INDEX  
  
DAY-1: 27.2

He explained about the things like what is database(memory card or sim) and dbms(phone) sql acts as a programming language to get data from the database it is called as query. we are mainly going to concentrate on relational database as it deals with tables with relationship etc. statements or commands in sql or ddl dml dcl tcl and dql.  
data definition language is used to define the structure of the table, with the commands like create, drop, alter, truncate  
------------------------------------------  
create table students(  
  name varchar(100),   -- it is used to create table  
  age int,  
  department (50))

drop table student -- it deletes entire table

alter table student add city varchar(100)  -- it is used to modify the stucture of the table like rename, delete column etc

Truncate deletes the data in the table by keeping its structure like header

Data manipulation language: it is used to modify the structure in the database the commands are update, delete, insert

update student  
set name = 'kia'  
where age = 23

delete from student where name ='Thanu'

insert into student values('Thara', 23, 'EEE')

sql is a std database language which is used to perform operations like crud in the data from the relational dbms like mysql, postgresql, oracle and sql server.

--learning starts--

Sql: Structured Query language which is used to interact with the database in the database management systems with the help of crud(create,read,update,delete) operations and also includes manage, fetching query, inserting or removing data from the table.  
sql keywords are not case sensitive.

Database: It is the structure form of storing collection of similar data in an organized way

Database management system: It is a software where we can access or manage the database with the help of sql queries

Relational dbms: It is used to manage or access the relational database, where the relational database is organized in structure of table with rows and columns where the tables are connected with the logical relationships within them. There are many relational database management system like mysql, sqlite, postgresql, Microsoft sql server and oracle.

RDBMS uses [SQL queries](https://www.w3schools.com/sql/default.asp) to access the data in the database.

Eg customer and order table with pk and foreign key relationship

Applications of sql:  
 we can create, delete, update data in the tables; modify, access, manipulate data within the table  
 retrieve, or summarize necessary info from table or several tables

5 commands or subsets of sql are DDL,DML,DCL,DQL,TCL

**Data Definition Language**: It is used to define the structure of table with the commands like create, alter, drop, truncate where alter is used to modify the structure of the table since it comes under ddl.

CREATE TABLE Students(  
 student\_id int,  
 name varchar(50),  
 city varchar(50)  
)

CREATE DATABASE dbname; --CREATE DATABASE EMP;(ddl command)  
within in the database we can create table and perform other actions  
SHOW DATABASES; -- it is a command which is used to view what are the databases available  
to fix with at which data we are available we need to use   
USE database\_name within that we can start creating table

DROP DATABASE EMP;

ALTER TABLE Students Add rank int; -- alter table tablename add columnname datatype;  
ALTER TABLE Students rename name to student\_name  
ALTER TABLE Students drop city;  
ALTER TABLE Students modify column city varchar(100)

DROP table Students

Truncate table Students

**Data Manipulation Language:** insert, update, delete merge

insert into Students(student\_id, name, city)  
values(101,’Anusha’,’Cbe’)

INSERT INTO Customers (CustomerName, City, Country)  
SELECT SupplierName, City, Country FROM Suppliers  
WHERE Country='Germany';

update students  
set name =’Anaka’  
where student\_id =101;  
update Students  
set city = Null  
where student\_id =101;

delete from Students  
where student\_id =101;

**DQL** -select

**TCL** – Transaction control language: COMMIT, SET TRANSACTION, ROLLBACK, and SAVEPOINT  
commit – it saves all the changes in the transactions permanently  
begin   
insert statements ---  
end  
commit;

Savepoint Savepoint\_name – we can divide the transactions like insert statements, delete statements  
the SAVEPOINT command in SQL, we can save these different parts of the same transaction using different names. For example, we can save all the insert related queries with the savepoint named INS. To save all the insert related queries in one savepoint, we have to execute the SAVEPOINT query followed by the savepoint name after finishing the insert command execution  
**Savepoint INS  
Rollback to INS**

Rollback: While carrying a transaction, we must create savepoints to save different parts of the transaction. According to the user's changing requirements, he/she can roll back the transaction to different savepoints. Consider a scenario: We have initiated a transaction followed by the table creation and record insertion into the table. After inserting records, we have created a savepoint INS. Then we executed a delete query, but later we thought that mistakenly we had removed the useful record. Therefore in such situations, we have an option of rolling back our transaction. In this case, we have to roll back our transaction using the ROLLBACK command to the savepoint INS, which we have created before executing the DELETE query.

