|  |  |
| --- | --- |
| DDL | DML |
| Creating DB | Insert (Copy) |
| Creating Table | Read |
| Altering Table | Update |
| Adding Columns to existing Table | Delete |
| Drop db |  |
| Drop table |  |

MySQL [DB] [Sequential Query Language]

**Query:**

What is query?

* Anything that we ask from DB Table

Types of Queries:

* DDL [Data Definition Language]
* Defining the structure of DB or Table or Columns
* DML [Data Manipulation Language]
* Any query that deals with data present in DB or Table or Column

**Structure of DB:**

* SQL Instance
* DB
* Table
* Column
* Column
* Table
* Column
* Column
* DB
* Table
* Column
* Column
* Table
* Column
* Column

**DDL:**

* Database:
  + create database MYDB;
  + show databases;
  + use mydb;
  + drop database mydb;
* Table:
  + create table student(id int, name varchar(50), marks int, report varchar(1));
  + show tables;
  + describe student;
  + alter table student add column grade varchar(1);
  + drop table student;

**DML:**

* Insertion/Copy -------C
* Read/FETCH -------R
* Updation/Modification----U
* Delete ------------------D

**C:**

* insert into student(id,name,marks,report) values (1,A,10,P);

or

* insert into student values(1,'A',10,'P');

**R:**

* select \* from student;
* select name,marks from student;

**U:**

* update student set marks=11;
* update student set marks=12, name='ANAND';

**D:**

* delete from student;

**Aggregation Operations:**

* select count(\*) from student; -------- count
* select min(marks) from student; ------ min
* select max(marks) from student; ------ max
* select avg(marks) from student; ----- avg
* select sum(marks) from student; ----- sum
* select \* from student limit 1; ------ row limitation for viewing

**Filtering Operations: [where clause --- condition clause]**

*Filtering on rows of the table as per conditioning column*

* select \* from student where report='P';
* select \* from student where report in ('p','f');
* select \* from student where report='p' and name='A' or marks=10;
* select \* from student where report='p' or name='A';

**Ordering Operations:**

* select \* from student order by marks asc; --ascending order
* select \* from student order by marks desc; --descending order

**Group By Operations:**

*Group By clause would be applied when we are grouping the rows based on columns and it will return always an aggregated output*

* select report, count(\*) from student group by report;
* select report , count(\*) as cnt from student group by report, marks;
* select report , count(\*) as cnt from student where report in ('p','f') group by report, marks;
* select report , count(\*) as cnt from student where report in ('p','f') group by report, marks order by cnt desc;

**Having Operations:**

*Having clause is applied only when group by clause is used in the query and also when filtering needs to be applied on aggregated output*

* select report , count(\*) as cnt from student where report in ('p','f') group by report, marks having cnt> 5 order by cnt desc;

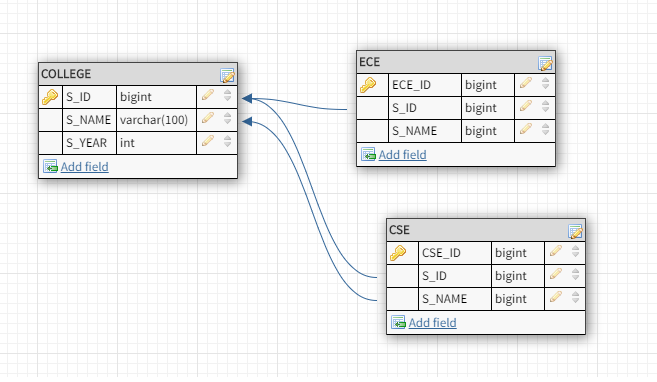
**Distinct Function:** *This function is used when you want to check the distinct values for a given column. It would not result aggregated function as output*

Select distinct(report) from student;

**Indexing:**

* Primary Key
* The PK is used to store and identify the every record in a table as unique
* One can assign only one column as PK
* One can give any number of columns as FK [Foriegn Key]
* Secondary Key / Foreign Key

PK in master table becomes FK in child table to ensure **uniqueness of the record in child** table and also **reference to master** table



