

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | | Database Planning and Development | | |  | |
|  |  | | | | |  |
|  | | |  |  | | |
|  | | | Abdullah Alabdullah |  | | |
|  | | | —14-05-2021 |  | | |

Table of Contents

[1.1 Compare the organisation of data in a ‘flat file’ database and a relational database. 3](#_Toc70610105)

[1.2 Explain relational database concepts, including entity, attribute, table, record, relationship and referential integrity. 3](#_Toc70610106)

[1.3 Define the term ‘primary key’ and explain the advantage of using fields in relational database. 3](#_Toc70610107)

[2.1 Analyse your database system to: identify entities and attributes; explain relationships (one-to-one, one-to-many, many to many). 3](#_Toc70610108)

[2.3 Explain the term ‘normalisation’. 3](#_Toc70610109)

# Compare the organisation of data in a ‘flat file’ database and a relational database.

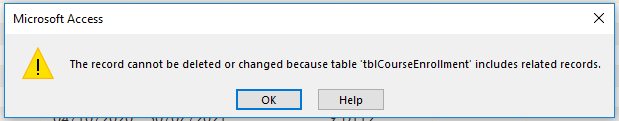
A database is a collection of data that is organized systematically. However, not all databases are organized identically. Two categories of the most widely used databases are flat-file database and relational database. A flat-file database is a single-table database that compiles data in consecutive order. Fields in this type of database are isolated using delimiters. This database can be created manually or using computer processors like Microsoft Excel or Microsoft Word. On the other hand, a relational database is a multi-table database where data in different tables are grouped using relationships. Relational databases can be created from a flat-file database by dividing it into smaller tables and recognising the relationships between them.

In contrast to relational databases, it is vastly problematic to update, insert or delete data from a flat-file database. One issue with flat-file databases is data redundancy. To elucidate, imagine you have a database for a vet practice. When an owner has four pets, four records should be created for each pet which in turn results in significant duplicated data. Moreover, updating, inserting, or deleting any data is very laborious as multiple records need to be updated/deleted. Compounding this issue further, when adding data for one entity more than once in a flat-file database it can result in inconsistent records. On the other hand, in a relational database, two separate tables, for owners and pets, can be created and linked using a one-to-many relationship. This way, any data can be updated or deleted effortlessly, hence, the database maintains its integrity. What's more, new owners will only need to be entered once. Thanks to the robust organization of relational databases, searching for data is nothing but straightforward.

# Explain relational database concepts, including entity, attribute, table, record, relationship and referential integrity.

The relational database is a type of database that allows us to link data in a database relationally. Before diving into the details of this type of database, let us understand some core terminology. As a data repository, collections of data in a relational database are stored in what is called tables (or referred as ‘relations’). Tables are the heart of any database. In fact, all data about an object entered into a relational database should be compiled and retrieved from tables. As a multi-table database, each table in a relational database should by necessity have at least one entity. An entity is an object, person, place or anything about which the data should be inserted and stored. An entity can be any vital object in a database system. For example, in a college database, examples of entities comprise Students, Courses, Staff, Department etc. Typically, every single entity has one table in the database. One should not forget here that an entity without data stored about them is merely but a name. In a relational database, data (properties) stored about an entity are referred to as attributes. Back to the previous example, an entity like "Student" can have the following attributes such as a First Name, Last Name, Date of Birth and Email. These attributes should be written in the column headers and all the data stored in the fields under these headers (attributes) are called attribute values.

