14. DATABASES

Databases are used by many organisations to organise, search and manipulate data:

- Schools hold data about students.
- Supermarkets hold data about stock levels of the items they sell and customer buying patterns.
- Companies hold data about the goods and services they produce and sell, the customers who buy them and the suppliers who provide the components.
- Websites and mobile apps like Twitter and Instagram hold data about users, the content they create and all the related links across a whole range of social media.

It is important that the data is ORGANISED and structured into records; for example, your school database will have a record for each individual student.

EXERCISE 32: DATABASES

1. What data will be in the school database about each student?
2. What data could be used to uniquely identify each student?

14.1 WHAT IS A DATABASE?

A database is a structured collection of data which uses tables, records and fields to organise that data. For example, your school will have a database containing all the personal details of the students, e.g. contact details, date of birth, address.

Each record will have a field with a unique reference or value so that each record is unique and there are no duplicate records.

The database structure will be organised so that users can easily extract information; for example, entering an employee code to find out when an employee's health and safety training needs to be renewed.

14.2 AVOIDING REPEATING DATA

This is part of a list of employees working at different branches held by a Human Resources department. A database structure must avoid repeating data to ensure that there are no duplicates or inconsistences, e.g. spelling errors.

	A	B	C	D	E	F	G	H	I
1	EmployeeID	FirstName	LastName	Address1	Address2	Town	PostCode	Training Level	Activity
2	CN41T7019	Todd	Vega	12 Eta Road	Royton	Oldham	OL7 3DF	BA1	Basic training
3	CNU4BZZ19	Michael	Carpenter	98 Tempor Road	Middleton	Oldham	OL10 8CS	HG2	Sell new products
4	CNQGZFN19	Hayley	Puckett	2 Aliquet St.	Chadderton	Oldham	OL5 9KM	MD1	Current products sales
5	CNXKWNP19	Harry	Lancaster	38 Tristique Avenue	Chadderton	Oldham	OL32 4BL	MD2	Info on new products
6	CN7ZXT619	Olivia	Carlson	12 Est Avenue	Failsworth	Manchester	M24 6HG	BA1	Training on products
7	CN4KZGK19	Haley	Duke	25 Libero. St.	Royton	Oldham	OL9 3TE	HG1	New product info to customers
8	CNHPR2G19	Anne	Burgess	51 Neque Road	Wythenshawe	Manchester	M15 4LK	HG2	New product sales to customer
9	CNIOVRC19	Henry	Haney	43 Risus. Av.	Denton	Oldham	OL15 8FG	BA1	Basic product training
10	CNNZDRN19	Montana	Holmes	39 Lobortis Ave	Burnage	Manchester	M23 5JM	MD1	Current product info to customer
11	CNF188719	Kirk	Sherman	43 Quisque Road	Royton	Oldham	OL4 6DE	BA1	Product training

You should be able to identify three fields where there could be duplication or inconsistency.

The problem with this system is:

1. **Inconsistent data:** The description of the activity for several training levels is different.
2. **Data redundancy:** Duplicate data may be held for employees' addresses, with areas and town names being repeated.

KEY POINT: Data duplication leads to data inconsistency

The aim of a database is that all data should be stored just ONCE; this is called a RELATIONAL DATABASE.

The problem above can be solved by splitting the data into separate tables. When the tables are linked together they create a RELATIONSHIP between the two tables through the use of the PRIMARY KEY.

The employee table will have a link to the training table by including the PRIMARY KEY of the training table (which is now called the FOREIGN KEY) in the employee table

	A	B	C	D	E	F	G	H
1	**EmployeeID**	**FirstName**	**LastName**	**Address1**	**Address2**	**Town**	**PostCode**	**TrainingLevel**
2	CN41T7019	Todd	Vega	12 Eta Road	Royton	Oldham	OL7 3DF	BA1
3	CNU4BZZ19	Michael	Carpenter	98 Tempor Road	Middleton	Oldham	OL10 8CS	HG2
4	CNQGZFN19	Hayley	Puckett	2 Aliquet St.	Chadderton	Oldham	OL5 9KM	MD1
5	CNXKWNP19	Harry	Lancaster	38 Tristique Avenue	Chadderton	Oldham	OL32 4BL	MD2
6	CN7ZXT619	Olivia	Carlson	12 Est Avenue	Failsworth	Manchester	M24 6HG	BA1
7	CN4KZGK19	Haley	Duke	25 Libero. St.	Royton	Oldham	OL9 3TE	HG1
8	CNHPR2G19	Anne	Burgess	51 Neque Road	Wythenshawe	Manchester	M15 4LK	HG2
9	CNIOVRC19	Henry	Haney	43 Risus. Av.	Denton	Oldham	OL15 8FG	BA1
10	CNNZDRN19	Montana	Holmes	39 Lobortis Ave	Burnage	Manchester	M23 5JM	MD1
11	CNF188719	Kirk	Sherman	43 Quisque Road	Royton	Oldham	OL4 6DE	BA1

Foreign Key

Primary Key

	A	B	C
1	**TrainingCode**	**Level**	**Activity**
2	BA1	Introductory	Product training
3	MD1	Intermediate 1	Current product info to customer
4	MD2	Intermediate 2	Current product sales to customer
5	HG1	Higher1	New product info to customer
6	HG2	Higher2	New product sales to customer

Other tables can also be linked together using suitable UNIQUE values in a record.

EXERCISE 33: PRIMARY KEYS

1. Identify the PRIMARY KEY in the original employee table.
2. The school is creating a database of staff cars to ensure that only authorised staff can park on the school site.
    a. Write down five possible fields in the cars table.
    b. State which field would be the primary key and give a reason.

