DATABASE FUNDEMENTALS

Topic 1 : An Overview of database

What Is a Database?

A database is an organized collection of structured information, or data, typically stored electronically in a computer system. A database is usually controlled by a database management system (DBMS). Together, the data and the DBMS, along with the applications that are associated with them, are referred to as a database system, often shortened to just database.

Data within the most common types of databases in operation today is typically modeled in rows and columns in a series of tables to make processing and data querying efficient. The data can then be easily accessed, managed, modified, updated, controlled, and organized. Most databases use structured query language (SQL) for writing and querying data.

What is Structured Query Language (SQL)?

SQL is a programming language used by nearly all relational databases to query, manipulate, and define data, and to provide access control. SQL was first developed at IBM in the 1970s with Oracle as a major contributor, which led to implementation of the SQL ANSI standard, SQL has spurred many extensions from companies such as IBM, Oracle, and Microsoft. Although SQL is still widely used today, new programming languages are beginning to appear.

Evolution of the database

Databases have evolved dramatically since their inception in the early 1960s. Navigational databases such as the hierarchical database (which relied on a tree-like model and allowed only a one-to-many relationship), and the network database (a more flexible model that allowed multiple relationships), were the original systems used to store and manipulate data. Although simple, these early systems were inflexible. In the 1980s, relational databases became popular, followed by object-oriented databases in the 1990s. More recently, NoSQL databases came about

as a response to the growth of the internet and the need for faster speed and processing of unstructured data. Today, cloud databases and self-driving databases are breaking new ground when it comes to how data is collected, stored, managed, and utilized.

Types of databases

There are many different types of databases. The best database for a specific organization depends on how the organization intends to use the data.

- **Relational databases**

Relational databases became dominant in the 1980s. Items in a relational database are organized as a set of tables with columns and rows. Relational database technology provides the most efficient and flexible way to access structured information.

- **Object-oriented databases**

Information in an object-oriented database is represented in the form of objects, as in object-oriented programming.

- **Distributed databases**

A distributed database consists of two or more files located in different sites. The database may be stored on multiple computers, located in the same physical location, or scattered over different networks.

- **Data warehouses**

A central repository for data, a data warehouse is a type of database specifically designed for fast query and analysis.

- **NoSQL databases**

A NoSQL, or nonrelational database, allows unstructured and semistructured data to be stored and manipulated (in contrast to a relational database, which defines how all data inserted into the database must be composed). NoSQL databases grew popular as web applications became more common and more complex.

- **Graph databases**

A graph database stores data in terms of entities and the relationships between entities.

OLTP databases. An OLTP database is a speedy, analytic database designed for large numbers of transactions performed by multiple users.

These are only a few of the several dozen types of databases in use today. Other, less common databases are tailored to very specific scientific, financial, or other functions. In addition to the different database types, changes in technology development approaches and dramatic advances such as the cloud and automation are propelling databases in entirely new directions. Some of the latest databases include

- **Open source databases**

An open source database system is one whose source code is open source; such databases could be SQL or NoSQL databases.

- **Cloud databases**

A cloud database is a collection of data, either structured or unstructured, that resides on a private, public, or hybrid cloud computing platform. There are two types of cloud database models: traditional and database as a service (DBaaS). With DBaaS, administrative tasks and maintenance are performed by a service provider.

- **Multi-model database**

Multi-model databases combine different types of database models into a single, integrated back end. This means they can accommodate various data types.

- **Document/JSON database**

Designed for storing, retrieving, and managing document-oriented information, document databases are a modern way to store data in JSON format rather than rows and columns.

- **Self-driving databases**

The newest and most groundbreaking type of database, self-driving databases (also known as autonomous databases) are cloud-based and use machine learning to automate database tuning, security, backups, updates, and other routine management tasks traditionally performed by database administrators.

What is database software?

Database software is used to create, edit, and maintain database files and records, enabling easier file and record creation, data entry, data editing, updating, and reporting. The software also handles data storage, backup and reporting, multi-access control, and security. Strong database security is especially important today, as data theft becomes more frequent. Database software is sometimes also referred to as a "database management system" (DBMS).

Database software makes data management simpler by enabling users to store data in a structured form and then access it. It typically has a graphical interface to help create and manage the data and, in some cases, users can construct their own databases by using database software.

What is a database management system (DBMS)?

A database typically requires a comprehensive database software program known as a database management system (DBMS). A DBMS serves as an interface between the database

and its end users or programs, allowing users to retrieve, update, and manage how the information is organized and optimized. A DBMS also facilitates oversight and control of databases, enabling a variety of administrative operations such as performance monitoring, tuning, and backup and recovery.

Some examples of popular database software or DBMSs include MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Database, and dBASE.

What is a MySQL database?

MySQL is an open source relational database management system based on SQL. It was designed and optimized for web applications and can run on any platform. As new and different requirements emerged with the internet, MySQL became the platform of choice for web developers and web-based applications. Because it's designed to process millions of queries and thousands of transactions, MySQL is a popular choice for ecommerce businesses that need to manage multiple money transfers. On-demand flexibility is the primary feature of MySQL.

MySQL is the DBMS behind some of the top websites and web-based applications in the world, including Airbnb, Uber, LinkedIn, Facebook, Twitter, and YouTube.

