**11/30/2021 SQL JOINS**

**what is join in sql ?**

It's corresponding to a join operation in relational algebra .

It combines the columns from one or more tables into a new table.

**why joins are used or need of join in sql?**

**SQL join** is used to combine data or rows from two or more tables

based on the common filed between the tables.

**Types of joins in sql?**

there are 3 types of joins are there in sql .

1) INNER JOIN

2) OUTER JOIN

3) CROSS JOIN

**1) INNER JOIN :**

we can only select the data which is common in both the tables ,

if some rows re having same data in both the tables then this join

would select that rows.

**2) OUTER JOINS :**

It can select the data which matching in the particular table .

**LEFT OUTER JOIN:**

If the left table can match the same data in the right table, if both

the data or rows are matched then it will print the rows matched

rows in left table .

**RIGHT OUTER JOIN:**

If the right table can match the same data in the left table, then

return data or rows are matched in the right table.

**FULL OUTER JOIN :**

This join combines the results of **LEFT OUTER JOIN &**

**RIGHT OUT JOIN .**

The joined table will contain all records from both the tables and

fill in nulls for missing mathces on either side.

if mysql doesn't support **FULL JOIN then**  you can use **UNION ALL**

to combine these two joins.

**CROSS JOIN :**

Tis join is used to generate a paired combinations of each row of

the first table with each row of the second table.

It's also called as cartesian join , the main idea of this join is that

returns the cartesian product of the joined tables.

**MYSQL SUPPORTED JOINS:**

**1) INNER JOIN**

**2) LEFT JOIN ( LEFT OUTER JOIN )**

**3) RIGHT JOIN ( RIHGT OUTER JOIN )**

**4) FULL JOIN ( FULL OUTER JOIN )**

**5) UNION ALL ( it's not a join )**

**12/ 01/2021 Normalization**

**what is normalization ?**

Normalization is a technique of organizing the data in database , it is approch of minimizing the table to eliminate data redundancy and undesirable characterstics like insertion ,updation and deletion anomalies.

**Why we use normalization?**

* To eliminate redundant ( useless) data.
* Ensuring data dependencies make sense , data is logically stored .

#### **Insertion Anomaly**

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.

Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.

These scenarios are nothing but **Insertion anomalies**.

#### **Updation Anomaly**

What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.

#### **Deletion Anomaly:**

In our **Student** table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

**Types of normalizations:**

**1) 1NF**

**2) 2NF**

**3) 3NF**

**4) BCNF ( Boyce codd normalization form)**

**5) 4NF**

**1NF :**

## **Rules for First Normal Form**

#### **Rule 1: Single Valued Attributes**

Each column of your table should be single valued which means they should not contain multiple values. We will explain this with help of an example later, let's see the other rules for now.

#### **Rule 2: Attribute Domain should not change**

This is more of a "Common Sense" rule. In each column the values stored must be of the same kind or type.

**For example:** If you have a column dob to save date of births of a set of people, then you cannot or you must not save 'names' of some of them in that column along with 'date of birth' of others in that column. It should hold only 'date of birth' for all the records/rows.

#### **Rule 3: Unique name for Attributes/Columns**

This rule expects that each column in a table should have a unique name. This is to avoid confusion at the time of retrieving data or performing any other operation on the stored data.

If one or more columns have same name, then the DBMS system will be left confused.

#### **Rule 4: Order doesn't matters**

This rule says that the order in which you store the data in your table doesn't matter.

**Example:**

|  |  |  |
| --- | --- | --- |
| **Roll\_No** | **Name** | **Subject** |
| 101 | Raj | Python , java |
| 103 | Yash | C++ |
| 102 | Anil | C, Php |

the above table staisfied 3 rules in 1NF that is

* all column name should contain unique name
* Column have contain the same data type ,
* we have stored data in oreder we wanted .

But this table couldn’t follow the 4th rule that is each column contain single valued attribute only, but here we have multiple valued attributes .