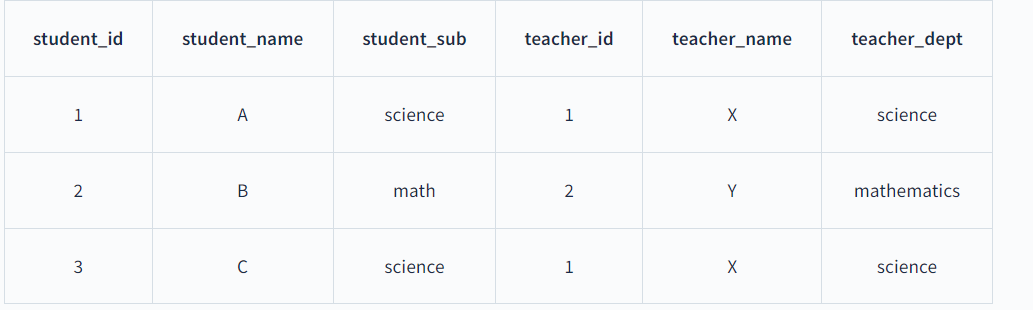
**Inner Queries:**

SELECT \* FROM (SELECT report AS rp,COUNT(\*) AS cnt FROM student WHERE STATUS IN ('g','b') GROUP BY report HAVING cnt>1 ORDER BY cnt) A WHERE A.rp IN ('r','n') ORDER BY A.cnt DESC;

**Normalization:**

**Normalization in SQL** is the process to eliminate data redundancy and enhance [data integrity](https://www.scaler.com/topics/data-integrity-in-sql/) in the table. Normalization also helps to organize the data in the database. It is a multi-step process that sets the data into tabular form and removes the duplicated data from the relational tables.

As we know, we’ve relations (or tables) where we store our data, and if you remember one part of DBMS talks about management, we need to manage this data efficiently. Let’s see the wrong way first.



If we see clearly, there are 2 entities, one is a student and the second is a teacher. It can be taken in 2 ways,

1. They both are added to a single table.
2. One entity is merged with another. In the above example, in the student table, we stuffed the teacher’s data or vice-versa.

The problem with the first assumption is clear. We don’t have a proper way to query the data. Because we need a key to get the data but if we make Student\_id a key then we get the teacher’s data too which we don’t want or vice-versa. So, it’s kinda like a heap of data with no organization.

But what’s wrong with another scenario? Actually, to solve this kinda problem normalization is introduced originally. So please have a nice look at this Section.

The three problems or anomalies are:

1. **Insertion Anomaly**
2. **Deletion Anomaly**
3. **Modification Anomaly**

* **Insertion anomaly** means cases where you can’t insert data unless you have another dependent piece of information. To see this from the above example, we can’t insert the teacher’s data unless we have student data (assuming the key is student\_id).
* **Deletion anomaly** means cases where deleting a record can mistakenly delete other prominent information which is in reality independent in nature. In the above example, deleting a student record will also delete the corresponding teacher’s record and that’s not what we want. That’s because let’s say if a student leaves the school and we’ve to delete their data then doing it would also delete the corresponding teacher’s data and that’s not what we want because it doesn’t mean the teacher also left the school.
* **Modification anomaly** means cases when instead of updating a single piece of information at one place we’ve to update it at multiple places. For example, student 1 and student 3 have the same subject, and that’s why they are allotted the same teacher. If we ever want to change this teacher's name (for some reason) we’ve to do it in 2 places. Assume if they’re duplicated at tons of places then we’ve to do it at every place to provide consistent data. and this you can guess is really difficult here.

So let’s see now what normalization really is and how it solves these anomalies.

**Normal Form**

“It divides a big table into smaller tables until it doesn’t follow a set of rules ”. What are these rules? Actually, we divide the normalization process into a set of normal forms. A normal form is nothing but a form of a table that follows some norms (rules) that prevent the above anomalies to some extent. We have a list of normal forms.

1. **1NF**
2. **2NF**
3. **3NF**
4. **BCNF**

For a table attributes (A,B,C,D), functional dependency could be like,A->BC.It says from attribute A we can get the values of A, B, and C.

### 1st Normal Form (1NF)

It’s the first normal form and almost all tables already follow rules specified in it.

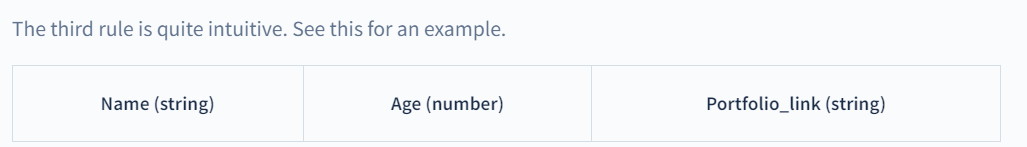
Rules are,

1. Each cell of a row must have a single value.
2. Each record should be unique.
3. Each column should have the same type of data.