Database challenges

Today's large enterprise databases often support very complex queries and are expected to deliver nearly instant responses to those queries. As a result, database administrators are constantly called upon to employ a wide variety of methods to help improve performance. Some common challenges that they face include:

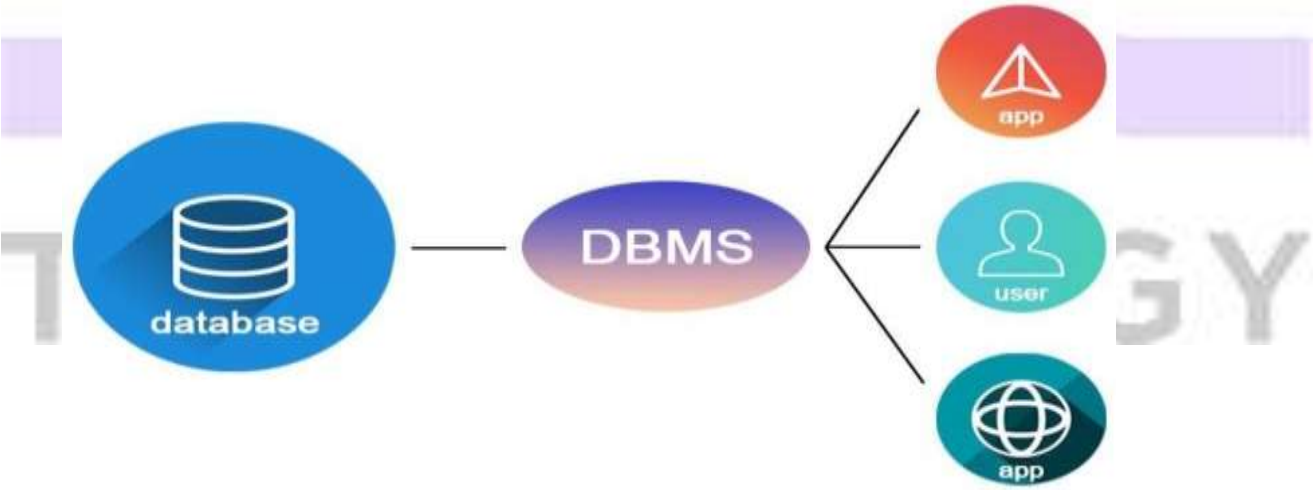
- Absorbing significant increases in data volume. The explosion of data coming infrom sensors, connected machines, and dozens of other sources keeps database administrators scrambling to manage and organize their companies' data efficiently.
- Ensuring data security. Data breaches are happening everywhere these days, and hackers are getting more inventive. It's more important than everto ensure that data is secure but also easily accessible to users.
- Keeping up with demand. In today's fast-moving business environment, companies need real-time access to their data to support timely decision-making and to take advantage of new opportunities.
- Managing and maintaining the database and infrastructure. Database administrators must continually watch the database for problems and perform preventative maintenance, as well as apply software upgrades and patches. As databases become more complex and data volumes grow, companies are faced with the expense of hiring additional talent to monitor and tune their databases.

- Removing limits on scalability. A business needs to grow if it's going to survive, and its data management must grow along with it. But it's very difficult for database administrators to predict how much capacity the company will need, particularly with on- premises databases.

- Ensuring data residency, data sovereignty, or latency requirements. Some organizations have use cases that are better suited to run on-premises. In those cases, engineered systems that are pre-configured and pre-optimized for running the database are ideal. Customers achieve higher availability, greater performance and up to 40% lower cost with Oracle Exadata, according to Wikibon's recent analysis (PDF).

Topic 2: Introduction to DBMS

DBMS is database management system. Databases are the collection of data in order to store and retrieve data. The database consists of data which can be a numeric, alphabetic and also alphanumeric form. Analyzing data is a key feature of database management system that is DBMS. DBMS allows the definition, creation, querying, update, and administration of databases. Language supported and widely used for querying and accessing the database is SQL.



Some of the common terminologies of DBMS are

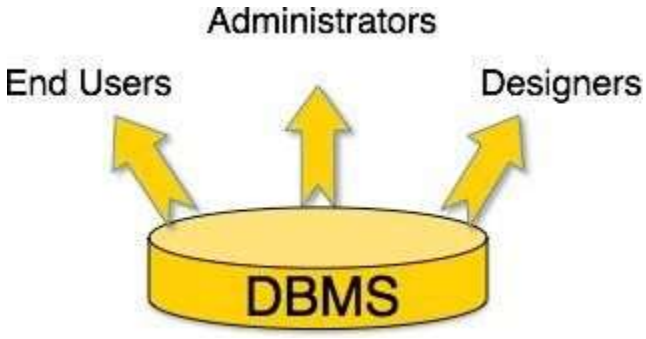
- Tuple: The rows in the database are often known as tuples.
- Table: Table is a collection of tuples and related information along with a key to distinguish the data. Although a table can have duplication of data tuples.
- Schema: Schema is the structure of the relation or a table.
- Data redundancy: Data redundancy ensures there are no multiple occurrences of same data hence avoids data duplication.
- Keys: Keys in a table are used to identify the unique attribute of the table.

A **database management system** stores data in such a way that it becomes easier to retrieve, manipulate, and produce information. Following are the important characteristics and applications of DBMS.

- **ACID Properties** − DBMS follows the concepts of **A**tomicity, **C**onsistency, **I**solation, and **D**urability (normally shortened as ACID). These concepts are applied on transactions, which manipulate data in a database. ACID properties help the database stay healthy in multi-transactional environments and in case of failure.
- **Multiuser and Concurrent Access** − DBMS supports multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when users attempt to handle the same data item, but users are always unaware of them.
- **Multiple views** − DBMS offers multiple views for different users. A user who is in the Sales department will have a different view of database than a person working in the Production department. This feature enables the users to have a concentrate view of the database according to their requirements.
- **Security** − Features like multiple views offer security to some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into the database and retrieving the same at a later stage. DBMS offers many different levels of security features, which enables multiple users to have different views with different features. For example, a user in the Sales department cannot see the data that belongs to the Purchase department. Additionally, it can also be managed how much data of the Sales department should be displayed to the user. Since a DBMS is not saved on the disk as traditional file systems, it is very hard for miscreants to break the code.

