SQL

Database 

Database is collection of data in a format that can be easily accessed (Digital)

A software application used to manage our DB is called DBMS (Database Management System)

Types of Databases

Relational

Data stored in tables

Non-relational (NoSQL)

data not stored in tables

** We use SQL to work with relational DBMS

All the RDBMS like MySQL, Informix, Oracle, MS Access and SQL Server use SQL as their standard database language. • SQL allows users to query the database in a number of ways, using English-like statements

What is SQL?

Structured Query Language

SQL is a programming language used to interact with relational databases.

It is used to perform **CRUD** operations :

Create

Read

Update

Delete

SQL follows the following rules:

- *Structure query language is not case sensitive. Generally, keywords of SQL are written in uppercase.*
- *Statements of SQL are dependent on text lines. We can use a single SQL statement on one or multiple text line.*
- *Using the SQL statements, you can perform most of the actions in a database.*
- *SQL depends on tuple relational calculus and relational algebra.*

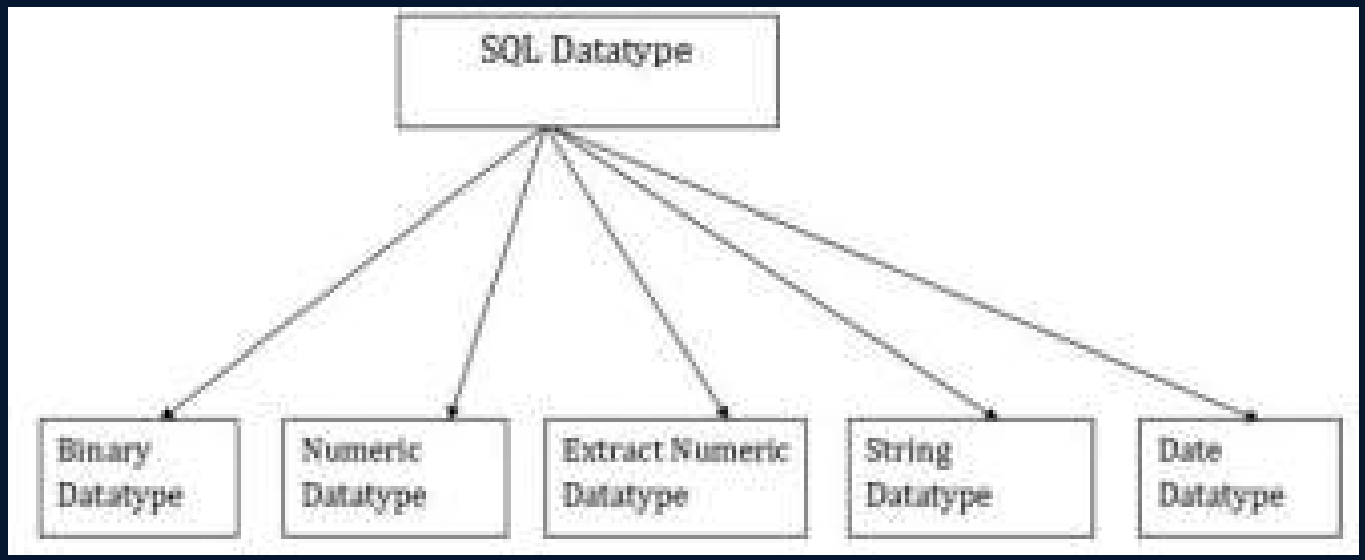
What is Advantages of SQL?

- *High speed*
- *No coding needed*
- *Well defined standards*
- *Portability*
- *Interactive language*
- *Multiple data view*

What is SQL Datatype?

• SQL Datatype is used to define the values that a column can contain.

• Every column is required to have a name and data type in the database table



Exact Numeric Data Types

DATA TYPE	FROM	TO
bigint	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
int	-2,147,483,648	2,147,483,647
smallint	-32,768	32,767
tinyint	0	255
bit	0	1
decimal	-10^38 +1	10^38 -1
numeric	-10^38 +1	10^38 -1
money	-922,337,203,685,477.5808	+922,337,203,685,477.5807
smallmoney	-214,748.3648	+214,748.3647

Approximate Numeric Data Types

DATA TYPE	FROM	TO
float	-1.79E + 308	1.79E + 308
real	-3.40E + 38	3.40E + 38

Date and Time Data Types

DATA TYPE	FROM	TO
datetime	Jan 1, 1753	Dec 31, 9999
smalldatetime	Jan 1, 1900	Jun 6, 2079
date	Stores a date like June 30, 1991	
time	Stores a time of day like 12:30 P.M.	

Character Strings Data Types

DATA TYPE	Description
char	Maximum length of 8,000 characters.(Fixed length non-Unicode characters)
varchar	Maximum of 8,000 characters.(Variable-length non-Unicode data).
varchar(max)	Maximum length of 231characters, Variable-length non-Unicode data (SQL Server 2005 only).
text	Variable-length non-Unicode data with a maximum length of 2,147,483,647 characters.