Think about the apps you use on your mobile phone. Each time you use Twitter or Instagram on your phone to search for information you are using a database.

For example, you can search Instagram for people by name or username and the results you get will be based on people you follow, what types of photos you have liked or who you are connected to. All this information is stored in a record about you and is constantly updated whenever you add to your Instagram account.

In order to link all the content and searches that belong to you, you have a 'unique' identifier in the database, your username, which is used as a PRIMARY KEY. This unique identifier is used to link all the details of users and their content. When you add new content it will be linked to your username by using location stickers, location or hashtags.

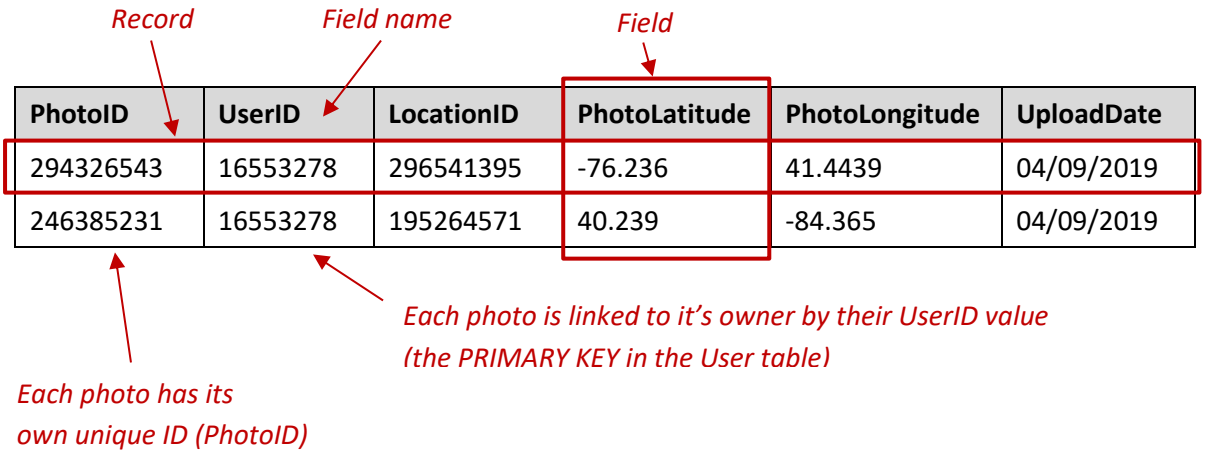
In each table, the data is divided into fields and records. Each record will have all the information about, for example, an Instagram user. The table is also divided into fields; each entry in a field will be the same type of data, e.g. date of birth.

We can think about the table as a set of rows and columns; this might be the table that stores username information.

Possible Instagram user table

UserID	Username	Email	DateOfBirth	CreationDate	LastLogin
1698732	manda	a_winfield@virgin.net	23.6.1992	07.05.2009	15.12.2019
16553278	AndyCaillon	Caillon_a@btinternet.com	5.2.1987	16.11.2011	24.12.2019

This might be a table that stores photo details:



14.2 DATABASE DATA TYPES

We have already discussed data types in Python (Topic 3). When we create a database, we also need to decide which data type to use for each field in the database so the computer knows what actions or calculations can be performed on that data. Common data types used in database software are:

Data Type	Description
Text	Text including telephone numbers (pseudo-numbers)
Number	Any type of number
Date/Time	Date and/or time
Boolean	True or False, Yes or No, On or Off (only two options)
Currency	Numbers formatted as a currency
Auto number	Automatically generated number – usually used to create unique values for a primary key

The data types used in the examples above:

- Text e.g. 'AndyCaillon', 'a_winfield@virgin.net'
- Number e.g. -76.236, 1698732
- Date/Time e.g. 4.9.2019, 23.6.1992

14.3 PRIMARY KEY

Databases can be complex data structures with many tables linking to data in other tables. The method that databases use to ensure that data is consistent – for example, that a username or customer name is always spelt correctly – is to have each piece of data stored just once and have numerical links to any other tables that need to display the data. This is achieved through the use of a PRIMARY KEY.

In the Instagram user table example shown above, the UserID is a PRIMARY KEY, a unique value that no other record in that table will ever use. Similarly, the photo details table uses a unique PhotoID number for each photo; again, this is unique and will never be used by any other record in the table.

EXERCISE 34: DATA TYPES

Look at the Python code used to create the 'games' table on the following page.

1. Which field is the most suitable as a primary key?
2. What data type is used in this field?
3. Explain why you have chosen this field.
4. Identify the data type used in the 'price' field.
5. Explain why you think this data type has been chosen.

14.4 SQL AND DATABASES

The universal language used to create and manipulate databases is called structured query language, or SQL (pronounced 'sequel').

In order to use Python and create a simple database, we need to know some basic SQL instructions:

These are the SQL statements we will be using:

- CREATE TABLE – this will create a table in a database
- INSERT INTO – insert new data into a table in a database
- SELECT – display data that is in a table in a database
- UPDATE – change data in a table in a database
- DELETE – remove data from a table in a database