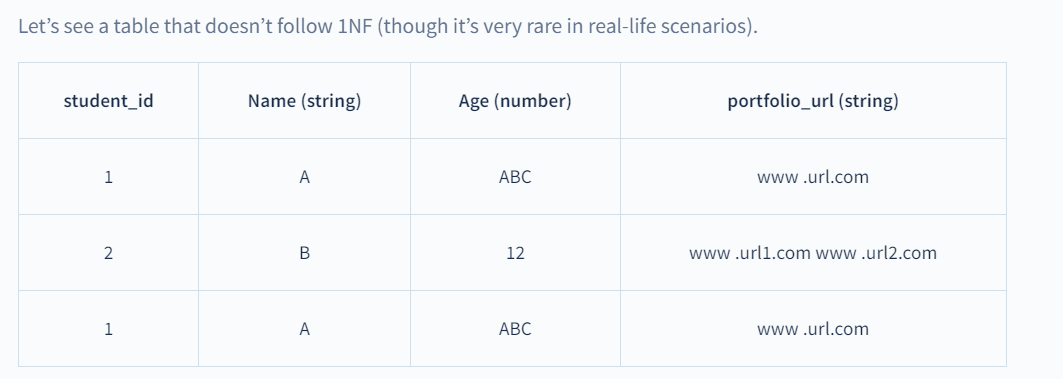
Let’s understand them.

The first rule is easy. It says, a cell should have a single value or you can say it as a cell shouldn’t contain composite values. For example, if an attribute asks for the name of a student, we shouldn’t add 2 or more names to it. Just a single value.

The second rule is related to the concept of keys that identifies each record uniquely. Basically, each record inside a table must have something unique in it because if it’s not then it’s nothing but redundant data (because of duplicity).

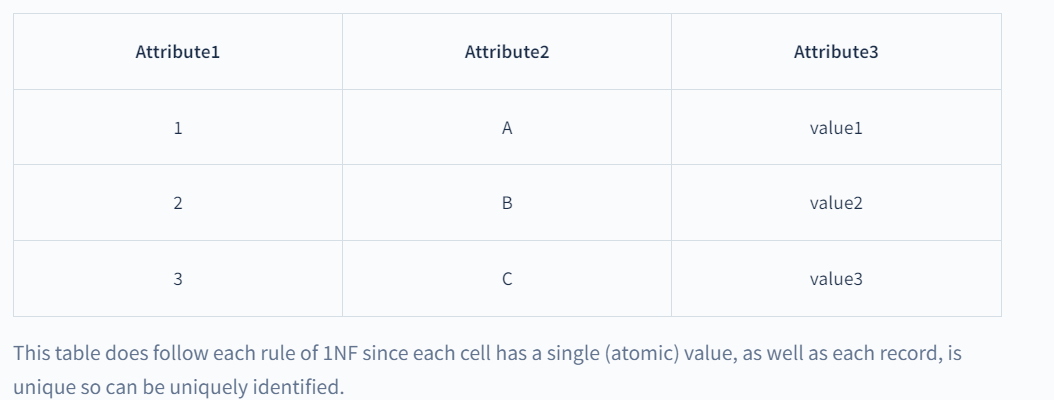


There are 3 attributes in the above table i.e. name, age, and portfolio\_link. And we define data type string, number, and a string for each corresponding attribute. Now, according to the rule, each value inside a name (string type) attribute must be a string. It can’t have numbers. Similarly, for age (number type), each value must be numerical.



**Why is It not in 1NF?**

1. The first row has defined ABC (string value) inside an age attribute (number type) – breaking rule 3
2. The second row defines multiple values in the portfolio\_link attribute. – breaking rule 1
3. The third row is identical to the first row. – breaking rule 2



### **2nd Normal Form (2NF)**

A table is in 2NF when,

1. It’s in 1NF.
2. It shouldn’t have a partial dependency.

**What is Partial Dependency?**

Suppose a relation R, having attributes A, B, C, D. To represent it,

R (A,B,C,D)

functional dependency defined on it is,

AB->D

B->C

From given functional dependencies, we can conclude that AB is a primary key because through AB we can find the value of each attribute.

Therefore, the closure of R (A, B, C, D) is AB.

Now, all the attributes which are part of the candidate key, i.e., A and B are called prime attributes. And all other attributes that are not part of the candidate key are called non-prime attributes.

So, according to 2NF, non-prime attributes should depend on all prime attributes, not on part of the prime attribute. From the above example,

AB->D

B->C

A and B are prime attributes. C and D are non-prime attributes.

D follows the rule because though it’s a non-prime it depends on all prime attributes.

But, C doesn’t follow the rule since it depends on B only (part of the prime attribute) and this is called partial dependency.

So, according to 2NF, a table shouldn’t have a partial dependency.

To make this relation into 2NF,

1. Make a relation that has prime attributes as well as those non-prime attributes that are totally dependent on prime attributes.
2. Make a separate relation for attributes that are in partial dependency.