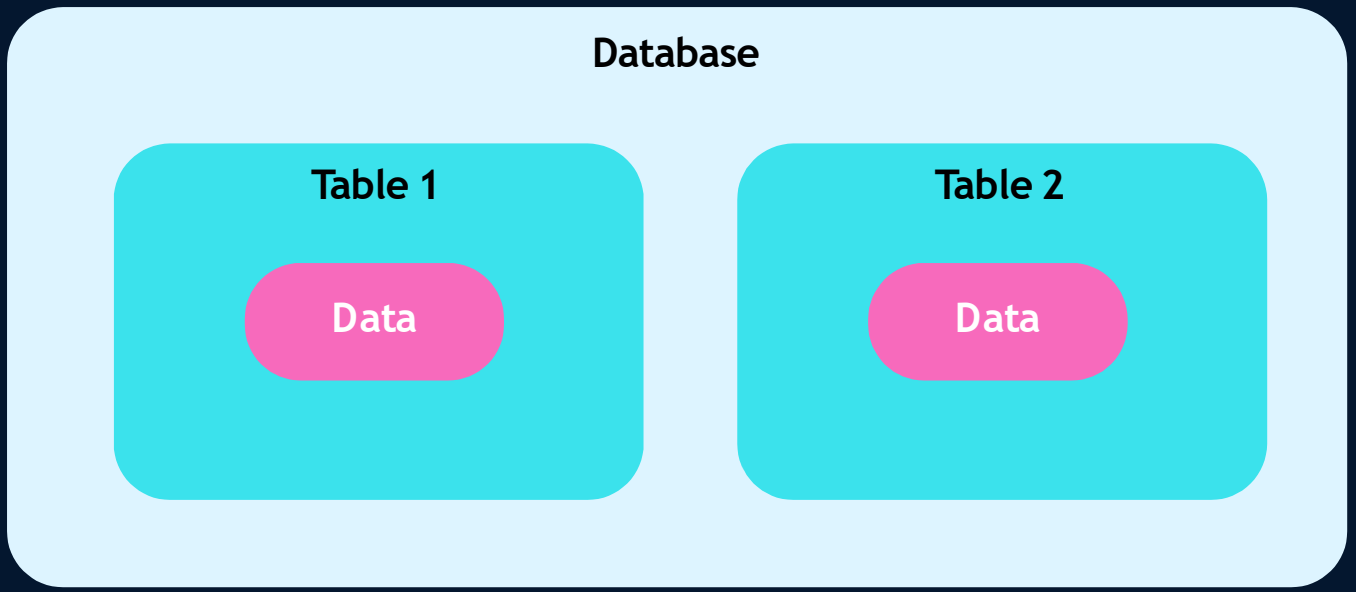
Binary Data Types

DATA TYPE	Description
binary	Maximum length of 8,000 bytes(Fixed-length binary data)
varbinary	Maximum length of 8,000 bytes.(Variable length binary data)
varbinary(max)	Maximum length of 231 bytes (SQL Server 2005 only). (Variable length Binary data)
image	Maximum length of 2,147,483,647 bytes. (Variable length Binary Data)

Misc Data Types

DATA TYPE	Description
sql_variant	Stores values of various SQL Server-supported data types, except text, ntext, and timestamp.
timestamp	Stores a database-wide unique number that gets updated every time a row gets updated
uniqueidentifier	Stores a globally unique identifier (GUID)
xml	Stores XML data. You can store xml instances in a column or a variable (SQL Server 2005 only).
cursor	Reference to a cursor object
table	Stores a result set for later processing

Database Structure



What is a table?

Student table

RollNo	Name	Class	DOB	Gender	City	Marks
1	Nanda	X	1995-06-06	M	Agra	551
2	Saurabh	XII	1993-05-07	M	Mumbai	462
3	Sonal	XI	1994-05-06	F	Delhi	400
4	Trisla	XII	1995-08-08	F	Mumbai	450
5	Store	XII	1995-10-08	M	Delhi	369
6	Marisla	XI	1994-12-12	F	Dubai	250
7	Neha	X	1995-12-08	F	Moscow	377
8	Nishant	X	1995-06-12	M	Moscow	489

Creating our First Database

Our first SQL Query

```
CREATE DATABASE db_name;
```

```
DROP DATABASE db_name;
```

Creating our First Table

```
USE db_name;

CREATE TABLE table_name (
    column_name1 datatype constraint,
    column_name2 datatype constraint,
    column_name2 datatype constraint
);
```

```
CREATE TABLE student (
    id INT PRIMARY KEY,
    name VARCHAR(50),
    age INT NOT NULL
);
```

SQL Datatypes

They define the type of values that can be stored in a column

DATATYPE	DESCRIPTION	USAGE
CHAR	string(0-255), can store characters of fixed length	CHAR(50)
VARCHAR	string(0-255), can store characters up to given length	VARCHAR(50)
BLOB	string(0-65535), can store binary large object	BLOB(1000)
INT	integer(-2,147,483,648 to 2,147,483,647)	INT
TINYINT	integer(-128 to 127)	TINYINT
BIGINT	integer(-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807)	BIGINT
BIT	can store x-bit values. x can range from 1 to 64	BIT(2)
FLOAT	Decimal number - with precision to 23 digits	FLOAT
DOUBLE	Decimal number - with 24 to 53 digits	DOUBLE
BOOLEAN	Boolean values 0 or 1	BOOLEAN
DATE	date in format of YYYY-MM-DD ranging from 1000-01-01 to 9999-12-31	DATE
YEAR	year in 4 digits format ranging from 1901 to 2155	YEAR

SQL Datatypes

Signed & Unsigned

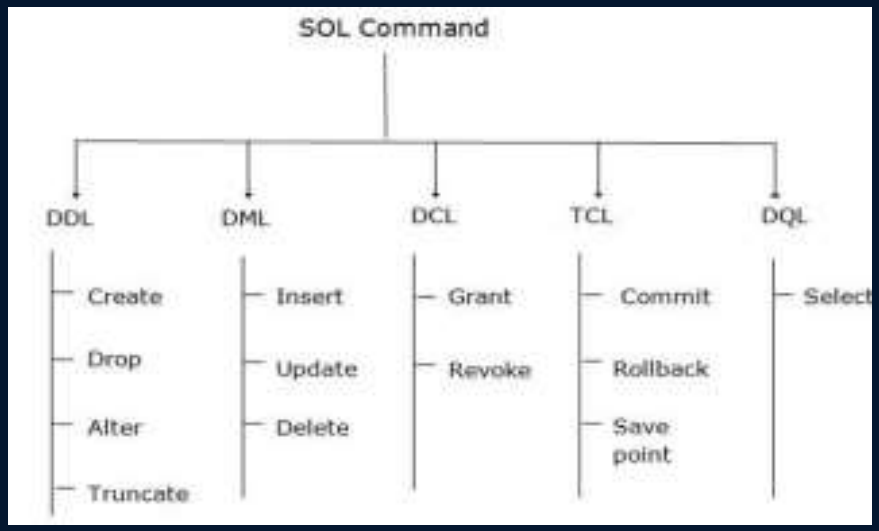
TINYINT UNSIGNED (0 to 255)

TINYINT (-128 to 127)

SQL COMMANDS

SQL commands are instructions. It is used to communicate with the database. It is also used to perform specific tasks, functions, and queries of data.

