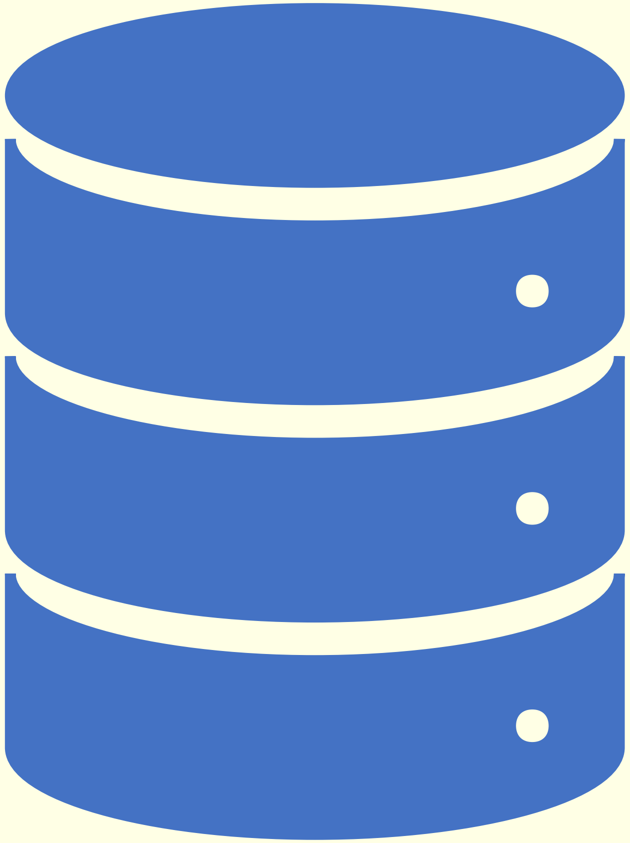
Files, Records, Length and Hashing

- Explain the concept of a relational database
- Define the terms: flat file, entity, attribute, primary key, foreign key, secondary key, entity relationship modelling, referential integrity



Recap

Key term

Record A single unit of information in a database. It is normally made up of *fields*. So a student file would be made up of many records. Each record is about one student and holds fields such as student number, surname, date of birth, gender, and so on.

Flat File Databases

A flat file database consists of data stored within a plain text file.

Records are stored one per line and each attribute of the record is separated by a delimiter – often in the form of a comma or tab. Often saved as a CSV.

A flat file is the simplest form of database and simplicity is its biggest advantage.

Typical uses:

- Storing contact details
- Small product database
- Music files

Example

A typical example of a flat-file database is an address book. Here is a view of part of one:

First name	Last name	Telephone	Street	City	Post code	DOB
Claire	Pate	1 55 791 7964-8421	1434 Aenean Road	Iowa City	K3I 1RF	6/28/1999
Virginia	Landry	1 61 306 9087-9418	404 Morbi Road	Rock Island	EI3O 7QR	1/23/1974
Orli	Goodwin	1 51 119 4068-1665	704-6375 Varius St.	Lynwood	CG12 9LQ	9/26/1984
Callie	Hodge	1 70 829 9014-9968	PO Box 362, 5198 Vulputate, St	Wichita Falls	D1Z 9AN	07/05/1978
Rhonda	Pugh	1 44 202 4884-7705	PO Box 250, 7653 Fusce Road	West Covina	S5 9OD	6/23/1984
Dara	Goff	1 70 115 3175-0607	844-4722 Felis St	Knoxville	KE9C 7XR	10/03/1999

You can easily understand the concept of a flat-file database by envisaging it as a spreadsheet or document table.

Serial File

- Data is stored in the order in which it is entered.
- For example, a text file to store a playlist
- This is a database containing only a single table of information
- Each new record is added to the end of the file
- Simple way to store date, if all we want to do is keep a record of songs in no particular order.

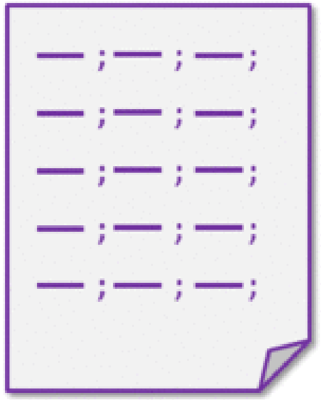
main.py playlist.txt

```
I Gotta Feeling;The Black Eyed Peas;4:05
Hey Brother;Avicii;4:15
This is the life;Amy MacDonald;3:06
Wolrd, Hold On;Bob Sinclar;6:41
Paradise;Coldplay;4:23
Memories;David Guetta;3:30
Hot 'n Cold;Katy Perry;3:40
Our House;Madness;3:12
Timber;Pitbull;3:25
```

Python example

```
file = open("file.csv","r")

for line in file:

    print( line )

file.close()
```

```
file = open("file.csv,"r")

for line in file:

    fields = line.split("; ")

    field1 = fields[0]

    field2 = fields[1]

    field3 = fields[2]

    print(field1 + " " + field2 + " " + field3)
```



```python

#Repeat for each song in the text file
for line in file:

  #Let's split the line into an array called "fields" using the ";" as a separator:
  fields = line.split(";")

  #and let's extract the data:
  songTitle = fields[0]
  artist = fields[1]
  duration = fields[2]

  #Print the song
  print(songTitle + " by " + artist + " Duration: " + duration)

#It is good practice to close the file at the end to free up resources
file.close()
```

main.py playlist.txt

```
I Gotta Feeling;The Black Eyed Peas;4:05
Hey Brother;Avicii;4:15
This is the life;Amy MacDonald;3:06
Wolrd, Hold On;Bob Sinclar;6:41
Paradise;Coldplay;4:23
Memories;David Guetta;3:30
Hot 'n Cold;Katy Perry;3:40
Our House;Madness;3:12
Timber;Pitbull;3:25
```

Drawbacks of serial files

- To locate a particular record, it is necessary to start at the beginning of the file and examine each record in turn until the required record is found or the end of the file is reached.
- If the file is large then this could take some time!

Sequential File

- This is when the data in a file can be sorted by a field.
- For example, sorting the records by the 'Amount' field

Transactions.csv

```
Sender,Recipient,Amount
Sam,George,150
Billy,Sam,100
George,Billy,50
```

- This makes searching a little easier as the records are ordered, alphabetically or numerical order, for example by ordered by Amount

Drawbacks of sequential files

- In order to generate a sequential file, at intervals the data in the file has to be sorted.
- This would involve writing the date in order to another file.
- The file has to be sorted before searched.

Searching using Indexes

- Sequential files can be searched more quickly by producing a separate index file
- The data is divided up into categories, e.g. n
- Each category is linked to a position in the d
- The number of records that must be searche

Index file			Data file	
Category	Data file start position		Position	Data
A	1		1	Abbott
B	10		2	Abby
C	20		3	Abercrombie
D	45		4	Agamemnon
E	80		5	Albemarle
			6	Alvarez
			7	Angstrom
			8	Anthracite
			9	Avery
			10	Baird
			11	Barr
			12	Barry
			13	Barton
			14	Brennan
			15	Buckley
			16	Bullock
			17	Bush

Drawbacks of Flat file databases

- Duplicate data is stored so storage is wasted ie. Data redundancy
- Changes to data may require rewriting the entire file

Fixed and variable length fields

Fields in records can be fixed or variable in length, and this in turn gives rise to fixed or variable length records.

