Chapter 19 Databases

Contents

Syllabus Learning Outcomes

3.3 Databases and Data Management

Understand, create and use SQL and NoSQL databases, as well as understand techniques to protect the privacy and integrity of data.

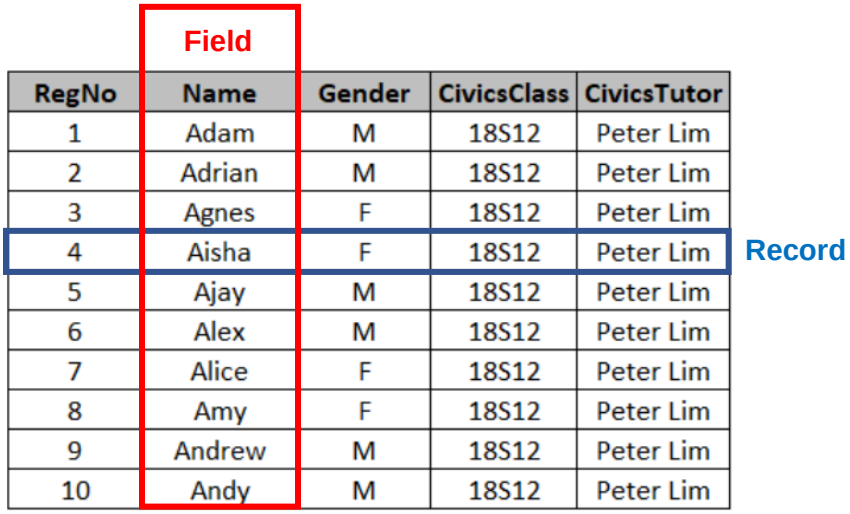
3.3.1 Determine the attributes of a database: table, record and field.
3.3.2 Explain the purpose of and use primary, secondary, composite and foreign keys in tables.
3.3.3 Explain with examples, the concept of data redundancy and data dependency.
3.3.4 Reduce data redundancy to third normal form (3NF).
3.3.5 Draw entity-relationship (ER) diagrams to show the relationship between tables.
3.3.6* Understand how NoSQL database management system addresses the shortcomings of relational database management system (SQL).
3.3.7* Explain the applications of SQL and NoSQL.
3.3.8* Use a programming language to work with both SQL and NoSQL databases.

*Note: NoSQL will be addressed in later chapter

1 How is data stored?

Data can be stored in a flat file database. A **database** is a collection of data stored in an organised or logical manner. Storing data in a database allows us to access and manage the data.

A **flat file database** is a database that contains a single table.



RegNo	Name	Gender	CivicsClass	CivicsTutor
1	Adam	M	18S12	Peter Lim
2	Adrian	M	18S12	Peter Lim
3	Agnes	F	18S12	Peter Lim
4	Aisha	F	18S12	Peter Lim
5	Ajay	M	18S12	Peter Lim
6	Alex	M	18S12	Peter Lim
7	Alice	F	18S12	Peter Lim
8	Amy	F	18S12	Peter Lim
9	Andrew	M	18S12	Peter Lim
10	Andy	M	18S12	Peter Lim

A **table** contains a collection of **records** for a particular theme. In the `Student` table above, it is a collection of student data from a civics class.

A **record (a row in a table)** is a complete set of data about a single item. In the table above, each record contains the data of one student.

A table has a number of fields. A column in a table is called a **field**. **Attributes** are the describing characteristics or properties that define all items pertaining to a certain category applied to all cells of a column.

In the example above, the table has 10 records and 5 fields, and the attributes are `RegNo`, `Name`, `Gender`, `CivicsClass` and `CivicsTutor`.

Are there any potential issues that might arise from storing the data in a flat file database?

Peter Lim is the Civics Tutor for the Civics Class 18S12. In a flat file database, his details would need to be entered many times as long as a student is in the class 18S12.

This is known as **data redundancy**.

2 Data redundancy

Data redundancy refers to the same data being stored more than once.