USERS

A typical DBMS has users with different rights and permissions who use it for different purposes. Some users retrieve data and some back it up. The users of a DBMS can be broadly categorized as follows −



- **Administrators** − Administrators maintain the DBMS and are responsible for administrating the database. They are responsible to look after its usage and by whom it should be used. They create access profiles for users and apply limitations to maintain

isolation and force security. Administrators also look after DBMS resources like system license, required tools, and other software and hardware related maintenance.

- **Designers** − Designers are the group of people who actually work on the designing part of the database. They keep a close watch on what data should be kept and in what format. They identify and design the whole set of entities, relations, constraints, and views.
- **End Users** − End users are those who actually reap the benefits of having a DBMS. End users can range from simple viewers who pay attention to the logs or market rates to sophisticated users such as business analysts.

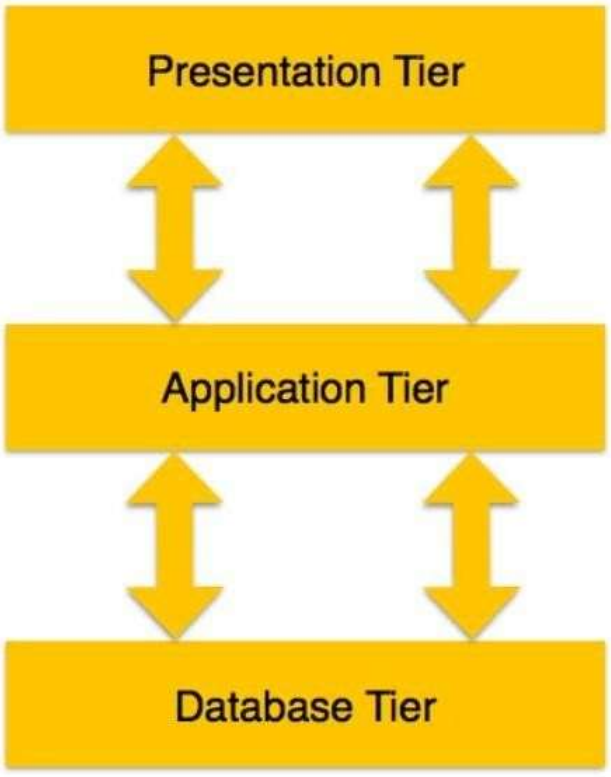
Design of the DBMS

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced.

In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.

If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

Three (3) Tier Architecture: A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



1. **Database (Data) Tier** − At this tier, the database resides along with its queryprocessing languages. We also have the relations that define the data and their constraints at this level.

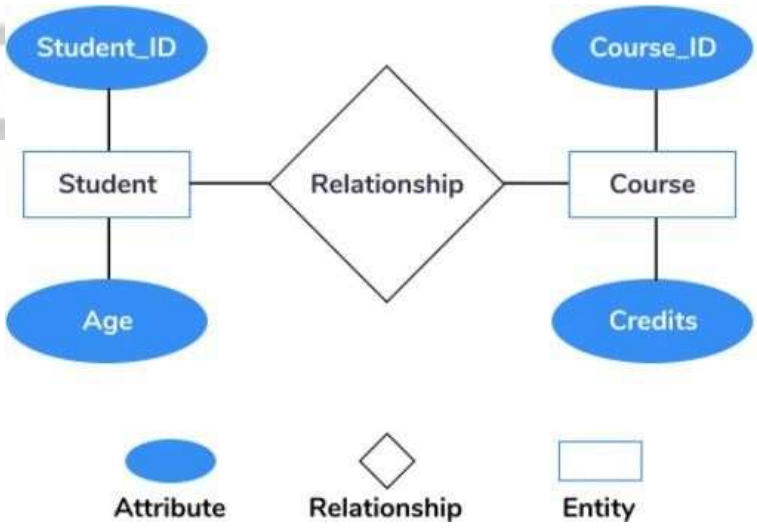
2. **Application (Middle) Tier** − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not awareof any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

3. **User (Presentation) Tier** − End-users operate on this tier and they know nothingabout any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Entity-Relationship (ER) Model is based on the notion of real-world entities and relationships among them. While formulating real-world scenario into the database model, the ER Model creates entity set, relationship set, general attributes and constraints.

ER Model is best used for the conceptual design of a database.
ER Model is based on −

- **Entities** and their *attributes.*
- **Relationships** among entities.