S	m	i	t	h										

Fixed – each field has the same number of bytes in length – easy to program but wasteful of space.

Variable length records are more efficient in terms of space (memory), but can be harder to process.

A common file type with variable length records is the CSV file.

CSV stands for comma-separated variable, where each field is separated by a comma. An obvious drawback with this approach is that the data within a field must not contain a comma.

Some systems allow a different separator to be used or for a comma within the data to be flagged as data in some way.

Here is a possible structure of part of a student record in CSV format, showing surname, forename, gender and student number.

S	m	i	t	h	,	J	o	h	n	,	m	,	2	1	6	6	8	,

Hashing

- Hashing is a method of transforming a string of characters in a record into a shortened form that can be used as a disk address.
- This shortened form (has value) can be used to access a record from a database more quickly than by using the complete string.
- Typically, multiple records can usually produce some hash values that are the same.
- In this case, the data is located in the next available space (or block) on the storage medium, so some serial searching may be necessary.

Hashing Example

- Account number 2563546 generates the disk address 546. This leads to a block of records beginning at position 546. The disk address 546 is accessed and the record is written at that location.
- Of course, the account number 5756546 will also generate the same address. In this case, if the position is already occupied, the record is written to the next sequentially available location.
- If the block is full, then any records that generate that address will be written to an overflow area specially designated for such data collisions.

Learning Aims

- Explain the concept of a relational database
- Define the terms: flat file, entity, attribute, primary key, foreign key, secondary key, entity relationship modelling, referential integrity
- Produce an entity relationship model for a simple scenario involving multiple entities

A **flat file database** is one that stores all data in a single table

It is simple and easy to understand but causes data redundancy, inefficient storage and is harder to maintain

ID	first_name	last_name	Personal tutor	FormRoom
1	sam	smith	Roger Hinds	6b
2	fred	lynch	Jess Little	8j
3	depak	noor	Roger Hinds	6b
4	archie	henns	Mary Kent	8k
5	helga	jordan	Mary Kent	8k
6	lizzy	bell	Mary Kent	8k
7	xavier	horten	Jack Berry	3m

- This table has redundant data - the tutor and form room information repeats
- This is inefficient
- If a tutor changed their name we would need to find all instances of that name and change them all
- Missing any would mean the **table** had inconsistent data

What is a relational database

- The key difference between a relational database and a flat-file database is that in a relational database the data is **grouped into entities** and stored in **multiple linked tables.**
- It uses keys to connect related data which reduces data **redundancy**, makes efficient use of storage and is easier to maintain

Relational database

- A new **table** could be created to store the tutor information and the tutor information in the student table could be moved to the new **table**. Then a **foreign key** in the student **table** (TutorID) could link a student to their tutor

ID	first_name	last_name	TutorID
1	sam	smith	1
2	fred	lynch	2
3	depak	noor	1
4	archie	henns	3
5	helga	jordan	3
6	lizzy	bell	3
7	xavier	horten	4

TutorID	Tutor Name	FormRoom
1	Roger Hinds	6b
2	Jess Little	8j
3	Mary Kent	8k
4	Jack Berry	3m

- Now the name of each tutor and their form room is stored only once
- This means if they change only one piece of data, the data is updated in the entire **database** and **Inconsistency** is avoided

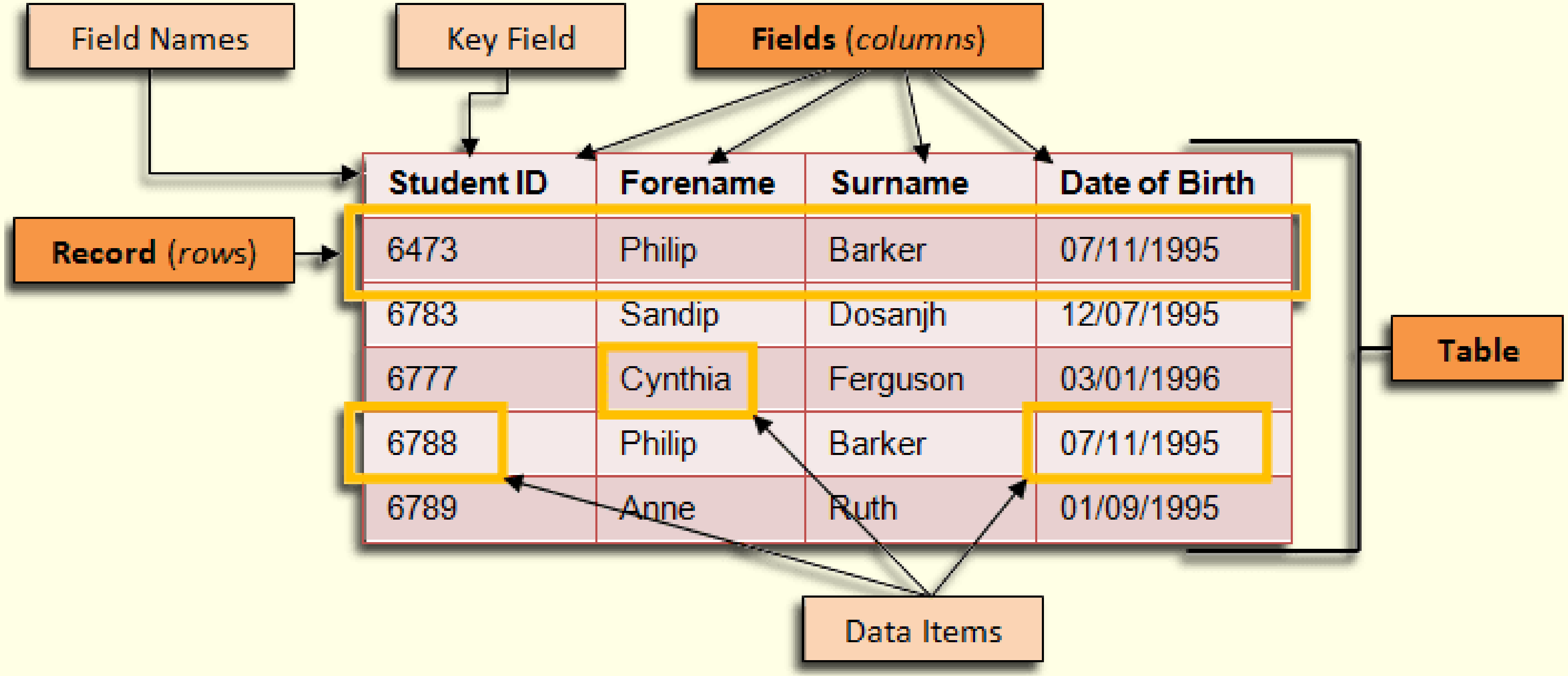
Key terms

- **Entity** - A real-world thing that is modelled in a database. For example, a customer, stock item, sale.

- **Attributes** - Each entity in a database system has attributes, for example: Title, Firstname, Surname

What is a database table?

- In a relational database, **each table represents the attributes of one entity**. Database tables are made up of **record** and **attributes** (**fields)**.



Key terms

Term	Definition
Field	A single piece of data in a record
Record	A group of related fields, representing one data entry
Table	A collection of records with a similar structure
Primary key	A unique identifier for each record in a table. Usually an ID number
Compound primary key (sometimes called composite)	A combination of (2 or more) fields that is unique for all records
Foreign key	A field in a table that refers to the primary key in another table. Used to link tables and create relationships
Secondary key	A field or fields that are indexed for faster searching
Database Management System (DBMS)	Software used to manage databases. Examples include MySQL, Oracle, Microsoft SQL Server, PostgreSQL

What is an entity and how are they related to database tables?