Every row in a relational database is called a record or tuple. A record is a row that contains a unique instance of the entity attributes. In relational databases, every record should be exclusive. In the previous example, every student must have a unique record with at least one exclusive field (usually Student ID). This field is also called the identifier. As the name implies, the whole concept behind relational databases is to define meaningful relationships that allow us to arrange the data in the database. There are three types of relationships that can be formed between tables in a relational database, one-to-one, one-to-many and many-to-many relationships. Relationships enable indistinguishable data in two tables (primary and foreign keys) to be matched. After a relationship is created, any amendments to the data in the first table (primary key) will be automatically reflected in the second table (foreign key). This also leads us to another important term, referential integrity. This term implies that common fields (usually primary and foreign keys) in two tables should always have identical data. Referential integrity is very important as it ensures the accuracy and consistency of a database. For example, if you want to delete a record from a primary table, it should not have any associated data (foreign key) in another table. If two tables are connected to each other using keys, you will not be able to delete any record that has associated data in another table (see screenshot below). If you try to enforce this, you will end up with orphaned records.



# Define the term ‘primary key’ and explain the advantage of using fields in relational database.

In order to understand how tables in relational databases are related to each other, we should first appreciate two integral relationship identifiers, primary key and foreign key. These keys are used to define all the different relationships in relational databases. A primary key is a unique table of important data. One example of a primary key in a student table is the student ID. Another key thing to remember when assigning primary keys is to maintain the uniqueness of the primary key. In other words, each record in the table must be represented distinctively using the primary key. Every table in a relational database can only have one primary key with no null values. As the primary key is used to link identical data in two tables, its values cannot be deleted from the parent table unless we remove the relationship or erase any linked data first.

On the other hand, a foreign key is a column in a table that contains identical data (usually the primary key) from the linked table. Using foreign keys, relationships can be set between tables. Furthermore, all the data about a record which has been linked using a foreign key can be accessed from the table that has the primary key. Unlike primary keys, you can have more than one foreign key in a table. This in turn allows you to import data from two or more tables. Another major difference is that records that have a foreign key can be deleted without removing the relationship between the two tables. In addition to the great advantages mentioned above, the main advantage of using key fields is to define relationships between tables which in turn allows us to maintain data integrity and accuracy. Moreover, key fields can help us avoiding data redundancy.

# 2.1 Analyse your database system to: identify entities and attributes; explain relationships (one-to-one, one-to-many, many to many).

In my database, I have a number of entities that also have various attributes. The table below lists all the database entities (column headers) and all the attributes which stem from these entities.



As a relational database, entities in my database are connected using a number of relationships including one-to-one, one-to-many and many-to-many relationships. The students table and student records table are linked using a one-to-one relationship (Figure-1). This relationship indicates that one student can only have one record and one record can only be assigned to one student. This relationship allows us to access the data stored in the student record table easily.

The one-to-many relationship can be seen between the department table and the courses table (Figure-2). In this relationship, each department can have many courses. After defining this relationship, we can view all the courses in a department easily from the department table. This type of relationship also exists between the staff table and the course table (Figure-3).

On the other hand, a many-to-many relationship is defined between the course table and the student table (Figure-4). As the name implies, this relationship means one course can have many students and one student can study more than one course. It should be noted that this relationship cannot be set between two tables directly. Instead, we use what is called a conjunction table. In this relationship, the conjunction table is the course enrolment table. This table allows data in both tables to be matched using key fields.