CREATE TABLE

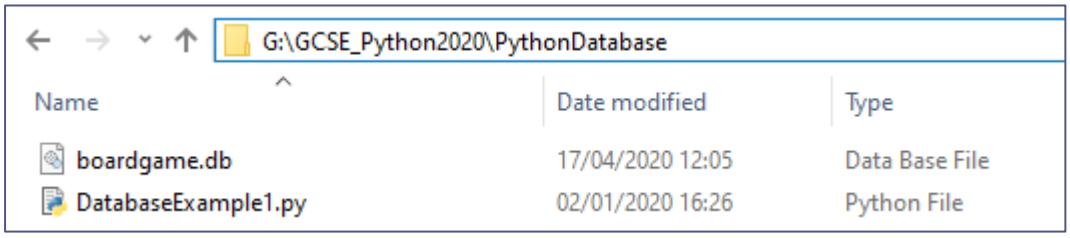
When the Python code is saved and run it creates an empty database with the fields specified:

```
import sqlite3 #  import the database module

#  create a connection'object' and name the database
conn = sqlite3.connect('boardgame.db')

#  create a cursor 'object' to perform the SQL instructions
c = conn.cursor()

#  Create the table with primary key and field names
c.execute("""CREATE TABLE games
             (code INTEGER PRIMARY KEY, name TEXT, price MONEY, date_bought TEXT)""")

#  Save the database
conn.commit()

#  Close the database connection
conn.close()
```

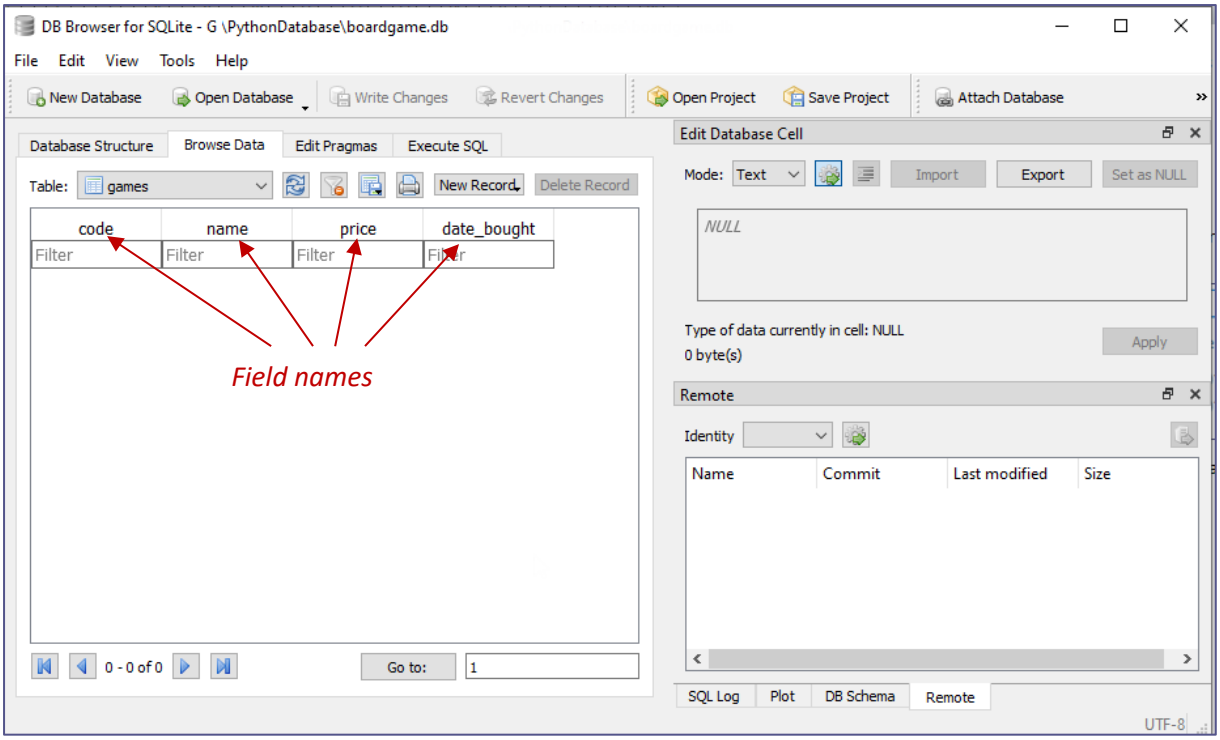
- code
- name
- price
- date_bought

Note: the data type used for the 'date_bought' field is text as no calculations will be made using this field.

This will be saved in the same folder as the Python file used to create it.



Using the SQLite Browser, we can see the empty table:



INSERT INTO

The database table is currently empty; we can now add the data using the INSERT INTO statement.

This can be done by adding an individual record or several records at the same time.