- An entity is a “real-world thing” about which data can be held in a database.
- In a relational database, each entity corresponds to a separate table in the database.

Examples of entities include:

- Customers
- Products
- Pupils
- Suppliers
- Loans
- Videos/DVD’s
- Flights
- Employees
- Treatments
- Contracts
- Library books
- Cars
- Orders
- Zoo animals
- Rentals

What is an attribute and how are they related to database tables?

- Attributes are the facts, aspects, properties, or details about an entity.

Examples of attributes include:

Entity	**Library books**	**Flights**	**employee**
Attributes	ISBN number author category	Flight No aircraft type departure Arrival date/time destination	Name Gender address DOB qualifications job title

In a database, each attribute corresponds to a separate field in the database.

Entity descriptions

An entity description is normally written using the format
Entity1 (Attribute1, Attribute2…)

The entity description for Dentist is therefore written:

Dentist (Title, Firstname, Surname, Qualification)

Entity identifier and primary key

Each entity needs to have an **identifier** which uniquely identifies the entity. In a relational database, the entity identifier is known as the **primary key** and it will be referred to as such in this section.

Clearly none of the attributes so far identified for **Dentist** and **Patient** is suitable as a primary key.

A numeric or string ID such as D13649 could be used.

In the entity description, the primary key is underlined:

Dentist (<u>DentistID</u>, Title, Firstname, Surname, Qualification)

Is National Insurance Number a suitable primary key for Patient? If not, why not?

NI number is not a suitable primary key because many patients may not know their NI number and some patients may not have an NI number, e.g. if not British.

What is a primary key and how are they used to link database tables?

- Primary key is used to store an attribute that makes that particular entity entry in the database unique.

For example:

- NHS number
- passport number
- vehicle registration
- booking reference
- flight number

Secondary key

A database needs to be set up so that it can be searched quickly. An **index** of all the primary keys in the database, and where the record is held, is automatically maintained by the database software.

However, more than one index may be needed.

If for example a patient rings up to make an appointment with the dentist, they are unlikely to know their patient ID, **A secondary index** on surname is likely to be held.

Rules

Rules:

- Each field in a table has a **unique name**.
- Each field stores a **single item of data** – For example, a field called Date of Birth would store no more than one date of birth value.
- Each field has a particular **data type** – for example, text, Boolean, integer, date/time, etc.
- Each field can have its own **validation rules** – these ensure that data recorded meets certain rules.

Example

Here is part of a data table. It is designed to store details of hotel-room bookings. It shows three rows and four columns.

room_number	date	room_type	customer_ref
101	21/03/2015	double	26335
310	22/03/2015	single	45335
250	23/03/2015	double	36587

Note that a combination of room number and date is sufficient to make a primary key field. Many tables make use of a special reference such as student_number to produce a key field.

Relationships

- The tables of a relational database are linked though relationships.
- **The tables in a relational database are linked by the Primary key in one table being added as a foreign key in another table to form a relationship between the entities.**

Example

Here, the field customer_ref forms the primary key in tblCustomer, but is a foreign key in tblRoom. It allows a relationship to link the tables.

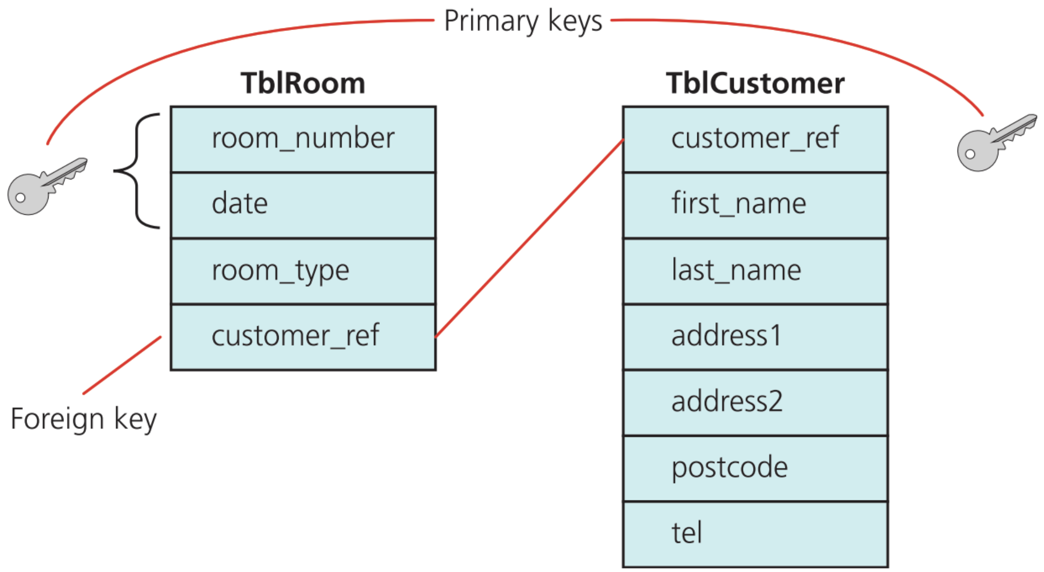


Figure 15.2 Relational database

Relationships between entities

The different entities in a system may be linked in some way, and the two entities are said to be related.

There are only three different 'degrees' of relationship between two entities.

A relationship may be

One-to-one	Examples of such a relationship include the relationship between Husband and Wife, Country and Prime Minister.
One-to-many	Examples include the relationship between Mother and Child, Customer and Order, Borrower and Library Book.
Many-to-many	Examples include the relationship between Student and Course, Stock Item and Supplier, Film and Actor.

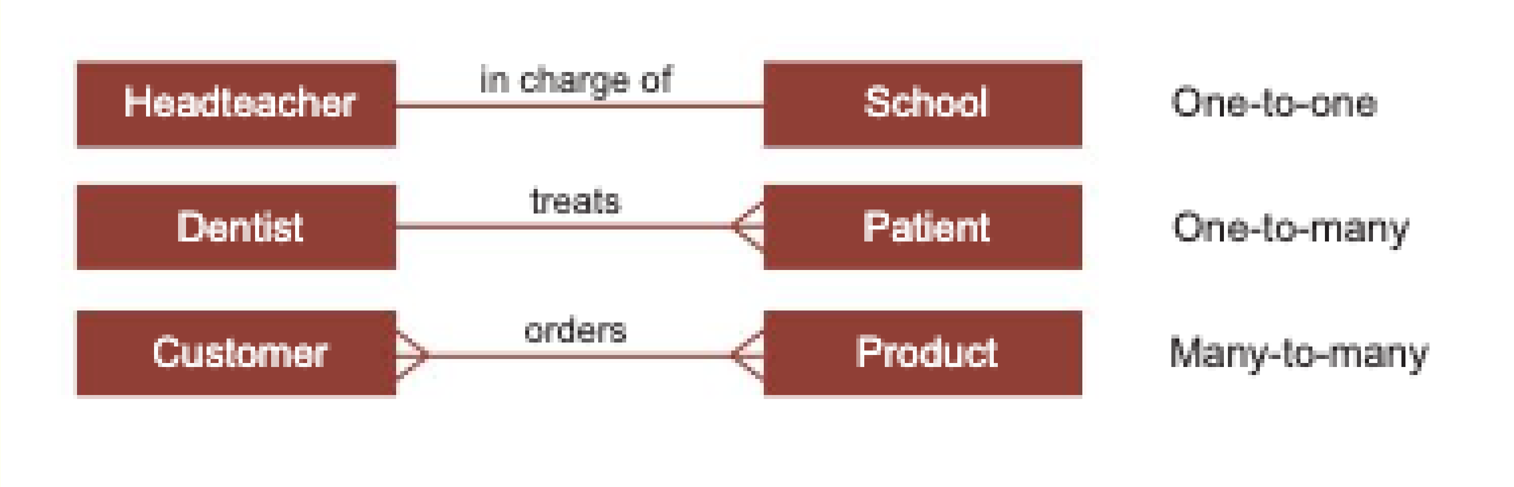
Entity relationship modelling

The relationship between entities can be modelled graphically.