So, to make the above example into 2NF, we do,

1. R (A,B,D) AB->D
2. R (B,C) B->C

### **3rd Normal Form (3NF)**

A table is in 3NF,

1. It must be in 2NF.
2. It shouldn’t have a transitive dependency.

**What is Transitive Dependency?**

Let’s take a relation, R (A,B,C) with functional dependencies defined as,

A->B

B->C

From the given dependencies, we can conclude that A is the primary key because from A we can get both B and C attributes (from A we get B and from B we get C). Therefore, the closure of R (A, B, C) for given dependencies is A.

Now, A is a prime attribute, and B and C are non-primes.

It’s in 2NF because there is no non-prime attribute that is partially dependent on part of the primary key since B (non-prime) is dependent on A (primary key) and C (non-prime) is dependent on B (non-prime). Also, if a given relation has only a single attribute as a primary key then partial dependency is not even possible because it’s single. Got the point?

But here, a non-prime attribute C is dependent on non-prime attribute B. And, this is called transitive dependency when a non-prime attribute is dependent on another non-prime.

To make this relation into 3NF,

1. Make a separate table of prime attributes and all other attributes directly dependent on it.
2. Make a separate table for non-prime attributes that are dependent on other non-primes and make determining non-prime the primary key in this new table.

So, to make above example into 3NF, we do,

1. R (A,B) A->B
2. R (B,C) B->C

Let’s see this relation, R (A,B,C,D)

A->B

B->C

C->D

Closure of this relation is A. Why?

Now, is it in a 2NF? Yes, because there is no partial dependency.

Is it in a 3NF? No, because there are 2 non-primes that are dependent on other non-primes.

To make it into 3NF,

1. R (A,B)
2. R (B,C)
3. R (C,D

How?

### Boyce CoddNormal Form (BCNF)

Boyce-Codd's normal form comes into the picture because of some anomalies that are not handled by above mentioned normal forms.

If we notice, 2NF and 3NF both deal with non-prime attributes. If there is a functional dependency from A->B then whatever the A is B is always non-prime. They both didn’t deal with situations when B is prime.

So, that’s why BCNF.

A table is in BCNF,

1. It must be in 3NF
2. For any functional dependency A -> B, A must be a super key.

Let’s take a relation, R (A,B,C)

AB->C

C->B

Here, we can conclude AB and AC are 2 candidate keys.

Is it in 2NF? Let’s check it.

AB -> C is not a partial dependency because AB is prime and C is also a prime attribute.

Similarly, C -> B is not a partial dependency. Though C (part of prime attribute) is determining an attribute but since B is also prime so partial dependency doesn’t exist here. If you remember, partial dependency exists when there is a dependency from X->Y and X is prime and Y is non-prime. Here dependency is from prime to prime not prime to non-prime.

**Is it in 3NF?**

According to transitive dependency, a non-prime attribute is dependent on other non-prime but here it’s not the case. So, it’s definitely in 3NF.

But what is the problem? Now, we’re determining even prime attributes. But that shouldn’t happen. Ideally, prime attributes should be the one that determines others. Right? Because that’s why we make keys to uniquely identify the record. That’s what they’re capable of.

So, is the above relation in BCNF?

AB -> C holds true because AB is a candidate key and also super key.

C -> B doesn’t hold because C is a prime attribute which is determining another prime attribute B, and C is not a super key as well. Therefore, the given relation is not in BCNF.

To make it into BCNF

1. R (C,B) C->B
2. R (A,C) AC->AC

This decomposition might be confusing at first but it’s logical. Let’s see how.

See, whenever we decompose a table into smaller tables then we’ve to do it in a way that we are able to get the original table once we join these smaller tables again. This is called lossless decomposition.

So, for the above example, if we make tables,

1. R (C, B) C -> B
2. R (A, B) AB -> AB

We can’t have A, B, and C in a single table because that’s what we’re solving. A separate table for C -> B (C being a super key) is clear because according to BCNF any functional dependency from X -> Y, X should be a super key.

Now, there is confusion between R (A,C) and R (A,B). Why did we choose R (A,C)? It’s because of lossless decomposition. If we choose R (A,B) then how do we connect R (A,B) and R (C,B)? To combine them, there must be one common prime attribute which is a key because then only we are able to relate the two. Isn’t it? In R (A,B) AB is the key and in R (C,B) C is the key (no common prime attribute found).

But in R (A,C) AC is the key and in R (C,B) C is the key. As you can see, C is the prime attribute which is common and also super key in the other table. So, that’s why we chose R (A,C) over R (A,B).

**Note:** Lossless decomposition is a separate topic altogether. We just discussed a glimpse of it here. Please have a look at it on a separate blog.