• SQL can perform various tasks like create a table, add data to tables, drop the table, modify the table, set permission for users.



Types of SQL Commands

DDL (Data Definition Language) : create, alter, rename, truncate & drop
DDL changes the structure of the table like creating a table, deleting a table, altering a table, etc. All the command of DDL are auto-committed that means it permanently save all the changes in the database.

DQL (Data Query Language) : select
DQL is used to fetch the data from the database. It uses only one command.

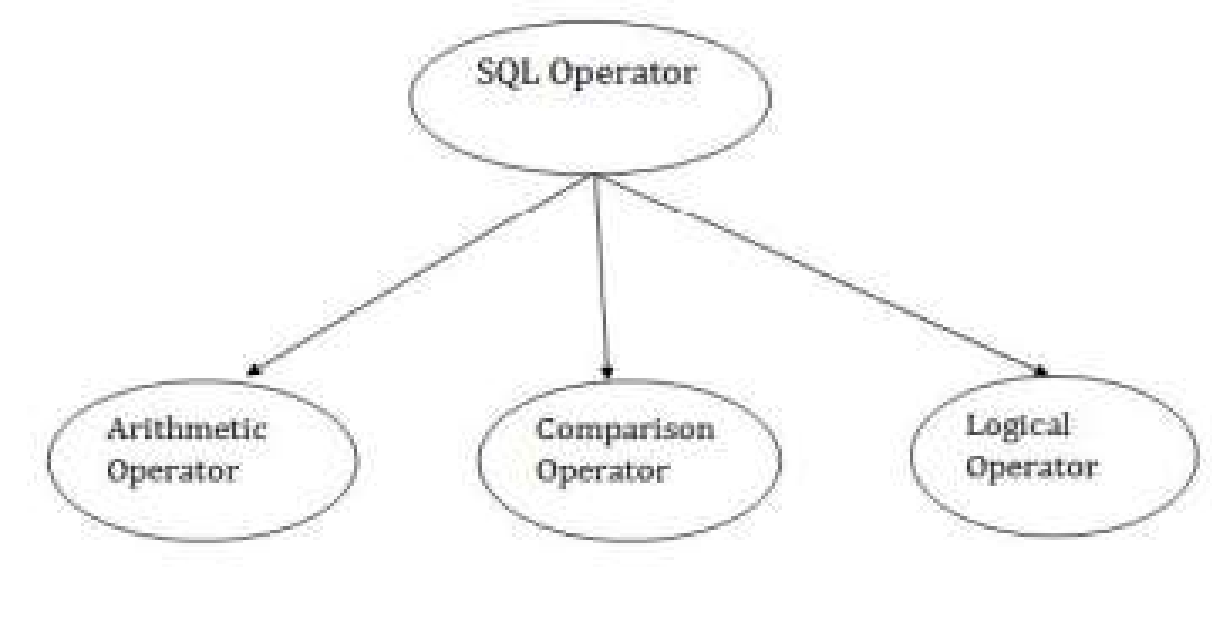
DML (Data Manipulation Language) : select, insert, update & delete
DML commands are used to modify the database. It is responsible for all form of CHANGES in the database. The command of DML is not auto-committed that means it can't permanently save all the changes in the database. They can be rollback.

DCL (Data Control Language) : grant & revoke permission to users

DCL commands are used to GRANT and TAKE BACK authority from any database user.

TCL (Transaction Control Language) : start transaction, commit, rollback etc.
TCL commands can only use with DML commands like INSERT, DELETE and UPDATE only. These operations are automatically committed in the database that's why they cannot be used while creating tables or dropping them

SQL OPERATORS



Arithmetic OPERATORS

Operator	Description
+	It adds the value of both operands.
-	It is used to subtract the right-hand operand from the left-hand operand.
*	It is used to multiply the value of both operands.
/	It is used to divide the left-hand operand by the right-hand operand.
%	It is used to divide the left-hand operand by the right-hand operand and returns reminder.

Compariosn operators

Operator	Description
<=	It checks if the left operand value is less than or equal to the right operand value, if yes then condition becomes true.
!<	It checks if the left operand value is not less than the right operand value, if yes then condition becomes true.
!>	It checks if the left operand value is not greater than the right operand value, if yes then condition becomes true.

Logical operators

Operator	Description
All	It compares a value to all values in another value set.
AND	It allows the existence of multiple conditions in an SQL statement.
ANY	It compares the values in the list according to the condition.
Between	It is used to search for values that are within a set of values.
IN	It compares a value to that specified list value.
NOT	It reverses the meaning of any logical operator.
OR	It combines multiple conditions in SQL statements.
EXIST	It is used to search for the presence of a row in a specified table.
LIKE	It compares a value to similar values using wildcard operator.

Database related Queries

CREATE DATABASE *db_name;*

CREATE DATABASE IF NOT EXISTS *db_name;*

```
CREATE DATABASE IF NOT EXISTS college;
```

DROP DATABASE *db_name;*

DROP DATABASE IF EXISTS *db_name;*

SHOW DATABASES;

SHOW TABLES;

Table related Queries

Create

CREATE TABLE *table_name (*
    column_name1 datatype constraint,
    column_name2 datatype constraint,
);

```
CREATE TABLE student (
  rollno INT PRIMARY KEY,
  name VARCHAR(50)
);
```

Table related Queries

Insert

INSERT INTO *table_name*
(colname1, colname2);
VALUES
(col1_v1, col2_v1),
(col1_v2, col2_v2);

```
INSERT INTO student
(rollno, name)
VALUES
(101, "karan"),
(102, "arjun");
```

Table related Queries

Select & View ALL columns

SELECT * FROM *table_name;*

```
SELECT * FROM student;
```

Keys

Primary Key

It is a column (or set of columns) in a table that uniquely identifies each row. (a unique id)

There is only 1 PK & it should be NOT null.

