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.

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

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.

Some of the latest databases include

- Open source databases
- Cloud databases
- Multimodel database
- Document/JSON database

In [ ]:

```
1
```

In [ ]:

```
1
```

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.

In [ ]:

```
1
```

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.
  - Ensuring data security.
  - Keeping up with demand.
  - Managing and maintaining the database and infrastructure.
  - Removing limits on scalability.
  - Ensuring data residency, data sovereignty, or latency requirements.

In [ ]:

```
1
```

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.

MySQL is a widely used relational database management system (RDBMS).

MySQL is free and open-source.

MySQL is ideal for both small and large applications.

In [ ]:

```
1
```

SQL DATABASE vs NOSQL DATABASE

- SQL DB will store the data in tables(row/colums) | NOSQL it will store in JSON/graphs/Key:value
- SQL DB are schema fixed | NOSQL : schema less

In [ ]:

```
1
```

Installation

Please click on the below link for MySql Community Server download

- https://dev.mysql.com/downloads/installer/ (https://dev.mysql.com/downloads/installer/)
- WorkBench
- Shell
- Connectors

Popular commands

- https://www.mysqltutorial.org/mysql-cheat-sheet.aspx (https://www.mysqltutorial.org/mysql-cheat-sheet.aspx)

Datatypes

- https://www.w3schools.com/mysql/mysql_datatypes.asp (https://www.w3schools.com/mysql/mysql_datatypes.asp)

Connecting with Shell

- check the version

  mysql --version
- Open Command Prompt;

  mysql -u root -p

In [ ]:

```
1
```

creating a database

```
CREATE DATABASE databasename;

ex: CREATE DATABASE pyDB;
```

dropping a database

```
DROP DATABASE databasename;

ex: DROP DATABASE pyDB;
```

table creation

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
   ....
);

ex:

     CREATE TABLE person (
        person_id int,
        last_name varchar(255),
        first_name varchar(255),
        address varchar(255),
        city varchar(255)
    );

-- Follow snake_case or camelCase for column-names
-- Follow lowercase for table-names
```

creating table from another existing table

```
CREATE TABLE new_table_name AS
    SELECT column1, column2,...
    FROM existing_table_name
    WHERE ....;

ex:
    CREATE TABLE TestTable AS
    SELECT customername, contactname
    FROM customers;
```

drop a table

```
DROP TABLE table_name;

ex:
    DROP TABLE person;
```

Truncate the table

- The TRUNCATE TABLE statement is used to delete the data inside a table, but not the table itself.

```
    TRUNCATE TABLE table_name;

ex:
    TRUNCATE TABLE TestTable
```

Alter the table

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table.
- The ALTER TABLE statement is also used to add and drop various constraints on an existing table.

```
ALTER TABLE table_name
ADD column_name datatype;

ALTER TABLE table_name
DROP COLUMN column_name;

ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

ex:

```
ALTER TABLE Customers
ADD email varchar(255);

ALTER TABLE Customers
DROP COLUMN email;

ALTER TABLE Persons
ADD DateOfBirth date;

ALTER TABLE Persons
MODIFY COLUMN DateOfBirth year;
```

MySQL Constraints

- SQL constraints are used to specify rules for data in a table.
- Constraints 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 table. If there is any violation between the constraint and the data action, the action is aborted.
- Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.
- Constraints can be specified when the table is created with the CREATE TABLE statement, or after the table is created with the ALTER TABLE statement.

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Prevents actions that would destroy links between tables
- CHECK - Ensures that the values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column if no value is specified.

- CREATE INDEX - Used to create and retrieve data from the database very quickly

NOT NULL

- By default, a column can hold NULL values.
- The NOT NULL constraint enforces a column to NOT accept NULL values.

> This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

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

```
ALTER TABLE Persons
MODIFY Age int NOT NULL;
```

UNIQUE Constraint

- The UNIQUE constraint ensures that all values in a column are different.
- Both the UNIQUE and PRIMARY KEY constraints provide a guarantee for uniqueness for a column or set of columns.
- A PRIMARY KEY constraint automatically has a UNIQUE constraint.