An entity relationship diagram is a diagrammatic way of representing the relationships between the entities in a database.

To show the relationship between two entities, both the degree and the name of the relationship need to be specified.

In the first relationship shown below, the degree is one-to-one, the name of the relationship is in charge of.



The concept of a relational database

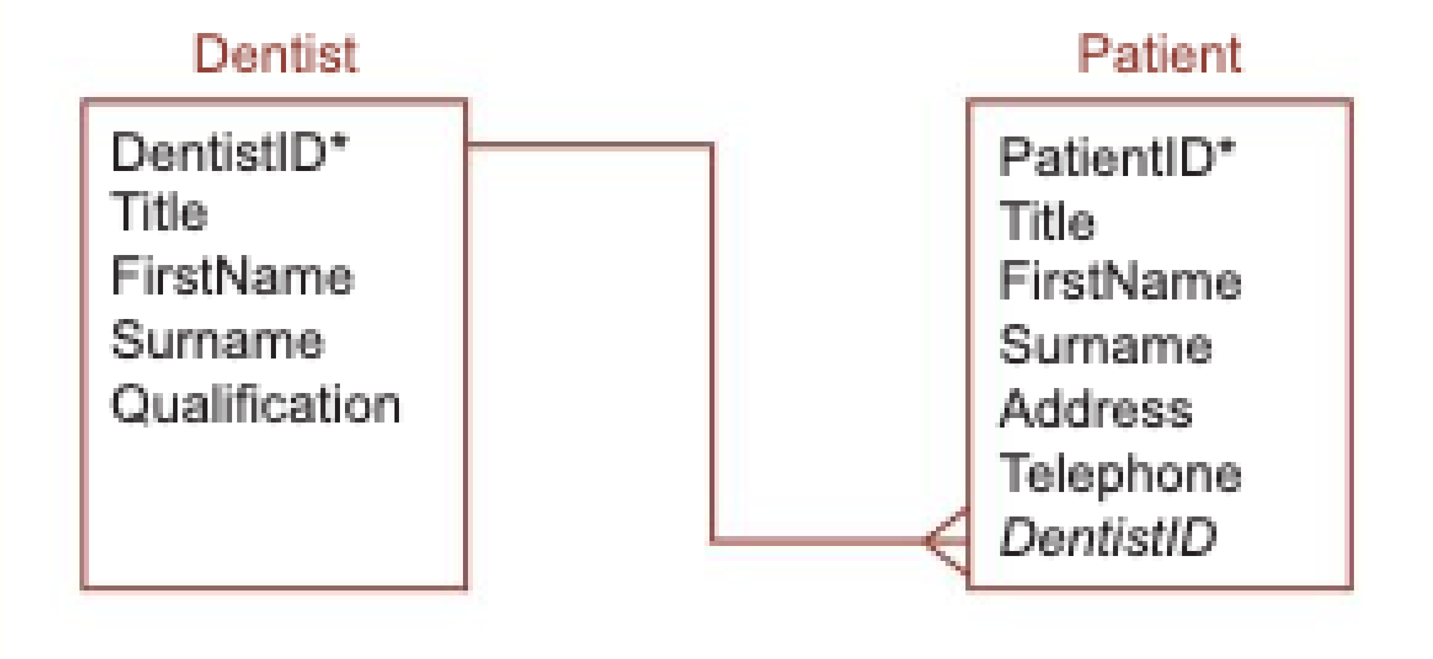
- In a relational database, a separate table is created for each entity identified in the system.

- Where a relationship exists between entities, an extra field called a **foreign key** links the two tables.

Foreign key

A foreign key is an attribute that creates a join between two tables. It is the attribute that is common to both tables, and the primary key in one table is the foreign key in the table to which it is linked.

Example: In the one-to-many relationship between Dentist and Patient, the entity on the 'many' side of the relationship will have **DentistID** as an extra attribute. This is the foreign key.

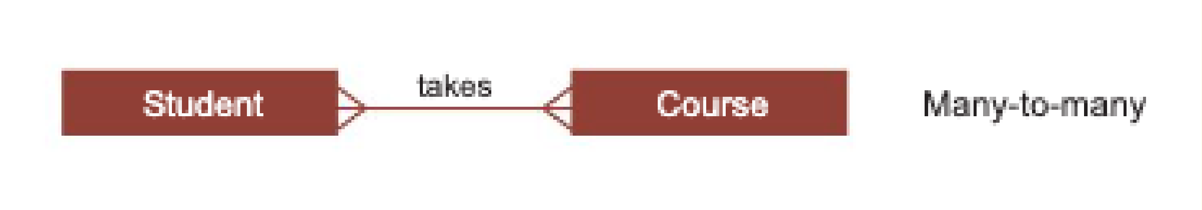


Linking tables in a many-to-many relationship

When there is a many-to-many relationship between two entities, tables **cannot** be directly linked in this way.

For example, consider the relationship between Student and Course.

A student takes many courses, and the same course is taken by many students.



In this case, an extra table is needed to link the Student and Course tables. We could call this StudentCourse, or Enrolment, for example.

The three tables will now have attributes something like those shown below:

Student (StudentID, Name, Address)

Enrolment (StudentID, CourseID)

Course (CourseID, Subject, Level)

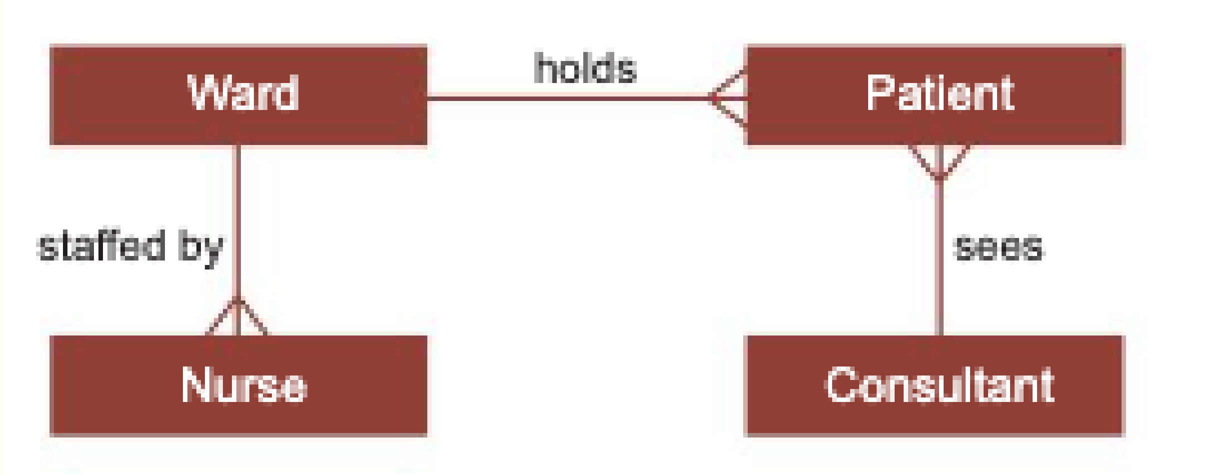
Composite key
In this data model, the table linking **Student** and **Course** has two foreign keys, each linking to one of the two main tables. The two foreign keys also act as the primary key of this table.