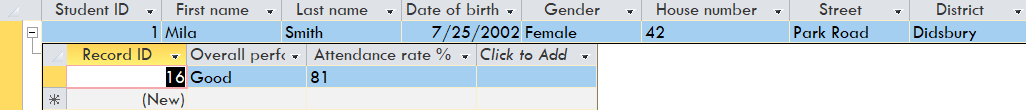
****

Figure 1

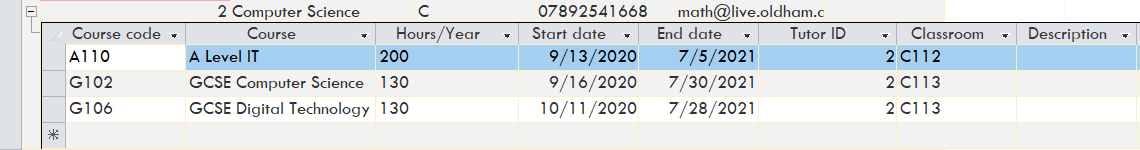


Figure 2

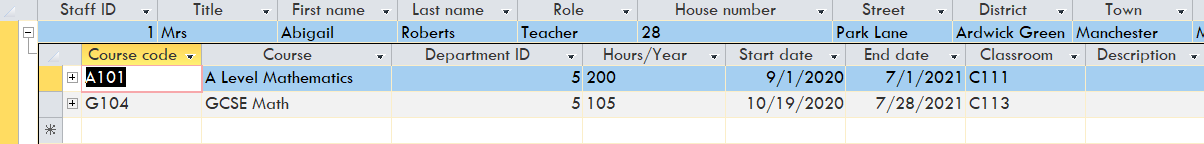


Figure 3

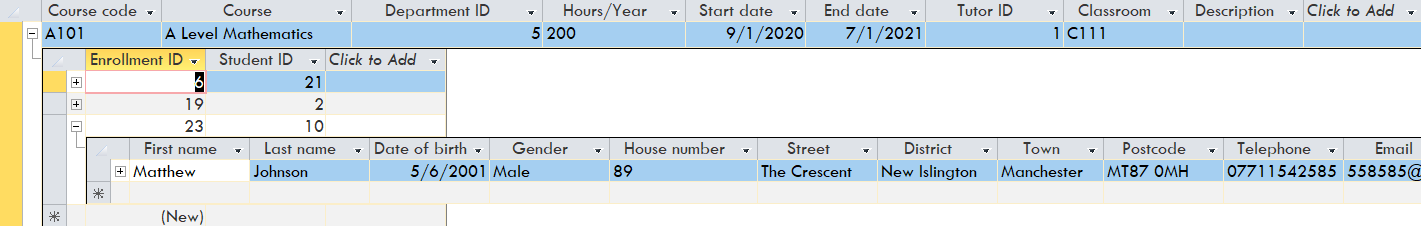
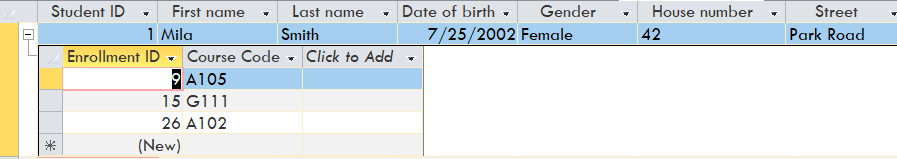
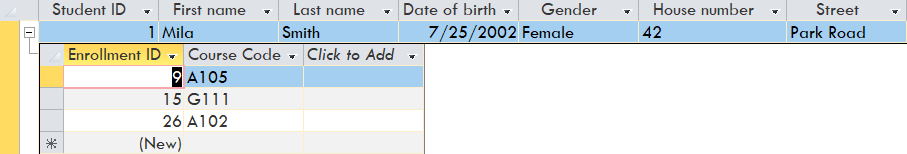
****

Figure 4

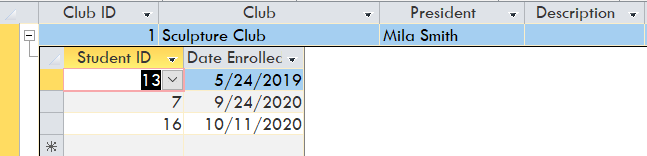
One-to-many relationship between the students table and the Course enrolment table.



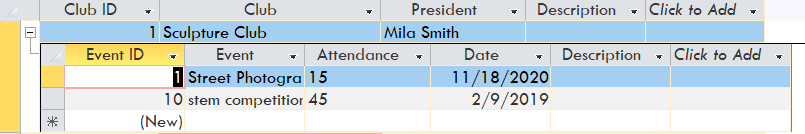
One-to-many relationship between the students table and the club membership table.