Foreign Key

A foreign key is a column (or set of columns) in a table that refers to the primary key in another table.

There can be multiple FKs.

FKs can have duplicate & null values.

Keys

table1 - Student

id	name	cityId	city
101	karan	1	Pune
102	arjun	2	Mumbai
103	ram	1	Pune
104	shyam	3	Delhi

table2 - City

id	city_name
1	Pune
2	Mumbai
3	Delhi

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.

Constraints

SQL constraints are used to specify rules for data in a table.

NOT NULL columns cannot have a null value

```
col1 int NOT NULL
```

UNIQUE all values in column are different

```
col2 int UNIQUE
```

PRIMARY KEY makes a column unique & not null but used only for one

```
id int PRIMARY KEY
```

```
CREATE TABLE temp (
  id int not null,
  PRIMARY KEY (id)
);
```

Constraints

FOREIGN KEY prevent actions that would destroy links between tables

```
CREATE TABLE temp (
  cust_id int,
  FOREIGN KEY (cust_id) references customer(id)
);
```

DEFAULT sets the default value of a column

```
salary INT DEFAULT 25000
```

Constraints

CHECK it can limit the values allowed in a column

```
CREATE TABLE city (
  id INT PRIMARY KEY,
  city VARCHAR(50),
  age INT,
  CONSTRAINT age_check CHECK (age >= 18 AND city="Delhi")
);
```

```
CREATE TABLE newTab (
  age INT CHECK (age >= 18)
);
```

INDEX: Used to create and retrieve data from the database very quickly.

Example

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns, three of which, are ID NAME and AGE, In this we specify not to accept NULLs –

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Example

For example, the following SQL creates a new table called CUSTOMERS and adds five columns. Here, the SALARY column is set to 5000.00 by default, so in case the INSERT INTO statement does not provide a value for this column, then by default this column would be set to 5000.00.

```
CREATE TABLE CUSTOMERS(
  ID   INT              NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT              NOT NULL,
  ADDRESS  CHAR (25) ,

  SALARY   DECIMAL (18, 2) DEFAULT 5000.00,
  PRIMARY KEY (ID)
);
```

Example

For example, the following SQL query creates a new table called CUSTOMERS and adds five columns. Here, the AGE column is set to UNIQUE, so that you cannot have two records with the same age.

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL UNIQUE,
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

Example

Consider the structure of the following two tables.

CUSTOMERS table

```
CREATE TABLE CUSTOMERS(
  ID   INT              NOT NULL,
  NAME VARCHAR (20)     NOT NULL,
  AGE  INT              NOT NULL,
  ADDRESS  CHAR (25) ,
  SALARY   DECIMAL (18, 2),
  PRIMARY KEY (ID)
);
```

ORDERS table

```
CREATE TABLE ORDERS (
  ID        INT       NOT NULL,
  DATE      DATETIME,
  CUSTOMER_ID INT references CUSTOMERS(ID),
  AMOUNT    double,
  PRIMARY KEY (ID)
);
```

Example

For example, the following program creates a new table called CUSTOMERS and adds five columns. Here, we add a CHECK with AGE column, so that you cannot have any CUSTOMER who is below 18 years.

```
CREATE TABLE CUSTOMERS(
   ID   INT              NOT NULL,
   NAME VARCHAR (20)     NOT NULL,
   AGE  INT              NOT NULL CHECK (AGE >= 18),
   ADDRESS  CHAR (25) ,
   SALARY   DECIMAL (18, 2),
   PRIMARY KEY (ID)
);
```

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. It is your choice to take it further and go to the fourth normal form, fifth normal form and so on, but in general, the third normal form is more than enough.

- *First Normal Form (1NF)*
- *Second Normal Form (2NF)*
- *Third Normal Form (3NF)*

Database -First Normal Form (1NF

The First normal form (1NF) sets basic rules for an organized database:

- *Define the data items required, because they become the columns in a table.*
- *Place the related data items in a table.*
- *Ensure that there are no repeating groups of data.*
- *Ensure that there is a primary key.*

First Rule of 1NF

You must define the data items. This means looking at the data to be stored, organizing the data into columns, defining what type of data each column contains and then finally putting the related columns into their own table. For example, you put all the columns relating to locations of meetings in the Location table, those relating to members in the MemberDetails table and so on.

Second Rule of 1NF

The next step is ensuring that there are no repeating groups of data. Consider we have the following table:

```
CREATE TABLE CUSTOMERS(
       ID   INT              NOT NULL,
       NAME VARCHAR (20)     NOT NULL,
       AGE  INT              NOT NULL,
       ADDRESS  CHAR (25),
       ORDERS   VARCHAR(155)
);
```

So, if we populate this table for a single customer having multiple orders, then it would be something as shown below:

ID	NAME	AGE	ADDRESS	ORDERS
100	Sachin	36	Lower West Side	Cannon XL-200
100	Sachin	36	Lower West Side	Battery XL-200

But as per the 1NF, we need to ensure that there are no repeating groups of data. So, let us break the above table into two parts and then join them using a key as shown in the following program:

CUSTOMERS Table

```
CREATE TABLE CUSTOMERS(
       ID   INT              NOT NULL,
       NAME VARCHAR (20)     NOT NULL,
       AGE  INT              NOT NULL,
       ADDRESS  CHAR (25),
       PRIMARY KEY (ID)
);
```

This table would have the following record:

ID	NAME	AGE	ADDRESS
100	Sachin	36	Lower West Side

ORDERS Table

```
CREATE TABLE ORDERS(
       ID   INT              NOT NULL,
       CUSTOMER_ID INT       NOT NULL,
       ORDERS   VARCHAR(155),
       PRIMARY KEY (ID)
);
```

This table would have the following records:

ID	CUSTOMER_ID	ORDERS
10	100	Cannon XL-200
11	100	Battery XL-200

Third Rule of 1NF The final rule of the first normal form, create a primary key for each table which we have already created.

