Carbon Accounting and Sustainable Designs in Product Lifecycle Management

Prof. Deepu Philip
Department of Management Sciences

Dr. Amandeep Singh Oberoi
Imagineering Laboratory

Dr. Prabal Pratap Singh
Department of Management Sciences

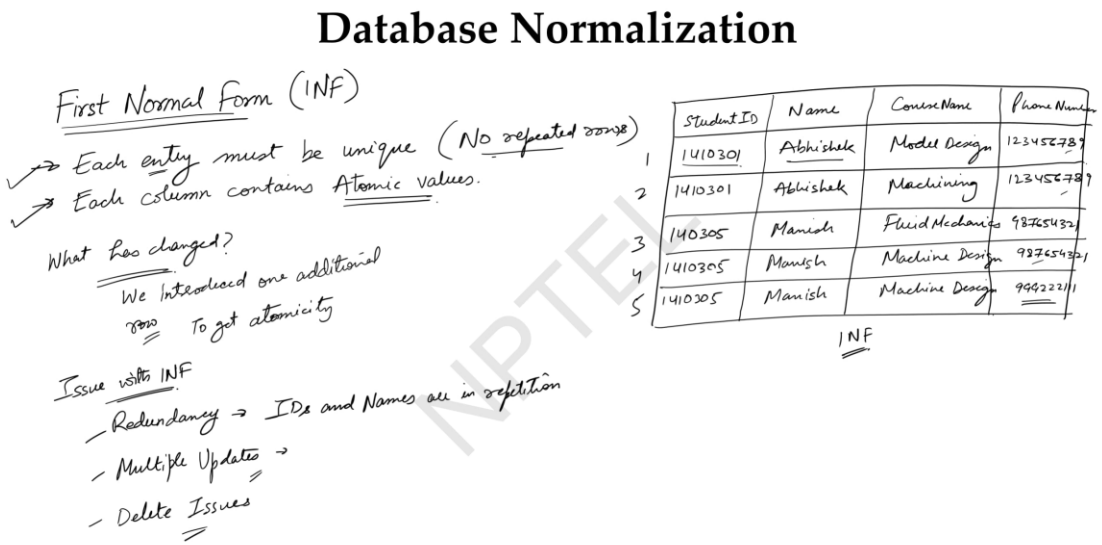
Indian Institute of Technology, Kanpur

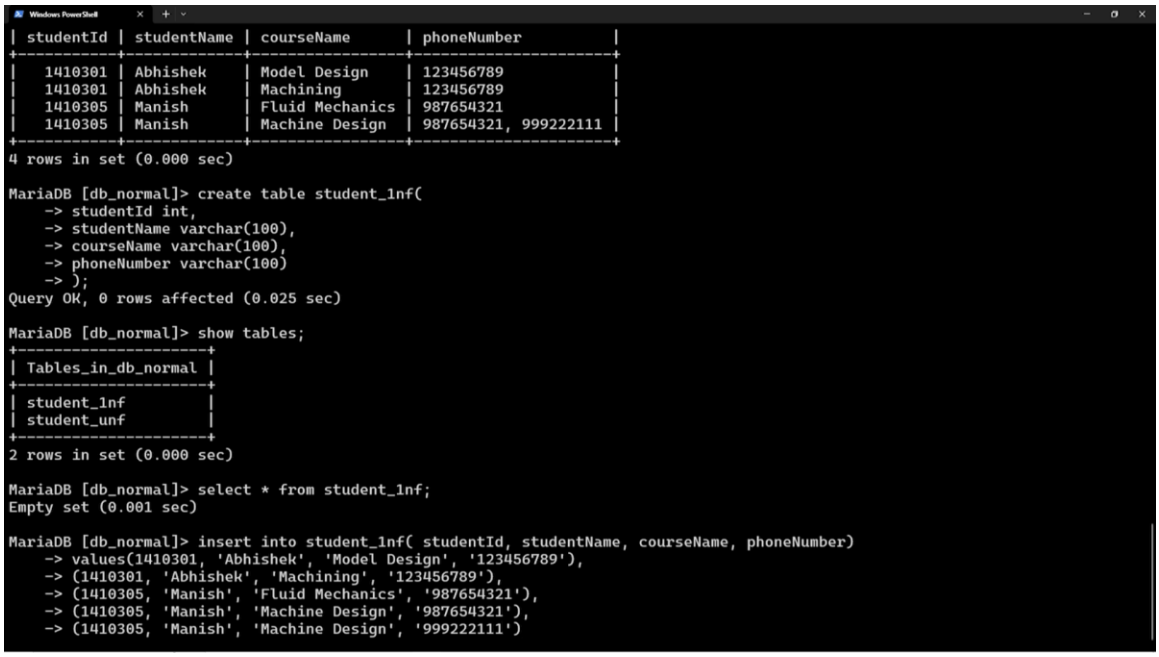
Week 11

Lecture 51

Database Normalization (Part-2)

