Data – Random facts and figures which can be stored is data. For ex- 1 , 1.2 , ‘E’ , “IA”.

Information – Processed data and meaningful data is information.

Database – Place where we store our data or collection of data is called database.

DBMS – Software which helps us to create and manages database is called DBMS.

DBMS are of different type

1. Relational DataBase Management System. For ex - MYSQL
2. Non relational database management System. For ex- MongoDB

SQL (Structured Query Language)

* It is language which is used by RDBMS to communicate with the Relational databases. Communicate means performing CRUD operation on D.B

(just for understanding)

Analogy:

Think of a **DBMS (MySQL)** as a car and **SQL** as the steering wheel or dashboard controls.

* The **DBMS (car)** provides the underlying functionality to move, turn, and stop.
* **SQL (controls)** lets you interact with and drive the car to perform specific actions.

**RDBMS(Relational Database management System)** relation means table

* RDBMS is a type of DBMS which helps us to create or manage relational databases.
* Relational database means where data is stored in form of table(means row and column)
* Example of RDBMS – MySQL , OracleSQL, PostgresSQL

**Table**

* Logical representation of data that is stored in D.B
* Combination of rows and columns.
* Rows are also called record, tuple.
* Columns are also called attribute, field, properties.
* A Single row is also called record.

while adding and modifying anything column and constraint keyword will not used in command.

**Important points**

* SQL is not case sensitive. But data will be case-sensitive.
* char, varchar, date type of data comes in single(‘ ‘) quotes.

**RELATIONAL OPERATOR**

Relational operators in SQL are used to compare values in queries, typically in WHERE or HAVING clauses. These operators return TRUE or FALSE based on the comparison.

**NOT OPERATOR !**

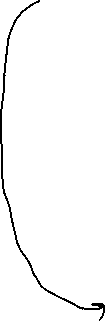
Not operator will **reverse the result**. If the result is true it will make it false and if the result is false it will make it true.

(just for understanding)

Whenever we write any CONDITION, be it in sql or programming, we always write true condition so if the condition is true that record will be selected, but whenever you are using NOT operator, always write false condition because NOT operator will change it from false to true.



Because if we write true condition, it will make it false which we don’t want, so write false condition NOT operator will make it true.



For ex - Select the all emp details, except Martin, Miller



(query without NOT operator)

select \*



from emp



where ename!=’MARTIN’ AND ename!=’MILLER’ ;

(query with NOT operator)

select \*

from emp

where not( ename=’MARTIN’ or ename=’MILLER’) ;

**IS OPERATOR**

Is operator is used to evaluate null values

For ex

Select details of employee who did not earn any commission

Select \*

From emp

Where comm is null ;

Select details of employee who are in some department

Select \*

From emp

Where deptno is not null ;

**IN OPERATOR**

* Shorthand property of OR operator.
* IN will group multiple values.
* IN operator works like = operator
* And results are logically combined using OR

Syntax - columnName IN (value1,value2,value3,value4)



Here we should have only one value here we can have n number of values

**NOT IN OPERATOR**

- Shorthand property of AND operator.

-It will group multiple values.

- Not In operator works like != , it rejects the value of specified list.

- The results are logically combined using AND

Syntax - columnName NOT IN (value1,value2,value3,value4)



**TRIM**

Trim function is used to remove a character from starting, ending, or from both places in a string.

Syntax - (leading/trailing/both ‘s’ from ‘sandeep’)



character you want to remove string

**REPLACE**



Replace function is used to replace and remove character from given string.

Syntax - Replace(main string , search string , replace string)

Note – if you don’t pass third argument it will remove the search string.

* It will look for the search string as a whole, in the given string, not character by character . It will replace the search string only when it finds the whole search string as it is, in the original string.

For example -

* 1. SELECT REPLACE('SANDEEP' , 'SA' , 'ZY')

FROM DUAL ;

OUTPUT

REPLACE

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

ZYNDEEP

* 1. SELECT REPLACE('SANDEEP' , 'SD' , 'ZY')

FROM DUAL ;

OUTPUT

REPLACE

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

SANDEEP

**Translate**

Translate is used to replace character from given string.

Syntax – Translate(main string, search string, replace string)

Note - 3rd argument is compulsory to pass.