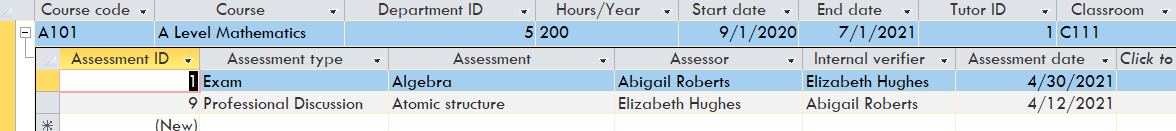
One-to-many relationship between the club table and the club membership table.

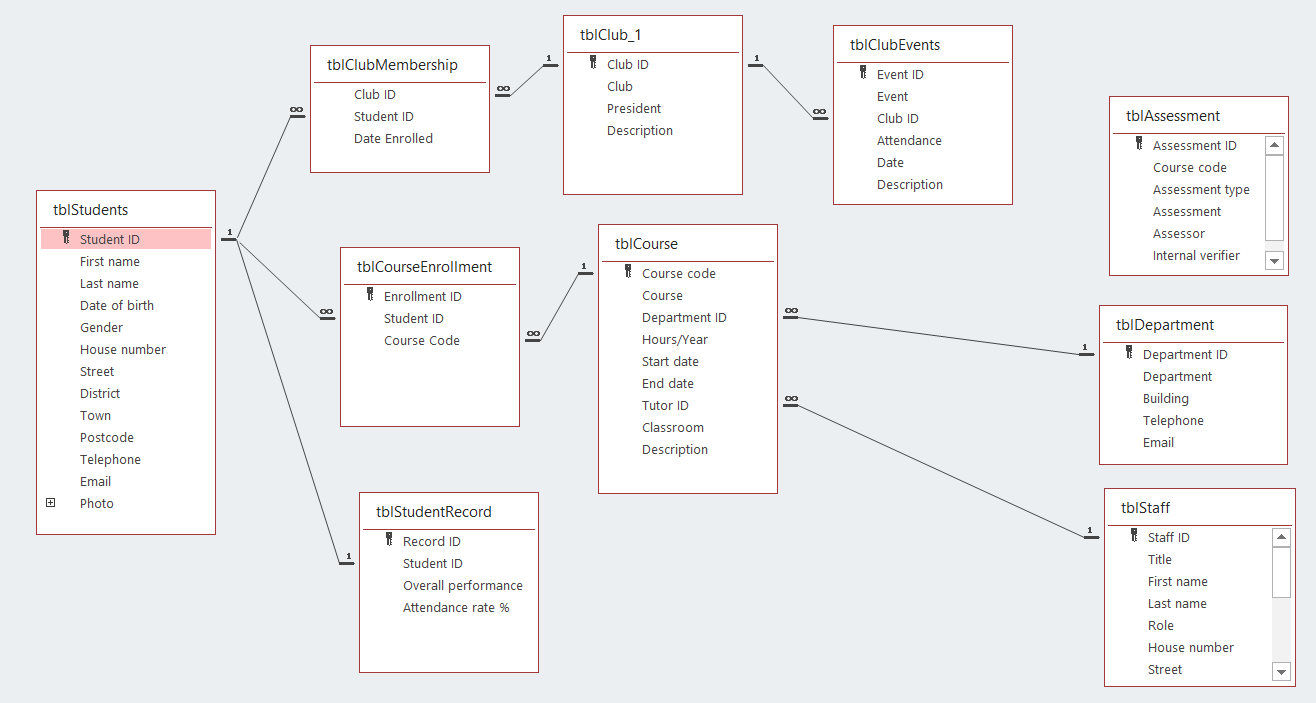


One-to-many relationship between the club table and the club events table.



One-to-many relationship between the course table and the assessment table.





# 2.3-1 Explain the term ‘normalisation’.

Normalization is an essential notion in database systems. Although there are many definitions for normalization, it can be comprehensively explained as a set of processes aimed to minimize data redundancy (unnecessary duplication of data). But why do we need to reduce data redundancy? When a database contains unnecessary duplication of data, conflicting data can occur which in turn results in data inconsistency. If a database is not normalized, it is more likely to encounter complications when deleting, updating or inserting new data. Additionally, normalizing data can optimize memory effectiveness and allow for a faster search for data. There are three types (levels) of normalization forms, 1st normal form, 2nd normal form, and 3rd normal form.