Database -Second Normal Form (2NF)

The Second Normal Form states that it should meet all the rules for 1NF and there must be no partial dependences of any of the columns on the primary key: Consider a customer-order relation and you want to store customer ID, customer name, order ID and order detail and the date of purchase:

```
CREATE TABLE CUSTOMERS(
        CUST_ID     INT                 NOT NULL,
        CUST_NAME VARCHAR (20)          NOT NULL,
        ORDER_ID    INT                 NOT NULL,
        ORDER_DETAIL VARCHAR (20)   NOT NULL,
        SALE_DATE   DATETIME,
        PRIMARY KEY (CUST_ID, ORDER_ID)
);
```

This table is in the first normal form; in that it obeys all the rules of the first normal form. In this table, the primary key consists of the CUST_ID and the ORDER_ID. Combined, they are unique assuming the same customer would hardly order the same thing. However, the table is not in the second normal form because there are partial dependencies of primary keys and columns. CUST_NAME is dependent on CUST_ID and there's no real link between a customer's name and what he purchased. The order detail and purchase date are also dependent on the ORDER_ID, but they are not dependent on the CUST_ID, because there is no link between a CUST_ID and an ORDER_DETAIL or their SALE_DATE. To make this table comply with the second normal form, you need to separate the columns into three tables. First, create a table to store the customer details as shown in the code block below:

```
CREATE TABLE CUSTOMERS(
        CUST_ID     INT              NOT NULL,
        CUST_NAME VARCHAR (20)       NOT NULL,
        PRIMARY KEY (CUST_ID)
```

The next step is to create a table to store the details of each order:

```
CREATE TABLE ORDERS(
       ORDER_ID    INT              NOT NULL,
       ORDER_DETAIL VARCHAR (20)  NOT NULL,
       PRIMARY KEY (ORDER_ID)
);
```

Finally, create a third table storing just the CUST_ID and the ORDER_ID to keep a track of all the orders for a customer:

```
CREATE TABLE CUSTMERORDERS(
       CUST_ID     INT              NOT NULL,
       ORDER_ID    INT              NOT NULL,
       SALE_DATE   DATETIME,
       PRIMARY KEY (CUST_ID, ORDER_ID)
);
```

Database -Third Normal Form (3NF)

A table is in a third normal form when the following conditions are met:

- ***It is in the second normal form.***
- ***All non-primary fields are dependent on the primary key. The dependency of these non-primary fields is between the data.***

For example, in the following table – the street name, city and the state are unbreakably bound to their zip code.

```
CREATE TABLE CUSTOMERS(
       CUST_ID        INT              NOT NULL,
       CUST_NAME      VARCHAR (20)      NOT NULL,
       DOB            DATE,
       STREET         VARCHAR(200),
       CITY           VARCHAR(100),
       STATE          VARCHAR(100),
       ZIP            VARCHAR(12),
       EMAIL_ID       VARCHAR(256),
       PRIMARY KEY (CUST_ID)
```

The dependency between the zip code and the address is called as a transitive dependency. To comply with the third normal form, all you need to do is to move the Street, City and the State fields into their own table, which you can call as the Zip Code table

```
CREATE TABLE ADDRESS(
        ZIP             VARCHAR(12),
        STREET          VARCHAR(200),
        CITY            VARCHAR(100),
        STATE           VARCHAR(100),
        PRIMARY KEY (ZIP)
);
```

The next step is to alter the CUSTOMERS table as shown below.

```
CREATE TABLE CUSTOMERS(
        CUST_ID         INT                 NOT NULL,
        CUST_NAME       VARCHAR (20)         NOT NULL,
        DOB             DATE,
        ZIP             VARCHAR(12),
        EMAIL_ID        VARCHAR(256),
        PRIMARY KEY (CUST_ID)
);
```

The advantages of removing transitive dependencies are mainly two-fold. First, the amount of data duplication is reduced and therefore your database becomes smaller. The second advantage is data integrity. When duplicated data changes, there is a big risk of updating only some of the data, especially if it is spread out in many different places in the database. For example, if the address and the zip code data were stored in three or four different tables, then any changes in the zip codes would need to ripple out to every record in those three or four tables

Create this sample table

```sql
CREATE DATABASE college;
USE college;

CREATE TABLE student (
   rollno INT PRIMARY KEY,
   name VARCHAR(50),
   marks INT NOT NULL,
   grade VARCHAR(1),
   city VARCHAR(20)
);
```

Insert this data

```sql
INSERT INTO student
(rollno, name, marks, grade, city)
VALUES
(101, "anil", 78, "C", "Pune"),
(102, "bhumika", 93, "A", "Mumbai"),
(103, "chetan", 85, "B", "Mumbai"),
(104, "dhruv", 96, "A", "Delhi"),
(105, "emanuel", 12, "F", "Delhi"),
(106, "farah", 82, "B", "Delhi");
```

Select in Detail

used to select any data from the database

Basic Syntax

```
SELECT col1, col2 FROM table_name;
```

To Select ALL

```
SELECT * FROM table_name;
```

you want to select every column in a table, you can use * instead of the column names

There are two required ingredients in any SQL query: SELECT and FROM—and they have to be in that order. SELECT indicates which columns you'd like to view, and FROM identifies the table that they live in. if you want to select every column in a table, you can use * instead of the column names

Where Clause

To define some conditions

SELECT *col1, col2* **FROM** *table_name*
WHERE *conditions;*

```
SELECT * FROM student WHERE marks > 80;
SELECT * FROM student WHERE city = "Mumbai";
```

Once you know how to view some data using SELECT and FROM, the next step is filtering the data using the WHERE clause. When using SQL, entire rows of data are preserved together. If you write a WHERE clause that filters based on values in one column, you'll limit the results in all columns to rows that satisfy the condition. The idea is that each row is one data point or observation, and all the information contained in that row belongs together.