* It will look for search string character by character in the given string and replace character by character.
* We can remove the character using Translate but atleast one character we have to replace and other character can be removed, as giving 3rd argument is mandatory in Translate.(see example 2)

For ex

1. SELECT TRANSLATE('SANDEEP' , 'NP' , 'KJ')

FROM DUAL ;

OUTPUT

TRANSLA

-------

SAKDEEJ

1. SELECT TRANSLATE('SANDEEP' , '\_DP' , 'Z')

FROM DUAL ;

OUTPUT

TRANSL

------

SKNEE

**INSTR**

Instr returns us the index value, of the character’s occurrence in given string.

**DATE FUNCTION**

Date written like this ’21-MAR-25’ is considered string type not date type. So to convert ’21-MAR-25’ to date type we use TO\_DATE(’21-MAR-25’) function.

So we cannot add any number in string type… for ex- ’21-MAR-25’+ 10 …. is error …..so to add any number to the date, we have to convert string type i.e ….’21-MAR-25’ to date type using TO\_DATE(’21-MAR-25’).

If I add any number to the date, it will be considered adding days to date. So it will return us new date.

For ex -

* SELECT TO\_DATE(’15-JAN-25’) + 8

FROM DUAL ;

OUTPUT

23-JAN-25

1. ADD\_MONTHS() FUNCTION

Syntax - ADD\_MONTHS(DATE , MONTHS)

* It will add/subtract months in the given date and give us the new date.
* Used when we want to know what will be the date after 3 months or 3years from given date.



by converting years into months

For ex

1. Adding months in the date

SELECT ADD\_MONTHS(’15-JAN-25’ , 9))

FROM DUAL ;

1. Adding years in date 🡪 convert year into months and then use add\_month() function only

SELECT ADD\_MONTHS(CURRENT\_DATE , 12\*3)

FROM DUAL ;



Converting years into months

1. MONTHS\_BETWEEN()

Syntax - MONTHS\_BETWEEN( HIGHER\_DATE , SMALLER\_DATE )

* It will give us number of months exist between two dates.
* And months can be converted to years also by dividing by 12, this means we can also find out no. of years between two dates.

**HANDLING TWO DIGIT YEAR IN SQL**

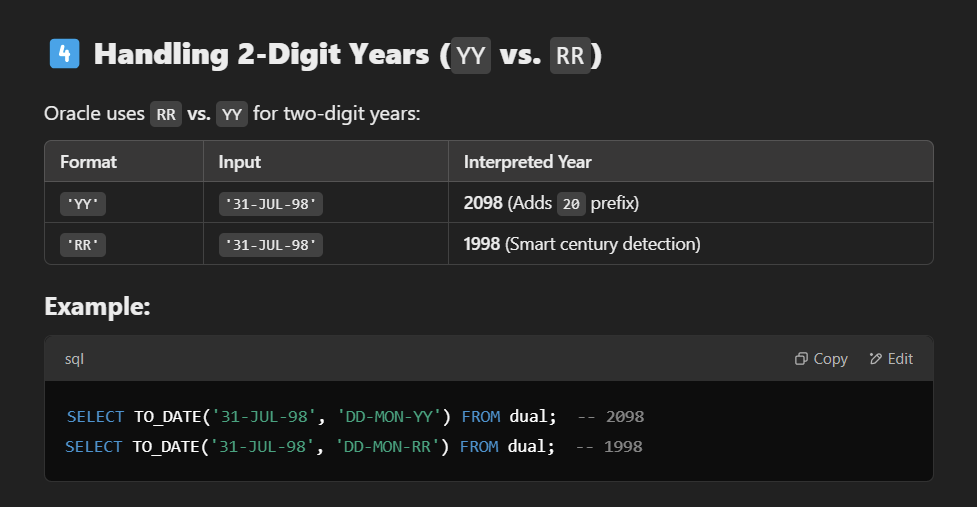
Whenever we mention YEAR in SQL in two digits, sql will detect century based on RR format instead of YY format.

So YY format means it will detect century based on system date.

But in RR it will detect century based on belowe criteria

00-49 🡪 2000 to 2049

50-99 🡪 1950 to 1999



That why it is said always use four digits for mentioning year in sql to avoid any confusion.