So, let us switch to the first normalized form and let us see how we can transfer our unnormalized form to the first form.





```
| studentId | studentName | courseName      | phoneNumber          |
+-----------+-------------+-----------------+----------------------+
|   1410301 | Abhishek    | Model Design    | 123456789            |
|   1410301 | Abhishek    | Machining       | 123456789            |
|   1410305 | Manish      | Fluid Mechanics | 987654321            |
|   1410305 | Manish      | Machine Design  | 987654321, 999222111 |
+-----------+-------------+-----------------+----------------------+
4 rows in set (0.000 sec)

MariaDB [db_normal]> create table student_1nf(
    -> studentId int,
    -> studentName varchar(100),
    -> courseName varchar(100),
    -> phoneNumber varchar(100)
    -> );
Query OK, 0 rows affected (0.025 sec)

MariaDB [db_normal]> show tables;
+---------------------+
| Tables_in_db_normal |
+---------------------+
| student_1nf         |
| student_unf         |
+---------------------+
2 rows in set (0.000 sec)

MariaDB [db_normal]> select * from student_1nf;
Empty set (0.001 sec)

MariaDB [db_normal]> insert into student_1nf( studentId, studentName, courseName, phoneNumber)
    -> values(1410301, 'Abhishek', 'Model Design', '123456789'),
    -> (1410301, 'Abhishek', 'Machining', '123456789'),
    -> (1410305, 'Manish', 'Fluid Mechanics', '987654321'),
    -> (1410305, 'Manish', 'Machine Design', '987654321'),
    -> (1410305, 'Manish', 'Machine Design', '999222111')
```

So, it is also known as 1 NF . So the requirements of this normal form is that each entry in your table, each entry must be unique. Unique in the sense that there should be no repeated rows. This is the first requirement and each column entry means the row that is a tuple and each column that is an attribute.

So each column contains atomic values that is individual values. So to create our first normal form from the previous table we can again create a table our fields are student id name course name phone number. So, the first row should be 1410301. Name is Abhishek. Fourth name is Mortal Design.

Phone number is 123. Next row also belongs to Abhishek. He is doing machining course as well the next student was Manish so his student id was 305 Manish fluid mechanics. So now you may ask what has changed from the previous unnormalized form. So this is the first normal form 1 NF table right for the table which we are studying.

So in the unnormalized form we had four rows only and there are composite values but in the first normal form we have 12345 rows, right. So we have increased one row removed the composite variable so these were the requirements for first normal form each entry must be unique. So in the unnormalized form if you see there are no repetitions repetitions could be that the same let's say the third row could be again inserted into the table. So, all the values of this row could be repeated again. So, your unnormalized form have these kinds of repetitions.

You should directly remove those rows because you don't need redundancy, right. So, the first requirement of the 1 NF is you have no repeated rows. So, we don't have any repeated rows. Now, the next is each column contains atomic values. So, if you see the unnormal form, you have student ID are unique and there is nothing you can do about it.

Same with the name. But in the phone numbers, there are some rows where there are multiple phone numbers. So, these are not atomic, right. So to create atomic columns, you need to divide these into multiple columns. So that is what we have done here.

We used this row and divided into another row as well. So all the values remain the same, but the phone number is different from this. So what has changed? We introduced one additional row to get atomicity, right. So there are issues with this one in a form as well.

First is redundancy. Like you can see there are multiple rows where IDs and names are in repetition. And you need to do multiple updates. How? So if any student changes his or her phone number, then you need to update in each row where the phone number is stored. So this is the problem that you need to do multiple updates.

And the last is delete issues. So let's say that student gets unenrolled from these courses that they have enrolled in. But all the information about students are stored in this table. And if let's say Abhishek drops his course on model design and machining, then his phone number is also lost. But maybe he is not doing these courses or maybe he is not doing any courses in this current semester.

But his phone number that is his details will also get lost from the school or college management system, right. So this is the problem of daily tissues. So now let us create our first normal form table in the MariaDB server. So we can create our table again. Create table student 1 right.

And your schema remains the same. So you can check how many tables you have now. There are two tables. Student Unnormalized Form and Student 1NF. Now you can check.

