Data capture

Before you can create a database, you need to collect data. This is known as data capture. One of the most common ways is to use and **optical mark reader (OMR**).

An **OMR sensor** is used for marking multiple choice tests and for collecting the results of surveys or questionnaires. The user fills out a form, usually by shading in the box they wish to select. The OMR shines a light at the paper and as the shaded mark will reflect less light than empty boxes, signals are sent back to the computer system that identifies which box has been shaded. An algorithm interprets this so can work out which box is selected and the information stored in a database.

An alternative to an OMR is an **OCR (optical character recognition).** Optical character readers are used to **convert paper-based hardcopies of text into digital**, **editable copies of the same document**. Alternatively, they are used so a computer can identify and interpret handwritten text.

They work by **scanning a hardcopy document and storing a picture**. The **optical character recognition software** looks at **each individual character** and **compares it to a database of characters** to look up the possible value. Some algorithms work by **pattern recognition** and **feature detection** of characters. Once a **match is found**, the **equivalent ASCII value is placed in a digital file.**

A Flat file database

A flat file database consists of just one file/table.

It is suitable for storing **smaller sets of data** such as all the names and addresses of people at a sports club or all the information about a personal DVD collection.
The main problem with these databases is that the **larger they grow the more data becomes redundant**. Data redundancy simply means having lots of **replicated unnecessary** data.

Most databases, however are about **more than just one entity**. There are **multiple relationships** between the entities. For example, in a database about DVDs, you may wish to keep data on which of the main actors starred in each film. The **actor** would be a **second entity alone**.

e.g. A Dentist's surgery with multiple dentists may have a system that allows patients to book appointments. Entities' in the system are: Dentist, Patient, Appointment.

- The attributes of Dentist are: Title, First name, Surname, Qualification, Age, Surgery name.
- The attributes of Patient are: Title, First name, Surname, Date of birth, Address, Telephone, Medical conditions, Teeth on watch, Dentist history.
- The attributes of Appointment are: Dentist, Patient, Date, Cost, Time, Specific requirement

Problems with a flat file database (data redundancy)

- If an item of data becomes outdated or needs to change then data redundancy may mean that the data has to be changed in numerous locations. This is not only time consuming but also means that it is easy to miss a record in the database. This leads to a loss of data integrity (data not reflecting the real world). This, in turn, could lead to legal problems with the Data Protection Act.
-

  Moreover, if we wanted to store extra information (another attribute) and there is data redundancy, the same data will have to be entered in numerous times. This increases the probability of entering invalid/inaccurate data by mistake.

Databases

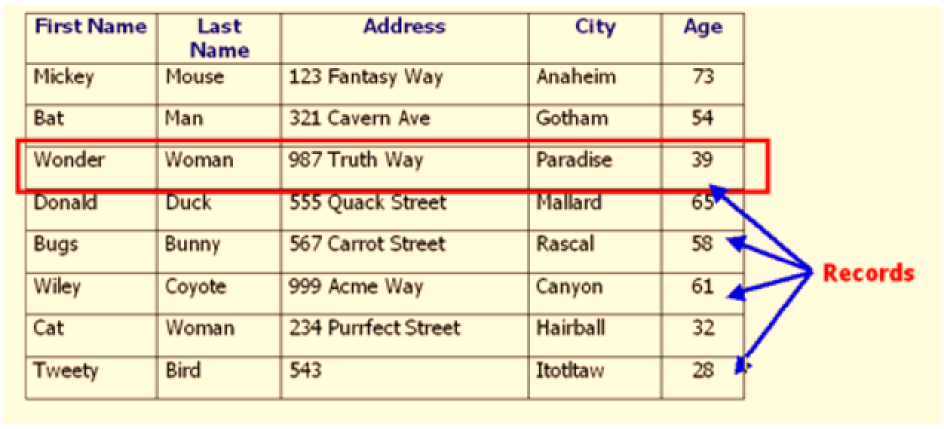
- **Database**: An organised collection of data held in a computer system, especially one that is accessible in various ways.
- **A relational database** = a collection of tables in which relationships are modelled by shared attributes.
- **A relation =** A set of attributes and records, modelling an entity (a table).