**SPECIAL FUNCTION**

CONVERSION FUNCTION

1. **TO\_CHAR() FUNCTION**

Syntax TO\_CHAR(DATE, DATE\_FORMAT)



mention string in which you want to

change like (‘DAY’, ’MON’ etc)

* This function will change date to particular type of string.
* The returned string will have leading and trailing spaces so remove spaces from both side before comparing the returned string to any other string.

1. **TO\_DATE() FUNCTION**

whatever string date you have given in 1st argument

mention same date format here. (See example)



Syntax TO\_DATE(STRING , DATE FORMAT)

* Convert string type to date type.

For example

1. TO\_DATE(’31-JUL-98’ , ‘DD-MON-YY’)

FROM DUAL ;

OUTPUT

31-JUL-98

1. SELECT TO\_DATE('31/07/1998' , 'DD/MM/YYYY')

FROM DUAL ;

OUTPUT

31-JUL-98

1. SELECT TO\_DATE('2003-27-03' , 'YYYY-DD-MM')

FROM DUAL ;

OUTPUT

27-MAR-03

1. SELECT TO\_DATE('2003-27-JUL' , 'YYYY-DD-MON')

FROM DUAL ;

OUTPUT

27-JUL-03

**HOW SET CLAUSE IS EXECUTED IN UPDATE STATEMENT**

Set clause in update statement is updating each row so it expects n value for n rows means one value for each row but , but aggregate function returns one value for n number of rows so database did not know how to map one value with each row…

So directly using aggregate function in set clause is not allowed… solution is to use subquery

Subquery also returns one value only but now it will be considered as constant by database for each row and each row will be updated with that value.

**GROUP BY CLAUSE**

* It will group same values, of a column.

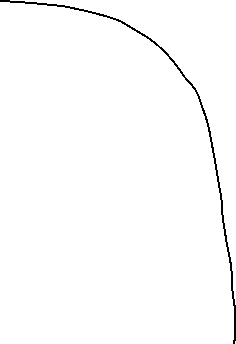
Syntax- GROUP BY COLUMN\_NAME1 , COLUMN\_NAME\_2

Any clause executed after group by clause will be executed group by group.

If we have used Group By clause in our query, then in Select Clause we can write only MRF() function and column written in group by clause.

Order of execution Reason

1. From tableName
2. Where condition(optional)
3. Group by columnName1 , columnName2
4. Select



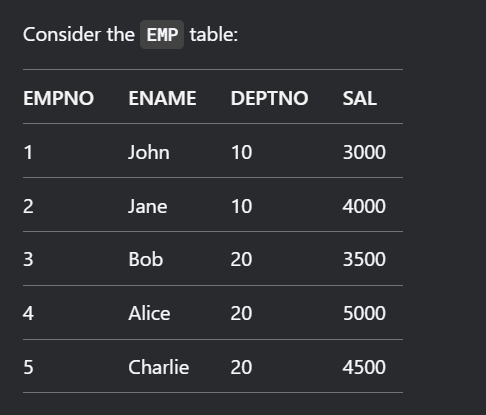
(just for understanding)