1st Normal Form (1st NF)

The 1st normal form must meet some rules. Firstly, each column should contain atomic values. For example, one field must not contain both the first and last name. The second rule states that data in a column should maintain the same type. For example, a column for data of birth should not contain a name. Moreover, the column designated with names must also have varying names.

Before normalization (0 NF)

|  |  |  |  |
| --- | --- | --- | --- |
| Student Name | Date of Birth | Course | Subject |
| Mila Smith | 01/02/2001 | A level, GCSE | Math |
| Jack Jones | 30/05/2000 | Access course, GCSE | IT |
| Aria Williams | January 2003 | GCSE | English |

After normalization (1st NF)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| First name | Last name | Date of Birth | Course | Subject |
| Mila | Smith | 01/02/2001 | A level | Math |
| Jack | Jones | 30/05/2000 | Access course | IT |
| Aria | Williams | 01/01/2003 | GCSE | English |
| Mila | Smith | 01/02/2001 | GCSE | Physics |
| Jack | Jones | 30/05/2000 | GCSE | Biology |

2nd Normal Form (2nd NF)

After normalizing the data in the 1st normal form, all that is required to meet the 2nd normal form criteria is to eliminate data partial dependency. This task can be executed efficiently by creating a unique column of data which can be a unique identifier for each record. This is also referred to as the primary key. After achieving this rule, the table can be divided into two or more logically connected tables.

Before 2nd Normal Form

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| First name | Last name | Date of Birth | Course | Subject | Department | Building |
| Mila | Smith | 01/02/2001 | A level | Math | Math | A |
| Jack | Jones | 30/05/2000 | Access course | IT | Technology and Computing | C |
| Aria | Williams | 01/01/2003 | GCSE | English | Linguistics | B |

This table will now be divided into two tables:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Student ID | First name | Last name | Date of Birth | Course ID |
| 01 | Mila | Smith | 01/02/2001 | 101 |
| 02 | Jack | Jones | 30/05/2000 | 102 |
| 03 | Aria | Williams | 01/01/2003 | 103 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Course ID | Course name | Subject | Department | Building |
| 101 | A level | Math | Math | A |
| 102 | Access course | IT | Technology and Computing | C |
| 103 | GCSE | English | Linguistics | B |

3rd Normal Form (3rdNF)

The most important rule in this form is to have the table in 2nd normal form. Secondly, the table should also be free of any transitive dependency. According to (Chapple, 2019), "a transitive dependency in a database is an indirect relationship between values in the same table that causes a functional dependency". In other words, we must create a separate table for the data that does not have a direct dependency. In the previous example, there is no transitive dependency for the department name and the building with the course table. For this example, it is best to have a separate table for the departments. One advantage of this process is that more attributes can be added to the department table without causing any confusion or data duplication. Another great advantage of the 3rd Normal Form is that multiple tables can be linked together without duplicating any data. Additionally, changing the data in a normalized database of the third form is straightforward. In other words, when you make any modifications to the data, the data will be updated in all linked tables automatically. Moreover, performing this level of normalization can also help us to prevent any inconsistency in the data after deleting, modifying or inserting new data. It can be noticed from the table below that the Department ID has been deleted as it can be easily accessed through the course ID. In the example below, the Department ID has been deleted from the course enrolment table because it can be accessed through the course code.

