

Higher Computing Science

Database structures

Summary notes

A database is a collection of data stored in a structured, organised manner.

Flat file databases

A **flat file** database has all data is contained in one **file** (aka table or entity).

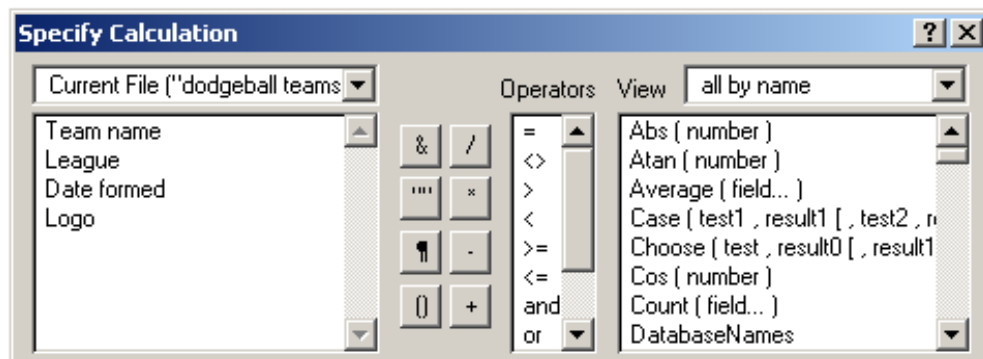
Data is organised into **fields** (aka columns or attributes) and **records** (aka rows or occurrences).

Field types

Basic field types include: text, number, date, time.

Other field types include:

- Object (aka container) – can contain graphic, audio, video, etc.
- Link – contains a hyperlink.
- Calculation – contains a formula to calculate its contents from other data in the record. Automatic calculation of data reduces the chance of human error.



- Boolean – stores Yes/No or True/False.
- Summary – calculates data based on data from multiple records in database. For example, totals up the contents of a particular field across all records.

Relational databases (linked tables)

Linked tables (aka relational databases) have two or more files linked together. Using linked tables reduces unnecessary duplication of data, therefore reducing the opportunity for error.

Good database design **avoids data duplication** (by using linked tables where appropriate) and **reduces errors in data entry** (by using suitable validation techniques).

Relationships

There are 3 types of relationships between tables in a relational database.

One-to-one (1:1)

One record in the first table can only link to exactly one record in the second table. One-to-one relationships are rarely used, except as a means of splitting up a large, unwieldy table or treating part of a table differently (e.g. keeping some fields secure).

Example

In a database with a *Customer Details* table linked to an *Account Details* table – one customer can only have one account, and vice versa. However, the *Customer Details* table can be made available to all staff whilst the *Account Details* table could only be available to senior staff.

One-to-many (1:M)

One record in the first table can link to two or more records in the second table. Each record in the second table can only be linked to exactly one record in the first table.

This is the most common type of relationship.

Example

In a database with a *Customers* table linked to a *Tickets* table - one customer can have many tickets, but each ticket may only have one customer.

Many-to-many (M:M)

One record in the first table can link to two or more records in the second table. One record in the second table can also link to two or more records in the first table. Many-to-many relationships are usually undesirable in relational database design.

Example

In a database with a *Pupils* table linked to a *Teachers* table – one pupil can have many teachers, and vice versa.

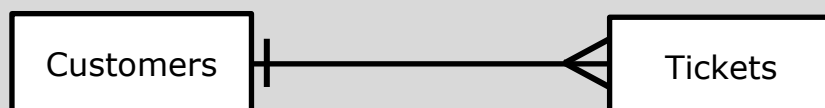
ER diagrams

Relationships can be represented graphically using an Entity-Relationship (ER) diagram.

There are many different notations for ER diagrams, we will use crow's foot notation.

- The names of entities are written in boxes joined by straight lines.
- At the "many" end the line forks.
- At the "one" end the line has a single line through it.

Example



This ER diagram shows a one-to-many relationship between *Customers* and *Tickets*.

Primary keys / foreign keys / compound keys

A **primary key** is a field which is a unique identifier for each record.

A file may have only one primary key.

At the design stage, primary keys are usually identified by underlining.

A **foreign key** is the primary key copied from another file, used to link two files.

In order to create a link, the primary key from the "one" entity gets copied into the "many" entity as a foreign key.

At the design stage, foreign keys are usually identified using an asterisk.

A **compound key** is a key which consists of two or more fields in order to create a unique identifier. This is required when no single field can be used to uniquely identify a record.

Example

Litters (Litter ID
Sire
Dame
Number in litter
DOB)

Puppies (Puppy ID
Puppy name
Sex
Cost of puppy
Litter ID*)

Customers (Customer name
Address
Puppy ID*)

There are 3 tables in this database.

- The *Litters* table has *Litter ID* as a primary key.
- The *Puppies* table has *Puppy ID* as a primary key.
- The *Customers* table uses *Customer name* and *Address* as a compound key.
- *Litter ID* is a foreign key in the *Puppies* table.
This creates a one-to-many relationship between *Litters* and *Puppies*.
- *Puppy ID* is a foreign key in the *Customers* table.
This creates a one-to-many relationship between *Puppies* and *Customers*.