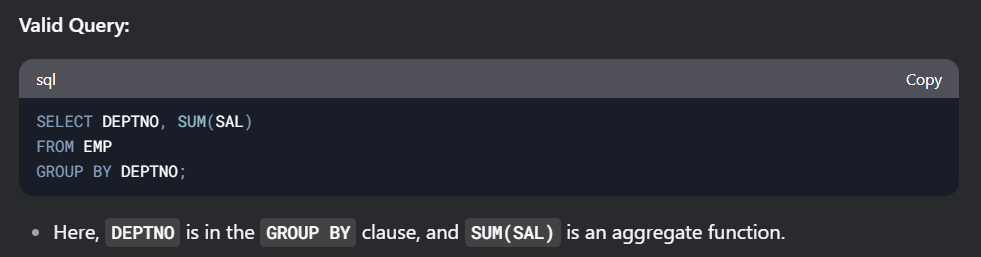


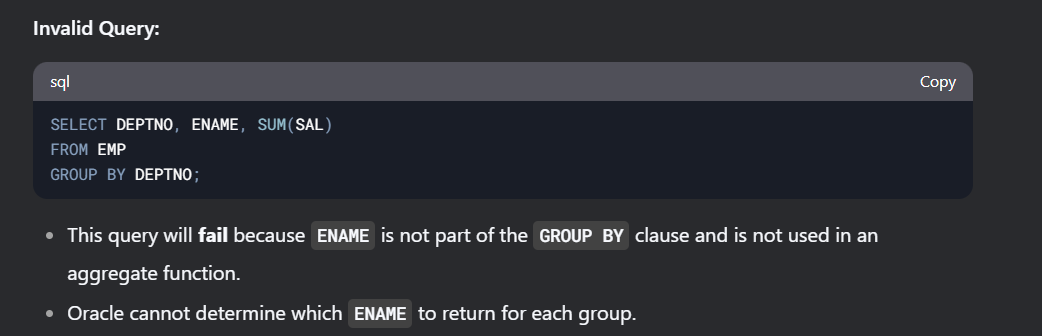
because after group by clause, every clause will be executed group by group, so if we write any other column in select clause, database will not be able to determine which one value to return from that column from each group.

(because that column written in select clause will have multiple values to return)

For ex-







**HAVING CLAUSE**

* Use to filter groups. (By writing a condition)
* Written and executed after group by clause.
* In having clause we can’t use srf function because having executes after group by clause which means having execute group by group means from each group one value should be returned, but srf function can return multiple values from each group, by which database will get confused to select which value for the group, from those multiple values.

Syntax

HAVING CONDITION\_1 , CONDITION\_2

**Difference between where clause and having clause**

|  |  |
| --- | --- |
| **Where clause** | **Having clause** |
| 1. Use to filter records of columns | * Use to filter records of groups |
| 1. Executes row by row (only srf function can be used here) | * Executes group by group |
| 1. Written and excuted before group by clause | * Written and executed after group by clause |
| 1. Where can be written without group by clause | * Having clause cannot be written without group by clause (because there will be no meaning of having clause withour group by clause) |
| 1. Mrf function cannot be written in where clause. | * Mrf function can be used in having clause. |

**ORDER BY CLAUSE**

Why?

* Used to sort the data in ascending or decending.

How?

Syntax - ORDER BY Column\_Name\_1 asc/desc , Column\_Name\_2 asc/desc

* We can use Column\_Number also in place of Column\_Name(Column\_Number will be given on the basis of 🡪on which number that column is present on select clause) and if we don’t mention asc/desc then by default order by sort data in asc.
* If there are two column written in Order by clause, then first priority will be given to first column.
* Srf() and mrf() both can be used in order by clause

Order of execution

1. FROM
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. ORDER BY

Note – we can skip where, grouby , having if not needed

**ANY OPERATOR , ALL OPERATOR**

* **ANY, ALL operator both are used with relational operator**
* ANY operator returns true, if any one condition is satisfied
* ALL operator returns true, if all conditions are satisfied.

**SUBQUERY**

When to use subquery

When we have to fetch the data from one table only but which data to fetch depends on another table output.

We can write subquery in any clause and n number of times

**CO-RELATED SUBQUERY**

When to use

When I want to compare each row of one table with each row of same table or another table then we can use co-related subquery.

WHAT

When inner query is dependent on outer query, and outer query execute(partially) before inner query these types of query are called co-related subquery.

How co-related subquery works, (flow of co-related subquery)

* First outer query will be executed(its FROM clause… then WHERE clause in WHERE clause… inner Query/sub-query will be executed first, outer table each row will be compared with each row of inner table then inner-query/sb-query will give a result then WHERE clause of outer query will compare the result and select or deselect row of outer query. Then for each row of outer query this process will repeat.
* Inner query will be using, value from outer query.

**JOINS**

What?

When we join two or more tables rows, based on related column is called joins

Why?

Because when we have to fetch data from multiple we use join, it helps in avoiding writing multiple queries.

Syntax

SELECT

FROM tableName1 joinName tableName2

ON <join condition> (in ON clause when mentioning column name, use

tablename before columnName as multiple tables exist in query)



Writing join condition is mandatory and join condition is written based on related column(column present in both tables).

We don’t have to write join condition in cross join, natural join.

**SELF JOIN**

What ?

When we join a table with itself is known as self join.

WHY ?

When we have to MERGE row with the row of same table we use self join

Points to remember in self join while writing query

* Name is self join but when we mention self join in query write only join.
* Alias the table name whenever using table name in query, as same table are merging, so for avoiding confusion.
* Join condition will be written on the basis of rows we want to compare.