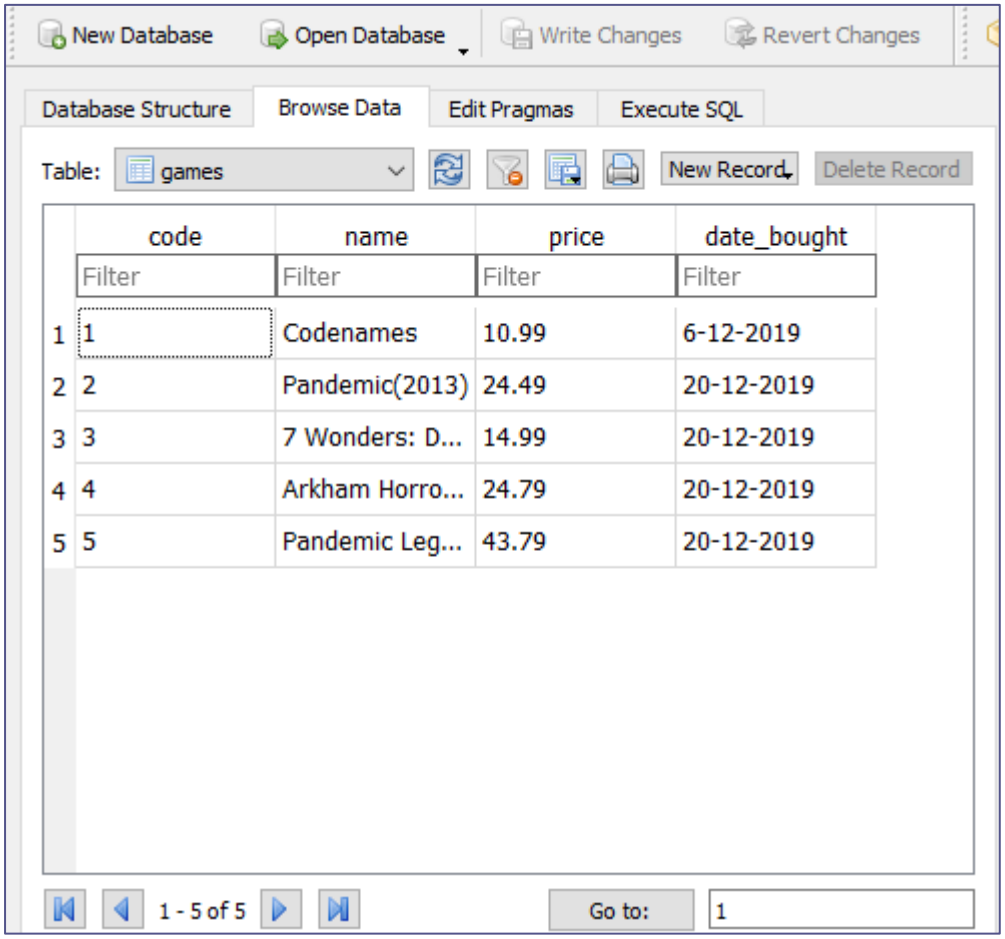
The code specifies the fields and the data but NOT the PRIMARY KEY. This will be automatically added each time a record is entered.

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Insert data into the table

c.execute("INSERT INTO games (name, price, date_bought)VALUES('Codenames', 10.99, '6-12-2019')")

#  save the table
conn.commit()

#  close the database connection
conn.close()
```


Adding several records to the database table using a list; again, the PRIMARY KEY will automatically be added.

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Insert multiple records

new_games = [('Pandemic(2013)', 24.49, '20-12-2019'),
             ('7 Wonders: Duel', 14.99, '20-12-2019'),
             ('Arkham Horror: The Card Game', 24.79,'20-12-2019'),
             ('Pandemic Legacy 1 Red', 43.79, '20-12-2019')
             ]

c.executemany('INSERT INTO games (name, price, date_bought) VALUES ( ?, ?, ?)',new_games)

#  save the table
conn.commit()

#  close the database connection
conn.close()
```

The database table now contains five records:



The table can be viewed using the DB Browser for SQLite.

All the records in the table can also be selected and printed out using Python.

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Use SQL statement to get the data
c.execute("SELECT code, name, price,date_bought FROM games")

#  Save the data in a variable called 'results'
results = c.fetchall()

#  Iterate (loop) through each record and print
for row in results:
    print(row)

#  close the database connection
conn.close()
```

SELECT

We can use the SELECT statement to choose which data from the table to view in a printed format.

The records are printed out:

```
(2, 'Pandemic(2013)', 24.49, '20-12-2019')
(4, 'Arkham Horror: The Card Game', 24.79, '20-12-2019')
(5, 'Pandemic Legacy 1 Red', 43.79, '20-12-2019')
>>>
```

You can use a symbol called a wildcard (*) in your SELECT statement to print all the records.

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Use SQL statement to get the data
c.execute("SELECT * FROM games")

results = c.fetchall()

for row in results:
    print(row)

#  close the database connection
conn.close()
```

```
(1, 'Codenames', 10.99, '6-12-2019')
(2, 'Pandemic(2013)', 24.49, '20-12-2019')
(3, '7 Wonders: Duel', 14.99, '20-12-2019')
(4, 'Arkham Horror: The Card Game', 24.79, '20-12-2019')
(5, 'Pandemic Legacy 1 Red', 43.79, '20-12-2019')
```


QUERY BY EXAMPLE GRID (QBE)

The query by example grid was developed in the 1970s as the use of databases in business became much more common. Since the people now using databases were not experts in using SQL, the simple grid layout allowed non-specialist users to easily find data without needing any detailed knowledge of how to write the SQL statements.

Using a query by example grid to find all the data in a table is simple:

1. Choose the fields you want to see and the table containing those fields
2. Include suitable search criteria (where relevant)
3. Choose 'show' for each data field that you want to see in the results
4. Execute or run the query

If you want to search for a specific record, you need to add the search criteria in the criteria row for that field. For example, if I want to know which games cost more than £15, I would add >15 in the criteria row of the 'price' field.

Field:	code	name	price	date_bought
Table:	games	games	games	games
Sort:				
Show:	☑	☑	☑	☑
Criteria:				
or:				

EXERCISE 35: QBE GRIDS

Using the query by example templates at the back of this resource:

1. Write a query that will find all games for which the cost is less than £20.
2. Write a query that will select the game where the code equals 'arkhr01' and print out the title and price only.

SELECT ... WHERE

The select statement can now be extended to add selection criteria using the keyword WHERE.