Having to enter data multiple times means the database is at an **increased risk** of having inaccurate data. It would be hard to update because every occurrence of the data item will need to be changed. This can lead to **data inconsistency**.

This will cause issues when inserting, updating and deleting data from the database.

RegNo	Name	Gender	CivicsClass	CivicsTutor
1	Adam	M	18S12	Peter Lim
2	Adrian	M	18S12	Peter Lim
3	Agnes	F	18S12	Peter Lim
4	Aisha	F	18S12	Peter Lim
5	Ajay	M	18S12	Peter Lim
6	Alex	M	18S12	Peter Lim
7	Alice	F	18S12	Peter Lim
8	Amy	F	18S12	Peter Lim
9	Andrew	M	18S12	Peter Lim
10	Andy	M	18S12	Peter Lim

Referring to the `Student` table above, see below for possible issues.

Inserting data	A new student from the same Civics Class to be inserted will need to insert duplicated data on the Civics tutor.
Updating data	If Peter Lim decide to leave the college, all the records with his name as the Civics Tutor in the `Student` table would need to be updated. If we miss any record, it will lead to inconsistent data.
Deleting data	If all the records of the `Student` table are deleted, information on civics class and civics tutor will be lost.

To address this issue, a more efficient way of storing data has to be adopted.

3 Relational database

The most common model for a database is a relational database model.

A relational database is a collection of relational tables where data are organised in one or more tables with relationships between them.

In general, a table in a relational database can be expressed as:

```
TableName (Attribute1, Attribute2, Attribute3, …)
```

For example,

```
Student (RegNo, Name, Gender, CivicsClass, CivicsTutor)
```

A table needs to fulfil the following conditions:

- Values are atomic (Information in each column cannot be an array or another table, it only contains one piece of information which cannot be broken down further.)
- Columns are of the same kind
- Rows are unique
- The order of columns is insignificant
- Each column must have a unique name

It is important to be able to identify each record in table uniquely using a key field.

A **key field**, or **key** in short, is a combination of one or more columns in a database that uniquely identifies a row in a table.

Keys allows for the establishment of relationships between tables and allows for the identification of relation between tables. Keys also help to enforce identity and integrity in the relationship.

There are different type of keys.

Candidate keys

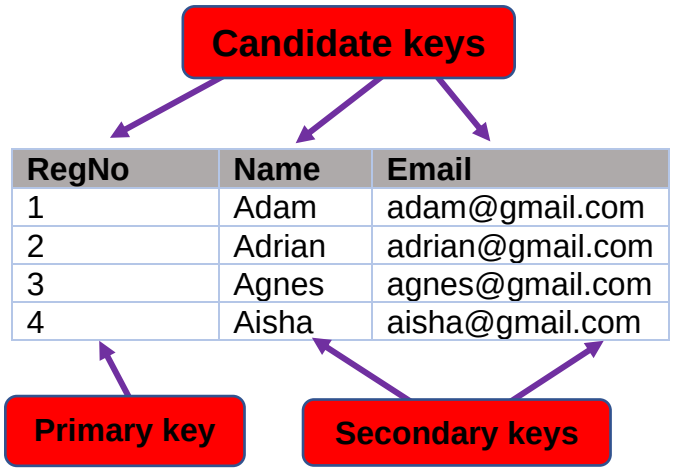
A **candidate key** is defined as a minimal set of fields which can uniquely identify each record in a table. It tells a particular record apart from another record. A candidate key should never be NULL or empty. There can be more than one candidate key for a table. A candidate key can also be a combination of more than one field.

Primary keys

A **primary key** is a field or a set of fields in a table whose values uniquely identifies each record in a table and should not change over time. That is, a primary key tells a particular record apart from another record. It is also a candidate key that is most appropriate to become the main key for a table.