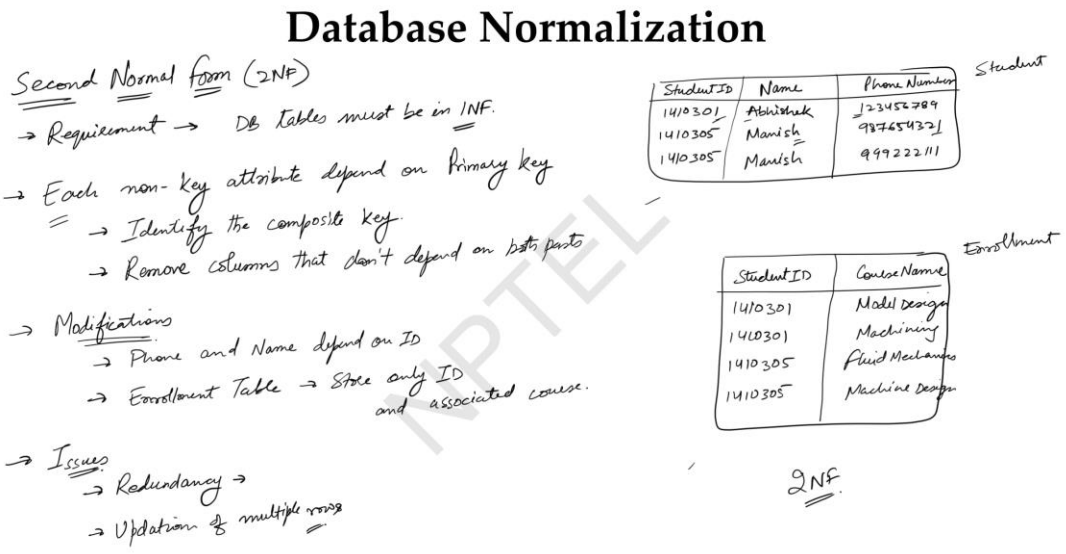
Select star from Student 1NF. And there is nothing in this table right now. So you can start inserting the table. Insert N2 student one ns. Now we will fill the values in a single command this time so you can write the first is 1410301.

So this is the one complete row the first row second will be 1410301 name is again abhishek. Third row is about Manish 1410305 is the row number he is enrolled in fluid mechanics this is string we should use single quotation marks. And his phone number is

98765 again it is about Manish. So this is the last row, which is the new row for the first normal form. So this is 1410305.

It is about Manish. Controlled in machine design. And his second phone number is 999. So we have provided five rows and there are five records in this there are no duplicates and zero warnings now you can again check okay. We can check the contents of our first normal form by providing select star from student_1 minus.

So now there is an atomicity in this table and the only difference is we have a new row, right.



So now let us move to the second normal form is also known as 2NF. And the very first requirement of this is that your database table requirement your database table must be in the first normal form DB tables. Which means that whatever is the requirement of the first normal form like this no repeated rows and no atomic values these should be present in your database table. After that only you can use the second normal form and make your tables more efficient right.

So now what do we do here we try to remove the non-key attributes and try to connect it with the primary keys only. So each non key attribute should depend on primary key. So we already know what is a primary key it is the identifying information for a particular row in the database table. So to do this step we need to do the following we first need to

identify the composite key. So in our table that is in our first normal form the composite key could be student ID and the name of the student.

So with these two identifying information every other thing is related to this, right. And then we should remove columns that don't depend on both parts of the primary K. So now we will at this stage we will divide a single table. This single table of five rows into two different tables to get it into second normal form. So the first table would be student ID, name and phone number.

And his phone number was 123. Sir, student was 1410305. His name was Manish. And using these two composite keys, the next non-key attribute is his phone number, which was 9876. Since Manish has two phone numbers, so we have one more row in this table 1410305.