**EXISTS NOT EXISTS OPERATORS**

What

These two are operators used with corelated subquery.

**EXISTS** operator returns true if inner query returns some row.

When to use

* When we want to check record present in one table is also present in another table.

**NOT EXISTS** operator returns true if inner query do not return any row.

When to use

* When we want to check record present in one table should not present in another table.

**FUNCTIONAL DEPENDENCY**

It is a relationship between two attribute where one attribute’s value determines other attribute’s value.

Notation A🡪 B

Determinant dependent

Value of A can determine, the value of B

For ex – In emp table, if you have empid you can determines the emp name, deptno etc

empId 🡪 ename

empId 🡪 deptno

Types of functional dependency

1. Trivial functional dependency

A🡪 B

A can determine B and B is a subset of A. This is called trivial functional dependency

For example

(emp\_id, ename) 🡪 ename

(emp\_id,ename) can determine (ename), and (ename) is subset of (emp\_id, ename)

(emp\_id,ename) 🡪 emp\_id

(emp\_id,ename) can determine (emp\_id), and (emp\_id) is subset of (emp\_id, ename)

1. Non-trivial functional dependency

A 🡪 B

A can determine B but B is not subset of A. This is called non-trivial functional dependency

For example

(emp\_id,ename) 🡪 deptno

(emp\_id,ename) can determine deptno but deptno is not subset of (emp\_id,ename) so this is non- trivial functional dependency.

NOTE 🡪 means DETERMINE

Rules of Functional dependency (Armstrong’s axioms)

1. Reflexive

If A is a set of attribute and B is a subset of A then A🡪B (means A can determine B)

For example

A= {a,b,c,d,e}

B={a,b,c}

B is a subset of A

So A can determine B.

1. Augumentation

If A 🡪 B dependency is given, then multiplying attributes on both sides will not affect the dependency, it will still be valid dependency.

For example

Attributes of table(A,B,Z)

A 🡪 B (given)

THEN

(multiplying Z on both side)

AZ 🡪 BZ …. AZ can determine BZ (so it is still valid dependency)

3 ) Transitivity

If A can determine B, and B can determine C then A can also determine C.

If A🡪B and B🡪 C then A🡪C also

**Normalisation**

What is normalisation

Normalisation is a D.B optimisation technique which reduces data redundancy and increases data consistency.

How to optimise your D.B

Decompose(breakdown) your table into multiple tables until we achieve SRP(single Responsibilty Principle). SRP means one table should represent one entity only.

Why to use Normalisation

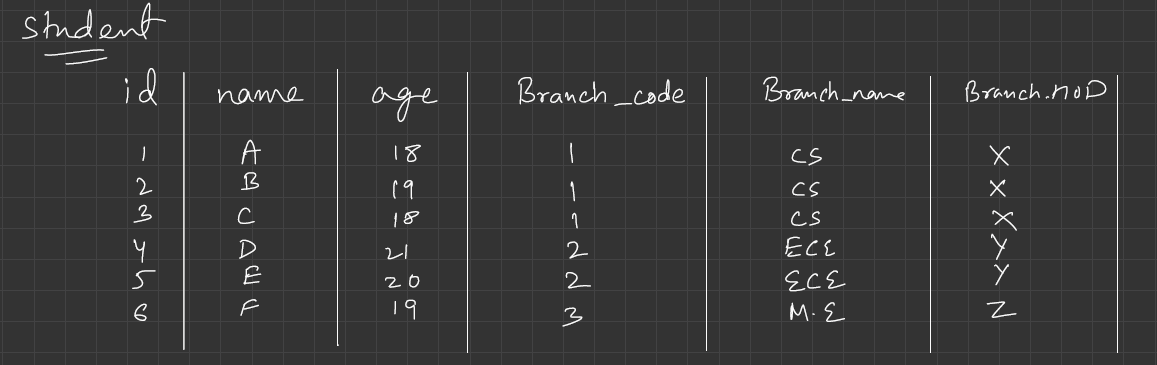
* To avoid Data redundancy as it causes anomalies like

1. Insertion Anomalies
2. Deletion Anomalies
3. Updation Anomalies

* Data redundancy also increase the size of database because of which database performance will be low.

Insertion Anomalies

Without the presence of one data we cannot enter the other data.