Secondary keys

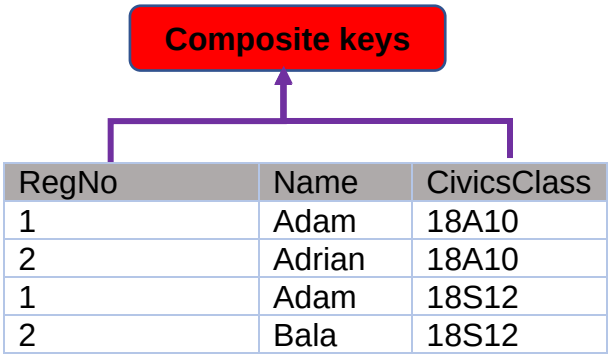
Candidate keys that are not selected as primary key are known as **secondary keys**. A secondary key is an additional key, or alternate key, which can be used in addition to the primary key to locate specific data.



In the given table above, `RegNo`, `Name`, and `Email` are candidate keys which help us to uniquely identify the student record in the table. Any one of the candidate keys can be selected as the primary key, for example, RegNo. The rest of the two keys would be the secondary keys.

Composite primary keys

Sometimes, more than one field is needed to uniquely identify a record. A **composite primary key** is a combination of two or more fields in a table that can be used to uniquely identify each record in a table. Uniqueness is only guaranteed when the fields are combined. When taken individually, the fields do not guarantee uniqueness.



In the example above, both `RegNo` and `CivicsClass` are needed to form a composite key to uniquely identify a record.

Foreign keys

A **foreign key** is an attribute (field) in one table that refers to the primary key in another table. It links to a primary key in the other table and form a relationship between the tables.

Another table, `ClassInfo`, as shown below, stores the information about each civics class.

CivicsClass	CivicsTutor	HomeRoom
18S12	Peter Lim	BlkA-L2-3
18A10	Pauline Lee	BlkC-L1-4

Notice that the `CivicsClass` field is the primary key in the table `ClassInfo` and is related or linked to the `CivicsClass` field in table `Student`. This makes `CivicsClass` field in table `Student` a foreign key (FK).

One benefit of a relational database is that all data can be updated at the original **source**. Data doesn't need to be repeated and relationships can be made using the source table's **unique identifier**.

4 Normalisation

Normalising the tables in the relational database is an important process of organising the tables so as to reduce data redundancy and prevent inconsistent data. There are at least three normal forms associated with normalisation: first normal form (1NF), second normal form (2NF), and third normal form (3NF).

You might want to take a quick look at this article before reading on: https://www.freecodecamp.org/news/database-normalization-1nf-2nf-3nf-table-examples/

First Normal Form

For a table to be in 1NF, all columns must be atomic. This means there can be no multi-valued columns – i.e., columns that would hold a collection such as an array or another table. In other words, the information in each column cannot be broken down further.

Consider the following table:

MatricNo	Name	Gender	CivicsClass	CivicsTutor	HomeRoom	CCAInfo
1	Adam	M	18S12	Peter Lim	TR1	Tennis Teacher IC = Adrian Tan
2	Adrian	M	18S12	Peter Lim	TR1	Choir Teacher IC = Adeline Wong, Student Council Teacher IC = David Leong
3	Adam	M	18A10	Pauline Lee	TR2	Rugby Teacher IC = Andrew Quah
4	Bala	M	18A10	Pauline Lee	TR2	Badminton Teacher IC = Lilian Lim
5	Bee Lay	F	18A10	Pauline Lee	TR2	Choir Teacher IC = Adeline Wong, Chess Club Teacher IC = Edison Poh

It is assumed that every civics class only has one civics tutor, and each CCA has only one teacher in-charge (IC).

This table is not in 1NF because the `CCAInfo` column contains multiple values.

In order for the table to be in 1NF, we split `CCAInfo` into two single-value columns `CCAName` and `CCATeacherIC`. Notice that the students with `MatricNo` 2 and `MatricNo` 5

have multiple CCAs. We keep this information intact by splitting their records into multiple records, each corresponding to a different CCA. The resulting table is shown below.

