Temasek Junior College

2024 JC2 H2 Computing

Database 1 – Fundamental Relational Database Concepts

Section	3	Data and Information
Unit	3.3	Databases and Data Management
Objectives	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.

1 What is a Database?

A database is **an organised collection of data**. There many different categories of databases such as

- flat file databases
- relational databases,
- non-relational (NoSQL) databases, and
- hierarchical databases,

to name but a few.

Organising and storing data within a database facilitates the extraction of different datasets and the combination of these datasets in constructive ways to derive useful information (for further processing and analyses).

2 Flat File Databases

A flat file database is a database where all the required data is stored in a single table, within a single file. They are generally considered as the simplest kind of databases. Examples of flat file databases include but are not restricted to only:

- comma-separated-values (CSV) files,
- tab-separated-values (TSV) files,
- plain text files, and
- JavaScript Object Notation (JSON) files.

The example below shows a flat file database that holds data on films shown in a cinema:

Title	Rating	Duration (mins)	Theatre	Time
Minions 2: The Rise of Gru	PG	87	3	1745
Jumanji: Welcome to the Jungle	PG	119	Gold Class 1	1815
Minions 2: The Rise of Gru	PG	87	4	1845
Thor: Love and Thunder	PG13	911	Gold Class 2	1930
Jumanji: Welcome to the Jungle	NC16	119	Gold Class 1	2015
Thor: Love and Thunder	PG13	911	3	2030

Storing all the data in a single table in this way can lead to problems, especially as the dataset becomes larger. Data is often repeated and this can result in **poor data integrity**.

Herein, **data integrity** refers to the reliability of data in terms of its accuracy, completeness and consistency.

2.1 Data Inconsistency

When data in a database is not stored correctly and/or is not updated correctly, in all of the relevant places, **data inconsistency** occurs.

Observe that the first occurrence of the film *Jumanji: Welcome to the Jungle* has a rating of PG while the second occurrence of the film has a rating of NC16. Such data inconsistency compromises data integrity as the accuracy of the data becomes questionable.

In fact, where there is inconsistent data, it might not be even possible to determine which piece of data is inaccurate.

A flat file database is therefore only suitable for a small volume of data e.g. keeping a record of the birthdays of all your friends. A small volume of data makes it easier to see all the data at one glance, making it easier to check for data inconsistencies.

2.2 Data Redundancy

Data redundancy occurs when there is unnecessary repetition of data within a database.

Observe that there are several instances where the same piece of data appears in more than one place. In fact, the title, rating and duration of each film are repeated, as are the names of the theatres.

Unnecessarily repeated data results in more time being consumed to keep the database updated. For example, if the length of the film *Thor: Love and Thunder* had been incorrectly listed as 911 minutes instead of the correct duration of 119 minutes, every occurrence of it would need to be updated. If one of these instances was missed, the data would become inconsistent, compromising data integrity due to data inaccuracy.

3 Relational Databases

In a relational database, the overall dataset is split across more than one **table**. The tables are linked together by forming **relationships** between them.

Each table in a relational database represents an **entity**. Herein an **entity** refers to an object in the real world for which data needs to be stored in the database.

For the flat file database in the previous section, the relevant entities can be identified as films, theatres, and screenings. Hence the data can be organised across three tables as follows:

`Flims`

FilmID	Title	Genre	Rating	Duration
1234	Thor: Love and Thunder	Action	PG13	119
1235	Minions 2: The Rise of Gru	Animation	PG	87
1236	Jumanji: Welcome to the Jungle	Adventure	PG	119

`Screenings`

ScreeningID	FilmID	TheatreID	ScreenDate	ScreenTime
201	1235	3	21062024	1745
202	1236	1	21062024	1815
203	1235	4	21062024	1845
204	1234	2	21062024	1930
205	1236	1	21062024	2015
206	1234	3	21062024	2030

`Theatres`

TheatreID	TheatreName	Capacity
1	Gold Class 1	30
2	Gold Class 2	30
3	3	200
4	4	200

Notice that in the **`Films`** table, there is only a single record for each film i.e. the details of each film are only stored once. Similarly, in the **`Theatres`** table, there is only a single record for each theatre the cinema has.

In the **`Screenings`** table, there is a record for each screening of a film. The **`FilmID`** is recorded for each screening, but none of the other details about the film being screened are repeated. Similarly the **`TheatreID`** is recorded, rather than the name of the theatre.

Now, if the details about a film or a theater needs to be changed, this only needs to be done in **one** place (i.e. the data is updated in the **`Films`** or the **`Theatres`** table).

This approach eliminates **data redundancy**, reducing the likelihood of storage and/or update errors leading to **data inconsistency**. It is much simpler to keep data updated and the data is far more reliable, preserving **data integrity**.