Data is stored in tables (called relations) and modelled by shared attributes.

Databases store data in tables like the one below……

- **A table** = a data structure that stores all the records for a particular category or person.
- **A field** = One specific piece of information about an entity. (i.e. the title of a column).
- **Attribute** = The properties, facts, details of characteristics which represent that entity (named column in table)
- **A record**: all of the data or information about one person or thing. Records stored as rows (horizontal).
- Small databases have very few records compared to a large company which has thousands to millions of records.

e.g.

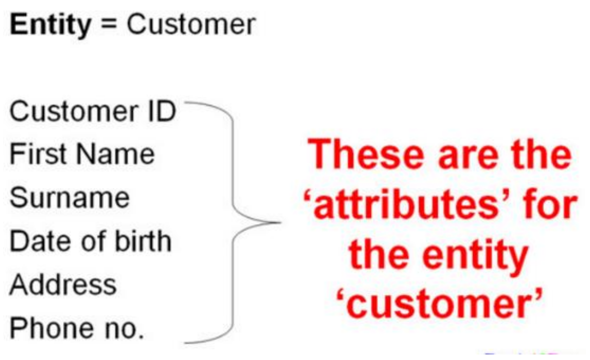


First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

Data is stored in the tables in records (rows), that can be divided into fields (columns). Each table must have its own primary key and name.

Relational databases arrange data as sets of tables with linked or shred attributes through primary and foreign keys.
The tables contain records (horizontal rows).
Each record consists of several fields; the fields of all records form the columns. Examples of fields: name, gender, hair colour.

e.g.



Entity = Customer

Customer ID
First Name
Surname
Date of birth
Address
Phone no.

These are the 'attributes' for the entity 'customer'

Conceptually, one row of a table holds one record. Each column in the table holds one attribute (with many records of data about it).

Database concepts

When a systems designer begins to work on a new proposed computer system, one of the first things they need to do is to examine the data that needs to be input, processed and stored and determine what the **data entities are.**

Entity = A category of object, person, event or thing of interest to an organisation about which data (attributes) are to be recorded.

Examples of such entities are Film, Actor, Product, Recipe, Ingredient and each entity has attributes.

Entity Descriptions

An entity description is usually written with the format:

Entity1 (Attribute1, Attribute2, ……..)

e.g. entity description for Appointment would be……..

Appointment: Appointment(<u>AppointmentID</u>, Dentist, Patient, Date, Cost, Time, Specific requirement)

<u>AppointmentID</u> = Primary Key

Entity identifier and primary key

Each entity requires an **entity identifier** to **uniquely identify the entity**. In a relational database, this is called a **primary key**. So far, none of the attributes of the entities identified so far (Dentist/Patient/Appointment) would be suitable. A **unique ID** such as D13649 should be used.

- We use a **numeric code or string**
- In the entity description, the **primary key is <u>underlined.</u>**

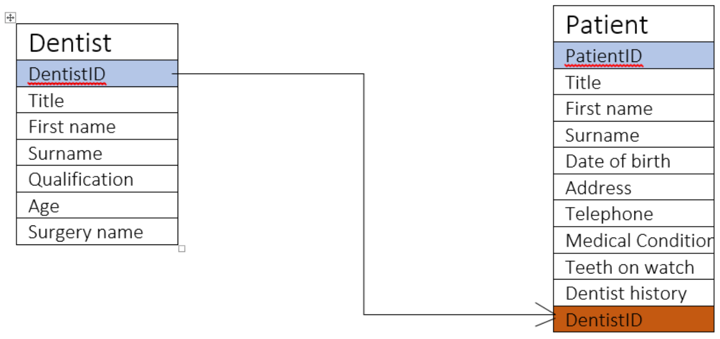
Relational database keys

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 (column) called foreign key links** the two tables.

Relationship = when tables within a database are **related/linked** so data can be **accessed between them**. One table has a **foreign key** that **references the primary key** of the other table.