MatricNo	Name	Gender	CivicsClass	CivicsTutor	HomeRoom	CCAName	CCATeacherIC
1	Adam	M	18S12	Peter Lim	TR1	Tennis	Adrian Tan
2	Adrian	M	18S12	Peter Lim	TR1	Choir	Adeline Wong
2	Adrian	M	18S12	Peter Lim	TR1	Student Council	David Leong
3	Adam	M	18A10	Pauline Lee	TR2	Rugby	Andrew Quah
4	Bala	M	18A10	Pauline Lee	TR2	Badminton	Lilian Lim
5	Bee Lay	F	18A10	Pauline Lee	TR2	Choir	Adeline Wong
5	Bee Lay	F	18A10	Pauline Lee	TR2	Chess Club	Edison Poh

The values for `CCAName` and `CCATeacherIC` are now atomic for each row. The primary key for the above table will be the composite key formed by `MatricNo` and `CCAName`.

Second Normal Form

For a table to be in 2NF, it must satisfy two conditions:

1) The table should be in 1NF
2) Every non-key attribute must be functionally dependent on the entire primary key.

> Attribute Y is **functionally dependent** on attribute X (usually the primary key), if for every valid instance of X, the value of X uniquely determines the value of Y (X Y).
>
> This means that for every non-key attribute, the entire primary key uniquely determines the value of that attribute, i.e. no attribute can depend on only part of the primary key.

The table above is not in 2NF. `Name`, `Gender`, `CivicsClass`, `CivicsTutor` and `HomeRoom` is dependent on only part of the primary key, `MatricNo`. `CCATeacherIC` is dependent only on `CCAName`. Thus, we decompose the table into three tables as shown below.

Student

MatricNo	Name	Gender	CivicsClass	CivicsTutor	HomeRoom
1	Adam	M	18S12	Peter Lim	TR1
2	Adrian	M	18S12	Peter Lim	TR1
3	Adam	M	18A10	Pauline Lee	TR2
4	Bala	M	18A10	Pauline Lee	TR2
5	Bee Lay	F	18A10	Pauline Lee	TR2

StudentCCA

MatricNo	CCAName
1	Tennis
2	Choir
2	Student Council
3	Rugby
4	Badminton
5	Choir
5	Chess Club

CCAInfo

CCAName	CCATeacherIC
Tennis	Adrian Tan
Choir	Adeline Wong
Student Council	David Leong
Rugby	Andrew Quah
Badminton	Lilian Lim
Chess Club	Edison Poh

The primary key for table `Student` will be `MatricNo`. `MatricNo` and `CCAName` will form a composite key for table `StudentCCA` and `CCAName` will be the primary key for table `CCAInfo`.

Third Normal Form
For a table to be in 3NF, it must satisfy two conditions:

1) The table should be in 2NF
2) The table should not have transitive dependencies – all fields must only be determined by the primary/composite key, not by other non-key attributes

   A functional dependency X → Z is said to be **transitive** if there exists an attribute Y such that X → Y and Y → Z
   Note that X → Y **does not** necessarily imply the converse Y → X.

Looking at `Student` table, `CivicsTutor` and `HomeRoom` are dependent on `CivicsClass` and `CivicsClass` is dependent on `MatricNo`. Therefore, `CivicsTutor` and `HomeRoom` are transitively dependent on `MatricNo`. To remove the transitive dependency, we decompose the `Student` table as follows:

Student

MatricNo	Name	Gender	CivicsClass
1	Adam	M	18S12
2	Adrian	M	18S12
3	Adam	M	18A10
4	Bala	M	18A10
5	Bee Lay	F	18A10

Civics

CivicsClass	CivicsTutor	HomeRoom
18S12	Peter Lim	TR1
18A10	Pauline Lee	TR2

`MatricNo` remains the primary key for `Student` table and `CivicsClass` will be the primary key of the newly formed Civics table.