PRIMARY KEY

Suppose college has introduced a new branch and till now no student is enrolled in that branch, but that branch data cannot be inserted into this table because this table Primary key is StudentId, so without Student details, branch details cannot be added.

Deletion Anomalies

Deletion of certain data results in unintended deletion of other data also.

For example

In above table, Suppose All the Students of C.S branch has passed out and now we want to delete the data of those Students from the table but deleting the students, results in deleting the branch, even though the branch exist in the college.

Updation Anomalies

* Update in a single data, requires multiple rows to be updated
* so if one forgots to update any one row also data inconsistency arises.

For example

In above table, C.S (computer science) branch name changed to S.E (software engineering) branch, now change in single data requires multiple rows to be updated, so if we forgot to update any one row data inconsistency arises.

**TYPES OF NORMAL FORMS**

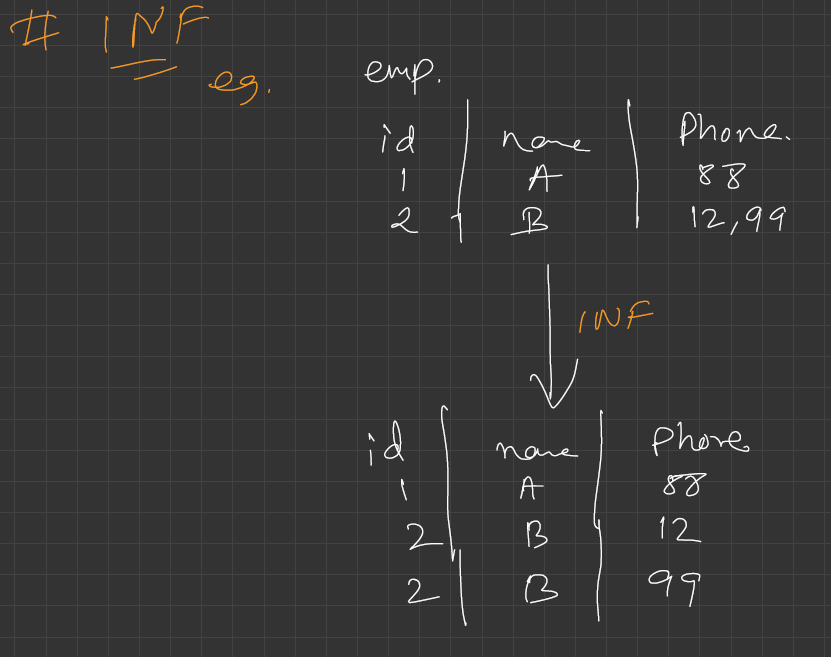
1NF

* Every cell in table should contain only one value, Table should not have multivalued attribute

Multivalued attribute means attribute which can have more than one value for a single entity.

Here phone number is a multivalued attribute as it consist more than one value for a single entity.







2NF

* Relation/Table must be in 1NF
* Remove partial dependency of non-prime attributes.

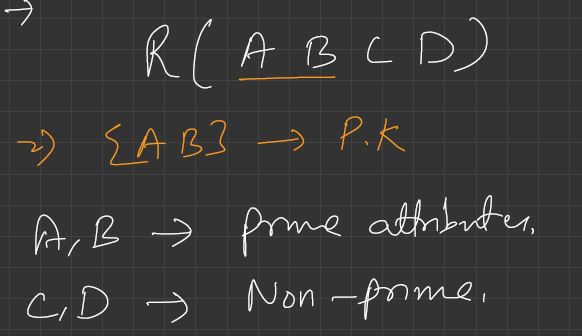
Non prime attribute should be dependent on full Primary Key, not on the part of Primary Key.

Prime Attributes are those attribute which are part of primary key.

Non Prime Attributes are those attributes which are not part of primary key.

For example

Four Column of table





C,D(Non-prime Attribute) should be determined by (A,B) combined, by the whole Primary Key then we can say non-prime attributes are dependent on full Primary Key.



(A,B) 🡪 C

(A,B) 🡪 D

If C(non-prime Attribute) is determinded by A alone( by a part of Primary Key) then it means Non-prime attributes are partially dependent on primary Key.