However, we can have many UNIQUE constraints per table, but only one PRIMARY KEY constraint per table.

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

```
!-- To name a UNIQUE constraint, and to define a UNIQUE constraint on multiple columns, use the following SQL syntax:
```

PRIMARY KEY Constraint

- The PRIMARY KEY constraint uniquely identifies each record in a table.
- Primary keys must contain UNIQUE values, and cannot contain NULL values.
- A table can have only ONE primary key; and in the table, this primary key can consist of single or multiple columns (fields).

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

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

```
ALTER TABLE Persons
ADD PRIMARY KEY (ID);

ALTER TABLE Persons
ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);

ALTER TABLE Persons
DROP PRIMARY KEY;
```

CHECK Constraint

- The CHECK constraint is used to limit the value range that can be placed in a column.

- defining a CHECK constraint on a column it will allow only certain values for this column.
- defining a CHECK constraint on a table it can limit the values in certain columns based on values in other columns in the row.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);


CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255),
    CONSTRAINT CHK_Person CHECK (Age>=18 AND City='Sandnes')
);


ALTER TABLE Persons
ADD CHECK (Age>=18);


ALTER TABLE Persons
ADD CONSTRAINT CHK_PersonAge CHECK (Age>=18 AND City='Sandnes');


ALTER TABLE Persons
DROP CHECK CHK_PersonAge;
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

DEFAULT Constraint

- The DEFAULT constraint is used to set a default value for a column.
- The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Sandnes'
);


-- The DEFAULT constraint can also be used to insert system values, by using fu
nctions like CURRENT_DATE():


CREATE TABLE Orders (
    ID int NOT NULL,
    OrderNumber int NOT NULL,
    OrderDate date DEFAULT CURRENT_DATE()
);



-- To create a DEFAULT constraint on the "City" column when the table is alread
y created.


ALTER TABLE Persons
ALTER City SET DEFAULT 'Sandnes';


-- To drop a DEFAULT constraint,


ALTER TABLE Persons
ALTER City DROP DEFAULT;
```

In [ ]:

```
1
```

AUTO INCREMENT Field

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table.
- This is the primary key field that we would like to be created automatically every time a new record is inserted.
- MySQL uses the AUTO_INCREMENT keyword to perform an auto-increment feature.
- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record.

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

-- To let the AUTO_INCREMENT sequence start with another value
```

MySQL Dates

- The most difficult part when working with dates is to be sure that the format of the date you are trying to insert, matches the format of the date column in the database.
- MySQL comes with the following data types for storing a date or a date/time value in the database:

```
DATE - format YYYY-MM-DD
DATETIME - format: YYYY-MM-DD HH:MI:SS
TIMESTAMP - format: YYYY-MM-DD HH:MI:SS
YEAR - format YYYY or YY
```

What is a NULL Value?

- A field with a NULL value is a field with no value.

If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value.

```
Note: A NULL value is different from a zero value or a field that contains spaces. A field with a NULL value is one that has been left blank during record creation!
```

- It is not possible to test for NULL values with comparison operators, such as =, <, or <>.
- We will have to use the IS NULL and IS NOT NULL operators instead.

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

FOREIGN KEY Constraint

- The FOREIGN KEY constraint is used to prevent actions that would destroy links between tables.

- A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the PRIMARY KEY in another table.
- The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

example

Persons Table

PersonID	LastName	FirstName	Age
1	Hansen	Ola	30
2	Svendson	Tove	23
3	Pettersen	Kari	20

Orders Table

OrderID	OrderNumber	PersonID
1	77895	3
2	44678	3
3	22456	2
4	24562	1

Notice that the "PersonID" column in the "Orders" table points to the "PersonID" column in the "Persons" table.

The "PersonID" column in the "Persons" table is the `PRIMARY KEY` in the "Persons" table.