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Use SQL statement to get the data
c.execute("SELECT code, name, price,date_bought FROM games WHERE price > 15")

results = c.fetchall()

for row in results:
    print(row)

#  close the database connection
conn.close()
```

prints the results:

```
(2, 'Pandemic(2013)', 24.49, '20-12-2019')
(4, 'Arkham Horror: The Card Game', 24.79, '20-12-2019')
(5, 'Pandemic Legacy 1 Red', 43.79, '20-12-2019')
>>>
```

There is a range of operators that can be used with the 'where' clause:

Operator	Meaning
=	Equal to
<> or !=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
BETWEEN	Value between a given range including the bounds
LIKE	Matches a given pattern
IN	To find more than one value in a column

ORDER BY

If the dataset returned from your search contains many records, it may be easier to view the data by sorting on a particular field in ascending (from A to Z) or descending (from Z to A) order. Additional games have been added to the database to demonstrate this:

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Use SQL statement to get the data and ORDER BY
c.execute("SELECT name,price,date_bought FROM games ORDER BY name ASC")

results = c.fetchall()

for row in results:
    print(row)

# save the updated table
conn.commit()

#  close the database connection
conn.close()
```

```
('7 Wonders: Duel', 14.99, '20-12-2019')
('Arkham Horror: The Card Game', 24.79, '20-12-2019')
('Articulate', 32, '10/01/2020')
('Betrayal at House on the Hill', 21.99, '01/09/2019')
('Carcassonne', 31.97, '15/11/2019')
('Codenames', 10.99, '6-12-2019')
('Cosmic Encounter', 71.38, '01/09/2019')
('Fallout', 44.99, '06/10/2019')
('Flamme Rouge', 30.13, '06/10/2019')
('Gloomhaven', 170, '15/11/2019')
('Jaws', 24.99, '06/10/2019')
('King of Tokyo', 25.99, '10/01/2020')
('Mysterium', 32.99, '06/10/2019')
('One Night Ultimate Werewolf', 20.99, '15/11/2019')
('Pandemic Legacy 1 Red', 43.79, '20-12-2019')
('Pandemic(2013)', 24.49, '20-12-2019')
('Photosynthesis', 32.73, '06/10/2019')
("Pinch'n'Pass", 25, '06/10/2019')
('Quacks of Quedlinburg', 30.17, '01/09/2019')
('Root', 130, '06/10/2019')
('Scythe', 74.34, '15/11/2019')
('Splendor', 21.99, '01/09/2019')
('Ticket to Ride', 20.67, '15/11/2019')
('Villainous', 34.99, '10/01/2020')
```



```
import sqlite3

conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Use SQL statement to get the data and ORDER BY

c.execute("SELECT name,price,date_bought FROM games ORDER BY price DESC")

results = c.fetchall()

for row in results:
    print(row)

# save the updated table
conn.commit()

#  close the database connection
conn.close()
```

```
('Gloomhaven', 170, '15/11/2019')
('Root', 130, '06/10/2019')
('Scythe', 74.34, '15/11/2019')
('Cosmic Encounter', 71.38, '01/09/2019')
('Fallout', 44.99, '06/10/2019')
('Pandemic Legacy 1 Red', 43.79, '20-12-2019')
('Villainous', 34.99, '10/01/2020')
('Mysterium', 32.99, '06/10/2019')
('Photosynthesis', 32.73, '06/10/2019')
('Articulate', 32, '10/01/2020')
('Carcassonne', 31.97, '15/11/2019')
('Quacks of Quedlinburg', 30.17, '01/09/2019')
('Flamme Rouge', 30.13, '06/10/2019')
('King of Tokyo', 25.99, '10/01/2020')
("Pinch'n'Pass", 25, '06/10/2019')
('Jaws', 24.99, '06/10/2019')
('Arkham Horror: The Card Game', 24.79, '20-12-2019')
('Pandemic(2013)', 24.49, '20-12-2019')
('Betrayal at House on the Hill', 21.99, '01/09/2019')
('Splendor', 21.99, '01/09/2019')
('One Night Ultimate Werewolf', 20.99, '15/11/2019')
('Ticket to Ride', 20.67, '15/11/2019')
('7 Wonders: Duel', 14.99, '20-12-2019')
('Codenames', 10.99, '6-12-2019')
```


UPDATE

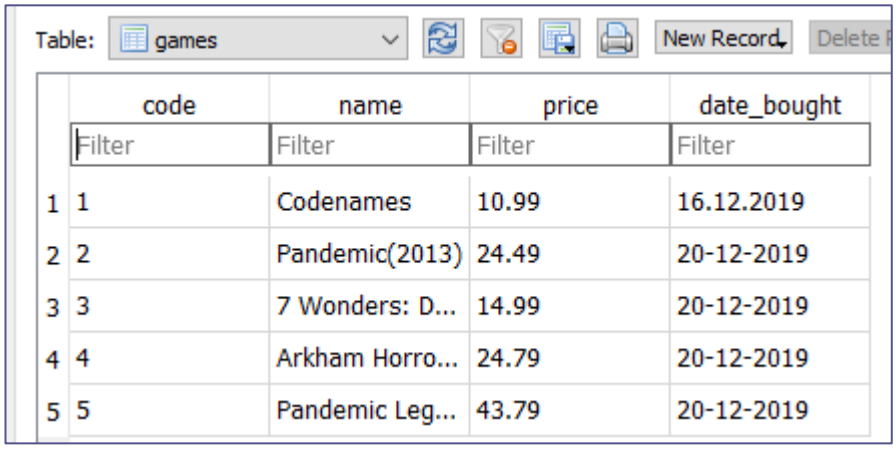
Sometimes data in our database needs to be changed; perhaps a customer has changed their address or the name of a product has changed.

In this example, the 'date_bought' has been entered incorrectly for the 'Codenames' game and needs to be changed to 16.12.2019.