A 🡪 C

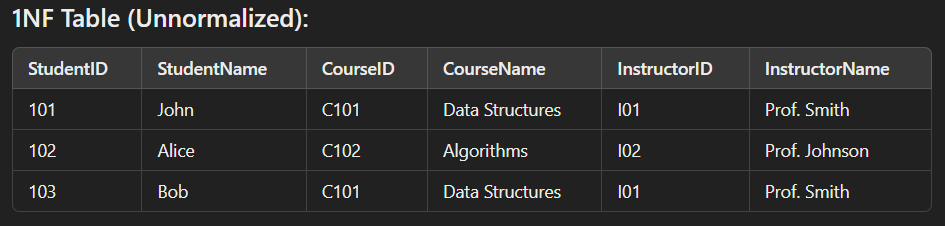
B 🡪 D

Q- Why Non-prime Attribute should be fully dependent on Primary key And not Partially dependent?

Ans So from above example….. A column can hold null values also , B column can hold null value also but at the same time both column can’t hold null because primary key can’t be null. And if A alone is determining C so if A is null,how can A determine C. So to avoid this problem Non-prime Attribute should be fully dependent on Primary Key because both values can’t be null at same time. So it is better to have A,B both determine C

**Real Life example of converting a table from 1NF to 2NF**

Target 🡪 to remove partial dependency of non-prime attribute on primary key.

 Primary key: **(StudentID, CourseID)** (composite key because a student can enroll in multiple courses).

**Problems in 1NF:**

1. Partial Dependency:
   * StudentName depends only on StudentID, not the full composite key.
   * CourseName and InstructorName depend only on CourseID, not the full composite key.

**Tables Converted to 2NF**

**Student Table**:

|  |  |  |
| --- | --- | --- |
| **StudentId** | **StudentName** | **CourseId** |
| 101 | John | C101 |
| 102 | Alice | C102 |
| 103 | Bob | C103 |

Primary Key 🡪 StudentId

**Course Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **CourseId** | **CourseName** | **InstructorId** | **InstructorName** |
| C101 | Data Structure | 101 | Prof. Smith |
| C102 | Algorithms | 102 | Prof. Johnson |

Primary Key 🡪 CourseId

Now non-prime Attributes are dependent on full primary Key.

3NF

* Relation must in 2NF
* Remove transitive dependency

Means non-prime attributes should not find non-prime attributes

For example

R(A,B,C) (in a table there are three column A,B,C)

Primary key 🡪 A

F.D(given)

A🡪B

B🡪C

then transitivity say A 🡪 C also….. so if A🡪B and A🡪 C also ..so we remove B🡪C functional ‘ dependency.

Q- How to remove transitive dependency.

We remove transitive dependency in 3NF by making a separate table for B and C attribute and making B as Primary Key in that table.

**Real Life Example of converting a table from 2NF to 3NF**

**Tables in 2NF**

**Student Table**:

|  |  |  |
| --- | --- | --- |
| **StudentId** | **StudentName** | **CourseId** |
| 101 | John | C101 |
| 102 | Alice | C102 |
| 103 | Bob | C103 |

Primary Key 🡪 StudentId

In this table we need no changes as transitive dependency do not exist.

**Course Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **CourseId** | **CourseName** | **InstructorId** | **InstructorName** |
| C101 | Data Structure | 101 | Prof. Smith |
| C102 | Algorithms | 102 | Prof. Johnson |

Primary Key 🡪 CourseId

Problem in this table……. transitive dependency exist

CourseId 🡪 InstructorId

InstructorId 🡪 InstructorName (remove this transitive dependency)

CourseId 🡪 InstructorId

Solution Make a separate table for InstructorId And InstructorName and make InstructorId primary key.

**Tables converted to 3NF**

**Student Table**:

|  |  |  |
| --- | --- | --- |
| **StudentId** | **StudentName** | **CourseId** |
| 101 | John | C101 |
| 102 | Alice | C102 |
| 103 | Bob | C103 |

Primary Key 🡪 StudentId

Foreign Key 🡪 CourseId

**Course Table:**

|  |  |  |
| --- | --- | --- |
| **CourseId** | **CourseName** | **InstructorId** |
| C101 | Data Structure | 101 |
| C102 | Algorithms | 102 |

Primary Key 🡪 CourseId

InstructorID 🡪 ForeignKey

**Instructor Table:**

|  |  |
| --- | --- |
| **InstructorId** | **InstructorName** |
| 101 | Prof. Smith |
| 102 | Prof. Johnson |

Primary Key 🡪 Instructor Name

No Transitive dependency exist in any table now..