Example:

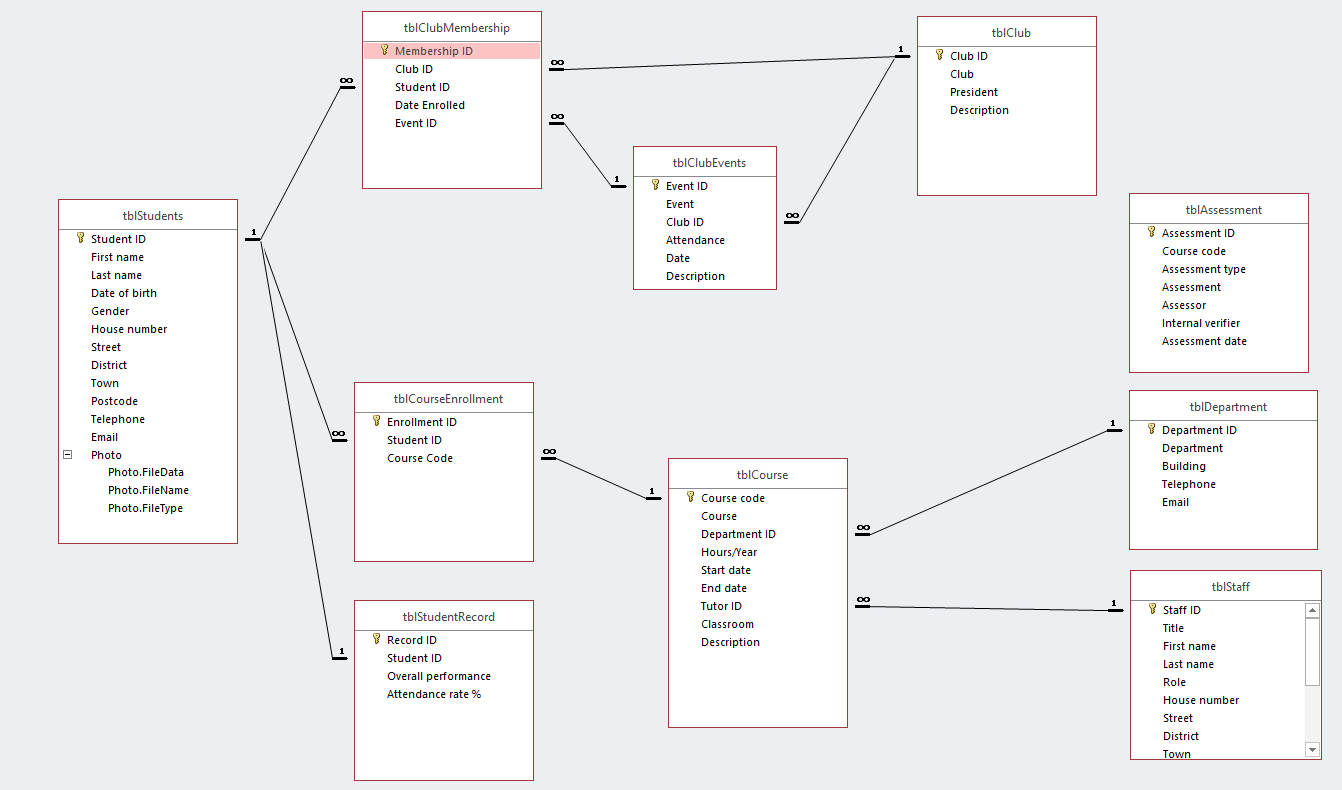
The table below is **not** in 3rd NF (before):

|  |  |  |  |
| --- | --- | --- | --- |
| Enrolment ID | Student ID | Course Code | Department ID |
| 1 | 4 | A109 | 2 |
| 2 | 10 | A105 | 1 |

The table below is in 3rd NF (after):

|  |  |  |
| --- | --- | --- |
| Enrolment ID | Student ID | Course Code |
| 1 | 4 | A109 |
| 2 | 10 | A105 |

All the tables in my database are in 3rd NF and meet all the criteria listed above. This can be seen clearly from the relationship diagram below.



References

1. Chapple, M. 2019. What Is Transitive Dependency in a Database [online] Available at: <<https://www.lifewire.com/transitive-dependency-1019760#:~:text=A%20transitive%20dependency%20in%20a,must%20eliminate%20any%20transitive%20dependency>.> [Accessed 28 April 2021].