A primary key is composed of one or more attributes that uniquely identify a particular record in the table (when describing an entity, this is called an entity identifier).

A foreign key is an attribute that creates a link between two tables. It is the attribute that is common to both tables and is used as a primary key in the other table it uniquely identifies.



Note that the primary key of Dentist (DentistID) is used as a foreign key in Patient table (DentistID).

A secondary Key (indexing) *(a secondary key may not be unique to record)*

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

One or more **secondary indexes** may be needed when the database is created, **this is the case for any attribute that is often used as search criteria**. E.g. Author and Title in a database that uses BookIDs as the primary key may be used as **secondary keys.** This would **speed up searches** on either of these fields, which would otherwise be searched sequentially.

e.g. A patient is unlikely to know their patientID so a **secondary index** (e.g. **surname**) is likely.

Indexing = An index is a **data structure** used to shorten the length of time it takes to search a database. It allows the user to gain access to specific records that are indexed in a file.

For example, the index might contain the 'surname' column of a database or even the column of primary keys. This would mean that when you are searching for a student, if you know their surname or primary key (StudentID) , you can find the information you want much faster than if you were to sequentially sort through all the items. This is similar to a contents page. Sometimes you may not know the primary key but you do know the surnames of the students so you can use the index table (or indexing software) to quickly jump to the correct record.

Linking database tables:

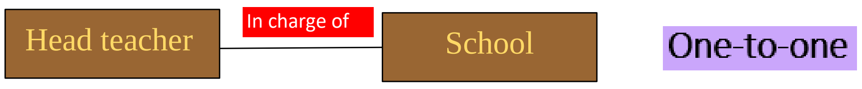
Tables are linked through the use of **common attributes**. The attribute must be the primary key of one of the tables, and is known as a foreign key in the secondary table.

Relationships between entities

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

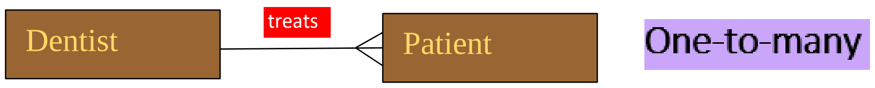
- **One-to-one (1:1):** When one table (entity) is related to another using a common attribute as a primary and foreign key.

Examples include the relationship between a Husband and Wife or Country and Prime Minister .



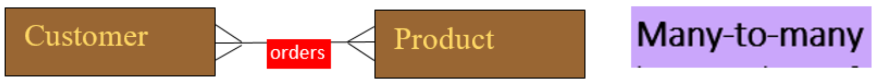
- **One-to-many (1:n):** When one table (entity) is related to multiple other tables (entities). This means that the (an attribute) primary key of the main table is used as a foreign key in multiple other tables (common attribute to many).

Examples of relationships include Mother and Child, Customer and Order, Borrow and Library Book.



- **Many-to-many (n:n):** When multiple tables (entities) are linked to many other tables (entities). In this case an extra table is required to store foreign keys of the tables. In this table there is a composite primary key containing multiple foreign keys used to link other tables in a many-to-many relationship.

Examples of relationships include Student and Course, Stock Item and Supplier, Film and Actor.



❖ These are known as entity relationship diagrams and always require the degree and name of relationship to be specified. They are a diagrammatic way of representing the relationship between entities in a database.

Linking tables in a many-to-many relationship

In this data model, we can't simply link two tables in a many to many fashions.
An extra table is used linking say a Student and Course table. We could call this StudentCourse or Enrolment.