Where Clause

Using Operators in WHERE

Arithmetic Operators : +(addition) , -(subtraction), *(multiplication), /(division), %(modulus)

Comparison Operators : = (equal to), != (not equal to), > , >=, <, <=

Logical Operators : AND, OR , NOT, IN, BETWEEN, ALL, LIKE, ANY

Bitwise Operators : & (Bitwise AND), | (Bitwise OR)

Operators

AND (to check for both conditions to be true)

```
SELECT * FROM student WHERE marks > 80 AND city = "Mumbai";
```

OR (to check for one of the conditions to be true)

```
SELECT * FROM student WHERE marks > 90 OR city = "Mumbai";
```

Operators

Between (selects for a given range)

```
SELECT * FROM student WHERE marks BETWEEN 80 AND 90;
```

In (matches any value in the list)

```
SELECT * FROM student WHERE city IN ("Delhi", "Mumbai");
```

NOT (to negate the given condition)

```
SELECT * FROM student WHERE city NOT IN ("Delhi", "Mumbai");
```

The SQL LIKE operator

LIKE is a logical operator in SQL that allows you to match on similar values rather than exact ones.

```
SELECT * FROM
tutorial.billboard_top_100_year_end WHERE
"group_name" LIKE 'Snoop%'
```

The IS NULL operator

IS NULL is a logical operator in SQL that allows you to exclude rows with missing data from your results.

Some tables contain null values—cells with no data in them at all. This can be confusing for heavy Excel users, because the difference between a cell having no data and a cell containing a space isn't meaningful in Excel.

```
SELECT * FROM tutorial.billboard_top_100_year_end
WHERE artist IS NULL
```

Limit Clause

Sets an upper limit on number of (tuples)rows to be returned

```
SELECT * FROM student LIMIT 3;
```

SELECT *col1, col2* FROM *table_name*
LIMIT *number;*

Order By Clause

To sort in ascending (ASC) or descending order (DESC)

```
SELECT * FROM student
ORDER BY city ASC;
```

```
SELECT col1, col2 FROM table_name
ORDER BY col_name(s) ASC;
```

Aggregate Functions

Aggregare functions perform a calculation on a set of values, and return a single value.

- COUNT()
- MAX()
- MIN()
- SUM()
- AVG()

Get Maximum Marks

```
SELECT max(marks)
FROM student;
```

Get Average marks

```
SELECT avg(marks)
FROM student;
```

Group By Clause

Groups rows that have the same values into summary rows.
It collects data from multiple records and groups the result by one or more column.

*Generally we use group by with some ***aggregation function***.

Count number of students in each city

```
SELECT city, count(name)
FROM student
GROUP BY city;
```

Having Clause

Similar to Where i.e. applies some condition on rows.
Used when we want to apply any condition after grouping.

Count number of students in each city where max marks cross 90.

```
SELECT count(name), city
FROM student
GROUP BY city
HAVING max(marks) > 90;
```

General Order

```
SELECT column(s)
FROM table_name
WHERE condition
GROUP BY column(s)
HAVING condition
ORDER BY column(s) ASC;
```

Having Clause

Similar to Where i.e. applies some condition on rows.
Used when we want to apply any condition after grouping.

Count number of students in each city where max marks cross 90.

```
SELECT count(name), city
FROM student
GROUP BY city
HAVING max(marks) > 90;
```

DISTINCT CLAUSE

The **DISTINCT** clause in MySQL is used with a SELECT statement to return the distinct values (unique values) from a single or multiple of columns in a table. It ignores all the duplicates values present in the particular column(s) and returns only the distinct values.

```
CREATE TABLE CUSTOMERS ( ID INT NOT NULL, NAME
VARCHAR (20) NOT NULL, AGE INT NOT NULL, ADDRESS
CHAR (25), SALARY DECIMAL (18, 2), PRIMARY KEY
(ID) );

INSERT INTO CUSTOMERS(ID, NAME, AGE, ADDRESS, SALARY) VALUES (1, 'Ramesh', 32,
'Hyderabad', NULL), (2, 'Khilan', 25, 'Delhi', 1500.00), (3, 'Kaushik', 23,
'Hyderabad', 2000.00), (4, 'Chaital', 25, 'Mumbai', NULL), (5, 'Hardik', 27,
'Vishakapatnam', 8500.00), (6, 'Komal', 22, 'Vishakapatnam', 4500.00), (7, 'Muffy',
24, 'Indore', 10000.00);


SELECT ADDRESS FROM CUSTOMERS;


SELECT DISTINCT ADDRESS FROM CUSTOMERS;
```

Table related Queries

Update (to update existing rows)

```
UPDATE table_name
SET col1 = val1, col2 = val2
WHERE condition;
```

```sql
UPDATE student
SET grade = "O"
WHERE grade = "A";
```

Table related Queries

Delete (to delete existing rows)

DELETE FROM *table_name*
WHERE *condition;*

```
DELETE FROM student
WHERE marks < 33;
```

Table related Queries

Alter (to change the schema)

ADD Column

```
ALTER TABLE table_name
ADD COLUMN column_name datatype constraint;
```

DROP Column

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

RENAME Table

```
ALTER TABLE table_name
RENAME TO new_table_name;
```

Table related Queries

CHANGE Column (rename)

```
ALTER TABLE table_name
CHANGE COLUMN old_name new_name new_datatype new_constraint;
```

MODIFY Column (modify datatype/ constraint)

```
ALTER TABLE table_name
MODIFY col_name new_datatype new_constraint;
```

ADD Column

```sql
ALTER TABLE student
ADD COLUMN age INT NOT NULL DEFAULT 19;
```

DROP Column

```sql
ALTER TABLE student
DROP COLUMN stu_age;
```

MODIFY Column

```sql
ALTER TABLE student
MODIFY age VARCHAR(2);
```

RENAME Table

```sql
ALTER TABLE student
RENAME TO stu;
```

CHANGE Column (rename)

```sql
ALTER TABLE student
CHANGE age stu_age INT;
```

Table related Queries

Truncate (to delete table's data)

TRUNCATE TABLE *table_name ;*

CREATE VIEW

MySQL views are a type of virtual tables. They are stored in the database with an associated name. They allow users to do the following −

- Structure data in a way that users or classes of users find natural or intuitive.
- Restrict access to the data in such a way that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.

