Applied Databases Project Normalisation

May 4, 2020

Applied Databases Project
Angela Carpenter
Question 4.3 Normalisation

1 Database Design

Examine 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 - Employee Name - Employee Salary - Department Name - Department Location - Department Budget

Give your opinion, using examples from the data below, on whether or not the current database is good or bad.

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	55,000	R&D	Galway	1,500,000
107	Tom	50,000	R&D	Galway	1,500,000
108	Alice	44,500	HR	Limerick	250,000

A database is a collection of related data organised so that the data can be easily managed and updated. One of the advantages of using a DataBase Management System such as MySQL is that it can be used to control redundancy where data only needs to be stored once but can be accessed by many users. Normalisation organises the attributes and relations of a relational database to minimise such data redundancy by breaking tables with many attributes down into several smaller tables with relationships defined between the tables. Normalisation makes it easier to update a database and makes it less prone to anomalies such as update or deletion anomalies. Normalisation was developed by an IBM researcher E.F.Codd in the 1970's. Normalisation of a database is achieved by following a set of rules called **forms** when creating the database. (See references below) - First Normal Form (1NF) - each column is unique and each attribute should contain only a single value.

- Second Normal Form (2NF) - to be 2NF, an entity must already be in 1NF and all attributes

within the entity solely on the unique identifier of the entity.

- Third Normal Form (3NF) - to be 3NF an entity must already be in 2NF and no column entry should be dependent on any other entry other than the key for the table. If such an entity exists it should be moved outside the table.
- There are also BCNF (Boyce-Codd Normal Form) and Fourth Normal Forms but these cannot be achieved if the database does not conform to the earlier form rules.

There is already a lot of data duplication in this simple database structure consisting of a single table of data. This will result in increased storage being required and will decrease performance. It also makes it much more difficult to make changes. For example the same `DeptName` attribute, `DeptLocation` and `DeptBudget` are repeated across several rows. This current database structure is also very likely to give rise to anomalies and inconsistencies. The rows that store data in each table should have a unique primary key to identify the record. Each column / attribute should hold a single piece of data. The same attibutes should not be repeated across tables. The department details such as location and budget are not directly related to the employee whose employeeID is the primary key. In this table, the primary key is unique to the employee but it is not unique to the department. The same department, for example Sales has 3 primary keys which relate to 3 different employees. Only related data should be stored in the same table.

The same data here is recorded across several rows. Update anomalies are very likely. If an attribute had to be changed such changes would have to be made across many rows. For example, if a department budget is changed as it is likely to be over time, this would require all the records containing that department to be individually changed. If a department was to change name or even change location, then similarly changes would need to be made across all records in the database. For example the Sales or R&D departments might change locations or have an increase or decrease in the department's budget. This would require multiple updates to be made. If the updates were not made at the same time then the database would be in an inconsistent state. Updating many records with the same data is time-consuming and can be costly. If the Sales department's budget increased to 900,000 and this was updated for Sean with EmployeeID 100, without explicitly updating Mary and John's records also there would be inconsistencies in the data.

Deleting a record could result in more data being lost than was intended. For example if Alice with EmployeeID 108 who is the only employee in the HR department left the company, the data for the HR department including it's budget and location would also be lost. As there is no record with primary key employeeID 103, then this could indicate that a record for employee 103 had previously existed but was deleted and with it some details for another department that is now lost from the database. A deletion anomaly occurs when deletion of a data record results in losing some unrelated information that was stored as part of the record that was deleted from a table.

An insertion anomaly would occur when data cannot be recorded until more information is known. This problem occurs when the insertion of a record is not possible without adding some additional unrelated data to the record. For example, if a new department was to be created and was not yet populated with employees, then the record could not be created as there would be no primary key. MySQL does not allow null values for a primary key.

This database could be normalised by referring to the normal form rules mentioned above. This database contain only a single value in each column. While first normal form eliminates repeating groups, it does not reduce redundancy.

To make this database conform to 2NF rules the table should be divided into at least two tables. Having data in separate tables would prevent deletion, insertion and update anomalies. As it is, attributes that relate to a department do not depend solely on the unique identifier of each

attribute. Not all attributes in the table are fully dependent on the primary key EmployeeID. The only information that is fully dependent on the primary key is the employee information.