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Use SQL statement to get the data and update
c.execute("UPDATE games SET date_bought = '16.12.2019' WHERE name = 'Codenames'")

# save the updated record
conn.commit()

#  close the database connection
conn.close()
```

It is important to save the database after updating or deleting any records.

Here is the updated table viewed in DB Browser:



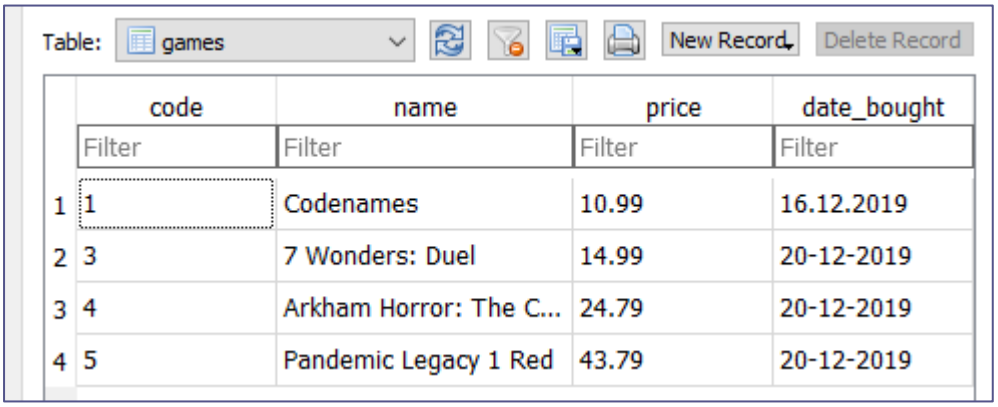
	code	name	price	date_bought
	Filter	Filter	Filter	Filter
1	1	Codenames	10.99	16.12.2019
2	2	Pandemic(2013)	24.49	20-12-2019
3	3	7 Wonders: D...	14.99	20-12-2019
4	4	Arkham Horro...	24.79	20-12-2019
5	5	Pandemic Leg...	43.79	20-12-2019

DELETE

Using the DELETE statement is very similar; the record is selected using search criteria, then removed.

```
import sqlite3


conn = sqlite3.connect('boardgame.db')

c = conn.cursor()