A view can be created from one or more tables, containing either all or selective rows from them. Unless indexed, a view does not exist in a database.

Syntax
Following is the syntax of the CREATE VIEW Statement −

```
CREATE VIEW view_name AS select_statements FROM table_name;
```

```
CREATE TABLE CUSTOMERS ( ID INT NOT NULL, NAME
VARCHAR(15) NOT NULL, AGE INT NOT NULL, ADDRESS
VARCHAR(25), SALARY DECIMAL(10, 2), PRIMARY
KEY(ID) );


INSERT INTO CUSTOMERS VALUES (1, 'Ramesh', '32',
'Ahmedabad', 2000), (2, 'Khilan', '25', 'Delhi', 1500),
(3, 'Kaushik', '23', 'Kota', 2500), (4, 'Chaitali',
'26', 'Mumbai', 6500), (5, 'Hardik','27', 'Bhopal',
8500), (6, 'Komal', '22', 'MP', 9000), (7, 'Muffy',
'24', 'Indore', 5500);


CREATE VIEW first_view AS SELECT * FROM CUSTOMERS;


SELECT * FROM first_view;

CREATE VIEW test_view AS SELECT * FROM CUSTOMERS WHERE SALARY>3000;
```

UPDATE VIEW

MySQL UPDATE View Statement

In MySQL, a view is a database object that can contain rows (all or selected) from an existing table. It can be created from one or many tables which depends on the provided SQL query to create a view. There is no direct statement to update a MySQL view. We use the UPDATE statement to modify all or selective records in a view. The results are reflected back in the original table as well.

Syntax

The basic syntax of the UPDATE query with a WHERE clause is as follows –

```
UPDATE view_name
SET column1 = value1, column2 = value2...., columnN = valueN
WHERE [condition];
```

Updating this view –

Now, through the view we created, we are trying to update the age of Ramesh to 35 in the original CUSTOMERS table, using the following query –

```
UPDATE CUSTOMERS_VIEW SET AGE = 35 WHERE name = 'Ramesh';
```

This will ultimately update the base table CUSTOMERS and the same would reflect in the view itself.

DROP view

DROP VIEW view_name;

Renaming Views in MySQL

The MySQL **RENAME TABLE** statement in MySQL is generally used to rename the name of a table. But this statement can also be used to rename views because views are typically virtual tables created by a query.

Syntax

Following is the basic syntax of the RENAME TABLE query to rename a view in MySQL –

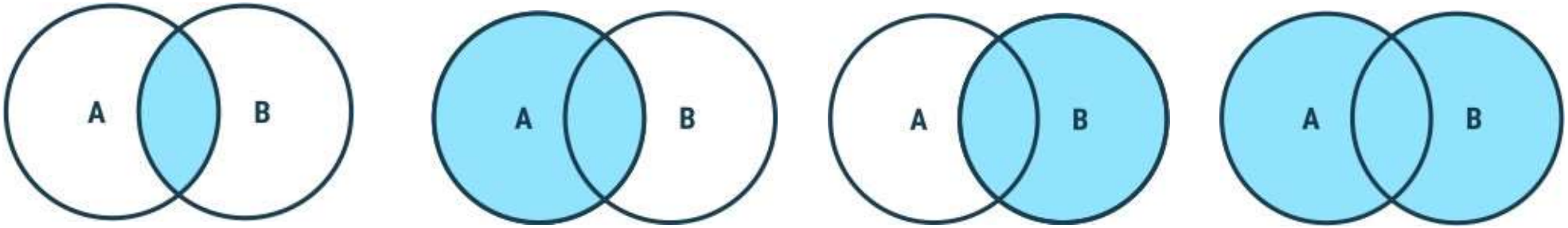
```
RENAME TABLE original_view_name TO new_view_name;
```

Example

Joins in SQL

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

Types of Joins



Inner Join

Left Join

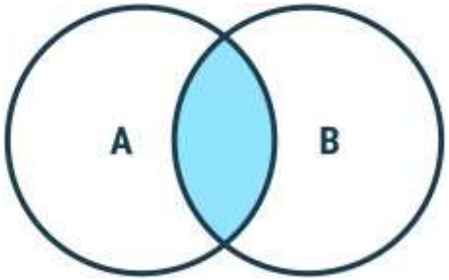
Right Join

Full Join

Outer Joins

Inner Join

Returns records that have matching values in both tables



Syntax

```
SELECT column(s)
FROM tableA
INNER JOIN tableB
ON tableA.col_name = tableB.col_name;
```

Inner Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

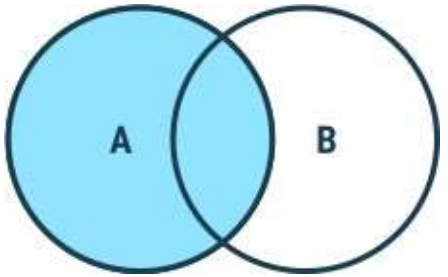
```
SELECT *
FROM student
INNER JOIN course
ON student.student_id = course.student_id;
```

Result

student_id	name	course
102	bob	english
103	casey	science

Left Join

Returns all records from the left table, and the matched records from the right table



Syntax

```
SELECT column(s)
FROM tableA
LEFT JOIN tableB
ON tableA.col_name = tableB.col_name;
```

Left Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

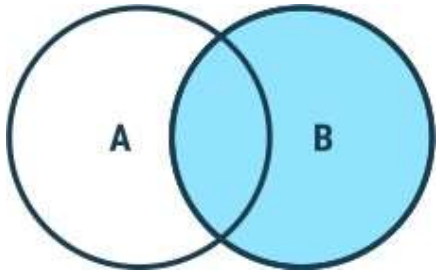
```
SELECT *
FROM student as s
LEFT JOIN course as c
ON s.student_id = c.student_id;
```

