AQA Computer Science A-Level

4.10 Fundamentals of databases

Intermediate Notes

Specification:

4.10.1 Conceptual data models and entity relationship modelling:

Produce a data model from given data requirements for a simple scenario involving multiple entities

Produce entity relationship diagrams representing a data model and entity descriptions in the form: Entity1 (Attribute1, Attribute2, ...)

4.10.2 Relational databases:

Explain the concept of a relational database

Be able to define the terms:

- attribute
- primary key
- composite primary key
- foreign key

4.10.3 Database design and normalisation techniques:

Normalise relations to third normal form

Understand why databases are normalised

4.10.4 Structured Query Language (SQL):

Be able to use SQL to retrieve, update, insert and delete data from multiple tables of a relational database

Be able to use SQL to define a database table

4.10.5 Client server databases:

Know that a client server database system provides simultaneous access to the database for multiple clients

Know how concurrent access can be controlled to preserve the integrity of the database

Data models

When creating a database, you might be given requirements from which you need to produce a data model: a plan of which things to store and what information about them should be recorded.

Entities and attributes

In database design, an entity is a thing about which data is to be stored, for example: a customer. Attributes are characteristics or other information about entities, for example: the customer's name or address.

Databases are formed of tables which are used to store multiple entities. Each entity usually has its own row in a table and fields of that row hold the entity's attributes.

Table: Customers		
CustomerID	**CustomerName**	**CustomerEmail**
8836	Sue B.	sue.b@gmail.com
3846	Jeremy F.	jeremy-f@hotmail.com
2003	Jackie R.	jackie@jackie-r.net

The table above stores information about a company's customers. Each row in the table holds information about one customer (one entity). The fields in each row hold information about the customers (attributes) such as their name and email address.

Entity identifiers

When creating a database, it's important to ensure that each entity has an unique identifier. An entity identifier is an attribute given to each entity which is unique within that table. In the example above, `CustomerID` is most likely to be a suitable entity identifier.

Synoptic Link

The idea of entity identifiers continues with primary and foreign keys which are covered later in these notes.

Entity description

When describing how information about an entity is to be stored in a database, an entity description can be used.

```
Customer (CustomerID, CustomerName, CustomerEmail)
```

The entity description above describes how information about customers is stored in the database table above. The name of the table is shown outside of brackets which contain each of the entity's attributes separated by commas.

Underlining can be used to identify the attribute or attributes which form the table's entity identifier. In this case, the attribute CustomerID has been underlined.

Relational Databases

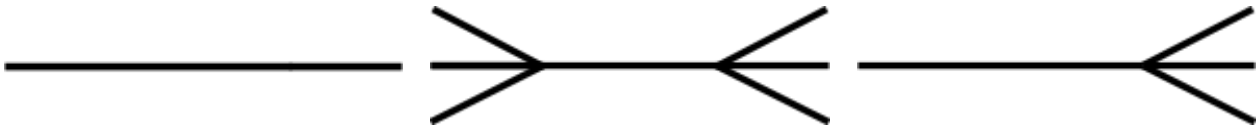
The tables in a database can be related to each other, linked by common attributes. There are three possible degrees of relationship between tables in a database: one-to-one, many-to-many and one-to-many.

Note

Beware - there is no such thing as a many-to-one relationship!

Entity relationship diagrams

Entity relationship diagrams (or ER diagrams) are used to graphically represent the relationships between tables in a database. Tables are shown as rectangles and are joined by lines which can represent different types of relationship.



One-to-one Many-to-many one-to-many

Each car has one owner, and each owner has one car (this example assumes that nobody owns multiple cars).

Each car has many passengers. Each passenger sits in one car.

Each driver can drive many different cars. Each car is driven by many different drivers.

Primary and foreign keys

A primary key is an attribute that provides an unique identifier for every entity in a database table. When tables are linked by a shared attribute, the attribute must be a primary key in one table and is called a foreign key in the other.

A foreign key is an attribute in a table which is the primary key in another, related, table.

Example: The following database tables are related to one another.

Table: Flights		
FlightNo	**PilotNo**	**Destination**
ESY8876	65587	Paphos
RYN4133	13584	Dublin
BRI1101	20547	Munich
ESY5655	65587	Edinburgh
BRI8989	20547	Athens

Table: Pilots	
PilotNo	**PilotName**
65587	Adam Triston
13584	Charlotte Green
20547	Orville Wright

The primary key in `Pilots` is `PilotNo` and is `FlightNo` in `Flights`. The tables are linked by the shared attribute `PilotNo`. This makes `PilotNo` a foreign key in `Flights`.