There should be an Department table specifically to hold the data relating to each of the company's departments, with a department id uniquely identifying each department as it's primary key. The data specific to the employee should be stored with other employee specific data in a separate table with EmployeeID as the primary key. The two tables could be linked using the department id as a foreign key. This department id does not need to have the exact same name in both tables.

Second normal form aims to reduce redundant data being stored in memory. The department name, location and budget does not need to be stored in memory for each employee in the department. Instead the data for each department can be stored just once.

- The Employee table would contain the EmployeeID, EmployeeName, Salary and DeptID attributes.
- A Department table could contain the DeptName, DeptLocation, DeptBudget and DeptID attributes.

The primary key of the department table could be called DeptID or something similar.

When multiple tables are created, the link between the tables is defined by the foreign key which acts as a constraint on what can be put in the table. The tables can be joined together using the foreign keys. Therefore while not all the same information will be stored in the same table, the tables can be very easily joined using the foreign keys and the information can be easily located and retrieved in this way. A foreign key constraint could be used so that a department id could not be added to an employees record if a department id does not already exist in the Department table.

The Department table could be joined with the Employee table in a query to see what employees belonged to a department. The Employee table could be joined with the Department table to see the department name, location and budget for employees.

Proposed Employee Table

EmployeeID	EmployeeName	Salary	DeptID
100	Sean	35,000	S100
101	Mary	36,000	S100
102	John	40,000	S100
104	Albert	55,000	RD200
105	Conor	52,000	RD200
106	Maeve	55,000	RD200
107	Tom	50,000	RD200
108	Alice	44,500	HR300

Proposed Department Table

DeptID	DeptName	DeptLocation	DeptBudget
S100	Sales	Dublin	750,000
RD200	R&D	Galway	1,500,000
HR300	HR	Limerick	250,000

To be 3NF an entity must already be in 2NF and no column entry should be dependent on any other entry other than the key for the table. If such an entity exists it should be moved outside the table.

In the early stages, the database structure here with 2 tables should work. Employee name depends on EmployeeID, DeptBudget and department name will depends on the department ID. However over time, as salaries and department budgets change, the salary and department budget fields no longer depend solely on the EmployeeID and DeptID respectively.

A Salaries table could be created that links to both the Employee and Department tables. The primary key of the Salaries table could be a composite key consisting of the EmployeeID and the Start date of the particular salary for that employee. A Budgets table could also be created whose primary key is a composite key using the Department id and a from date as a composite primary key. The Salaries table would link to the Employee table using the EmployeeID as a foreign key. The Salaries table could also link to the Department table using the department id as a foreign key. The budget for a department is related to the salaries of the employees in that department. The Departments table could link to the Budgets table using a foreign key.

Having the salaries data in a separate table might be better for confidentiality reasons also. Different users to the database could be given different access levels. An Employee table is likely to have more attributes such as birth-dates, first names, last names, gender, hire date.

Proposed Employee Table

EmployeeID	EmployeeName	DeptID
100	Sean	S100
101	Mary	S100
102	John	S100
104	Albert	RD200
105	Conor	RD200
106	Maeve	RD200
107	Tom	RD200
108	Alice	HR300

Proposed Department Table

DeptID	DeptName	DeptLocation
S100	Sales	Dublin
RD200	R&D	Galway
HR300	HR	Limerick

Proposed Salaries Table

EmployeeID	DepartmentID	Salary	Start_Date
100	S100	35,000	
101	S100	36,000	
102	S100	40,000	
104		RD200	55000
105	RD200	52000	
106	RD200	50000	

EmployeeID	DepartmentID	Salary	Start_Date
107	RD200	50000	
108	HR300	44500	

Proposed Budget Table

DeptID	DeptBudget	from_date
S100	750,000	
RD200	1,500,000	
HR300	250,000	

2 References

- https://en.wikipedia.org/wiki/Database_normalization
- https://www.w3schools.in/dbms/database-normalization/
- https://opentextbc.ca/dbdesign01/chapter/chapter-12-normalization/
- https://www.guru99.com/database-normalization.html
- https://www.geeksforgeeks.org/first-normal-form-1nf/