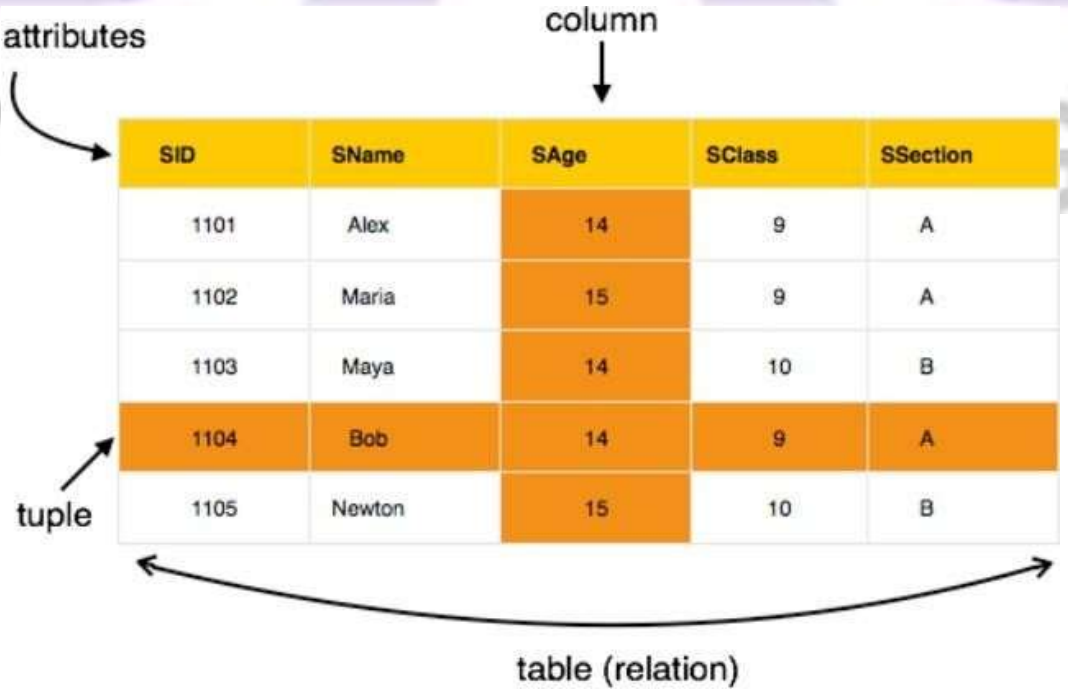
1. **Entity** − An entity in an ER Model is a real-world entity having properties called **attributes**. Every **attribute** is defined by its set of values called **domain**. For example, in a school database, a student is considered as an entity. Student has various attributes like name, age, class, etc.
2. **Relationship** − The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.
    a. Mapping cardinalities −

    - one to one
    - one to many
    - many to one
    - many to many

Relational Model

The most popular data model in DBMS is the Relational Model. It is more scientific a model than others. This model is based on first-order predicate logic and defines a table as an **n-ary relation**.
The main highlights of this model are −

- Data is stored in tables called **relations**.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in a relation contains a unique value.
- Each column in a relation contains values from a same domain.



Topic 3: RDBMS

What is RDBMS?

RDBMS stands for Relational Database Management System. RDBMS is the basis for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access.

What is a table?

The data in an RDBMS is stored in database objects which are called as tables. This table is basically a collection of related data entries and it consists of numerous columns and rows.

Remember, a table is the most common and simplest form of data storage in a relational database. The following program is an example of a CUSTOMERS table –

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kolkata	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

What is a field?

Every table is broken up into smaller entities called fields. The fields in the CUSTOMERS table consist of ID, NAME, AGE, ADDRESS and SALARY.

A field is a column in a table that is designed to maintain specific information about every record in the table.

What is a Record or a Row?

A record is also called as a row of data is each individual entry that exists in a table. For example, there are 7 records in the above CUSTOMERS table.

What is a column?

A column is a vertical entity in a table that contains all information associated with a specific field in a table.

For example, a column in the CUSTOMERS table is ADDRESS, which represents location description and would be as shown below –

```
+-------------+
| ADDRESS     |
+-------------+
| Ahmedabad   |
| Delhi       |
| Kolkata     |
| Mumbai      |
| Bhopal      |
| MP          |
| Indore      |
```

What is a NULL value?

A NULL value in a table is a value in a field that appears to be blank, which means a field with a NULL value is a field with no value.

It is very important to understand that a NULL value is different than a zero value or a field that contains spaces. A field with a NULL value is the one that has been left blank during a record creation.

SQL Constraints

Constraints are the rules enforced on data columns on a table. These are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the database.

Constraints can either be column level or table level. Column level constraints are applied only to one column whereas, table level constraints are applied to the entire table.

Following are some of the most commonly used constraints available in SQL −

1. NOT NULL Constraint − Ensures that a column cannot have a NULL value.
2. DEFAULT Constraint − Provides a default value for a column when none is specified.
3. UNIQUE Constraint − Ensures that all the values in a column are different.
4. PRIMARY Key − Uniquely identifies each row/record in a database table.
5. FOREIGN Key − Uniquely identifies a row/record in any another database table.
6. CHECK Constraint − The CHECK constraint ensures that all values in a column satisfy certain conditions.
7. INDEX − Used to create and retrieve data from the database very quickly.

Data Integrity : The following categories of data integrity exist with each RDBMS −

1. Entity Integrity − There are no duplicate rows in a table.
2. Domain Integrity − Enforces valid entries for a given column by restricting the type,

the format, or the range of values.

3. Referential integrity − Rows cannot be deleted, which are used by otherrecords.

4. User-Defined Integrity − Enforces some specific business rules that do notfall into entity, domain or referential integrity.

Database Normalization

Database normalization is the process of efficiently organizing data in a database. There are two reasons of this normalization process −

- Eliminating redundant data, for example, storing the same data in more than one table.

- Ensuring data dependencies make sense.

Both these reasons are worthy goals as they reduce the amount of space a database consumes and ensures that data is logically stored. Normalization consists of a series of guidelines that help guide you in creating a good database structure.

Normalization guidelines are divided into normal forms; think of a form as the format or the way a database structure is laid out. The aim of normal forms is to organize the database structure, so that it complies with the rules of first normal form, then second normal form and finally the third normal form.

Topic 4: SQL

What is SQL?

SQL stands for Structured Query Language. A query language is a kind of programming language that's designed to facilitate retrieving specific information from databases, and that's exactly what SQL does. To put it simply, SQL is the language of databases.

That matters because most companies store their data in databases. And while there are many types of databases (MySQL, PostgreSQL, Microsoft SQL Server), most of them speak SQL.