The final design after normalisation is represented below:

`Student` (<u>`MatricNo`</u>, `Name`, `Gender`, `CivicsClass`)

`Civics` (<u>`CivicsClass`</u>, `CivicsTutor`, `HomeRoom`)

`StudentCCA` (<u>`MatricNo`</u>, <u>`CCAName`</u>)

`CCAInfo` (<u>`CCAName`</u>, `CCATeacherIC`)

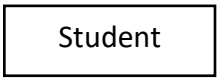
The primary key for each table is indicated by underlining one or more attributes using solid lines. Foreign keys are indicated by using a dashed underline.

This design could also be represented using an **Entity-Relationship** diagram, also known as an **E-R diagram**.

5 Entity-Relationship diagram

Designers of a large database will normally construct a diagram of the planned database. An example of this is entity-relationship (E-R) diagram.

An **entity** is a specific object of interest (usually indicating a table in the database). Collective nouns or nouns are usually used to name entities. Entities are represented by rectangles.

For example, 

Relationship describe a link between two entities.

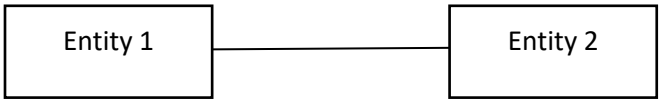
Relationships can be:

- One to one
- One to many
- Many to many

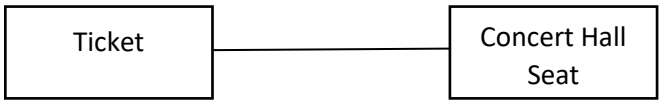
One of the following three relationships can exist between two entities.

1. One to One

   A **one to one** relationship means that **one record** in a table relates to a **single** record in another table.

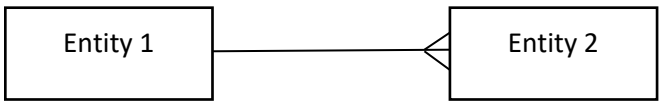
   

   For example, at a concert each ticket entitles you to a particular seat and each seat is linked to only one ticket.

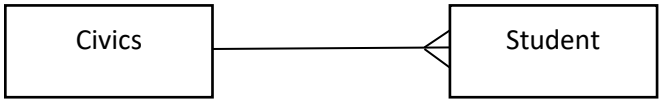
   

2. One to Many

   A **one to many** relationship means that **one record** in a table relates to **multiple** records in another table.

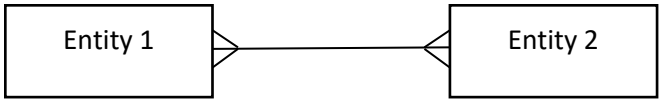
   

For example, a student can belong to only one civics class but a civics class can have many students.

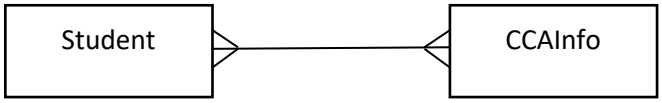


3. Many to Many

A **many to many** relationship means that **multiple** records in a table relate to **multiple** records in another table.



For example, a student can join many CCAs and one CCA can have many students.



As it is difficult to implement a many-to-many relationship in a database system as it may indicate transitive dependencies that must be removed for 3NF.

An example of many-to-many relationship tables is shown below.

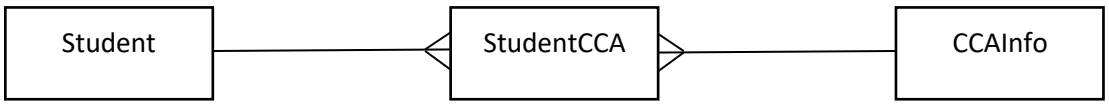
`Student` table

MatricNo	Name	Gender	CivicsClass	CCAName
1	Adam	M	18S12	Tennis
2	Adrian	M	18S12	Choir
2	Adrian	M	18S12	Student Council
3	Adam	M	18A10	Rugby
4	Bala	M	18A10	Badminton
5	Bee Lay	F	18A10	Choir
5	Bee Lay	F	18A10	Chess Club