The "PersonID" column in the "Orders" table is a `FOREIGN KEY` in the "Orders" table.

The `FOREIGN KEY` constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

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

-- To allow naming of a FOREIGN KEY constraint, and for defining a FOREIGN KEY
constraint on multiple columns, use the following SQL syntax:
```

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

In [ ]:

```
1
```

In [ ]:

```
1
```

In [ ]:

```
1
```

INSERTION

- The INSERT INTO statement is used to insert new records in a table.
- It is possible to write the INSERT INTO statement in two ways:

1. Specify both the column names and the values to be inserted , and to be inserted specified columsn

```
INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);
```

2. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query

```
INSERT INTO table_name
VALUES (value1, value2, value3, ...);
```

SELECTION

- The SELECT statement is used to select data from a database.
- The data returned is stored in a result table, called the result-set.

```
-- selects specified columns

SELECT column1, column2, ...
FROM table_name;

-- selects all columns

SELECT * FROM table_name;


-- The SELECT DISTINCT statement is used to return only distinct (different) va
lues.
-- Inside a table, a column often contains many duplicate values; and sometimes
wants to list the different (distinct) values.

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

-- To count distinct row count

SELECT COUNT(DISTINCT col1) FROM TABLE;
```

WHERE clause

- The WHERE clause is used to filter records.
- It is used to extract only those records that fulfill a specified condition.

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

Note: The WHERE clause is not only used in SELECT statements, it is also used in UPDATE, DELETE, etc.!

Operators

Operator	Description	E
=	Equal	
>	Greater than	
<	Less than	

AND, OR and NOT Operators

```
The WHERE clause can be combined with AND, OR, and NOT operators.

The AND and OR operators are used to filter records based on more than one cond
ition:

The AND operator displays a record if all the conditions separated by AND are T
RUE.
The OR operator displays a record if any of the conditions separated by OR is T
RUE.
The NOT operator displays a record if the condition(s) is NOT TRUE.
```

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 AND condition2 AND condition3 ...;
```

```
SELECT column1, column2, ...
FROM table_name
WHERE condition1 OR condition2 OR condition3 ...;

SELECT column1, column2, ...
FROM table_name
WHERE NOT condition;
```

ORDER BY Keyword

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- The ORDER BY keyword sorts the records in ascending order by default. To sort the records in descending order, use the DESC keyword.

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1, column2, ... ASC|DESC;
```

LIMIT Clause

- The LIMIT clause is used to specify the number of records to return.
- The LIMIT clause is useful on large tables with thousands of records.
- Returning a large number of records can impact performance.

```
SELECT column_name(s)
FROM table_name
WHERE condition
LIMIT number;
```

MIN() and MAX() Functions

- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;

---------------------------

SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

COUNT(), AVG() and SUM() Functions

- The COUNT() function returns the number of rows that matches a specified criterion.

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

- The AVG() function returns the average value of a numeric column.

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

- The SUM() function returns the total sum of a numeric column.

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator:
  - The percent sign (%) represents zero, one, or multiple characters
  - The underscore sign (_) represents one, single character
- The percent sign and the underscore can also be used in combinations!

```
SELECT column1, column2, ...
FROM table_name
WHERE columnN LIKE pattern;
```

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

IN Operator

- The IN operator allows you to specify multiple values in a WHERE clause.
- The IN operator is a shorthand for multiple OR conditions.

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);

SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

BETWEEN Operator

- The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
- The BETWEEN operator is inclusive: begin and end values are included.

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

MySQL Aliases

- Aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.

```
SELECT column_name AS alias_name
FROM table_name;


SELECT column_name(s)
FROM table_name AS alias_name;
```

MySQL Comments

- Comments are used to explain sections of SQL statements, or to prevent execution of SQL statements.

Single line comments start with --.

- Any text between -- and the end of the line will be ignored (will not be executed).

```
-- Select all:
SELECT * FROM Customers;
```

In [ ]:

```
1
```