A primary key which consists of more than one attribute is called a composite primary key.

Drawing an entity relationship diagram

A database system will frequently involve many different entities linked to each other, and an entity relationship diagram can be drawn to show all the relationships.

A hospital inpatient system may involve entities Ward, Nurse, Patient and Consultant. A ward is staffed by many nurses, but each nurse works on only one ward.
A patient is in a ward and has many nurses looking after them, as well as a consultant, who sees many patients on different wards.



What are the advantages of relational databases?

- **No data redundancy** – in a well-designed relational database there should be no duplicated data (other than the key field).
- **No data inconsistency** – as data is not duplicated, there is no risk of the same data item being stored differently in another record.
- **Flexibility** – A relational database can be queried with greater flexibility than a flat-file system. Relationships mean that data can be combined in a variety of ways to produce the views that different areas of an organisation require.

Explorer task

What is the suitability of flat files and relational databases for use by a family at home and for use in a large mail order company.

Flat files

Limited amount of data

Limited technical expertise available in family

Data format difficult to change

Security not a major issue for family compared with company

Relational database

Large volume of data for company

Requires technical knowledge

Easy to add data and search for data

Easy to link to other applications / e.g. address labels

Saves space / reduces data duplication / redundant data

Improves data consistency / integrity

Easy to change data format

Improves security / easy to control access to data

SUMMARY:

- The use of **foreign keys** enables tables to be **linked** to form relationships. A foreign key is a field in a table that is **also** a **key field** in another table.
- The relationships remove **data redundancy**, i.e. the need for data to be duplicated (as happens in the case in flat-file databases).
- To create a relationship between two tables, we use the **key field** (primary key) from one of the tables and place it as a **foreign key** in the other table.
- A **key field** is a field that is guaranteed to have a **unique** value for each record in the table.
- **Foreign key** relationships are used to ensure the **referential integrity** of data in a database.

Referential integrity

What is Referential Integrity?

- Ensures **consistency** between related tables in a relational database
- Maintains valid **relationships** between **primary** and **foreign** key s
- There should not be foreign keys for which a matching primary key in the linked table does not exist
- **Foreign key constraints**
  - Value in a foreign key field must either:
    - Match a primary key value in the related table, or
    - Be **null** (if allowed)
  - Enforce **referential integrity**

- Maintaining **referential integrity**
  - Use database management systems (DBMS) with **built-in support**
  - Implement triggers to **enforce** custom referential integrity rules
  - Regularly **validate** and clean up data to ensure **consistency**

Benefits and Drawbacks of Referential Integrity

Benefits	Drawbacks
Ensures data consistency and accuracy	Can impact performance due to additional checks
Prevents orphaned records	May require additional planning and design

Learning Aims

Relational databases and normalisation

- Describe the use of secondary keys and indexing
- Normalise relations to third normal form
- Understand why databases are normalised

Key Terms

Relational database is a collection of tables in which relationships are modelled by shared attributes.

Indexing is a method used to store the position of each record ordered by a certain attribute. This is used to look up and access data quickly.

Normalisation is a process used to come up with the best possible design for a relational database.

Relational database design

In a relational database, data is held in tables (also called **relations**) and the tables are linked by means of common attributes.

A **relational database** is a collection of tables in which relationships are modelled by shared attributes.

Conceptually then, one row of a table holds one record. Each column in the table represents one attribute.

A table holding data about an entity Book may have the following rows and columns:

BookID	DeweyCode	Title	Author	DatePublished
88	121.9	Mary Berry Cooks the Perfect	Berry, M	2014
123	345.440	The Paying Guests	Waters, S	2014
300	345.440	Fragile Lies	Elliot, L	2015
657	200.00	Learn French with stories	Bibard, F	2014
777	001.602	GCSE ICT	Barber, A	2010
etc				

To describe the table shown above, you would write:

Book (BookID, DeweyCode, Title, Author, DatePublished)

Indexing

In order that a record with a particular primary key can be quickly located in a database, an **index** of primary keys will be automatically maintained by the database software, giving the position of each record according to its primary key.

One or more secondary indexes may be defined when the database is created, for any attribute that is often used as a search criterion.

For example, table both **Author** and **Title** might be defined as **secondary keys**. This would speed up searches on either of these fields, which would otherwise have to be searched sequentially.

Linking database tables

Tables may be linked through the use of a common attribute.

This attribute must be a primary key of one of the tables, and is known as a **foreign key** in the second table.

There are three possible types of relationship between entities:

- one-to-one,
- one-to-many
- many-to-many

Normalisation

Normalisation is a process used to come up with the best possible design for a relational database.

Tables should be organised in such a way that:

- **no data is unnecessarily duplicated** (i.e. the same data item held in more than one table)
- **data is consistent** throughout the database (e.g. a customer is not recorded as having different addresses in different tables of the database). Consistency should be an automatic consequence of not holding any duplicated data. This means that anomalies will not arise when data is inserted, amended or deleted.
- the structure of each table is flexible enough to allow you to enter as many or as few items (for example, components making up a product) as required
- the structure should enable a user to make all kinds of complex queries relating data from
- different tables

There are three basic stages of normalisation known as **first, second and third normal form.**

Three types of normalisation

First normal form

- All **fields** names must be unique.
- Values in fields should be from the same domain.
- Values in **fields** should be atomic.
- No two **records** may be identical.
- Every **table** must have a **primary key**.

Second normal form

- The data must already be in 1NF.
- Remove any partial dependencies.
- Fix any M:M relationships created as a result.

Third normal form

- The data must already be in **2NF**.
- Remove any **transitive dependencies**.

  In other words, ensure that non-key fields are not dependent on each other.

Normalisation – 0NF (flat file before any normalisation)

A database for storing students' details

```
STUDENT (name, date of birth, gender, course number, course name, lecturer)
```

name	date of birth	gender	course number	course name	lecturer
Tony Gibbons	15/02/1979	M	F451 G403 P202	Computing History of Art Classics	CSA, Craig Sargent AOH, Austin O'Harel CSA, Craig Sargent
Mathew Robinson	14/03/1980	M	G403 Q947 P202	History of Art Textiles Classics	AOH, Austin O'Harel LCO, Linda Cox CSA, Craig Sargent
Claire Matthews	21/05/1974	F	F451 J564 P554	Computing Drama Physics	CSA, Craig Sargent LCO, Linda Cox JHA, James Hayes
Alfred Pillar-Hofman	22/03/1982	M	P202 H544 J390	Classics History English lit	CSA, Craig Sargent SRU, Simon Russel LCO, Linda Cox
James Applegate	01/02/1978	M	Q947 G403 J564	Textiles History of Art Drama	LCO, Linda Cox AOH, Austin O'Harel LCO, Linda Cox
...	...	...	...	...	...