**How to solve this problem ?**

It ‘s very simple break the multi valued attributes into single values

|  |  |  |
| --- | --- | --- |
| **Roll\_ No** | **Name** | **Subject** |
| 101 | Raj | Python |
| 101 | Raj | Java |
| 103 | Yash | C++ |
| 102 | Anil | C |
| 102 | Anil | Php |

here columns contain same data but subject column have atomic values .

**2NF: ( Second Normal Form )**

1.It should be in the First Normal form.

2.And, it should not have Partial Dependency.

**What is dependency?**

Dependency defines the relationalship between the two attributes .

it occures databse when information stored in the same database table uniquely determines other information stored in the same table .

Suppose we have student table in that there are different kind of columns are there like

**student\_id , student\_name , student\_address , student\_branch .**

Suppose I have made **student\_id** as primary so that we can access any column in the table by using the primary key , so that is the dependency ,

all columns are functionally depends on the **student\_id.**

**Partial dependency?**

It occurs when a non-prime attribues is functionally depend on the part of a candidate key .

**Example :**

**student table:-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| student\_id | Name | reg\_no | branch | address |
| 1 | A | CSE-18 | CSE | TN |
| 2 | A | IT-18 | IT | AP |
| 3 | B | CSE-18 | CSE | KN |
| 4 | C | CSE-18 | CSE | UP |

Here we student\_id is primary key

we can get any data by using primary key , but some times it’s not best every time .

**Subject table**

|  |  |
| --- | --- |
| **Subject\_Id** | **Subject\_Name** |
| 1 | Python |
| 2 | Java |
| 1 | C |
| 2 | C++ |
| 4 | Php |

here **Subject\_Id** is a primary key we can get the subject name by accessing the primary key

but here we have to make **canditate key** is **Student\_Id + Subject\_Id.**

To make another table to store student marks and the teacher name which subject they teach .

**Score table : Candidate key**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Score\_Id** | **Student\_Id** | **Subject\_Id** | **Marks** | **Teacher** |
| 1 | 1 | 1 | 80 | Mr.Pyhton |
| 2 | 1 | 2 | 77 | Mr.C++ |
| 3 | 2 | 1 | 82 | Mr.Java |
| 4 | 2 | 2 | 80 | Mr.C++ |
| 5 | 4 | 4 | 95 | Mr.Python |

here we have a primary key as **Score\_Id ,** But student\_id and subject\_id together makes a meaningful primary key .

Can we get the marks based on the student\_id no because we don’t know the which subject marks we want ,

like that we can’t get the marks by using subject\_id because we don’t know which student marks we want.

So we need **Student\_Id + Subject\_id** Combined together to get any row and column form the score table. Because it’s many to many relationships.

Here we have  **Teacher** column that can depend on the **subject\_id** . Not depend on the **Student\_Id ,** This is called parital dependency .

**To remove parital dependency ?**

Remove **Teacher column form Score table** and add to the subject table then it more appropriate or else make another table with **teacher\_id and teacher name**

|  |  |
| --- | --- |
| **Techer\_Id** | **Teacher\_Name** |
| 1 | Mr. Python |
| 2 | Mr. Java |
| 3 | Mr.C++ |
| 4 | Mr.C |
| 5 | Mr. Php |

**Third Normal Form (3NF)**

A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.

3NF is used to reduce the data duplication. It is also used to achieve the data integrity.

If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds atleast one of the following conditions for every non-trivial function dependency X â†’ Y.

X is a super key.

Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

**EMPLOYEE\_DETAIL table**:

**EMP\_ID EMP\_NAME EMP\_ZIP EMP\_STATE EMP\_CITY**

222 Harry 201010 UP Noida

333 Stephan 02228 US Boston

444 Lan 60007 US Chicago

555 Katharine 06389 UK Norwich

666 John 462007 MP Bhopal

**Super key in the table above:**

{EMP\_ID}, {EMP\_ID, EMP\_NAME}, {EMP\_ID, EMP\_NAME, EMP\_ZIP}....so on

Candidate key: {EMP\_ID}

**Non-prime attributes:** In the given table, all attributes except EMP\_ID are non-prime.

Here, EMP\_STATE & EMP\_CITY dependent on EMP\_ZIP and EMP\_ZIP dependent on EMP\_ID. The non-prime attributes (EMP\_STATE, EMP\_CITY) transitively dependent on super key(EMP\_ID). It violates the rule of third normal form.