Common Uses of SQL on the Web

As a user of any database-driven software program, you're probably using SQL, even if you don't know it. For example, a database-driven dynamic web page (like most websites) takes user input from forms and clicks and uses it to compose a SQL query that retrieves information from the database required to generate the next web page.
Consider the example of a simple online catalog with a search function. The search page might consist of a form containing just a text box in which you enter a search term and then click a search button. When you click the button, the web server retrieves any records from the product database containing the search term and uses the results to create a web page specific to your request.

Data Manipulation Language
The Data Manipulation Language (DML) contains the subset of SQL commands used most frequently — those that simply manipulate the contents of a database in some form. The four most common DML commands retrieve information from a database (the SELECT) command, add new information to a database (the INSERT command), modify information currently stored in a database (the UPDATE command), and remove information from a database (the DELETE command).

Data Definition Language
The Data Definition Language (DDL) contains commands that are less frequently used. DDL commands modify the actual structure of a database, rather than the database's contents. Examples of commonly used DDL commands include those used to generate a new database table (CREATE TABLE), modify the structure of a database table (ALTER TABLE), and delete a database table (DROP TABLE).

Data Control Language
The Data Control Language (DCL) is used to manage user access to databases. It consists of two commands: the GRANT command, used to add database permissions for a user, and the REVOKE command, used to remove existing permissions. These two commands form the core of the relational database security model.

Structure of an SQL Command
Fortunately for those of us who aren't computer programmers, SQL commands are designed to have a syntax similar to the English language. They normally begin with a command statement describing the action to take, followed by a clause that describes the target of the command (such as the specific table within a database affected by the command) and finally, a series of clauses that provide additional instructions.

Often, simply reading an SQL statement out loud will give you a very good idea of what the command is intended to do. Take a moment to read this example of a SQL statement:

```
DELETE
FROM students
WHERE graduation_year = 2020
```

Can you guess what this statement will do? It accesses the student's table of the database and deletes all records for students who graduated in 2020.

SQL is followed by a unique set of rules and guidelines called Syntax. This tutorial gives you a quick start with SQL by listing all the basic SQL Syntax.

All the SQL statements start with any of the keywords like SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE, USE, SHOW and all the statements end with a semicolon (;).

The most important point to be noted here is that SQL is case insensitive, which means SELECT and select have same meaning in SQL statements. Whereas, MySQL makes difference

in table names. So, if you are working with MySQL, then you need to give table names as they exist in the database.

All the examples given in this tutorial have been tested with a MySQL server.

SQL SELECT Statement

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

SQL DISTINCT Clause

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

SQL WHERE Clause

```
SELECT column1, column2
...columnN FROM table_name
WHERE CONDITION;
```

SQL AND/OR Clause

```
SELECT column1, column2
...columnN FROM table_name
WHERE CONDITION-1 {AND|OR} CONDITION-2;
```

SQL IN Clause

```
SELECT column1, column2
...columnN FROM table_name
WHERE column_name IN (val-1, val-2,.. val-N);
```

SQL BETWEEN Clause

```
SELECT column1, column2
...columnN FROM table_name
WHERE column_name BETWEEN val-1 AND val-2;
```

SQL LIKE Clause

```
SELECT column1, column2
...columnN FROM table_name
WHERE column_name LIKE { PATTERN };
```

SQL ORDER BY Clause

```
SELECT column1, column2
...columnN FROM table_name
WHERE CONDITION
ORDER BY column_name {ASC|DESC};
```

SQL GROUP BY Clause

```
SELECT
```

SUM(column_name) FROM
table_name
WHERE CONDITION
GROUP BY column_name;

SQL COUNT Clause

SELECT
COUNT(column_name) FROM
table_name
WHERE CONDITION;

SQL HAVING Clause

SELECT
SUM(column_name) FROM
table_name
WHERE CONDITION
GROUP BY column_name
HAVING (arithematic function condition);

SQL CREATE TABLE

Statement CREATE TABLE
table_name(column1 datatype,
column2 datatype,
column3 datatype,
.....
columnN datatype,
PRIMARY KEY(one or more columns)
);

SQL DROP TABLE Statement

DROP TABLE table_name;

SQL CREATE INDEX Statement

CREATE UNIQUE INDEX index_name
ON table_name (column1, column2,...columnN);

SQL DROP INDEX

Statement ALTER TABLE
table_name DROP INDEX
index_name;

SQL DESC Statement

DESC table_name;

SQL TRUNCATE TABLE Statement

TRUNCATE TABLE table_name;

SQL ALTER TABLE Statement

ALTER TABLE table_name {ADD|DROP|MODIFY} column_name {data_ype};

SQL ALTER TABLE Statement (Rename)
ALTER TABLE table_name RENAME TO new_table_name;

SQL INSERT INTO Statement
INSERT INTO table_name(column1, column2
...columnN) VALUES (value1, value2 ... valueN);

SQL UPDATE Statement
UPDATE table_name
SET column1 = value1, column2 = value2... columnN=valueN
[WHERE CONDITION];

SQL DELETE Statement
DELETE FROM
table_name WHERE
{CONDITION};

SQL CREATE DATABASE Statement
CREATE DATABASE database_name;

SQL DROP DATABASE Statement
DROP DATABASE database_name;

SQL USE Statement
USE database_name;

SQL COMMIT Statement
COMMIT;

SQL ROLLBACK Statement
ROLLBACK;

The following list may be helpful for the beginners to have a nice database performance:

- Use 3BNF database design explained in this tutorial in RDBMS Concepts chapter.
- Avoid number-to-character conversions because numbers and characters compare differently and lead to performance downgrade.
- While using SELECT statement, only fetch whatever information is required and avoid using * in your SELECT queries because it would load the system unnecessarily.
- Create your indexes carefully on all the tables where you have frequent search operations. Avoid index on the tables where you have less number of search operations and more number of insert and update operations.