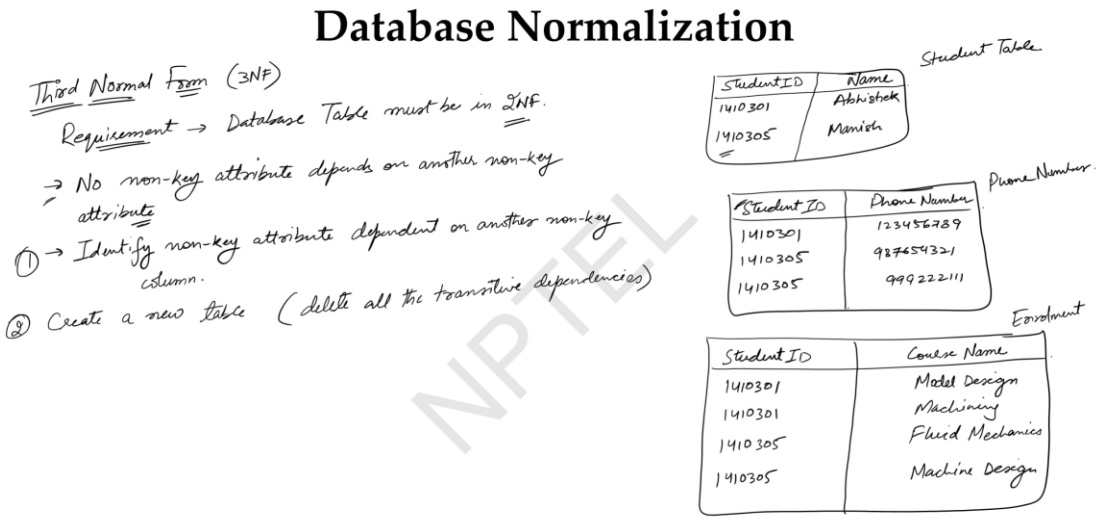
And his other phone number was okay. So this is the first table of the second normal form the second table could be the student id and the course name. So let's say for the first student 1410301 which was Abhishek. He is enrolled in machine design model design second he is also enrolled in machining course. The student ID 305 is associated with fluid mechanics, right.

And he is also associated with machine design. Now we have two tables. So this is the 2NF form and this student ID is associated with the course name and student ID and name is associated with the phone number in this first table. So what are the modifications to achieve this second normal form, phone and name depend on id. And we have created a new enrollment table.

So let's say this is the student table and this is the enrollment table enrollment of in courses. So enrollment tables will store only id and associated course, right. So this is the second normal form and this is an activity to all the students that they can write their queries in the MariaDB server to create these two different tables and try to connect these two with each other. Now let us switch to the issues because we have more normal forms to study. So what is the problem with the second normal form?

So there are some issues. The first issue is the redundancy and this is the major issue with this form. And this that let's say this user manish changes his phone number. Then we still need to update it into different rows. So this is a very small table but let's say you are managing a complete school or college management system. So if we are storing the

tables in this second normal form then the problem is redundancy and updation of multiple rows okay.




```
Windows PowerShell
| Database           |
+--------------------+
| db_normal          |
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.000 sec)

MariaDB [(none)]> use db_normal;
Database changed
MariaDB [db_normal]> show tables;
+----------------------+
| Tables_in_db_normal  |
+----------------------+
| student_1nf          |
| student_unf          |
+----------------------+
2 rows in set (0.000 sec)

MariaDB [db_normal]> select * from student_1nf;
+-----------+-------------+-----------------+-------------+
| studentId | studentName | courseName      | phoneNumber |
+-----------+-------------+-----------------+-------------+
|   1410301 | Abhishek    | Model Design    | 123456789   |
|   1410301 | Abhishek    | Machining       | 123456789   |
|   1410305 | Manish      | Fluid Mechanics | 987654321   |
|   1410305 | Manish      | Machine Design  | 987654321   |
|   1410305 | Manish      | Machine Design  | 999222111   |
+-----------+-------------+-----------------+-------------+
5 rows in set (0.000 sec)

MariaDB [db_normal]> create table student_3nf(
    -> studentId int
```

So now let us switch to the third normal form. Again, the requirement remains the same that you cannot create a 3NF table without reaching until 2NF. So, before doing the third normalization, the database table must be in 2NF, right. And what do we do in this stage

that there should be no non-key attribute that depends on another non-key attribute. So what does this mean?

So if you see that currently name is a non-key attribute and you can uniquely identify a row by using the student ID because usually the student ID is a unique identifier of a student. So this is a non-key attribute. Phone is a non-key attribute. The name is a non-key attribute. So there are multiple students with the name Abhishek.

So let's say there are two Abhishek and they have different phone numbers and they are different people. So if you assign name and phone numbers in a single table and you are saying that these two non-key attributes can be stored in a single table, then there should be many more redundancies. And these are not actual redundancies. So there are two different kinds of Abhishek, two different students with the same name Abhishek and they have different phone numbers. So that is why we must associate the non-key attributes like name or phone numbers in different tables with the student ID or a key attribute, right.