DCL – Data Control Language – Grant, revoke  
GRANT privilege\_name on objectname to user;-- GRANT SELECT, UPDATE ON employees TO Bhanu  
 REVOKE privilege\_name on objectname from user; --REVOKE SELECT, UPDATE ON employees TO Bhanu

--- commands over ----------------------------------

---QUERIES  
basic:  
SELECT \* FROM Students  
WHERE student\_id =101 and class = 10

SELECT \* From Students   
WHERE name =’Harshan’ or class= 10;

SELECT \* FROM Students  
WHERE NOT city =’cbe’ and NOT country =’Germany’

SELECT \* from student  
where class =10  
order by name desc; (by default order by will be asc if we desc we need to mention desc init)

SELECT \* FROM STUDENT  
where student\_id is null (it is neither empty nor zero) we should use student = null, because NULL doesn't equate to any value, not even itself. And null takes least value

SELECT \* FROM Customers  
LIMIT 3 OFFSET 3; (it starts from 4th record and provide only 3 values from that when we use simply limit it provides 3values)

Day2: 28.2  
**Single row functions:** max,min,avg,count,sum  
Select max/min(height) from students  
where class=10;

Select \* from Students  
where student\_id like ‘1%’  
select \* from students   
where student\_name like ‘B\_n%’ (% denotes to wildcard function)  
  
select distinct empname from employee  
where lower(empname) similar to ‘[aeiou]%’

select distinct empname from employee  
where lower(empname) regexp ‘^[aeiou]’  
SELECT DISTINCT(CITY) FROM STATION WHERE CITY NOT REGEXP '^[aeiou]' AND CITY NOT REGEXP '[aeiou]$';  
  
select \* from patients  
where diseases in (‘fever’,’cold’,’cough’)  
select \* from patients   
where roll\_no between 17 and 40  
select \* from students  
where birth\_date between ’21-10-2020’ to ’21-11-2020’  
  
select name as student\_name from Students (here as is used for aliasing column and also can be used for table name too)

Select s.roll\_no, s.name, t.name as teacher\_name from students s  
inner join/left join/ right join teacher t  
on s.school\_name = t.school\_name  
  
inner join : it provides only matching records from both the table  
left join: it provides matched record from right table and unmatched record from left table  
right join: it provides matched record from left table and unmatched record from right table  
cross join: it is also known as cartesian join it provides possible combination of records from the table  
for example if we table like color with blue, red, black and cars with kia, bmw and benz  
it provides output as 9  
self join: it is used to join within the table itself  
  
union: it will combine the table and provide the matching values with distinct  
SELECT City FROM Customers

UNION

SELECT City FROM Suppliers  
ORDER BY City;  
here it will provide the distinct values  
whereas if we use union all it allows duplicate values

-----------DAY-0------------ 29.2  
select class, count(\*) from Student  
group by class  
it will give count of each standard in the school. Here we can also give column name inside the count function. When we give column name it will not include null value and if we give \* it will also provide null value  
  
select country, count(city) from world  
where country like ‘A%’  
group by country  
having count(city) >10  
order by count(city) desc

This is the order of execution in SQL

------------**EXISTS**----------------------

SELECT SupplierName  
FROM Suppliers  
WHERE EXISTS (SELECT ProductName FROM Products WHERE Products.SupplierID = Suppliers.supplierID AND Price = 22);

In the above query if there is any value exists in the subquery then it will display the supplier name list

SELECT ProductName  
FROM Products  
WHERE ProductID = ANY  
 (SELECT ProductID  
 FROM OrderDetails  
WHERE Quantity = 10);  
it is provide the product name if any of the order details table have quantity of equal to 10

SELECT ProductName  
FROM Products  
WHERE ProductID = ALL  
 (SELECT ProductID  
 FROM OrderDetails  
WHERE Quantity = 10);  
it will check whether all the records in the order table will have the quantity of 10 else it will provide output as false and the output will be it will show no record found

Case Statements:  
SELECT OrderID, Quantity,  
CASE  
 WHEN Quantity > 30 THEN 'The quantity is greater than 30'  
 WHEN Quantity = 30 THEN 'The quantity is 30'  
 ELSE 'The quantity is under 30'  
END AS QuantityText  
FROM OrderDetails;

In the above query quantity text also act as a column.

SELECT CustomerName, City, Country  
FROM Customers  
ORDER BY  
(CASE  
    WHEN City IS NULL THEN Country  
    ELSE City  
END);  
check the above query for explanation

--MARCH -1---------- DAY1  
  
SELECT ProductName, UnitPrice \* (UnitsInStock + COALESCE(UnitsOnOrder, 0))  
FROM Products;

SELECT ProductName, UnitPrice \* (UnitsInStock + IFNULL(UnitsOnOrder, 0))  
FROM Products;

Nvl used in oracle, IFNULL used in mysql both are synonmy and this is used for only two arguments.  
Coalesce it will provide first nonnull output in the statement.