- A full-table scan occurs when the columns in the WHERE clause do not have an index associated with them. You can avoid a full-table scan by creating an index on columns that are used as conditions in the WHERE clause of an SQL statement.
- Be very careful of equality operators with real numbers and date/time values. Both of these can have small differences that are not obvious to the eye but that make an exact match impossible, thus preventing your queries from ever returning rows.
- Use pattern matching judiciously. LIKE COL% is a valid WHERE condition, reducing the returned set to only those records with data starting with the string COL. However, COL%Y does not further reduce the returned results set since %Y cannot be effectively evaluated. The effort to do the evaluation is too large to be considered. In this case, the COL% is used, but the %Y is thrown away. For the same reason, a leading wildcard %COL effectively prevents the entire filter from being used.
- Fine tune your SQL queries examining the structure of the queries (and subqueries), the SQL syntax, to discover whether you have designed your tables to support fast data manipulation and written the query in an optimum manner, allowing your DBMS to manipulate the data efficiently.
- For queries that are executed on a regular basis, try to use procedures. A procedure is a potentially large group of SQL statements. Procedures are compiled by the database engine and then executed. Unlike an SQL statement, the database engine need not optimize the procedure before it is executed.
- Avoid using the logical operator OR in a query if possible. OR inevitably slows down nearly any query against a table of substantial size.
- You can optimize bulk data loads by dropping indexes. Imagine the history table with many thousands of rows. That history table is also likely to have one or more indexes. When you think of an index, you normally think of faster table access, but in the case of batch loads, you can benefit by dropping the index(es).
- When performing batch transactions, perform COMMIT at after a fair number of records creations instead of creating them after every record creation.
- Plan to defragment the database on a regular basis, even if doing so means developing a weekly routine.

SQL has many built-in functions for performing processing on string or numeric data. Following is the list of all useful SQL built-in functions –

SQL has many built-in functions for performing processing on string or numeric data. Following is the list of all useful SQL built-in functions −

- **SQL COUNT Function** - The SQL COUNT aggregate function is used to count the number of rows in a database table.
- **SQL MAX Function** - The SQL MAX aggregate function allows us to select the highest (maximum) value for a certain column.

- **SQL MIN Function** - The SQL MIN aggregate function allows us to select the lowest (minimum) value for a certain column.
- **SQL AVG Function** - The SQL AVG aggregate function selects the average value for certain table column.
- **SQL SUM Function** - The SQL SUM aggregate function allows selecting the total for a numeric column.
- **SQL SQRT Functions -** This is used to generate a square root of a given number.
- **SQL RAND Function** - This is used to generate a random number using SQL command.
- **SQL CONCAT Function** - This is used to concatenate any string inside any SQL command.
- **SQL Numeric Functions** - Complete list of SQL functions required tomanipulate numbers in SQL.
- **SQL String Functions** - Complete list of SQL functions required to manipulate strings in SQL.

TOPIC 5: NoSQL

A NoSQL (Not Only SQL) database, referring to a "non SQL" or "non-relational" database, provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. NoSQL databases are increasingly used in big data and real-time web applications. NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages, or sit alongside SQL database in a polyglot persistence architecture.

What Is NoSQL

NoSQL is an approach to database design that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats. NoSQL, which stands for "not only SQL," is an alternative to traditional relational databases in which data is placed in tables, and data schema is carefully designed before the database is built. NoSQL databases are especially useful for working with large sets of distributed data.

- NoSQL encompasses a wide variety of different database technologies that were developed in response to the demands presented in building modern applications:
- Developers are working with applications that create massive volumes of new, rapidly changing data types – structured, semi-structured, unstructured and polymorphic data.

- Long gone is the twelve-to-eighteen month waterfall development cycle. Now small teams work in agile sprints, iterating quickly and pushing code every week or two, some even multiple times every day.

- Applications that once served a finite audience are now delivered as services that must be always-on, accessible from many different devices and scaled globally to millions of users.

- Organizations are now turning to scale-out architectures using open software technologies, commodity servers and cloud computing instead of large monolithic servers and storage infrastructure.

- Relational databases were not designed to cope with the scale and agility challenges that face modern applications, nor were they built to take advantage of the commodity storage and processing power available today.

NoSQL vs. RDBMS

The NoSQL term can be applied to some databases that predated the relational database management system, but it more commonly refers to the databases built in the early 2000s for the purpose of large-scale database clustering in cloud and web applications. In these applications, requirements for performance and scalability outweighed the need for the immediate, rigid data consistency that the RDBMS provided to transactional enterprise applications.

