Normalisation

Database normalization process is used to reduce data redundancy and improve data integrity. It was first proposed by Edgar F. Codd as part of his relational model.

Normalization is the process of organizing the columns (attributes) and tables (relations) of a relational database to minimize data redundancy.

It is a systematic approach of decomposing tables to eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.

It is a multi-step process that puts data into tabular form, removing duplicated data from the (relations) tables.

Normalization is used for mainly two purposes,

1. Eliminating redundant(useless) data.
2. Ensuring data dependencies make sense i.e data is logically stored.

If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anomalies are very frequent if database is not normalized.

We have to look at the following database (consisting of one table, with the primary key = EmployeeID) that was designed to store the following information on a company's employees and departments:

• Employee ID (primary key)
• Employee Name
• Employee Salary
• Department Name
• Department Location
• Department Budget

EmployeeID*	EmployeeName	Salary	DeptName	DeptLocation	DeptBudget
100	Sean	35,000	Sales	Dublin	750,000
101	Mary	36,000	Sales	Dublin	750,000
102	John	40,000	Sales	Dublin	750,000
104	Albert	55,000	R&D	Galway	1,500,000
105	Conor	52,000	R&D	Galway	1,500,000
106	Maeve	50,000	R&D	Galway	1,500,000
107	Tom	50,000	R&D	Galway	1,500,000
108	Alice	44,500	HR	Limerick	250,000

As It is obvious from the above table there is **redundant data** for dept (name, budget and location)

Insertion Anomaly

Suppose we need to add a new Department, until and unless an Employee is for that department is there a department cannot be inserted, or else we will have to set Employee as NULL. But we cannot put the NULL value for primary key.

EmployeeID*	EmployeeName	Salary	DeptName	DeptLocation	DeptBudget
100	Sean	35,000	Sales	Dublin	750,000
101	Mary	36,000	Sales	Dublin	750,000
102	John	40,000	Sales	Dublin	750,000
104	Albert	55,000	R&D	Galway	1,500,000
105	Conor	52,000	R&D	Galway	1,500,000
106	Maeve	50,000	R&D	Galway	1,500,000
107	Tom	50,000	R&D	Galway	1,500,000
108	Alice	44,500	HR	Limerick	250,000
NULL	NULL	NULL	finance	Limerick	200,000

Also, if we have to insert data of 100 Employee of same Department, then the Department information will be repeated for all those 100 Employee. These both scenarios are nothing but Insertion anomalies.

Updation Anomaly

What if any Department Location or budget is changed? In that case all the Employee records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.

EmployeeID*	EmployeeName	Salary	DeptName	DeptLocation	DeptBudget
100	Sean	35,000	Sales	Dublin	750,000
101	Mary	36,000	Sales	Dublin	750,000
102	John	40,000	Sales	Dublin	750,000
104	Albert	55,000	R&D	Galway	1,600,000
105	Conor	52,000	R&D	Galway	1,500,000
106	Maeve	50,000	R&D	Galway	1,600,000
107	Tom	50,000	R&D	Galway	1,600,000
108	Alice	44,500	HR	Limerick	250,000

Deletion Anomaly

In our Employee table, two different information are kept together, Employee information and Department information. Hence, if there is only one employee in a department and that employee leave job and we need to delete employee. If Employee records are deleted, we will also lose the Department information. This is Deletion anomaly. As if Alice left the job and if we delete his record from the table the information about the HR department well be deleted as well.