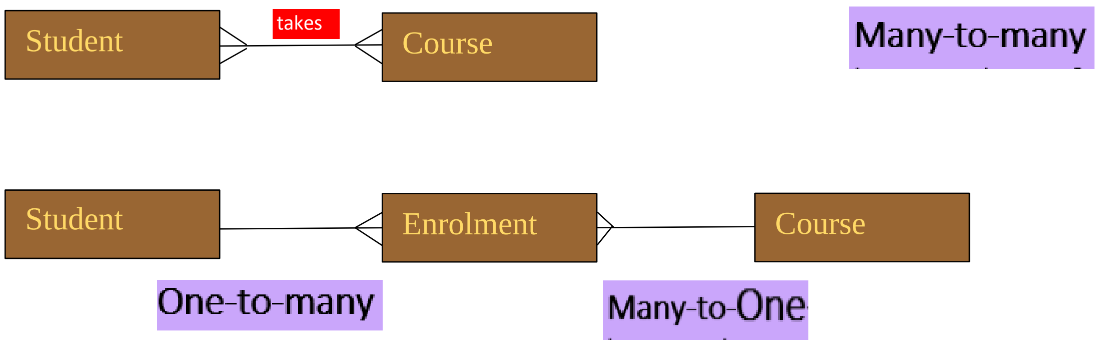
The primary key of the Student table and of the Course table become foreign keys in the third enrollment table. They form what is known as a composite primary key.

Composite primary key = a primary key consisting of more than one attribute.

For the relationship between Student and Course: A student takes many courses that are also taken by many other students.
In this case an extra table (e.g. Enrolment) is needed to link the two tables.

The three tables could be described as:

- Student (<u>StudentID</u>, Name, Address)
- Enrolment (<u>StudentID</u>, <u>CourseID</u>) *The composite key consists of two attributes StudentID, CourseID*
- Course (<u>CourseID</u>, Subject, Level) *NOTE: Entity name is held outside of these brackets*

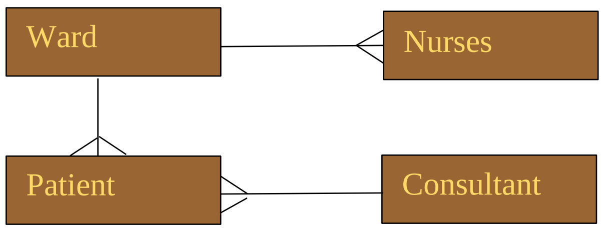


Drawing an entity relationship diagram

Databases will regularly have many entities linked together so an entity relationship diagram is used to show all relationships.

e.g. A hospital inpatient system may involve the entities Ward, Nurse, Patient, Consultant.

A ward is staffed by many nurses but each nurses only staffs only one ward. A patient in a ward has many nurses looking after them and only one consultant that looks after them. The Consultant sees many patients on many wards.



Referential integrity

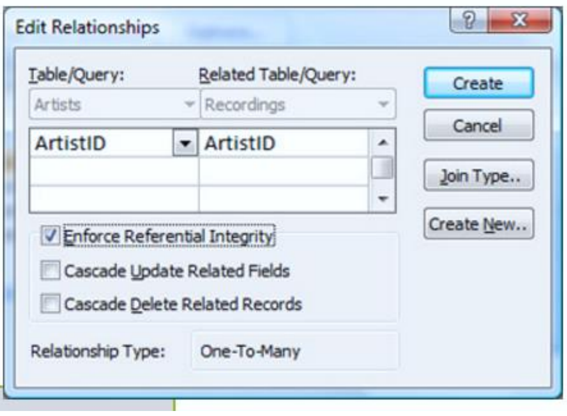
Imagine a database with entities Product and Component
It is important that when the tables are linked, a particular component is not deleted if it is used in a product in the Product table. This is known as referential integrity.

Every foreign key value has a matching value in the corresponding primary key.

Referential integrity uses these to ensure that there are no orphan records i.e. it prevents you from deleting related records.

Referential integrity can also alert you if you try to delete a record which is related to another one.

It can also be used to cascade changes made to the database.


Edit Relationships

Table/Query:	Related Table/Query:
Artists	Recordings
ArtistID	ArtistID

[✓] Enforce Referential Integrity
[ ] Cascade Update Related Fields
[ ] Cascade Delete Related Records

Relationship Type: One-To-Many

Examiner tips
The 'Many' side is usually the foreign key

The 'One' side is usually the primary key

Before you design or set up a database, you should work out:
- the entities
- the attributes
- the entity relationships

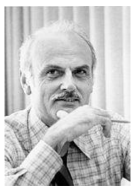
This process is called 'data modelling'

Normalisation

#Normalisation = a process used to devise the best possible design for a relational database.
#Normalised entities = a set of entities that contain no redundant data
#Referential integrity = If a value appears as a foreign key in one table then it must be a primary key of another table.