`CCAInfo` table

MatricNo	CCAName	CCATeacherIC
1	Tennis	Adrian Tan
2	Choir	Adeline Wong
2	Student Council	David Leong
3	Rugby	Andrew Quah
4	Badminton	Lilian Lim
5	Choir	Adeline Wong
5	Chess Club	Edison Poh

We will usually decompose a many-to-many relationship into two (or more) one-to-many relationships. For example,



Student

MatricNo	Name	Gender	CivicsClass
1	Adam	M	18S12
2	Adrian	M	18S12
3	Adam	M	18A10
4	Bala	M	18A10
5	Bee Lay	F	18A10

StudentCCA

MatricNo	CCAName
1	Tennis
2	Choir
2	Student Council
3	Rugby
4	Badminton
5	Choir
5	Chess Club

CCAInfo

CCAName	CCATeacherIC
Tennis	Adrian Tan
Choir	Adeline Wong
Student Council	David Leong
Rugby	Andrew Quah
Badminton	Lilian Lim
Chess Club	Edison Poh

The E-R Diagram can then be directly translated into database designs.

Representing the normalised tables,

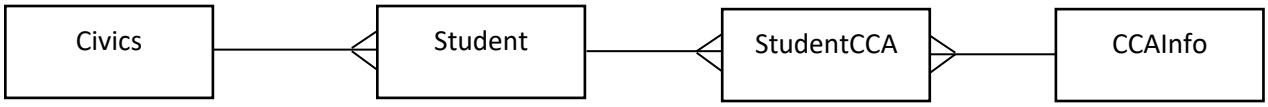
Student (MatricNo, Name, Gender, CivicsClass)

Civics (CivicsClass, CivicsTutor, HomeRoom)

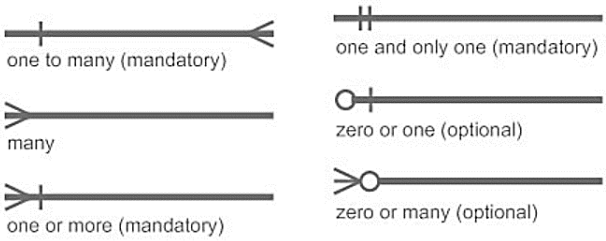
StudentCCA (MatricNo, CCAName)

CCAInfo (CCAName, CCATeacherIC)

Using an E-R diagram, we will have the following:



Other relationships which may come in handy for E-R diagrams:



References

1. Burdett, A. (2016). *BCS glossary of computing*. BCS Learning and Development Limited.
2. Captain, F. A. (2015). *Six-step relational database design: A step by step approach to relational database design and development*. Place of publication not identified: Publisher not identified.
3. LANGFIELD, S. D. (2015). *Cambridge International AS and A Level Computer Science Coursebook*. Place of publication not identified: Cambridge University Press.
4. Archiveddocs. (n.d.). Lesson 2: Designing a Normalized Database. Retrieved from https://docs.microsoft.com/en-us/previous-versions/tn-archive/cc505842(v=technet.10)
5. Introduction to Database Keys. (n.d.). Retrieved from https://www.studytonight.com/dbms/database-key.php
6. Watt, A. (2014, October 24). Database Design - 2nd Edition. Retrieved from https://opentextbc.ca/dbdesign01/
7. Singh, C., & Prasad. (2018, December 14). Functional dependency in DBMS. Retrieved from https://beginnersbook.com/2015/04/functional-dependency-in-dbms/
8. Chapter 6. Entity-Relationship Modelling. (n.d.). Retrieved from https://www.cs.uct.ac.za/mit_notes/database/htmls/chp06.html#one-to-one-relationships-between-two-entities
9. Transitive Dependency in DBMS. Retrieved from https://www.scaler.com/topics/transitive-dependency-in-dbms/