If a listing of all of the film screenings (identical to that presented in the flat file database in the previous section) is required, the relevant data from the three tables can be extracted and combined to display the desired information as follows:

Film screenings for 21062023

Title	Rating	Duration	Theatre	Time
Minions 2: The Rise of Gru	PG	87	3	1745
Jumanji: Welcome to the Jungle	PG	119	Gold Class 1	1815
Minions 2: The Rise of Gru	PG	87	4	1845
Thor: Love and Thunder	PG13	911	Gold Class 2	1930
Jumanji: Welcome to the Jungle	NC16	119	Gold Class 1	2015
Thor: Love and Thunder	PG13	911	3	2030

The above listing would be produced using **Structured Query Language (SQL)**, which is a standardised language for working with relational databases.

3.1 Tables, Records and Fields

Tables
A table is a collection of related data. As previously mentioned, each table represents an **entity** which is an object in the real world for which data needs to be stored in the database.

Records
Each row of a table represents a **record** in the database.

Fields
A record is split into **fields**. Each field contains a distinct piece of data.

Consider the **Films** table. Each row of the table is a record that contains data about a single film. The fields in each record are:

- FilmID
- Title
- Genre
- Rating
- Duration

Each field will have a data type. In most relational database systems, a field stores data of one of the following data types:

- Text
- Integer
- Real
- Boolean
- Date

There are also some database systems that support a far wider range of data types.

3.2 Table Descriptions, Primary Keys, Composite Keys and Foreign Keys

Table Descriptions

A **table description** in the following standardised format can be written to describe the **`Films`** table:

`Films(FilmID, Title, Genre, Rating, Duration)`

In this standardised format, each description starts with the name of the table. Then, within a pair of parentheses, the name of each field is written.

Similarly, table descriptions can be written for the **`Theatres`** and **`Screenings`** tables.

`Theatres(TheatreID, TheatreName, Capacity)`

`Screenings(ScreeningID, FilmID, TheatreID, ScreenDate, ScreenTime)`

Primary Keys

A **primary key** is a **field** or the **smallest combination of fields** in a relational database table that **uniquely identifies each record** in the table.

This primary key must contain a unique value or unique combination of values for every record in the table. In addition, the primary key should not contain null values.

The primary key of a table should be unique to the table i.e. each table can only have one primary key. There should not be multiple fields or multiple combinations of fields that can be used as the primary key.

Primary keys are identified in table descriptions using **full underlined text**. Using the **`Films`** table as example, the **`FilmID`** is the primary key, and it is identified as a full underlined text in the table description `Films(FilmID, Title, Genre, Rating, Duration)`

This means that every record in the **`Films`** table will have a unique value for the **`FilmID`** field. In other words, each row in the table will have a different value in the **`FilmID`** column.

Notice that **`Title`** is not suitable as the primary key for this table because it is possible for two films to have the same name e.g. the film *Missing* produced in 1982 is different from that produced in 2023.

Sometimes, there is a field that is an obvious choice for a primary key. For example, every published book has an International Standard Book Number (ISBN) that is unique to that book. Hence in designing a table to store books available at a library, the ISBN can be selected as the primary key.

In other situations, identification of a suitable primary key field may be more difficult. A common solution is to allocate an auto-incrementing number as the primary key. This means that each new record will be automatically given an ID that is one higher than that of the previous record.

Composite Keys

When a single field is insufficient to uniquely identify each record, the smallest combination of fields that can allow so will be chosen as the primary key. In such cases, the primary key is a **composite key**. In other words, a **composite key** is a key made up of more than one field.

Consider the case where the cinema accepts pre-bookings on top of off the counter purchase. Two more tables would be required. The table descriptions of these tables are as follows:

Customer(CustomerID, FirstName, LastName, MobileNo)

Booking(CustomerID, ScreeningID, BookingDate, NumOfSeats)

The **Customer** table has a single field **CustomerID** designated as the primary key while the **Booking** table uses **CustomerID** together with **ScreeningID** as a **composite primary key**. This is because neither the **CustomerID** nor **ScreeningID** is sufficient on its own to uniquely identify a record in the **Booking** table since a customer can make multiple bookings, one each for a different film screening. Likewise, the same film screening can be booked by multiple customers.

Using a combination of **CustomerID** and **ScreeningID** is guaranteed to be unique i.e. a customer booked tickets for a particular screening.

To better understand this, consider the following records stored in the **Booking** table:

CustomerID	ScreeningID	BookingDate	NumOfSeats
145543	206	21052024	2
012010	205	23052024	1
132099	202	28052024	2
131092	206	28052024	2
132099	205	05062024	3
012010	301	15062024	2
132099	206	16062024	2
145543	370	18062024	4
148765	299	18062024	2
145543	204	20062024	2

Observe from the data that none of the fields **on their own** can be guaranteed to be unique. The **CustomerID** is not unique for each record as there can be more than one booking by each customer. For example, the customer with **CustomerID 132099** has three bookings.