Tables should ideally be organised in such a way that: LEARN THESE 4 POINTS

1. **No data is unnecessarily duplicated** (i.e. same data held in more than one table)
2. **Data is consistent throughout** the database (e.g. customer is not recorded as having different addresses in different tables of the database). Consistency should be an automatic consequence of not duplicating data. This means that **no anomalies** will arise when data is **inserted, amended or deleted**.
3. The **structure of table is flexible** enough to allow you to enter **as many or as few items for an attribute** in a record as **required**.
4. The structure should **enable** a user to make all kinds of complex **queries relating to data from different tables.**



The relational database was invented by Dr Edgar F. Codd in 1970.

The main disadvantage with relational databases is that they are more complicated to create and maintain

Normalisation of a database

A serial database: Serial means that the data is in an ordered structure

Sequential means the database is ordered by a particular field. E.g. ordering a database of people by surname (alphabetical)

First normal form

Rule 1: Eliminate duplicate columns from the same table and each field/column has one piece of data.
Rule 2: Create separate tables for each group of related data (all attributes dependant on primary key)
Rule 3: Identify a column or combination that will uniquely identify each of the records in a table. i.e. Define and create primary keys in each table.

Second normal form

Rule 1: Check the data is in first normal form.
Rule 2: Remove any data sets that occur in multiple rows and transfer them to new tables.
Rule 3: Only at second normal form will the relationships be created between the separate tables (old and new) by means of a foreign key.

Third normal form

Rule 1: Check the data is in second normal form.
Rule 2: Remove any columns that are not dependent on the primary key.

To remove a many-to-many relationship it is necessary to insert another entity (sometimes called a linking entity) that breaks the relationship into two one-to-many relationships.

SQL language

You need to be able to retrieve the data from a database. This is done through querying the database.

A query = A search on the database allowing specified data to be extracted. Results are returned in the form of a new table or a report.

In the early computing days, there were **many different databases** that each had a **different way to query it**. This meant it was **difficult to transfer skills** from one database to another. There was **no standardisation** so developers were **locked into specific database implementations**. SQL became the standard for querying a database.

SQL = structured query language.
A query written in SQL is sent to a database query engine as plain text.

How do we select data

We use the command "SELECT ……….." to retrieve either a record, column (field) or an entire table.
e.g. let Students be a table in a database

Student ID	Student name	DOB	Gender	Postcode
ST6	James	30/01/2000	M	OX68TY
ST7	Charlotte	12/05/2000	F	CA80GH
ST8	Julie	03/10/2000	F	WR168UY

How do we narrow our selections

We can use the "**WHERE** …" command which will always come after the **SELECT** command.

How do we sort our selections

We can use the "ORDER BY" command to order the output. This will come after the FROM or WHERE commands.

We write: **ORDER BY** field (to determine order) **DESC**

If we leave out DESC then it will **automatically order ascending**. We can just write **ORDER BY** field

SQL	Description	Result
SELECT StudentID **FROM** Students	This will retrieve the entire student ID column	ST6, ST7, ST8
SELECT * FROM Students	* Means retrieve all items	Students table
SELECT * FROM Students **WHERE** StudentID = "ST7"	This will retrieve all the items of the record ST7. It is defined by the primary key ST7	ST7, Charlotte, 12/05/2000, F, CA80GH
SELECT * FROM Students **WHERE** Gender = "F"	This returns all records where the gender is female	ST7 and its fields ST8 and its fields
SELECT Studentname, DOB **FROM** Students **WHERE** Gender = "F"	This returns the student name and date of birth for all the female students.	Charlotte, 12/05/2000 Julie, 03/10/2000
SELECT Studentname **FROM** Students **WHERE** gender = "F" **ORDER BY** DOB **DESC**	This selects the students names who are female and it will display them in order of their date of birth descending	Julie Charlotte

It is also common practice to start a new line when using a new operator. E.g. **SELECT** *

FROM Students

How to create a table

We use the command:

```
CREATE TABLE table_name
(
 PrimaryKey    VARAIBLE,
 field1        VARAIBLE,
 field2        VARAIBLE,
 field3        VARAIBLE
 )
```

How to remove from a table

We use the command:

DROP table_name

This will delete the table and all of its contents

What variable type to use for each field?