Result

student_id	name	course
101	adam	*null*
102	bob	english
103	casey	science

Right Join

Returns all records from the right table, and the matched records from the left table



Syntax

```
SELECT column(s)
FROM tableA
RIGHT JOIN tableB
ON tableA.col_name = tableB.col_name;
```

Right Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

student_id	course
102	english
105	math
103	science
107	computer science

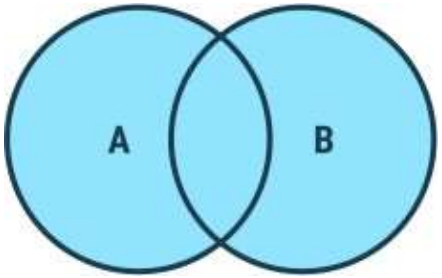
Result

student_id	course	name
102	english	bob
105	math	*null*
103	science	casey
107	computer science	*null*

```
SELECT *
FROM student as s
RIGHT JOIN course as c
ON s.student_id = c.student_id;
```

Full Join

Returns all records when there is a match in either left or right table



Syntax in MySQL

```
SELECT * FROM student as a
LEFT JOIN course as b
ON a.id = b.id
UNION
SELECT * FROM student as a
RIGHT JOIN course as b
ON a.id = b.id;
```

LEFT JOIN
UNION
RIGHT JOIN

Full Join

Example

student

student_id	name
101	adam
102	bob
103	casey

course

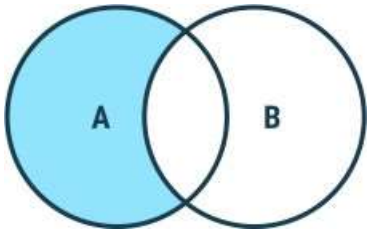
student_id	course
102	english
105	math
103	science
107	computer science

Result

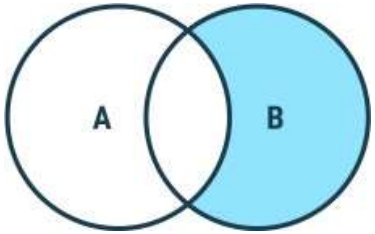
student_id	name	course
101	adam	*null*
102	bob	english
103	casey	science
105	*null*	math
107	*null*	computer science

Think & Ans

Qs: Write SQL commands to display the right exclusive join :



Left Exclusive Join



Right Exclusive Join

```sql
SELECT *
FROM student as a
LEFT JOIN course as b
ON a.id = b.id
WHERE b.id IS NULL;
```

Self Join

It is a regular join but the table is joined with itself.

Syntax

```
SELECT column(s)
FROM table as a
JOIN table as b
ON a.col_name = b.col_name;
```

Self Join

Example

Employee

id	name	manager_id
101	adam	103
102	bob	104
103	casey	*null*
104	donald	103

Result

```sql
SELECT a.name as manager_name, b.name
FROM employee as a
JOIN employee as b
ON a.id = b.manager_id;
```

Union

It is used to combine the result-set of two or more SELECT statements.
Gives UNIQUE records.

To use it :

- **every SELECT should have same no. of columns**
- **columns must have similar data types**
- **columns in every SELECT should be in same order**

Syntax

SELECT *column(s)* FROM ***tableA***
UNION
SELECT *column(s)* FROM **tableB**

INTERSECTION

Combines common rows

SELECT column1 FROM table1 INTERSECT SELECT column2 FROM table2;

EXCEPT/MINUS

Items in first select query not in second select query

SELECT column1 FROM table1 EXCEPT SELECT column2 FROM table2;

UNION ALL

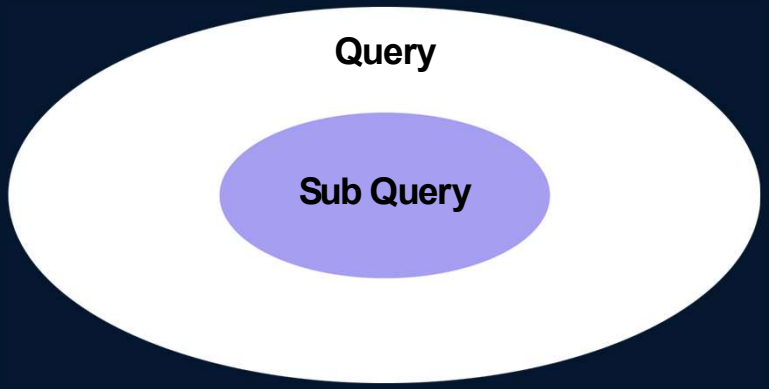
Same as UNION but do not remove duplicates

SELECT column1 FROM table1 UNION ALL SELECT column2 FROM table2;

SQL Sub Queries

A Subquery or Inner query or a Nested query is a query within another SQL query.

It involves 2 select statements.



Syntax

SELECT *column(s)*
FROM *table_name*
WHERE *col_name operator*
(subquery);

SQL Sub Queries

Example

Get names of all students who scored more than class average.

Step 1. Find the avg of class
Step 2. Find the names of students with marks > avg

rollno	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

SQL Sub Queries

Example

Find the names of all students with even roll numbers.

Step 1. Find the even roll numbers
Step 2. Find the names of students with even roll no

rollno	name	marks
101	anil	78
102	bhumika	93
103	chetan	85
104	dhruv	96
105	emanuel	92
106	farah	82

SQL Sub Queries

Example with FROM

Find the max marks from the students of Delhi

Step 1. Find the students of Mumbai
Step 2. Find their max marks using the sublist in step 1

rollno	name	marks	city
101	anil	78	Pune
102	bhumika	93	Mumbai
103	chetan	85	Mumbai
104	dhruv	96	Delhi
105	emanuel	92	Delhi
106	farah	82	Delhi

MySQL Views

A view is a virtual table based on the result-set of an SQL statement.

```
CREATE VIEW view1 AS
SELECT rollno, name FROM student;

SELECT * FROM view1;
```

*A view always shows up-to-date data. The database engine recreates the view, every time a user queries it.