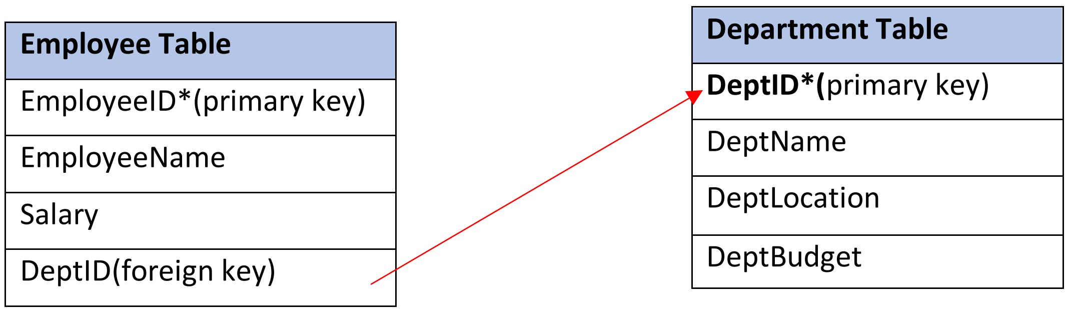
EmployeeID*	EmployeeName	Salary	DeptName	DeptLocation	DeptBudget
100	Sean	35,000	Sales	Dublin	750,000
101	Mary	36,000	Sales	Dublin	750,000
102	John	40,000	Sales	Dublin	750,000
104	Albert	55,000	R&D	Galway	1,500,000
105	Conor	52,000	R&D	Galway	1,500,000
106	Maeve	50,000	R&D	Galway	1,500,000
107	Tom	50,000	R&D	Galway	1,500,000
~~108~~	~~Alice~~	~~44,500~~	~~HR~~	~~Limerick~~	~~250,000~~

As we can see from the table that department information is not dependent on employee id and there is duplication in data to reduce that we can divide the table into two tables.

Split the table into two tables Employee and Department, In Employee we need to only put in the department ID for each employee

Employee ID*	Employee Name	Salary	DeptID
100	Sean	35,000	1
101	Mary	36,000	1
102	John	40,000	1
104	Albert	55,000	2
105	Conor	52,000	2
106	Maeve	50,000	2
107	Tom	50,000	2
108	Alice	44,500	3

Dept ID*	Dept Name	DeptLocation	DeptBudget
1	Sales	Dublin	750,000
2	R&D	Galway	1,500,000
3	HR	Limerick	250,000



Employee ID*	Employee Name	Salary	DeptID
100	Sean	35,000	1
101	Mary	36,000	1
102	John	40,000	1
104	Albert	55,000	2
105	Conor	52,000	2
106	Maeve	50,000	2
107	Tom	50,000	2
108	Alice	44,500	3
109	Mark	40,000	NULL

Dept ID*	DeptName	DeptLocation	DeptBudget
1	Sales	Dublin	750,000
2	R&D	Galway	1,500,000
3	HR	Limerick	250,000
4	Finance	Limerick	20,000

If there is a new department added before getting an employee than it will not be an issue and same way if a new employee joins without assigned department it can be added. It removes the insertion anomaly

The relationship between two tables is DeptID. DeptID is primary key in Department table and is it is as foreign key in employee table

As for updation if the budget or location of a department is changed there will be needed to update only one row in department table. this removes the updation anomaly

As there is foreign key reference constraint so if need to delete a department that has any employee in employee table related , it cannot be deleted, it improves the data consistency. If any employee left the job and we need to delete that record it does not delete the department data as we see above in deletion anomaly

We use foreign key to link the tables that will keep the data consistency as we put constraint (referential integrity)

Normalization rules are divided into the following normal forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF
5. Fourth Normal Form

First Normal Form (1NF):

For a table to be in the First Normal Form, it should follow the following 4 rules:

1. It should only have single(atomic) valued attributes/columns.
2. Values stored in a column should be of the same domain
3. All the columns in a table should have unique names.
4. And the order in which data is stored, does not matter.

Second Normal Form (2NF)

For a table to be in the Second Normal Form,

1. It should be in the First Normal form.
2. And, it should not have Partial Dependency.

Third Normal Form (3NF)

A table is said to be in the Third Normal Form when,

1. It is in the Second Normal form.
2. And, it doesn't have Transitive Dependency.(functional dependency)

Boyce and Codd Normal Form (BCNF)

Boyce and Codd Normal Form is a higher version of the Third Normal form. This form deals with certain type of anomaly that is not handled by 3NF. A 3NF table which does not have multiple overlapping candidate keys is said to be in BCNF. For a table to be in BCNF, following conditions must be satisfied:

R must be in 3rd Normal Form

and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Fourth Normal Form (4NF)

A table is said to be in the Fourth Normal Form when,

It is in the Boyce-Codd Normal Form.

And, it doesn't have Multi-Valued Dependency

References:

1. Normalization of Database online available at: https://www.studytonight.com/dbms/database-normalization.php accessed on 17/04/2020