So this is what we are saying here in the first requirement. So how to achieve this? So, first try to identify non-key attribute dependent on another non-key column. Then create a new table. So this new table will delete all the transitive dependencies

So in 3 NF we will further divide these two tables into one more table. So the first table will have a student id and name associated with it. So we have 1410301 and the name is abhishek for the student. The next student was 1410305 and the name was manish. Now this is a small table and it is only telling that we have two students in our school management system.

And this is a student table and this is the primary key and this is the data associated with the primary key right the second table again has a student id associated with it. Because it is the identifying information and the phone number right so phone number for the student was one two three. So now at this stage we have also divided the phone numbers and student id into another table. So let's say this table is phone number table and this is again the student id is the primary key attribute and the third table will store the information about courses where these roll numbers are enrolled. So this is the enrollment table it will also have student id and course name, right.

So this is the enrollment table. Now, these three tables will be efficiently stored on your MariaDB server. And this is not the ultimate form, but this is the most useful form. And

mostly database developers will at least try 3NF. After that, there are advanced forms as well like BCNF.

But 3NF is a very useful form and you can have multiple tables. So storing multiple tables is not an issue in your MariaDB or any DBMS server. But what they do is the links between these tables will try to retrieve or manipulate the data very efficiently and quickly. So let us try to create these tables in your MariaDB server now. So we will first create table student let's go 3NF and we have student id and the constraint on this.

So now we are first using our constraints which we studied earlier. So student id is a primary constraint so we can write primary key and the second attribute for this field is student name it is aware care with 100 characters limit and it has no constraint. So this is the simple table definition and now we have a student table for third normalized form. Now we can insert data into this table using insert into student 3NF student id, student name. What are the values which we want to store values and let us provide multiple values together.

So the first row was the 1410301 as a student id and the name was abhishek. The second row for the table was 1410305 and the name was Manish this is these are the only two rows in this table, right. So select star from student. So this is the first table for the student data, right. And let us now create the second table.

So create table phone 3NS and it should have student id. which should be int the phone number should be string. And now let us define the primary key for this table. Primary key is student id and phone number. And since this table is connected we should also write foreign key. Foreign key is student id and it references the table student 3NF and from there the student id field, right.

So you can check what is the schema of this table by using DESC that is described statement phone_3 NF . So there are two primary keys and these are connected okay. Now you can insert the data so insert into phone 3 NF _3 NF student id. So now we have error. Why?

Because while filling the student ID here, I have filled a wrong value that is 14101301 and while filling the current table, I was filling with 1410301. So why the database system has found that these two values are not correct because we have created a foreign key with the student ID. So if you are not providing the same student ID that is available in the previous table that means there is something wrong. And your DBMS is smart

enough to find out and then it is highlighting that there is an error either you should correct your previous value or you should correct that value in the current table. So, let us update your table.

As I have told you that we had a wrong value filled in the table, which was this 14101301, right. With the student name, associated student name was Abhishek. So to correct that we can write this update command that is update which table that student 3 nf table and set the student id to 1410301 where student id is this wrong value. So we are updating this information in the table. Now the current table is correct.

So, now we can input the same value. So, this was the actual thing that your RDBMS will tell you whenever you will try to connect it with the foreign key. So, foreign key assignment and the foreign key checks, these constraints will be helpful while filling your database. So, find the same statement, insert it into student 3NF . So, we were filling this phone table.

So, cert into phone 3NF. Thank you. So now we have our student information regarding their phone numbers in the phone 3NF. And this table is not a simple table. It is connected with foreign key from the with the student ID.

That is the actual student 3NF table, right. Now let us create the last table that is the table for the enrollment ids and the course enrollment information. So create table first we need to create the scheme of the table. So enrollment_3NS and it has student id that is int. And let us now define the primary key for this table the same student id.

The course name again define the foreign key to connect it with the actual tables defined above. So foreign key is student id and it references to the student 3 NF table from that table it will use the student id as the field attribute. We missed a comma here. Now both of these fields are assigned as a primary key and we have created the schema. So the last step is to insert the values insert into enrollment enrollment_3NF that has student id.

And course name what are the values so the first student id is 1410301 the course name was model design. Second entry is also about 1410301 course name was machining third is 1410305 course name was fluid mechanics. And the last entry was 1410305 and the course was machine design. So we can cross check our data in the enrollment table as well. And this is how we have the third normal form of this table, right.

So with this, we have learned about three normal forms of the table from unnormalized form to first normal form, then second normal form and the last is the third normal form.

Now, in the upcoming week, we will try to develop our own carbon accounting database. And we will try to understand how to create different tables and utilize that into a user interface of the database by creating that user interface using Python programming language.

Thank you.