Comments: -- it is used for single line and /\* it is used for multiple lines \*/

--Constraints:  
SQL constraints are used to specify rules for data in a table.  
Constrains are null, unique, primary Key, foreign key, check, default

CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    CONSTRAINT UC\_Person UNIQUE (ID,LastName) – unique constrain can be used for multiple column, for multiple column we need to use this syntax.  
 UNIQUE(ID) --syntax for single column  
);

In a table unique can be used in multiple column, whereas primary key need to used for only one column in a table

To drop unique key in a table  
ALTER TABLE Persons  
DROP INDEX UC\_Person;

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,  
    CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName) –multiple column or PRIMARY KEY(ID) –single column  
);

ALTER TABLE Persons  
DROP PRIMARY KEY;

ALTER TABLE Persons  
ADD CONSTRAINT PK\_Person PRIMARY KEY (ID,LastName);

A FOREIGN KEY is a field (or collection of fields) in one table, that refers to the [PRIMARY KEY](https://www.w3schools.com/MySQL/mysql_primarykey.asp) 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.

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);  
Check constrains  
CREATE TABLE Persons (  
    ID int NOT NULL,  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255),

CHECK (Age>=18 ) -- for single column  
    CONSTRAINT CHK\_Person CHECK (Age>=18 AND City='Sandnes') –- for multiple columns  
);  
Default:  
 it is a constrain which is used to set default values, where if new record is inserted in a table without any value is provided then default value is assigned for the particular column.

CREATE TABLE Persons (  
    ID int NOT NULL **AUTO\_INCREMENT**  
    LastName varchar(255) NOT NULL,  
    FirstName varchar(255),  
    Age int,  
    City varchar(255) DEFAULT 'Sandnes',  
 OrderDate date DEFAULT CURRENT\_DATE() we can also declare default value as function  
);  
for auto increment we can set a value from where to start by using alter statement.

INDEX:

Updating a table with indexes takes more time than updating a table without (because the indexes also need an update). So, only create indexes on columns that will be frequently searched against.

CREATE INDEX idx\_pname  
ON Persons (LastName, FirstName);  
-- create index index\_name  
on table\_name column , for multiple columns comma separated

VIEW:   
a view is a virtual table based on the result-set of an SQL statement.  
A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database.  
You can add SQL statements and functions to a view and present the data as if the data were coming from one single table.

CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName, Price  
FROM Products  
WHERE Price > (SELECT AVG(Price) FROM Products);

SELECT \* FROM [Products Above Average Price];

For updating view statement we can use create or replace view statement to add any column created in the table.

MY SQL STRING FUNCTIONS:  
lower,upper it converts the string case  
select upper(name) from emp;

Concat it joins two or more expressions  
SELECT CONCAT("SQL ", "Tutorial ", "is ", "fun!") AS ConcatenatedString; or we can also mention the column name.

INSTR: The INSTR() function returns the position of the first occurrence of a string in another string.  
SELECT INSTR("W3Schools.com", "3") AS MatchPosition; and provides output as 2

SELECT LEFT("SQL Tutorial", 3) AS ExtractString; extracts output from left and provides output from right.

SELECT name,length(name) from emp; it provides length of characters from the name

SELECT OrderID, Quantity, IF(Quantity>10, "MORE", "LESS")  
FROM OrderDetails;

SELECT IFNULL(NULL, "W3Schools.com");

SELECT DATEDIFF("2017-06-25", "2017-06-15"); it provides difference between the date

SELECT SUBSTR("SQL Tutorial", 5, 3) AS ExtractString;

SELECT REPLACE("SQL Tutorial", "SQL", "HTML");

Numeric functions: floor, ceil, round, trunc, mod, div and abs(it provides output irrespective of any sign)

SELECT ROUND(POWER(close, 2),2) AS squared\_close;

DATE FUNCTIONS:

CURRENT\_DATE, CURRENT\_TIME and CURRENT TIMESTAMP(it has both date and time) to return current date and time.  
EXTRACT() and DATE\_PART() to extract specific components of date.  
DATE\_TRUNC() to round down date or timestamp into specific level of precision.  
INTERVAL to add or subtract time intervals in calculations.  
TO\_CHAR() to convert date or timestamp into strings.  
::DATE, TO\_DATE(), ::TIMESTAMP, and TO\_TIMESTAMP() to convert strings into date or timestamp.