#  Use SQL statement to get the data and delete
c.execute("DELETE FROM games WHERE name = 'Pandemic(2013)'")

# save the updated table
conn.commit()

#  close the database connection
conn.close()
```


The database now has just four records:



	code	name	price	date_bought
	Filter	Filter	Filter	Filter
1	1	Codenames	10.99	16.12.2019
2	3	7 Wonders: Duel	14.99	20-12-2019
3	4	Arkham Horror: The C...	24.79	20-12-2019
4	5	Pandemic Legacy 1 Red	43.79	20-12-2019

The record with the unique PRIMARY KEY 2 has now been deleted, and that number will never be used again for any new records.

14.5 EXTENDING QUERIES USING OPERATORS

We can extend our queries using comparison and logical operators so that we can create more complex searches.

Examples:

To find employees who live in Oldham OR work in the Oldham branch:

Town = “Oldham” **OR** BranchCode = “OL1”

To find employees who live in Manchester **AND** work in the Manchester branch:

Town = “Manchester” **AND** BranchCode = “MA2”

EmployeeID	FirstName	LastName	Address1	Address2	Town	PostCode	BranchCode
CN41T7019	Todd	Vega	12 Eta Road	Royton	Oldham	OL7 3DF	OL1
CNU4BZZ19	Michael	Carpenter	98 Tempor Road	Middleton	Oldham	OL10 8CS	OL1
CNQGZFN19	Hayley	Puckett	2 Aliquet St.	Chadderton	Oldham	OL5 9KM	OL1
CNXKWNP19	Harry	Lancaster	38 Tristique Avenue	Chadderton	Oldham	OL32 4BL	RO3
CN7ZXT619	Olivia	Carlson	12 Est Avenue	Failsworth	Manchester	M24 6HG	MA2
CN4KZGK19	Haley	Duke	25 Libero. St.	Royton	Oldham	OL9 3TE	MA2
CNHPR2G19	Anne	Burgess	51 Neque Road	Wythenshawe	Manchester	M15 4LK	MA2
CNIOVRC19	Henry	Haney	43 Risus. Av.	Denton	Oldham	OL15 8FG	RO2
CNNZDRN19	Montana	Holmes	39 Lobortis Ave	Burnage	Manchester	M23 5JM	OL1
CNF188719	Kirk	Sherman	43 Quisque Road	Royton	Oldham	OL4 6DE	RO3

EXERCISE 36: DATABASE QUESTIONS

A database was set up to store details of books at a library. Below is a sample of the data.

Title	Author	Format	Pages	ISBN	PublicationDate
Drawing Manga	9colorstudio	Paperback	44	9781615645619	2013
Flashpoint	Johns, Geoff	Paperback	166	9781401233389	2012
House of M	Bendis, Brian	Paperback	224	9780785117216	2006
Punishermax: Homeless	Aaron, Jason	Paperback	128	9780785152101	2012
Punishermax: Frank	Aaron, Jason	Paperback	112	9780785152095	2012
Ripclaw	Aaron, Jason	Paperback	208	9781607060543	2009
Wallace & Gromit: The Bootiful Game	Rimmer, Ian	Paperback	48	9781840239485	2005

1. State the number of fields in each record.
2. Identify which field should be used as the primary key, and explain why.
3. State the title of the book(s) that are returned using the following search criteria:

   (Name = * Jason) AND (PublicationDate < 2012)

EXERCISE 37: COMPARISON OPERATORS

1. Copy and complete the following table by explaining the meaning of the following *comparison operators* used in queries.

Operator	Meaning
=	
>	
<	
>=	
<=	
<>	

EXERCISE 38: DATABASE QUERIES

The following diagram shows part of a membership database.

Member ID	Surname	Forename	Gender	Age	City	Year Joined
0001	Kirkland	Haley	M	18	York	2014
0002	Foley	Thor	M	19	Birmingham	2015
0003	Sanders	Nomlanga	M	17	Manchester	2017
0004	Allen	Edan	M	20	Leeds	2015
0005	Brewer	Jane	F	16	London	2016
0006	Abbott	Eagan	M	18	Birmingham	2017
0007	Kline	Ciara	M	18	Birmingham	2014
0008	Miles	Ivor	F	20	Bristol	2018
0009	Patrick	Gannon	M	16	York	2018
0010	Miranda	Jelani	M	17	Sheffield	2019
0011	Pena	Jelani	F	18	Sheffield	2014
0012	Austin	Cherokee	M	16	Bristol	2018
0013	Wilkins	Rafael	M	19	London	2014
0014	Cote	Briar	M	16	Manchester	2019
0015	Good	Josephine	F	16	Newcastle	2016
0016	Noble	Arthur	M	17	London	2018
0017	Good	Zachery	F	19	Leeds	2016
0018	Valencia	Solomon	F	18	York	2017
0019	Eaton	Brenden	M	17	London	2013
0020	Trevino	Andrew	M	16	Bristol	2016
0021	Mccarthy	Kalia	M	16	York	2015
0022	Beach	Beck	M	16	Bristol	2014
0023	David	Cora	M	17	York	2017
0024	Castillo	Kitra	F	18	Sheffield	2015
0025	Sawyer	Nomlanga	F	19	Manchester	2018
0026	Carver	Tyrone	M	16	Leeds	2019
0027	Bullock	Lisandra	M	19	Sheffield	2014
0028	Hurst	Camilla	F	20	Leeds	2016
0029	Mcfadden	Hannah	M	16	York	2015
0030	Mcintyre	Kareem	F	19	Birmingham	2015

For the following question:

State the search criteria which would have been input to produce the following results.

Member ID	Surname	Forename	Gender	Age	City	Year Joined
0002	Foley	Thor	M	19	Birmingham	2015
0004	Allen	Edan	M	20	Leeds	2015
0013	Wilkins	Rafael	M	19	London	2014
0027	Bullock	Lisandra	M	19	Sheffield	2014

The answer is: **Gender = 'M' AND Age >= 19**

Using the same membership database from the previous page, state the search criteria which would have been input to produce the following results:

1.

Member ID	Surname	Forename	Gender	Age	City	Year Joined
0004	Allen	Edan	M	20	Leeds	2015
0010	Miranda	Jelani	M	17	Sheffield	2019
0011	Pena	Jelani	F	18	Sheffield	2014
0017	Good	Zachery	F	19	Leeds	2016
0024	Castillo	Kitra	F	18	Sheffield	2015
0026	Carver	Tyrone	M	16	Leeds	2019
0027	Bullock	Lisandra	M	19	Sheffield	2014
0028	Hurst	Camilla	F	20	Leeds	2016

2.

Member ID	Surname	Forename	Gender	Age	City	Year Joined
0009	Patrick	Gannon	M	16	York	2018
0010	Miranda	Jelani	M	17	Sheffield	2019
0012	Austin	Cherokee	M	16	Bristol	2018
0014	Cote	Briar	M	16	Manchester	2019
0016	Noble	Arthur	M	17	London	2018
0026	Carver	Tyrone	M	16	Leeds	2019

3.

Member ID	Surname	Forename	Gender	Age	City	Year Joined
0017	Good	Zachery	F	19	Leeds	2016
0028	Hurst	Camilla	F	20	Leeds	2016

4.

Member ID	Surname	Forename	Gender	Age	City	Year Joined
0005	Brewer	Jane	F	16	London	2016
0008	Miles	Ivor	F	20	Bristol	2018
0011	Pena	Jelani	F	18	Sheffield	2014
0015	Good	Josephine	F	16	Newcastle	2016
0018	Valencia	Solomon	F	18	York	2017
0024	Castillo	Kitra	F	18	Sheffield	2015
0025	Sawyer	Nomlanga	F	19	Manchester	2018
0030	Mcintyre	Kareem	F	19	Birmingham	2015

14.6 SQL AND RELATIONAL DATABASES

The database examples used so far have only one table; all the databases used in real life will have multiple tables which link to each other using the PRIMARY KEY in each table as a field in a linked table. This linked field is called the FOREIGN KEY and it is the links between the tables that ensure that the data in the database is CONSISTENT and that there is only one copy of each item of data, i.e. there is no redundant data.

The following example uses five tables which are all linked together.

```
CREATE TABLE IF NOT EXISTS Customers(
CustomerID INTEGER PRIMARY KEY,
FirstName TEXT,
LastName TEXT,
Address1 TEXT,
Address2 TEXT,
Town TEXT,
PostCode TEXT);

CREATE TABLE IF NOT EXISTS Genre(GenreID INTEGER PRIMARY KEY,Genre TEXT);

CREATE TABLE IF NOT EXISTS Category(CategoryID INTEGER PRIMARY KEY,Category TEXT);

CREATE TABLE IF NOT EXISTS DVDS(
DVD_ID INTEGER PRIMARY KEY,
Title TEXT,
GenreID INTEGER,
CategoryID INTEGER,
YearReleased TEXT,
Director TEXT,
FOREIGN KEY(GenreID) REFERENCES Genre(GenreID),
FOREIGN KEY(CategoryID) REFERENCES Category(CategoryID));

CREATE TABLE  IF NOT EXISTS Rentals(
RentalID INTEGER PRIMARY KEY,
CustomerID INTEGER,
DVD_ID INTEGER,
DateRented TEXT,
DateReturned TEXT,
FOREIGN KEY(CustomerID) REFERENCES Customers(CustomerID),
FOREIGN KEY(DVD_ID) REFERENCES DVDS(DVD_ID));
```

In the DVDs table, a FOREIGN KEY provides links to the Category table (e.g. U, PG, 12) and the Genre table (e.g. comedy, action, sci-fi).

The Rentals table also uses a FOREIGN KEY to link to the Customer table and the DVDS table. This can be shown in the results from a search for all rentals:

```
SELECT dvds.Title,dvds.Director,dvds.YearReleased
FROM dvds,Category WHERE dvds.CategoryID=Category.CategoryID AND Category ='18'
```

Title	Director	YearReleased
Gangs of New York	Martin Scorcese	2002

The PRIMARY KEY for an 18 film is 5 and the PRIMARY KEY for historical films is 6.

Table: Category

	CategoryID	Category
	Filter	Filter
1	1	U
2	2	PG
3	3	12
4	4	15
5	5	18

	GenreID	Genre
	Filter	Filter
1	1	action
2	2	adventure
3	3	comedy
4	4	crime
5	5	drama
6	6	historical
7	7	horror
8	8	musicals
9	9	sci-fi
10	10	war
11	11	westerns

Table: DVDS

	DVD_ID	Title	GenreID	CategoryID	YearReleased	Director
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	Austin Powers: International Man of Mystery	3	4	1997	Jay Roach
2	2	Shaun of the Dead	3	4	2004	Edgar Wright
3	3	Be Kind Rewind	3	3	2008	Michel Gondry
4	4	No Country for Old Men	11	4	2007	Ethan & Joel ...
5	5	The Hurt Locker	10	4	2009	Kathryn Bigelow
6	6	Gangs of New York	6	5	2002	Martin Scorcese
7	7	Shrek	3	1	2001	Andrew Adam...
8	8	Harry Potter and the Sorcerers Stone	2	2	2001	Chris Columbus
9	9	WALL-E	9	1	2008	Andrew Stanton
10	10	Finding Nemo	3	1	2003	Andrew Stanton

14.7 SEARCHING MULTIPLE TABLES

Example:

Find the name, DVD title and date borrowed for all customers with DVDs not yet returned.

1. List the fields and the tables for the search.

```
SELECT Customers.FirstName, Customers.LastName, DVDS.Title , Rentals.DateRented
FROM Customers, DVDS, Rentals
```

   When you are searching for data from more than one table, you need to specify the table name followed by a 'dot' and then the field name.

2. Add the search criteria; we are looking for Rentals with no returned date.

```
SELECT Customers.FirstName, Customers.LastName, DVDS.Title , Rentals.DateRented
FROM Customers, DVDS, Rentals
WHERE Rentals.DateReturned = ''
```

3. Link the FOREIGN KEY in the Rentals table with the PRIMARY KEY in the Customer table.

```
SELECT Customers.FirstName, Customers.LastName, DVDS.Title , Rentals.DateRented
FROM Customers, DVDS, Rentals
WHERE Rentals.DateReturned = ''
AND Rentals.CustomerID = Customers.CustomerID
```

   This ensures that the customer data referenced by the Foreign Key in the Rental table can be extracted from the Customer table.

4. Link the FOREIGN KEY in the Rentals table with the PRIMARY KEY in the DVDS table.

```
SELECT Customers.FirstName, Customers.LastName, DVDS.Title , Rentals.DateRented
FROM Customers, DVDS, Rentals
WHERE Rentals.DateReturned = ''
AND Rentals.CustomerID = Customers.CustomerID
AND Rentals.DVD_ID = DVDS.DVD_ID
```

   Again, this ensures that the DVD data referenced by the Foreign Key in the Rental table can be extracted from the DVDS table.

Running the SQL query shows the seven records for DVDs that have not yet been returned:

Search Result:

FirstName	LastName	Title	DateRented
Todd	Vega	Shaun of the Dead	2020/03/26
Montana	Holmes	Shrek	2020/04/25
Kirk	Sherman	Harry Potter and the Sorcerers Stone	2020/04/28
Kirk	Sherman	WALL-E	2020/04/28
Olivia	Carlson	Be Kind Rewind	2020/04/01
Olivia	Carlson	Austin Powers: International Man of Mystery	2020/04/02
Haley	Duke	Gangs of New York	2020/04/02

Table View:

	RentalID	CustomerID	DVD_ID	DateRented	DateReturned
	Filter	Filter	Filter	Filter	Filter
1	1	2	1	2020/03/25	2020/04/05
2	2	1	2	2020/03/26	
3	3	2	5	2020/04/03	2020/04/11
4	4	7	4	2020/03/20	2020/03/25
5	5	9	7	2020/04/25	
6	6	10	8	2020/04/28	
7	7	10	9	2020/04/28	
8	8	5	3	2020/04/01	
9	9	5	1	2020/04/02	
10	10	6	6	2020/04/02	

Why do the CustomerID and DVD_ID fields just show numbers? These are the FOREIGN KEYS in the Rentals table. In the Customer table we can see that CustomerID 1 is Todd Vega, who has borrowed DVD_ID 2, which is Shaun of the Dead.

Table: Customers

	CustomerID	FirstName	LastName
	Filter	Filter	Filter
1	1	Todd	Vega
2	2	Michael	Carpenter
3	3	Hayley	Puckett

Table: DVDS

	DVD_ID	Title
	Filter	Filter
1	1	Austin Powers: International Man of Mystery
2	2	Shaun of the Dead
3	3	Be Kind Rewind
4	4	No Country for Old Men

EXERCISE 39: MULTIPLE TABLE SEARCH

Write the SQL code to search for:

1. Title and Director for all DVDs with a category rating of 15.
2. Title and YearReleased for all DVDs in the 'comedy' genre.

