4.3 Normalisation

- come back and tidy up before submitting!!!
- just some notes from looking up normalisation

4.3.1 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.

proposed_table

A database is a collection of related data which is organised in such a way 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 is the process of organising the attributes and relations of a relational database to minimise data redundancy. Normalisation breaks a tables that has many attributes down into several smaller tables and relationships are defined between the tables. Normalisation makes it easier to update a database and makes it less prone to anomalies. A database that is not normalised cannot be easily managed and may result in update or deletion anomalies. A normalised table has more than one table with links between the tables instead of all the data being stored in a single table. Instead of duplicating the same data, it can instead be referenced from another table.

The table here shows much duplication with the same `DeptName` attribute, `DeptLocation` and `DeptBudget` repeated. Not only does it make it harder to read, it also makes it harder to manage the database. 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, this would require all the records that have 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. 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.

- First Normal Form (1NF) - each column is unique and each attribute should contain only a single value. The proposed database table here does follow INF rules.
- Second Normal Form (2NF) - if the entity is already in 1NF and all attributes within the entity should depend solely on the unique identifier of the entity. This is not the case for this database. Attributes relating to a `Department` do not depend solely on the unique identifier of each attribute. A table specifically relating to the company's department with a department id uniquely identifying each department would follow 2NF.

- 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. As the database in question here does not follow 2NF then it cannot be considerd 3NF. There are also BCNF (Boyce-Codd Normal Form) and Fourth Normal Forms.

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

It would be better if the data was separated out into at least two tables with a table containing the personal details of an employee and a separate table for the department details. The two tables could be linked using a foreign key. The employee table would contain the EmployeeID, EmployeeName and Salary attributes. A department table could contain the DeptName, DeptLocation and DeptBudget attributes. This table would require a primary key which could be called DeptID or something suitable like that. The department could reference the employee table if a query was required to see what employees belonged to a department. The employee table could reference the department table to see the department name, location and budget for employees.

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 columns / 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.

Employee Table

EmployeeID	EmployeeName	Salary
100	Sean	35,000
101	Mary	36,000
102	John	40,000
104	Albert	55,000
105	Conor	52,000
106	Maeve	55,000

EmployeeID	EmployeeName	Salary
107	Tom	50,000
108	Alice	44,500

Department Table

DeptID	DeptName	DeptLocation	DeptBudget
100	Sales	Dublin	750,000
102	R&D	Galway	1,500,000
103	HR	Limerick	250,000

Come back to this!

maybe have a third table for salary details

Data duplication will result in increased storage being required and will decrease performance. It also makes it much more difficult to make changes.

- Only related data should be stored in the same table.

If an employee was to leave the company and their record was being dropped from the database, then deleting their record could result in the department details being deleted?? i think...

Insert anomalies.

An insert anomaly would occur when data cannot be recorded until more information is known. 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. The primary key being the employee ID does not make sense anyway for a department to use it as its primary key.

Update Anomalies

The same data here is recorded across several rows. 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. Otherwise inconsistencies would occur. 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.

Deletion Anomalies

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 might indicate that a record for employee 103 was deleted and with it some details for another department that is now lost.

Finding Information

When tables are created, the link between the tables is defined by the foreign key which is 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 foreigh keys.

Having data in separate tables would prevent deletion anomalies. A primary key could also be a composite key.

The `show create table` command could be used to see any foreign keys that might exist on a table.

A foreign key constraint could be used so that a deptID could not be added to an employees record if it does not already exist in the department table.

dept_no or deptID, department name.

Perhaps for confidentiality reasons it might be even better to have 3 tables.

a salaries table to hold the current salaries as well as previous salaries relating to each employee.

An employee table would probably have more attributes such as birthdates, first names, last names, gender, hire date. A salaries table might have attributes for previous salaries and dates.

These are the tables from the employee2 databases used for the lectures.

A `salaries` table had a `emp_no` as the primary key, a salary field, from and to dates and a dept_no. This had a composite primary key (`emp_no`,`from_date`) and two foreign key constraints.

```
  KEY `emp_no` (`emp_no`),
  KEY `dept_no` (`dept_no`),
  CONSTRAINT `salaries_ibfk_1` FOREIGN KEY (`emp_no`) REFERENCES
`employees` (`emp_no`) ON DELETE CASCADE,
  CONSTRAINT `salaries_ibfk_2` FOREIGN KEY (`dept_no`) REFERENCES `dept`
(`dept_no`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 |
```

The `employees` table had emp_no, birth_date, first_name, last_name, gender and hire_date with emp_no as the primary key The `dept` table had a dept_no as the primary key.

References

- Wikipedia - Database Normalisation
- W3Schools - Database Normalisation
- Database Design - 2nd Edition
- guru99 - Database Normalisation