SELECT   
 message\_id,   
 sent\_date,  
 EXTRACT(YEAR FROM sent\_date) AS extracted\_year,  
 DATE\_PART('year', sent\_date) AS part\_year  
from Order.

Similarly we can use for month, hour, day, minute in place of year in the above query

SELECT user\_id, MAX(post\_date::DATE) FROM posts  
GROUP BY user\_id; --to get recent date

SELECT

message\_id,

sent\_date,

sent\_date + INTERVAL '2 days' AS add\_2days,

sent\_date - INTERVAL '3 days' AS minus\_3days,

sent\_date + INTERVAL '2 hours' AS add\_2hours,

sent\_date - INTERVAL '10 minutes' AS minus\_10mins

FROM messages

LIMIT 3;  
adding or subtracting the time intervals as we required.

SELECT

message\_id,

sent\_date,

TO\_CHAR(sent\_date, 'YYYY-MM-DD HH:MI:SS') AS formatted\_iso8601,

TO\_CHAR(sent\_date, 'YYYY-MM-DD HH:MI:SS AM') AS formatted\_12hr,

TO\_CHAR(sent\_date, 'Month DDth, YYYY') AS formatted\_longmonth,

TO\_CHAR(sent\_date, 'Mon DD, YYYY') AS formatted\_shortmonth,

TO\_CHAR(sent\_date, 'DD Month YYYY') AS formatted\_daymonthyear,

TO\_CHAR(sent\_date, 'Month') AS formatted\_dayofmonth,

TO\_CHAR(sent\_date, 'Day') AS formatted\_dayofweek

FROM messages

LIMIT 3;

TO\_CHAR() converts a date or timestamp to a string with a specified format.

Cast(),:: it is used for conversion of datatypes  
for advanced refer data lemur and make note of it.

MARCH -3

CTE- Common Expression Table is a query within the query which is created with ‘with’ clause. It is a temporary table created to store results and it makes complex queries readable and maintainable.This query will exist only the duration of main query.  
Advantages:  
 Breakdown complex queries,  
 Reusing subquery results,  
 Recursive queries

Subquery: It is the query within the query, by nesting query within the parantheses we can create temporary table to perform calculations or to filter data within the query.

Advantages: Single-Value Comparison in WHERE Clauses  
 Column Creation and aggregation  
 Correlated queries  
SELECT

artist\_name,

genre,

concert\_revenue,

(SELECT AVG(concert\_revenue) FROM concerts) AS avg\_concert\_revenue,

(SELECT MAX(concert\_revenue) FROM concerts) AS max\_concert\_revenue

FROM concerts;

The above query create new column for enchancing the data analysis

In correlated query we can get output from inner query with that list we can get the required output from the outer query.

Window Function:  
1. Window fn with the aggregate fn like sum, avg, max, min etc  
SELECT

spend,

SUM(spend) OVER (

PARTITION BY product

ORDER BY transaction\_date) AS running\_total

FROM product\_spend;

SUM(): SUM(spend) is a typical aggregate function

OVER: OVER required for window functions

PARTITION BY: makes each product it's own section / window,

ORDER BY: the data is ordered by transaction\_date, and the running\_total accumulates the sum across the current row and all subsequent rows of spend

ORDER BY: ORDER BY essentially sorts the data by the specified column, similar to an ORDER BY clause.

Without ORDER BY, each value would be a sum of all the spend values without its respective product.  
SELECT

category,

product,

spend,

MAX(spend) OVER (

PARTITION BY product) AS max\_product\_spend

FROM product\_spend;

It provides the output of maximum amount spent within the partition window.

SELECT

category,

product,

user\_id,

spend,

transaction\_date,

FIRST\_VALUE(product) OVER (

ORDER BY transaction\_date) AS first\_purchase,

LAST\_VALUE(product) OVER (

ORDER BY transaction\_date) AS last\_purchase

FROM product\_spend;

It provides the first value of the purchases and last value of the purchases.

SELECT

RANK() / DENSE\_RANK() / ROW\_NUMBER() OVER ( -- Compulsory expression

PARTITION BY partitioning\_expression -- Optional expression

ORDER BY order\_expression) -- Compulsory expression

FROM table\_name;

Dense\_rank function does not skip the rank e.g if two people secure same marks it comes in the order that 1,2,2,3  
whereas in rank() it gives 1,2,2,4

Row\_number() it is a function which gives sequential number for the rows.

