYOFL1DNTH('2020 -O3—1 0')	Extracts the day of the nJ onth frDm the given date	10

You can refer to the following links to learn more about these concepts:

1. [String functions](#)
2. [Date-time functions](#)

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

VENN DIAGRAMS
(Refresher)

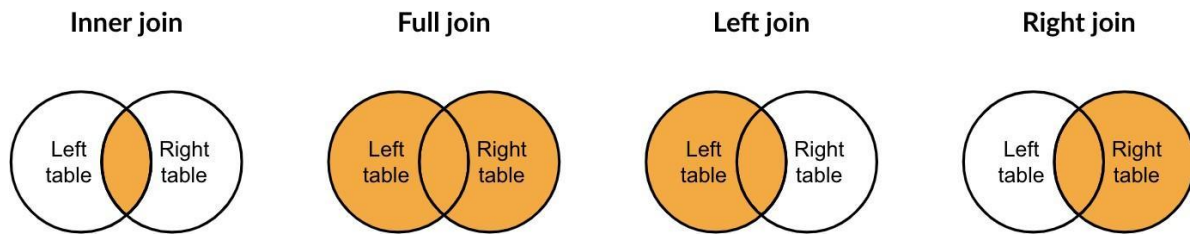


A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

- This is a method of linking data between one or more tables based on the values of the common column between the tables.
- It can be used to combine rows from two or more tables.
- It is used with the SELECT statement.

The different types of JOINS in SQL are listed below.

- **Inner Join:** It combines the rows of two tables that have a matching value or satisfy a given condition.
- **Outer Join / Full Join:** It combines all the rows of two tables regardless of whether matching values exist. If the value in one column does not have any corresponding matching value in another table, the records will be returned in the table with NULL values.
- **Left Join:** It preserves the left table's rows while attaching the rows of the right table whose values in the matching column exist in the corresponding column of the left table.
- **Right Join:** It preserves the rows of the right table while attaching the rows of the left table whose values in the matching column exist in the corresponding column of the right table.

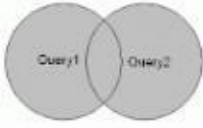

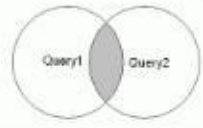
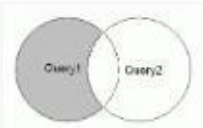


The syntax for different types of Joins are shown in this table.

INNER JOIN/JOIN	<pre>SELECT column_name(s) FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name;</pre>
LEFT JOIN	<pre>SELECT column_name(s) FROM table1 INNER JOIN table2 ON table1.column_name = table2.column_name;</pre>
RIGHT JOIN	<pre>SELECT column_name(s) FROM table1 RIGHT JOIN table2 ON table1.column_name = table2.column_name;</pre>
FULL JOIN	<pre>SELECT column_name(s) FROM table1 FULL OUTER JOIN table2 ON table1.column_name = table2.column_name</pre>

FULL OUTER JOIN and FULL JOIN are the same.

SQL also provides set-based operations.

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

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

When you create a view, there is no separate storage layer for storing this table. One major advantage of creating a view is 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 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.

The syntax for creating a view is as follows:

```
CREATE VIEW view_name
AS SELECT column1, column2,
... FROM table_name
WHERE condition;
```

A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

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 copies 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/unauthorized 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 rights not expressly granted in these terms are reserved.