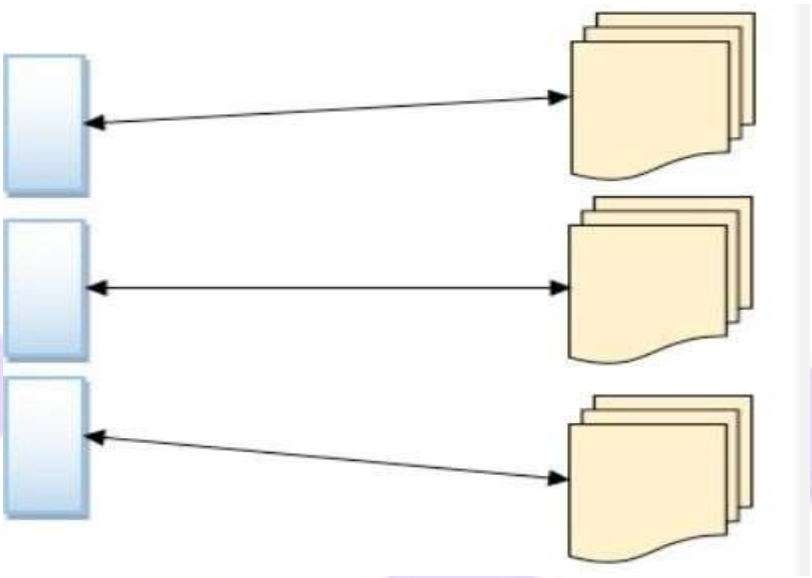
Notably, the NoSQL systems were not required to follow an established relational schema. Large-scale web organizations such as Google and Amazon used NoSQL databases to focus on narrow operational goals and employ relational databases as adjuncts where high-grade data consistency is necessary.

Early NoSQL databases for web and cloud applications tended to focus on very specific characteristics of data management. The ability to process very large volumes of data and quickly distribute that data across computing clusters were desirable traits in web and cloud design.

NoSQL Database Types

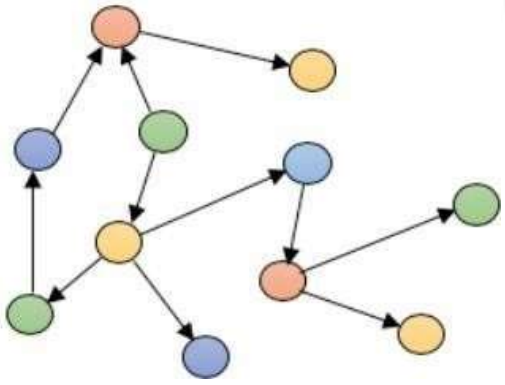
Document databases pair each key with a complex data structure known as a document. Documents can contain many different key-value pairs, or key-array pairs, or even nested documents. Document databases, also called document stores, store semi-structured data and descriptions of that data in document format. They allow developers to create and update programs without needing to reference master schema. Use of document databases has increased along with use of JavaScript and the JavaScript Object Notation (JSON), a data interchange format that has gained wide currency among web application developers, although

XML and other data formats can be used as well. Document databases are used for content management and mobile application data handling. Couchbase Server, CouchDB, DocumentDB, MarkLogic and MongoDB are examples of document databases.



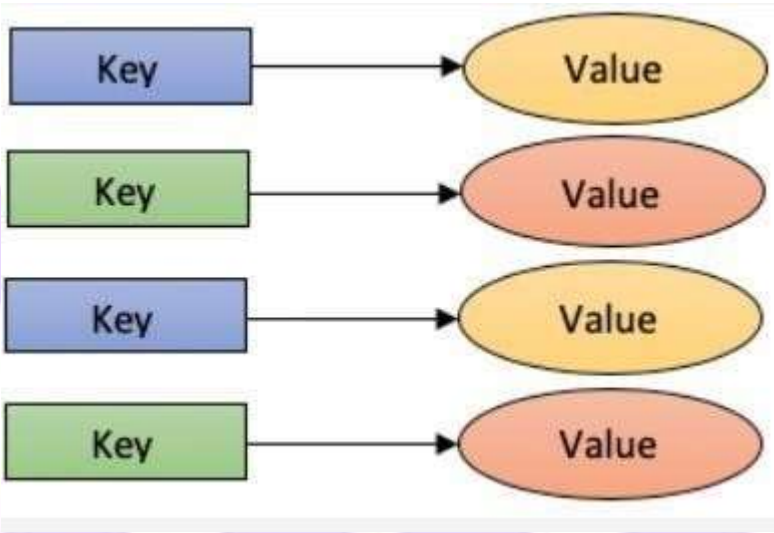
Graph stores

They are used to store information about networks of data, such as social connections. Graph data stores organize data as nodes, which are like records in a relational database, and edges, which represent connections between nodes. Because the graph system stores the relationship between nodes, it can support richer representations of data relationships. Also, unlike relational models reliant on strict schemas, the graph data model can evolve over time and use. Graph databases are applied in systems that must map relationships, such as reservation systems or customer relationship management. Examples of graph databases include AllegroGraph, IBM Graph, Neo4j and Titan.



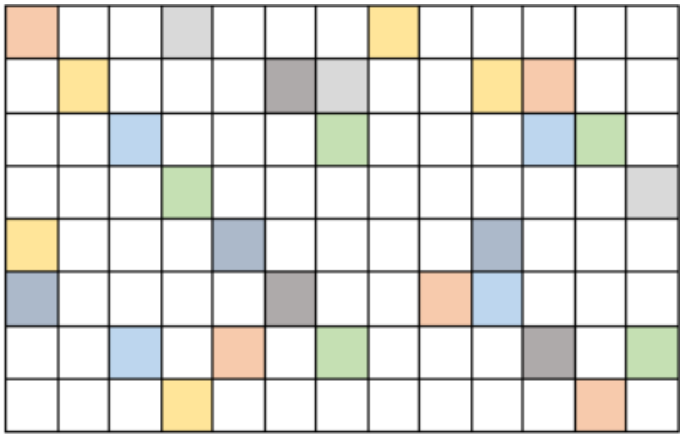
Key-value stores

They are the simplest NoSQL databases. Every single item in the database is stored as an attribute name (or 'key'), together with its value. Key-value stores, or key-value databases, implement a simple data model that pairs a unique key with an associated value. Because this model is simple, it can lead to the development of key-value databases, which are extremely performant and highly scalable for session management and caching in web applications. Implementations differ in the way they are oriented to work with RAM, solid-state drives or disk drives. Examples include Aerospike, Berkeley DB, MemchacheDB, Redis and Riak.