A **surrogate key** can be utilised when there is no natural occurring primary key. In other words, there is no data that can be used as a unique identifier. As a result a new field is created to serve as the unique identifier, removing the requirement for a compound key.

This can be implemented using an **autonumber** field type as this will automatically allocate a unique number to each record in a table, reducing the need for the user to create their own identifying data.

Field validation

Using appropriate validation reduces the chance of error when data is input.

Various validation techniques can be used to ensure data is appropriate:

- **Presence check** – ensures field cannot be left empty (good validation for a primary key or any data that must be part of the record). Data must be present in the field for the record to be stored. Appears in a data dictionary as 'Required'.
- **Restricted choice** – the user is presented with a list of options to choose from (using a drop-down menu, option buttons or similar). This is often used for the input of data for a foreign key, by automatically generating the list of options from the linked file (i.e. a look up validation).
- **Length check** – ensures an appropriate number of characters is input (e.g. minimum of 8 characters, maximum of 20). This is almost always used for fields with the **Text** data type.
- **Range check** – used on numeric fields to ensure number is within certain range (e.g. between 0 and 100). This is used on fields with a **Number** data type.

Queries

Queries (searches) allows the user to find information in a database.

Users may perform simple queries (looking at the contents of one field) or complex queries (looking at the contents of many fields).

When answering exam questions always state what data is being searched for and which field it should be in.

Examples	
To find all the male pupils who are over 12	SEARCH for "male" in the Gender field AND >12 in the Age field
To find all the people who live in Edinburgh or Glasgow	SEARCH for "Edinburgh" in the Town field OR "Glasgow" in the Town field
To find all the people born in the 1990s	SEARCH for >31/12/89 in the DOB field AND <1/1/00 in the DOB field

Sorting

Sorting puts database records in order based on the contents of particular fields.

- Ascending order goes from A to Z, smallest number to largest number.
- Descending order goes from Z to A, largest number to smallest number.

Examples	
Put the customers in alphabetical order	SORT on Surname field in Ascending order then the Forename field in Ascending order.
Find the tallest person in the database	SORT on Height field in Descending order then look at the first record in the list.

Remember, you also need to be able to look at a sorted database and identify the fields it has been sorted on.

Reports


Reports (layouts) allow the presentation of selected data from the database. The output can be customised in a variety of ways, including displaying only chosen fields and formatting the data in a particular way.

Example

Here is part of a table storing details about a company's employees.

Forename	Surname	Gender	Birth Date	Salary
Nancy	Davolio	F	8/12/68	32800
Carol	Paterson	F	9/5/75	25000
Andrew	Fuller	M	19/2/52	43700
Joseph	McDowall	M	27/1/81	19500
Janet	Leverling	F	30/8/63	36000

Here is a report which has been produced to show details of all employees who earn more than £30000.

Employees Report 			
First Name	Last Name	Gender	Birth Date
Nancy	Davolio	F	08-Dec-1968
Andrew	Fuller	M	19-Feb-1952
Janet	Leverling	F	30-Aug-1963
Margaret	Peacock	F	19-Sep-1958
Steven	Buchanan	M	04-Mar-1955
Michael	Suyama	M	02-Jul-1963
Robert	King	M	29-May-1960
Laura	Callahan	F	09-Jan-1958
Anne	Dodsworth	F	02-Jul-1969
Male Employees:		4	
Female Employees:		5	

- Only 4 of the 5 fields are included in the report, *salary* is not required.
- A header with a title has been added.
- The *forename* and *surname* fields have been given different headings.
- The *birth date* field has been formatted differently.
- Different fonts have been used.
- There is a footer with summary fields which automatically calculate male and female numbers.

Forms

Database creators can prepare forms (layouts) specially designed for inputting data, to help improve usability.

A well laid-out form, using suitable techniques such as drop-down menus or checkboxes, reduces the typing required and makes data input easier and faster.

Examples

Id:

Last Name:

First Name:

Job Role:

Street:

City:

State:

Zip:

Phone:

Fax:

Product development

First Name	<input type="text"/>	Last Name	<input type="text"/>
Address1	<input type="text"/>		
Address2	<input type="text"/>		
City	<input type="text"/>	State	<input type="text"/>
Phone (W)	<input type="text" value="() -"/>	Phone (H)	<input type="text" value="() -"/>

Employee ID:

Last Name:

First Name:

Title:

Title Of Courtesy:

Birth Date:

Hire Date:

First name:

Last name:

Password:

Sex: Male
 Female

Country:

Date of birth:

Data dictionaries

A data dictionary is a design notation used to show the fields required in each table of a relational database, including field types, validation required, etc.

Example

Table	Field	PK/FK	Data type	Data size	Unique	Required	Validation	Format
Pupils	Pupil ID	PK	Text	30	Y	Y	Presence	AB1234
	DOB		Date			Y		
	School name	FK	Text	40			Look up from School table	
Schools	School name	PK	Text	40	Y	Y	Restricted choice	
	Address		Text	11	Y	Y		