To get to **first normal form (1NF),** a table should follow five rules:

1. All **field** names must be unique.
2. Values in **fields** should be from the same **domain**.
3. Values in **fields** should be atomic.
4. No two **records** can be identical.
5. Each **table** needs a **primary key**.

Normalisation – 1NF

A database for storing students' details

STUDENT (name, date of birth, gender, course number, course name, lecturer initials, lecturer name)

name	date of birth	gender	course number	course name	lecturer initials	lecturer name
Tony Gibbons	15/02/1979	M	F451	Computing	CSA	Craig Sargent
Tony Gibbons	15/02/1979	M	G403	History of Art	AOH	Austin O'Harel
Tony Gibbons	15/02/1979	M	P202	Classics	CSA	Craig Sargent
Mathew Robinson	14/03/1980	M	G403	History of Art	AOH	Austin O'Harel
Mathew Robinson	14/03/1980	M	Q947	Textiles	LCO	Linda Cox
Mathew Robinson	14/03/1980	M	P202	Classics	CSA	Craig Sargent
Claire Matthews	21/05/1974	F	F451	Computing	CSA	Craig Sargent
Claire Matthews	21/05/1974	F	J564	Drama	LCO	Linda Cox
Claire Matthews	21/05/1974	F	P554	Physics	JHA	James Hayes
Alfred Pillar-Hofman	22/03/1982	M	P202	Classics	CSA	Craig Sargent
Alfred Pillar-Hofman	22/03/1982	M	H544	History	SRU	Simon Russel
Alfred Pillar-Hofman	22/03/1982	M	J390	English lit	LCO	Linda Cox
James Applegate	01/02/1978	M	Q947	Textiles	LCO	Linda Cox
James Applegate	01/02/1978	M	G403	History of Art	AOH	Austin O'Harel
James Applegate	01/02/1978	M	J564	Drama	LCO	Linda Cox

To get to **first normal form (1NF),** a table should follow five rules:

1. All **field** names must be unique.
2. Values in **fields** should be from the same **domain**.
3. Values in **fields** should be atomic.
4. No two **records** can be identical.
5. Each **table** needs a **primary key**.

The solution still needs work, but let's go with it for now and see if we can find a better one as we further **normalise** the **database**.

The final rule has been met, so our table is now in **first normal form (1NF)**.

Normalisation – 2NF

A database for storing students' details

STUDENT (<u>student number</u>, name, date of birth, gender)
COURSE (<u>course number</u>, course name, lecturer initials, lecturer name)
STUDENT_TAKES_COURSE(<u>student number</u>, <u>course number</u>)

student number	name	date of birth	gender
001	Tony Gibbons	15/02/1979	M
002	Mathew Robinson	14/03/1980	M
003	Claire Matthews	21/05/1974	F
004	Alfred Pillar-Hofman	22/03/1982	M
005	James Applegate	01/02/1978	M
...	...	...	...

student number	course number
001	F451
001	G403
001	P202
002	G403
002	Q947
002	P202
003	F451
...	...

course number	course name	lecturer initials	lecturer name
F451	Computing	CSA	Craig Sargent
G403	History of Art	AOH	Austin O'Harel
P202	Classics	CSA	Craig Sargent
Q947	Textiles	LCO	Linda Cox
J564	Drama	LCO	Linda Cox
P554	Physics	JHA	James Hayes
H544	History	SRU	Simon Russel
J390	English lit	LCO	Linda Cox

Which courses does Mathew Robinson take?

To get to **second normal form (2NF)**, a table should follow two rules:

1. The data is already in **1NF**.
2. Any **partial dependencies** have been removed.

Once we spot a **many-to-many** relationship, we can fix it by:

- Creating a linking table.
- Assigning the **primary keys** from the two initial **tables** as the **composite key** for the new linking **table**.
- Flipping the **M:M** crows-feet relationship to become two separate **1:M** relationships joined by the new **table.**

3NF

Normalisation – 3NF

A database for storing students' details

```
STUDENT (student number, name, date of birth, gender)
COURSE (course number, course name, lecturer initials, lecturer name)
STUDENT_TAKES_COURSE(student number, course number)
LECTURER (lecturer initials, lecturer name)
```

All our tables are now in **3NF**.

In moving from **1NF** to **3NF**, we have gone from one single table (a flat file) to a four-table **relational database**.

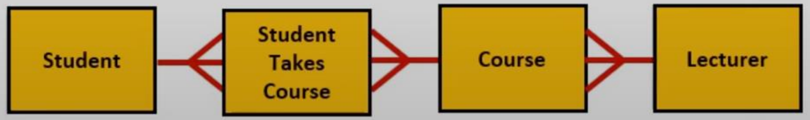
We have removed repeating data, and each table now serves a single purpose.

student number	name	date of birth	gender
001	Tony Gibbons	15/02/1979	M
002	Mathew Robinson	14/03/1980	M
003	Claire Matthews	21/05/1974	F
004	Alfred Pillar-Hofman	22/03/1982	M
005	James Applegate	01/02/1978	M
...	...	...	...

student number	course number
001	F451
001	G403
001	P202
002	G403
002	Q947
002	P202
003	F451
...	...

course number	course name	lecturer initials
F451	Computing	CSA
G403	History of Art	AOH
P202	Classics	CSA
Q947	Textiles	LCO
J564	Drama	LCO
P554	Physics	JHA
H544	History	SRU
J390	English lit	LCO

lecturer initials	lecturer name
CSA	Craig Sargent
AOH	Austin O'Harel
LCO	Linda Cox
JHA	James Hayes
SRU	Simon Russel



Database – SQL

A Level Computer Science: Exchanging Data

Learning Aims

- Be able to use SQL to retrieve data from multiple tables of a relational database
- Be able to interpret and modify SQL
- Be able to use SQL to define a database table
- Be able to use SQL to update, insert and delete data from multiple tables of a relational database

CRUD

All relational databases must have certain basic functionality to be useful. This is often summarised by the acronym CRUD. This stands for:

- Create
- Read
- Update
- Delete.

Each of these functions can be actioned by an equivalent SQL statement:

- INSERT/CREATE
- SELECT
- UPDATE
- DELETE.

SQL

SQL, or Structured Query Language (pronounced either as S-Q-L or Sequel) is a declarative language used for querying and updating tables in a relational database.

It can also be used to create tables.

We will look at SQL statements used in querying a database.

The tables shown will be used to demonstrate some SQL statements.