CustomerID	ScreeningID	BookingDate	NumOfSeats
132099	202	28052024	2
132099	205	05062024	3
132099	206	16062024	2

Neither is the **ScreeningID** unique for each record as there can be multiple bookings for each screening. For example, the screening with **ScreeningID 205** has two bookings.

CustomerID	ScreeningID	BookingDate	NumOfSeats
012010	205	23052024	1
132099	205	05062024	3

However, a customer can only make **one** booking for a specific showing. Therefore, the combination of the **CustomerID** and the **ScreeningID** fields is unique for every record in the table. Therefore, combining these two attributes forms a **composite primary key**.

Foreign Keys

A **foreign key** is a field in a table that is the primary key (or as part of a composite primary key) in **another table**.

Primary/foreign key pairs are used to make links between the tables and they are written as partial underlined text in the table descriptions.

Foreign keys help maintain the integrity of the data in the tables; for example, you can specify that a foreign key value can only be inserted in a table if this value exists as a primary key in another table.

Consider the cinema database which now have five tables. Each table has a primary key. The primary key in the booking table is a composite key.

```
Films(FilmID, Title, Genre, Rating, Duration)
Theatres(TheatreID, TheatreName, Capacity)
Screenings(ScreeningID, FilmID, TheatreID, ScreenDate, ScreenTime)
Customer(CustomerID, FirstName, LastName, MobileNo)
Booking(CustomerID, ScreeningID, BookingDate, NumOfSeats)
```

There is a **FilmID** field in the **Films** table which is a primary key of the table; there is also a **FilmID** field in the **Screenings** table which is a foreign key in the table. Hence the **Films** table is linked to the **Screenings** table by the **FilmID** primary/foreign key pair.

Similarly, the **Theatres** and **Screenings** tables are linked together by the **TheatreID** primary foreign key pair, where **TheatreID** is the primary key of the **Theatres** table and foreign key in the **Screenings** table.

Likewise, the **Customer** and **Booking** tables are linked together by the **CustomerID** primary foreign key pair, where **CustomerID** is the primary key of the **CustomerID** table and foreign key in the **Booking** table. Notice that **CustomerID** is both part of the composite primary key of the **Booking** table as well as a foreign key in the same table. This is in fact a possible scenario.

In addition, the **Screenings** and **Booking** tables are linked together by the **ScreeningID** primary/foreign key pair, where the **ScreeningID** is the primary key of the **Screenings** table and a foreign key in the **Booking** table. Incidentally, the **ScreeningID** is also part of the composite primary key in the **Booking** table.

You might have noticed that some tables are not directly linked together e.g. there is no direct link between the **Screenings** and **Customer** tables. Nevertheless, both tables are directly linked to the **Booking** table. As such the **Screenings** table is indirectly linked to the **Customer** table via the **Booking** table. In the study of relational databases, such indirect relationships are termed **transitive dependencies**. More on transitive dependencies shall be discussed in the next topic.

Now that we know that every primary/foreign key pair forms a link between the relevant tables, let us consider some data from the Booking table.

The first record of the table has **CustomerID 145543** and **ScreeningID 206**.

The **CustomerID** field specifies which customer made the booking. To find out the details of the customer with **CustomerID 145543,** we need to look with the **Customer** table. An example of the customer table is given as follows:

CustomerID	FirstName	LastName	MobileNo
012010	Ashley	Alexster	91234567
131092	Lucas Yong Han	Leow	92345678
132099	Jia Heng	Lee	93456789
145543	Asha	Solomon	94567890
148765	Marc Wen Xu	Soh	95678901

By looking for the record with **CustomerID 145543**, you can find the details of Asha Solomon who is the customer that made the booking.

The **ScreeningID** field of the **Booking** table specifies which screening has been booked. By looking at the **Screenings** table, we can find the record for the screening with **ScreeningID 204**.

From the **Screenings** table, we can identify the record to contain **FilmID 1234**. By looking at the **Films** table, we can discover that the film being screened is *Thor: Love and Thunder*.

Finally, by looking at the **Theatres** table, we can discover that the film will be screened in the theatre with **TheatreID 4** i.e. theater 4.

Hence from the data in the first record of the **Booking** table we can extract the following information:

"Asha Solomon booked two seats for a screening of Thor: Love and Thunder that will be shown on 21st June 2024 at 1930 hours in theater 4."

You can imagine that it is extremely time consuming if you had to look into every table and work out what data the foreign keys match up with in their respective tables. As such SQL can be used to query the database and extract this information for you. For example, using SQL, you can search for all the bookings for the screening of *Thor* on 21/06/23 at 19:30 in Grand Theatre B, and produce the following report:

FirstName	LastName	MobileNo	Seats
Asha	Solomon	94567890	2
Lucas Yong Han	Leow	92345678	2
Jia Heng	Lee	93456789	2