BCNF (Boyce Codd Normal Form) ………….. it basically focuses on prime attributes.

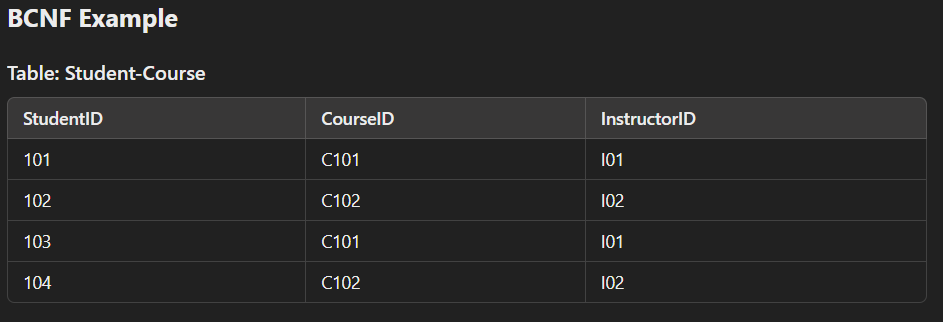
* Table should be in 3NF

- X🡪Y , X must be a super key.

- A non-prime attribute should not determine any prime attribute.

- A prime attribute also should not determine any other prime attribute. In other words prime attribute should also depend on full primary key, not on the part of primary key.

Real life example of BCNF



Functional Dependencies:

Primary key/super key : StudentID, CourseID

StudentID, CourseID 🡪InstructorID

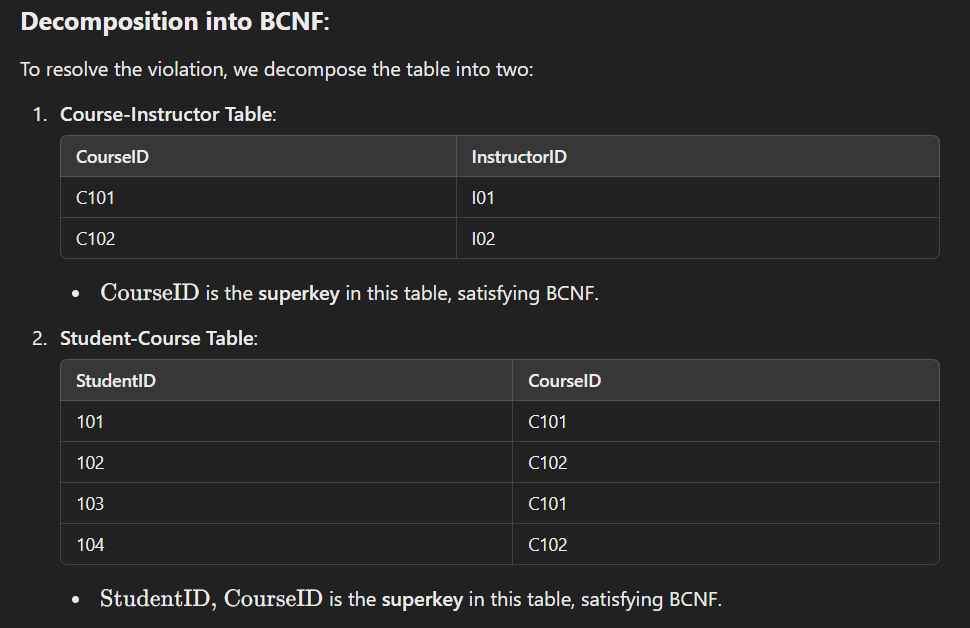
CourseID 🡪 InstructorID

Violation of BCNF:

CourseID determines InstructorID,but CourseID is not a superkey.

This violates BCNF because, in BCNF, every determinant (left-hand side of a functional dependency) must be a superkey.

Table converted to BCNF





Q- Why we have not made StudentID as primary Key and took both StudentId and CourseId as super key /primary key ?

StudentId, Course id is the superkey/primary key because same student can enroll in multiple courses so at that time StudentId alone will not be able to identify each row uniquely. So its better to make StudentId, CourseId as primary key/super key.