The tables are part of a database used by a retailer to store details of CDs in a database that will allow information about the CDs to be extracted. The four entities CD, CDSong, Song and Artist are connected by the following relationships:

Figure 18.1

CD Table

CD table

CDNumber	CDTitle	RecordCompany	DatePublished
CD14356	Shadows	ABC	06/05/2014
CD19998	Night Turned Day	GHK	24/03/2015
CD25364	Autumn	ABC	11/10/2015
CD34512	Basic Poetry	GHK	01/02/2016
CD56666	The Lucky Ones	DEF	16/02/2016
CD77233	Lucky Me	ABC	24/05/2014
CD77665	Flying High	DEF	31/07/2015

SELECT .. FROM .. WHERE

The SELECT statement is used to extract a collection of fields from a given table. The basic syntax of this statement is

SELECT	list the fields to be displayed
FROM	list the table or tables the data will come from
WHERE	list the search criteria
ORDER BY	list the fields that the results are to be sorted on (default is Ascending order)

Example 1

```
SELECT      CDTitle, RecordCompany, DatePublished
FROM        CD
WHERE       DatePublished BETWEEN #01/01/2015# AND #31/12/2015#
ORDER BY    CDTitle
```

This will return the following records:

CDTitle	RecordCompany	DatePublished
Autumn	ABC	11/10/2015
Flying High	DEF	31/07/2015
Night Turned Day	GHK	24/03/2015

Conditions

Conditions in SQL are constructed from the following operators:

Symbol	Meaning	Example	Notes
=	Equal to	CDTitle = "Autumn"	Different implementations use single or double quotes
>	Greater than	DatePublished > #01/01/2015#	The date is enclosed in quote marks or, in Access, # symbols.
<	Less than	DatePublished < #01/01/2015#	
!=	Not equal to	RecordCompany != "ABC"	
>=	Greater than or equal to	DatePublished >= #01/01/2015#	
<=	Less than or equal to	DatePublished <= #01/01/2015#	
IN	Equal to a value within a set of values	RecordCompany IN ("ABC", "DEF")	
LIKE	Similar to	CDTitle LIKE "S%"	Finds Shadows (wildcard operator varies and can be *)
BETWEEN… AND	Within a range, including the two values which define the limits	DatePublished BETWEEN #01/01/2015# AND #31/12/2015#	
IS NULL	Field does not contain a value	RecordCompany IS NULL	
AND	Both expressions must be true for the entire expression to be judged true	DatePublished > #01/01/2015# AND RecordCompany = "ABC"	
OR	If either or both of the expressions are true, the entire expression is judged true.	RecordCompany = "ABC" OR RecordCompany = "DEF"	Equivalent to RecordCompany IN ("ABC", "DEF")
NOT	Inverts truth	RecordCompany NOT IN ("ABC", "DEF")	

Specifying a sort order

ORDER BY gives you control over the order in which records appear in the Answer table. If for example you want the records to be displayed in ascending order of RecordCompany and within that, descending order of DatePublished, you would write, for example:

```
SELECT      *
FROM        CD
WHERE       DatePublished < #31/12/2015#
ORDER BY    RecordCompany, DatePublished Desc
```

This would produce the following results:

CDNumber	CDTitle	RecordCompany	DatePublished
CD25364	Autumn	ABC	11/10/2015
CD77233	Lucky Me	ABC	24/05/2014
CD14356	Shadows	ABC	06/05/2014
CD77665	Flying High	DEF	31/07/2015
CD19998	Night Turned Day	GHK	24/03/2015

Extracting data from several tables

So far we have only taken data from one table. The **Song** and **Artist** tables have the following contents:

Song table

SongID	SongTitle	ArtistID	MusicType
S1234	Waterfall	A318	Americana
S1256	Shake it	A123	Heavy Metal
S1258	Come Away	A154	Americana
S1344	Volcano	A134	Art Pop
S1389	Complicated Game	A318	Americana
S1392	Ghost Town	A123	Heavy Metal
S1399	Gentle Waves	A134	Art Pop
S1415	Right Here	A134	Art Pop
S1423	Clouds	A315	Art Pop
S1444	Sheet Steel	A334	Heavy Metal
S1456	Here with you	A154	Art Pop

Artist table

ArtistID	ArtistName
A123	Fred Bates
A134	Maria Okello
A154	Bobby Harris
A315	Jo Morris
A318	JJ
A334	Rapport

Extracting data from several tables

Using SQL you can combine data from two or more tables, by specifying which table the data is held in.

For example, suppose you wanted SongTitle, ArtistName and MusicType for all Art Pop music.

When more than one table is involved, SQL uses the syntax tablename.fieldname. (The table name is optional unless the field name appears in more than one table.)

```
SELECT    Song.SongTitle, Artist.ArtistName, Song.MusicType
FROM      Song, Artist
WHERE     (Song.ArtistID = Artist.ArtistID) AND (Song.MusicType = "Art Pop")
```

The condition Song.ArtistID = Artist.ArtistID provides the link between the Song and Artist tables so that the artist's name corresponding to the ArtistID in the Song table can be found in the Artist table. This will produce the following results:

SongTitle	ArtistName	MusicType
Volcano	Maria Okello	Art Pop
Gentle Waves	Maria Okello	Art Pop
Right Here	Maria Okello	Art Pop
Clouds	Jo Morris	Art Pop
Here with you	Bobby Harris	Art Pop

SQL JOIN

JOIN provides an alternative method of combining rows from two or more tables, based on a common field between them.

The query above could be written as follows:

```
SELECT    Song.SongTitle, Artist.ArtistName, Song.MusicType
FROM      Song
JOIN      Artist
ON        Song.ArtistID = Artist.ArtistID
WHERE     Song.MusicType = "Art Pop"
```

SQL JOIN

The fourth table in the database is the table CDSong which links the songs to one or more of the CDs.

We can make a search to find the CDNumbers and titles all the CDs containing the song Waterfall, sung by JJ.

```
SELECT    Song.SongID, Song.SongTitle, Artist.ArtistName, CDSong.CDNumber, CD.CDTitle
FROM      Song, Artist, CDSong, CD
WHERE     CDSong.CDNumber = CD.CDNumber
          AND CDSong.SongID = Song.SongID
          AND Artist.ArtistID = Song.ArtistID
          AND Song.SongTitle = "Waterfall"
```

This will produce the following results:

SongID	SongTitle	ArtistName	CDNumber	CDTitle
S1234	Waterfall	JJ	CD14356	Shadows
S1234	Waterfall	JJ	CD19998	Night Turned Day
S1234	Waterfall	JJ	CD34512	Basic Poetry

CDSong table

CDNumber	SongID
CD14356	S1234
CD14356	S1258
CD14356	S1415
CD19998	S1234
CD19998	S1389
CD19998	S1423
CD19998	S1456
CD25364	S1256
CD25364	S1392
CD34512	S1392
CD34512	S1234
CD34512	S1389
CD34512	S1444
CD77233	S1256
CD77233	S1344
CD77233	S1399
CD77233	S1456

Note

Note that in the SELECT statement, it does not matter whether you specify Song.SongID or CDSong.SongID since they are connected.

The same is true of CDSong.CDNumber and CD.CDNumber.

The Boolean conditions

CDSong.SongID = Song.SongID and Artist.ArtistID = Song.ArtistID are required to specify the relationships between the data tables.

Model Answer

1 A vehicle rental company holds their data in a database. The details of their vehicles are held in a table named `Vehicle`. An extract of the table is shown below.

VehicleID	VehicleType	FuelType	Seats
V6786	SUV	Diesel	7
C9879	Hatchback	Electric	4
C6689	Hatchback	Petrol	4
V6624	Saloon	Petrol	5
C5638	Hatchback	Electric	4
V3872	Saloon	Electric	5

(a) With reference to the data shown, describe what the following SQL statement will do.

```
SELECT VehicleID FROM Vehicle WHERE
VehicleType = 'Hatchback'
```

The SQL statement will output the VehicleID from the table Vehicle for the vehicles that match the criteria of VehicleType being Hatchback. For the data shown this would return C9879, C6689 and C5638. **[3]**

(b) Write an SQL statement that will show all vehicles that have five or more seats and are diesel or petrol.

*SELECT * FROM Vehicle WHERE Seats >= 5 AND (FuelType = 'Diesel' OR FuelType = 'Petrol')* **[3]**

Do you remember?

SQL (Structured Query Language) is a language used to create, query and update tables in relational databases.