Wide-column stores

Such as Cassandra and HBase are optimized for queries over large datasets, and store columns of data together, instead of rows. Wide-column stores organize data tables as columns instead of as rows. Wide-column stores can be found both in SQL and NoSQL databases.
Wide-column stores can query large data volumes faster than conventional relational databases. A wide-column data store can be used for recommendation engines, catalogs, fraud detection and other types of data processing. Google BigTable, Cassandra and HBase are examples of wide-column stores.



Benefits of NoSQL

When compared to relational databases, NoSQL databases are more scalable and provide superior performance, and their data model addresses several issues that the relational model is not designed to address:

Large volumes of rapidly changing structured, semi-structured, and unstructured data
Agile sprints, quick schema iteration, and frequent code pushes
Object-oriented programming that is easy to use and flexible
Geographically distributed scale-out architecture instead of expensive, monolithic architecture

Types and examples

There are various ways to classify NoSQL databases, with different categories and subcategories, some of which overlap. What follows is a basic classification by data model, with examples:

Types	Examples
Column	Accumulo, Cassandra, Scylla, Apache Druid, HBase, Vertica.
Document	Apache CouchDB, ArangoDB, BaseX, Clusterpoint, Couchbase, Cosmos DB, IBM Domino, MarkLogic, MongoDB, OrientDB, Qizx, RethinkDB
Key-value	Aerospike, Apache Ignite, ArangoDB, Berkeley DB, Couchbase, Dynamo, FoundationDB, InfinityDB, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak, SciDB, SDBM/Flat File dbm, ZooKeeper
Graph	AllegroGraph, ArangoDB, InfiniteGraph, Apache Graph, MarkLogic, Neo4J, OrientDB, Virtuoso

NoSQL vs. SQL Summary

	SQL Databases	NoSQL Databases
Types	One type (SQL database) with minor variations	Many different types including key- value stores, document databases, wide-column stores, and graph databases
Development History	Developed in 1970s to deal with first wave of data storage applications	Developed in late 2000s to deal with limitations of SQL databases, especially scalability, multi-structured data, geo-distribution and agile development sprints
Examples	MySQL, Postgres, Microsoft SQL Server, Oracle Database	MongoDB, Cassandra, HBase, Neo4j

Data Storage Model	Individual records (e.g., 'employees') are stored as rows in tables, with each column storing a specific piece of data about that record (e.g., 'manager,' 'date hired,' etc.), much like a spreadsheet. Related data is stored in separate tables, and then joined together when more complex queries are executed. For example, 'offices' might be stored in one table, and 'employees' in another. When a user wants to find the work address of an employee, the database engine joins the 'employee' and 'office' tables together to get all the information necessary.	Varies based on database type. For example, key-value stores function similarly to SQL databases, but have only two columns ('key' and 'value'), with more complex information sometimes stored as BLOBs within the 'value' columns. Document databases do away with the table-and-row model altogether, storing all relevant data together in single 'document' in JSON, XML, or another format, which can nest values hierarchically.
Schemas	Structure and data types are fixed in advance. To store information about a new data item, the entire database must be altered, during which time the database must be taken offline.	Typically dynamic, with some enforcing data validation rules. Applications can add new fields on the fly, and unlike SQL table rows, dissimilar data can be stored together as necessary. For some databases (e.g., wide-column stores), it is somewhat more challenging to add new fields dynamically.
Scaling	Vertically, meaning a single server must be made increasingly powerful in order to deal with increased demand. It is possible to spread SQL databases over many servers, but significant additional engineering is generally required, and core relational features such as JOINs, referential integrity and transactions are typically lost.	Horizontally, meaning that to add capacity, a database administrator can simply add more commodity servers or cloud instances. The database automatically spreads data across servers as necessary.
Development Model	Mix of open technologies (e.g., Postgres, MySQL) and closed source (e.g., Oracle Database)	Open technologies

Consistency	Can be configured for strong consistency	Depends on product. Some provide strong consistency (e.g., MongoDB, with tuneable consistency for reads) whereas others offer eventual consistency (e.g., Cassandra).

When should I use a NoSQL database instead of a relational database?

Relational databases enforces ACID. So, you will have schema based transaction oriented data stores. It's proven and suitable for 99% of the real world applications. You can practically do anything with relational databases.

But, there are limitations on speed and scaling when it comes to massive high availability data stores. For example, Google and Amazon have terabytes of data stored in big data centers. Querying and inserting is not performant in these scenarios because of the blocking/schema/transaction nature of the RDBMs. That's the reason they have implemented their own databases (actually, key-value stores) for massive performance gain and scalability.

If you need a NoSQL db you usually know about it, possible reasons are:

- client wants 99.999% availability on a high traffic site.
- your data makes no sense in SQL, you find yourself doing multiple JOIN queries for accessing some piece of information.
- you are breaking the relational model, you have CLOBs that store denormalized data and you generate external indexes to search that data.

Cursor and its types:

A cursor in SQL is a temporary work area created in system memory when a SQL statement is executed. A SQL cursor is a set of rows together with a pointer that identifies a current row. It is a database object to retrieve data from a result set one row at a time. It is useful when we want to manipulate the record of a table in a singleton method, in other words, one row at a time. In other words, a cursor can hold more than one row but can process only one row at a time. The set of rows the cursor holds is called the active set.

Types of Cursors in SQL:

There are the following two types of cursors in SQL:

1. Implicit Cursor
2. Explicit Cursor