Data type	Description	Example
CHAR(n)	Character string of fixed length n	ProductCode **CHAR(n)**
VARCHAR(n)	Character string of variable length, max, n	ProductName **VARCHAR(n)**
BOOLEAN	TRUE or FALSE	ReviewComplete **BOOLEAN**
INTEGER, INT	Integer values	Quality **INTEGER**
FLOAT (n,d)	Real numbers (floating point decimal) with max of n digits and d decimal places	Length FLOAT (10,2)
DATE	Dates in format dd/mm/yyyy	HireDate **DATE**
TIME	Stores hour, minute, second values	RaceTime **TIME**
CURRENCY	Formats the numbers in the currency used in your region (£pp.dd)	EntryFee **CURRENCY**

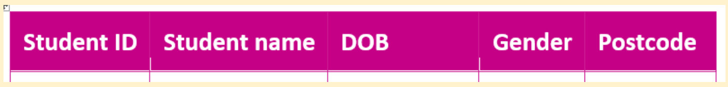
How to insert items into a table just created or an existing one

SQL also lets you add records using the **INSERT INTO…. VALUES….** command.

```
INSERT INTO table_name
(PrimaryKey, field1 ,field2, field3)
VALUES
("id1", "V1", "V2", "V3", PRIMARY KEY(id1))
```

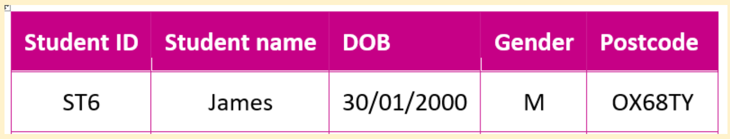
Be aware that when inserting a record you must include a value for the primary key (in this case StudentID). If you try to insert a new record with the same primary key as an existing record, the database will reject the query.

SQL	Description
CREATE Students(Student ID **CHAR(3),** Student name **VARCHAR(12**), DOB **DATE**, Gender **CHAR(1)**, Postcode **CHAR (7), PRIMARY KEY(Student ID)**)	This will create a new table called students. The primary key is student id and is a string of fixed length (3 characters). Likewise, Student name is a field that is a variable string of maximum length 12 characters. DOB is a field of dates. Gender is a field of strings of fixed length 1 character. Postcode is a field of strings of fixed length 7 characters. We have to define primary key at the end. We can also define foreign keys here. Note these will have already been mentioned in this list.



Student ID	Student name	DOB	Gender	Postcode

SQL	Description
DROP Students	Removes the student table and all items
INSERT INTO Students(Student ID, Student name, DOB, Gender, Postcode) **VALUES** ("ST6", "James", #30/01/2000#, "M", "OX68TY")	We use "" for CHAR (strings) and #...# for dates and just simply write integers as integers. This will insert into the students table a singular record that covers items in all fields.

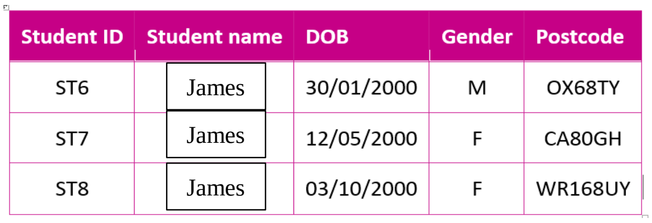


Student ID	Student name	DOB	Gender	Postcode
ST6	James	30/01/2000	M	OX68TY

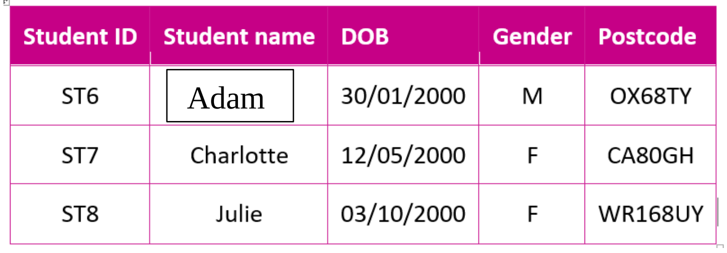
How to update items in a table

We use the **UPDATE….SET** … = .. Command to edit existing data in the database.