What does the following SQL statement mean?

```
SELECT * FROM User WHERE group=7
```

`SELECT` *is used to select columns from the database. In this case, the * is a wildcard which means select all the fields.*

`FROM` *is used to declare the table(s) where the data is located.*

`WHERE` *is used to set criteria the data needs to meet. In this case any records where the group field is 7.*

Defining a database table

The following example shows how to create a new database table.

Use SQL to create a table named Employee, which has four columns:

EmpID (a compulsory int field which is the primary key), EmpName (a compulsory character field of length 10), HireDate (an optional date field) and Salary (an optional real number field).

```
CREATE TABLE Employee
(
EmpID       INTEGER NOT NULL, PRIMARY KEY,
EmpName     VARCHAR(20) NOT NULL,
HireDate    DATE,
Salary      CURRENCY
)
```

Data types

Some of the most commonly used data types are described in the table below. (The data types vary depending on the specific implementation.)

Data type	Description	Example
CHAR(n)	Character string of fixed length n	ProductCode CHAR(6)
VARCHAR(n)	Character string variable length, max. n	Surname VARCHAR(25)
BOOLEAN	TRUE or FALSE	ReviewComplete BOOLEAN
INTEGER, INT	Integer	Quantity INTEGER
FLOAT	Number with a floating decimal point	Length FLOAT (10,2) (maximum number of digits is 10 and maximum number after decimal point is 2)
DATE	Stores Day, Month, Year values	HireDate DATE
TIME	Stores Hour, Minute, Second values	RaceTime TIME
CURRENCY	Formats numbers in the currency used in your region	EntryFee £23.50

Altering a table structure

The ALTER TABLE statement is used to add, delete or modify columns (i.e. fields) in an existing table:

To add a column (field):

```
ALTER TABLE Employee
ADD Department VARCHAR(10)
```

To delete a column:

```
ALTER TABLE Employee
DROP COLUMN HireDate
```

To change the data type of a column:

```
ALTER TABLE Employee
MODIFY COLUMN EmpName VARCHAR(30)NOT NULL
```

Defining linked tables

If you set up several tables, you can link tables by creating foreign keys.

Suppose that an extra table is to be added to the Employee database which lists the training courses offered by the company.

A third table shows which date an employee attended a particular course.

The structure of the Employee table is:

EmpID	Integer (Primary key)
Name	30 characters maximum
HireDate	Date
Salary	Currency
Department	30 characters maximum

Course Table and CourseAttenance

The structure of the Course table is:

CourseID 6 characters, fixed length (Primary key)

CourseTitle 30 characters maximum (must be entered)

OnSite Boolean

The structure of the CourseAttendance table is:

CourseID 6 characters, fixed length (foreign key)

EmpID Integer (foreign key) Course ID and EmpID form a composite primary key

CourseDate Date (note that the same course may be run several times on different dates)

The CourseAttendance table is created using the SQL statements:

```
CREATE TABLE CourseAttendance
(
CourseID     CHARACTER(6)NOT NULL,
EmpIDI              NTEGER NOT NULL,
CourseDate          DATE,
FOREIGN KEY         CourseID REFERENCES Course(CourseID),
FOREIGN KEY         EmpID REFERENCES Employee(EmpID)
PRIMARY KEY         (CourseID, EmpID)
)
```

Inserting, updating, and deleting data using SQL

The SQL INSERT INTO statement
This statement is used to insert a new record in a database table. The syntax is:

```
INSERT INTO tableName (column1, column2, ...)
VALUES (value1, value2, ...)
```

Example: add a record for employee number 1122, Bloggs, who was hired on 1/1/2001 for the technical department at a salary of £18000.

```
INSERT INTO Employee (EmpID, Name, HireDate, Salary, Department)
VALUES ("1122", "Bloggs", #1/1/2001#, 18000, "Technical")
```

Note that if all the fields are being added in the correct order you would not need the field names in the brackets above to be specified. INSERT INTO Employee would be sufficient

Example: add a record for employee number 1125, Cully, who was hired on 1/1/2001. Salary and Department are not known.

```
INSERT INTO Employee (EmpID, Name, HireDate)
VALUES ("1125", "Cully", #1/1/2001#)
```

The SQL UPDATE statement

This statement is used to update a record in a database table. The syntax is:

```
UPDATE tableName
SET column1 = value1, column2 = value2, …
WHERE columnX = value
```

Example: increase all salaries of members of the Technical department by 10%

```
UPDATE Employee
SET Salary = Salary*1.1
WHERE Department = "Technical"
```

Example: Update the record for Bloggs, who has moved to Administration.

```
UPDATE Employee
SET Department = "Administration"
WHERE EmpID = "1122"
```

The SQL DELETE statement

This statement is used to delete a record from a database table.

The syntax is:

DELETE FROM tableName
WHERE columnX = value

Example: Delete the record for Bloggs.

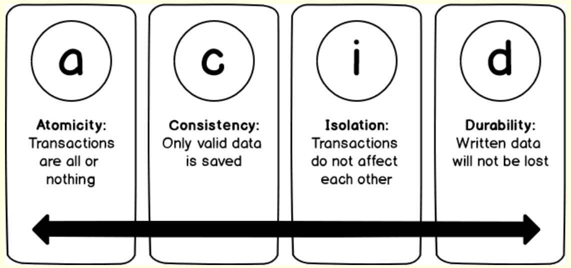
DELETE FROM Employee
WHERE EmpID = "1122"

How are transactions entered?

- **Forms** – Used to collect structured data from users, often in digital or paper format. Can include text fields, checkboxes, and dropdowns.
- **OMR (Optical Mark Recognition)** – Scans and detects marks on paper (e.g., multiple-choice exam sheets) to quickly process large amounts of data.
- **OCR (Optical Character Recognition)** – Converts printed or handwritten text into digital format, allowing for text processing and editing.
- **Sensors** – Devices that collect data from the environment (e.g., temperature, motion, light) and convert it into a digital format for processing.
- **Barcodes** – Encoded data stored in a series of lines or squares, scanned by barcode readers for quick identification (e.g., product labels in stores).
- **Data Mining** – The process of analysing large datasets to identify patterns, trends, and useful insights for decision-making.

What is a transaction?
A **transaction** in databases is a sequence of one or more operations (such as inserting, updating, or deleting data) that are executed as a single unit.

A transaction must follow the **ACID** properties to ensure data integrity.



Atomic A transaction should either succeed or fail but never partially succeed.

Consistent: The transaction should only change the database according to the rules of the database.

Isolation Each transaction shouldn't affect/overwrite other transactions concurrently being processed. (**Concurrency**: When multiple transactions are executed simultaneously, they might try to access or modify the same data, leading to inconsistencies)

Record locking can be used to ensure that the ACID principle of isolation is achieved when carrying out multiple transactions. Record locking allows one user/process to access/modify record level data at any one time

Durability Transaction is not lost in case of power / system failure. Completed transactions stored in secondary storage.

Transaction Processing

Data entered must be accurate in the first place.

Data entered should be validated (automatically checked it is sensible) and verified (checked that the data entered matches the original).

Security measures need to be in place to prevent malicious tampering of data.

Record Locking
Records should be locked when in use. If one transaction is amending a record, no other transaction should be able to until the first transaction is complete.

Transactions should maintain **referential integrity.**

Ensuring that changes are consistent across a database. if a record is removed all references to it are removed A foreign key value must have a corresponding Primary key value in another table.

Changes to data in one table must take into account data in linked tables.