LEAD() and LAG() are time-series window functions used to access data from rows that come after, or before the current row within a result set based on a specific column order. We can also provide offset value.  
Think of LEAD() as a function that lets you peek into the future 🔮, and LAG() as a way to glance into the past ⏪. They're like time-travel functions for your data! We frequently use this to create a 7-day moving average metric, or a 28-day rolling count metric.

SELECT

date,

close,

LEAD(close, offset) OVER (ORDER BY date) AS next\_month\_close,

LAG(close, offset) OVER (ORDER BY date) AS prev\_month\_close

FROM stock\_prices

WHERE EXTRACT(YEAR FROM date) = 2023  
 AND ticker = 'GOOG';

Lag is used to check previous marks scored by the students.

SELECT

b1.genre,

b1.book\_title AS current\_book,

b2.book\_title AS suggested\_book\_1,

b3.book\_title AS suggested\_book\_2

FROM goodreads AS b1

INNER JOIN goodreads AS b2

ON b1.genre = b2.genre

INNER JOIN goodreads as b3

ON b1.genre = b3.genre

WHERE b1.book\_id != b2.book\_id

AND b1.book\_id != b3.book\_id

AND b2.book\_id != b3.book\_id

ORDER BY b1.book\_title,

LIMIT 50;

It will provide book recommendation with one table the current book should not match with the suggesting books so in where clause we have mentioned book id != with other book id.

Union/Union All:

SELECT ingredient

FROM recipe\_1

UNION /UNION ALL/ INTERSECT

SELECT ingredient

FROM recipe\_2;

Lets take example of ingredients list, in union it will provide values present in both the table with out any duplicate  
for eg ingredient1: rice, floor, sugar, salt and Ingredient2: floor, salt, jaggery, milk

For union o/p as: rice, floor, sugar, salt, jaggery, milk  
whereas in union all it includes the above list with floor, salt which duplicates the value.  
in intersect it gives common value: floor, salt  
when it comes with EXCEPT: the ingredients in Recipe 1 that are not in recipe 2 are: Sugar.

**Pivoting** involves rotating a table by converting unique values from a single column into multiple columns. This rearrangement **turns rows into column values**, often involving aggregations on remaining columns.

SELECT

superhero\_alias,

MAX(CASE WHEN platform = 'Instagram' THEN engagement\_rate END) AS instagram\_engagement\_rate,

MAX(CASE WHEN platform = 'Twitter' THEN engagement\_rate END) AS twitter\_engagement\_rate,

MAX(CASE WHEN platform = 'TikTok' THEN engagement\_rate END) AS tiktok\_engagement\_rate,

MAX(CASE WHEN platform = 'YouTube' THEN engagement\_rate END) AS youtube\_engagement\_rate

FROM marvel\_avengers

WHERE superhero\_alias IN ('Iron Man', 'Captain America', 'Black Widow', 'Thor')

GROUP BY superhero\_alias

ORDER BY superhero\_alias;

String Functions:

SELECT

SUBSTRING('Spider-Man', 1, 6) AS substring\_1, -- Using positive index

SUBSTRING('Black Widow', 7) AS substring\_2, -- Using positive index

SUBSTRING('Spider-Man', -1) AS substring\_3, -- Using negative index

SUBSTRING('Black Widow', -2, 3) AS substring\_4 -- Using negative index

FROM marvel\_avengers;

| **substring\_1** | **substring\_2** | **substring\_3** | **substring\_4** |
| --- | --- | --- | --- |
| Spider | Widow | n | dow |
|  |  |  |  |

SELECT

TRIM(' Spiderman') AS full\_trim,

LTRIM('Iron Man', 'Iron ') AS left\_trim,

RTRIM('Scarlet Witch', ' Witch') AS right\_trim,

BTRIM(' Falcon ', ' ') AS combination\_trim1,

BTRIM('...Iron Man...', '.') AS combination\_trim2

FROM marvel\_avengers;

BTRIM trims both the sides, whereas ltrim and rtrim cuts the corresponding sides.

Slicing and Dicing using SUBSTRING():

SUBSTRING(string, start\_position [mandatory], length [optional]) SELECT

SUBSTRING('Spider-Man', 1, 6) AS substring\_1, -- Using positive index

SUBSTRING('Black Widow', 7) AS substring\_2, -- Using positive index

SUBSTRING('Spider-Man', -1) AS substring\_3, -- Using negative index

SUBSTRING('Black Widow', -2, 3) AS substring\_4 -- Using negative index

FROM marvel\_avengers;