SQL	Description	Result
UPDATE Students **SET** StudentName **=** "James"	This will edit all of the records under the field student name to have the same name "James" in the students table.	All student names = James
UPDATE Students **SET** StudentName = "Adam" **WHERE** Student ID **=** "ST6"	This will change the record defined by primary key ST6 to have the student name "Adam" (opposed to the previous "James"	ST6, Adam, 30/01/2000 , M, OX68TY



Student ID	Student name	DOB	Gender	Postcode
ST6	James	30/01/2000	M	OX68TY
ST7	James	12/05/2000	F	CA80GH
ST8	James	03/10/2000	F	WR168UY



Student ID	Student name	DOB	Gender	Postcode
ST6	Adam	30/01/2000	M	OX68TY
ST7	Charlotte	12/05/2000	F	CA80GH
ST8	Julie	03/10/2000	F	WR168UY

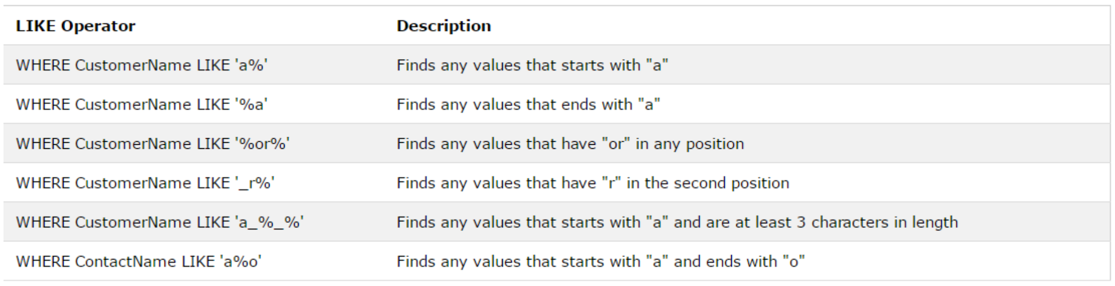
How to delete items from a table

As well as adding and editing data, SQL also enables you to delete data from a database using the **DELETE** command.
This uses similar syntax to the SELECT command

SQL	Description	Result
DELETE FROM Students **WHERE** StudentName = "James"	This will delta every record with student name James	All student names = James
DELETE * FROM Students	Deletes all items in the table but does not remove it	Empty table
DELETE DOB **FROM** Students	Deletes the field DOB from the table	Same table but without the field DOB

We can also increase our specified location of the WHERE operator.

If we want to search for something that we know has a pattern e.g. all the names beginning with the letter a.
SELECT StudentName
FROM Students
WHERE StudentName **LIKE** "a%"



LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

You can use all the common logical operators

You are familiar with in SQL statements, including Boolean operations such as AND, OR, NOT.

```
SELECT    StudentID, StudentName, Gender
FROM      Students
WHERE     DOB BETWEEN #01/04/1999# AND #31/07/2002#
```

In this case it would return the StudentID, StudentName, Gender of all of our students in the table as they all have this range of DOB.

DBSM = Database management system is a software application which handles a database at fundamental levels, allowing the database to interact with the user and other applications to:

1. **Additional security to the database**
2. **Integrity to ensure efficiency and structure is not compromised**
3. **A manipulation language to access and change data (SQL)**
4. **An interface for other programs to access and use the data with program/data independence. I.e. there is no concern for the underlying structure.**

They are available for small systems on PC's and for huge systems for large organizations. Microsoft access is a DBMS and ORACLE is another DBMS.