That's why we need to move the EMP\_CITY and EMP\_STATE to the new <EMPLOYEE\_ZIP> table, with EMP\_ZIP as a Primary key.

**EMPLOYEE table:**

**EMP\_ID EMP\_NAME EMP\_ZIP**

222 Harry 201010

333 Stephan 02228

444 Lan 60007

555 Katharine 06389

666 John 462007

**EMPLOYEE\_ZIP table:**

**EMP\_ZIP EMP\_STATE EMP\_CITY**

201010 UP Noida

02228 US Boston

60007 US Chicago

06389 UK Norwich

462007 MP Bhopal

**Boyce Codd normal form (BCNF):**

BCNF is the advance version of 3NF. It is stricter than 3NF.

A table is in BCNF if every functional dependency X â†’ Y, X is the super key of the table.

For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

**Example:** Let's assume there is a company where employees work in more than one department.

**EMPLOYEE table:**

**EMP\_ID EMP\_COUNTRY EMP\_DEPT DEPT\_TYPE EMP\_DEPT\_NO**

264 India Designing D394 283

264 India Testing D394 300

364 UK Stores D283 232

364 UK Developing D283 549

In the above table Functional dependencies are as follows:

EMP\_ID â†’ EMP\_COUNTRY

EMP\_DEPT â†’ {DEPT\_TYPE, EMP\_DEPT\_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP\_DEPT nor EMP\_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

**EMP\_COUNTRY table:**

**EMP\_ID EMP\_COUNTRY**

264 India

364 UK

**EMP\_DEPT table:**

**EMP\_DEPT DEPT\_TYPE EMP\_DEPT\_NO**

Designing D394 283

Testing D394 300

Stores D283 232

Developing D283 549

**EMP\_DEPT\_MAPPING table:**

**EMP\_ID EMP\_DEPT**

D394 283

D394 300

D283 232

D283 549

Functional dependencies:

EMP\_ID â†’ EMP\_COUNTRY

EMP\_DEPT â†’ {DEPT\_TYPE, EMP\_DEPT\_NO}

Candidate keys:

For the first table: EMP\_ID

For the second table: EMP\_DEPT

For the third table: {EMP\_ID, EMP\_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

**Fourth normal form (4NF):**

A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.

For a dependency A â†’ B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

**Example**

**STUDENT**

**STU\_ID COURSE HOBBY**

21 Computer Dancing

21 Math Singing

34 Chemistry Dancing

74 Biology Cricket

59 Physics Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU\_ID, 21 contains two courses, Computer and Math and two hobbies, Dancing and Singing. So there is a Multi-valued dependency on STU\_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

**STUDENT\_COURSE**

**STU\_ID COURSE**

21 Computer

21 Math

34 Chemistry

74 Biology

59 Physics

**STUDENT\_HOBBY**

**STU\_ID HOBBY**

21 Dancing

21 Singing

34 Dancing

74 Cricket

59 Hockey

**5th Normal Form :**

A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.

5NF is also known as Project-join normal form (PJ/NF).

**Example:**

**SUBJECT LECTURER SEMESTER**

Computer Akash Semester 1

Computer John Semester 1

Math John Semester 1

Math Akash Semester 2

Chemistry Praveen Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER SUBJECT

Semester 1 Computer

Semester 1 Math

Semester 1 Chemistry

Semester 2 Math

**P2**

Computer Anshika

Computer John

Math John

Math Akash

Chemistry Praveen

**P3**

**SEMSTER LECTURER**

Semester 1 Anshika

Semester 1 John

Semester 1 John

Semester 2 Akash

Semester 1 Praveen