The relationship in this example is one-to-many. Many different flight routes are operated by the same pilot.

Many-to-many relationships

When linking many-to-many relationships, a new table has to be created. This new table is called a link table.

The following example features the two tables `Products` and `Customer`. There is a many-to-many relationship here as many different types of products are each bought by many different customers.

Table: Products		
ProductID 🔑	**ProductName**	**ProductPrice**
155484765	Knife	£18.99
233145882	Rope	£4.45
366584554	Revolver	£124.98

Table: Customers	
CustomerName	**CustomerID** 🔑
Professor Plum	155484765
Miss Scarlet	233145882
Reverend Green	233145882

In order to model the relationship between the tables, a new table called `Orders` has to be created. This is a link table.

Table: Orders		
OrderID 🔑	**ProductID**	**CustomerID**
223	155484765	155484765
223	233145882	155484765
224	233145882	233145882
225	233145882	233145882
225	366584554	233145882

Database Normalisation

Databases are normalised so that they can be efficient without any compromise to the integrity of their data. Normalising databases involves ensuring that entities contain no redundant or repeated data.

A database that has been normalised allows for faster searching and sorting that an unnormalised database thanks to the smaller tables created in the normalisation process. Furthermore, normalised databases are easier to maintain than their unnormalised counterparts.

There are three levels of normalisation that you need to know: first, second and third normal form.

First normal form

When a database conforms to first normal form, it contains no repeating attributes. The database's data can be referred to as atomic (meaning that no single column contains more than one value).

This table contains repeating attributes so is not normalised to first normal form.

Table: Staff			
Name 🔑	**Department**	**Subject** 🔑	**DepartmentHead**
John Strode	Earth Sciences	Geography	Jackie Smith
Sarah Ng	Science	Chemistry	Brian Jones
Mary Marsh	Science	Physics, Biology	Brian Jones

Splitting the repeating attributes means that this database is now in first normal form.

Table: Staff			
Name 🔑	**Department**	**Subject** 🔑	**DepartmentHead**
John Strode	Earth Sciences	Geography	Jackie Smith
Sarah Ng	Science	Chemistry	Brian Jones
Mary Marsh	Science	Physics	Brian Jones
Mary Marsh	Science	Biology	Brian Jones

Second normal form

In order to meet second normal form, a database must also satisfy first normal form. In second normal form, partial key dependencies are removed.

A partial key dependency occurs in databases with composite primary keys (a primary key made up of multiple attributes combined) when a non-key attribute doesn't depend on the whole of the composite key.

In our example, the primary key is composite.

`Staff (Name, Department, Subject, DepartmentHead)`

Because the attribute DepartmentHead depends only on the attribute Department and Department depends only on the attribute Subject, the tables must be modified to meet second normal form.

Table: Staff	
Name 🔑	**Subject** 🔑
John Strode	Geography
Sarah Ng	Chemistry
Mary Marsh	Physics
Mary Marsh	Biology

Table: SubjectDepartments	
Subject 🔑	**Department**
Biology	Science
Chemistry	Science
Geography	Earth Sciences
Physics	Science

Table: HeadsOfDepartment	
Department 🔑	**Head**
Science	Brian Jones
Earth Sciences	Jackie Smith

Creating the two tables `SubjectDepartments` and `HeadsOfDepartment` has ensured that the database now conforms to second normal form as the partial key dependencies of Department and `DepartmentHead` have been removed from the `Staff` table.

Third normal form

In order to meet third normal form, in addition to conforming to second normal form, a database must have no non-key dependencies.

A database that meets third normal form can be described as follows:

All non-key attributes depend on the key, the whole key and nothing but the key

Our example meets third normal form as none of the attributes that do not form the key (or part of a composite key) depend on the anything other than the whole key.

Structured Query Language (SQL)

SQL is a language used with databases. SQL is easy to learn and use, partly because it is a declarative language, meaning that the programmer describes the result that's required rather than describing the process which should be followed.

There are four main SQL commands: SELECT, UPDATE, INSERT and DELETE.

The SELECT command

SELECT is used for retrieving data from a database table. Commands take the following form:

```
SELECT <attribute> FROM <table> WHERE <condition> ORDER BY <ASC/DESC>
```

Note that the ORDER BY clause is optional. Let's use the following table as an example.

Table: Flights		
FlightNo 🔑	**PilotNo**	**Destination**
ESY8876	13584	Glasgow
ESY1225	13584	Swansea
BRI1101	20547	Berlin

```
SELECT FlightNo FROM Flights WHERE Destination = 'Berlin'
>> BRI1101
```

```
SELECT Destination FROM Flights WHERE PilotNo = '13584' ORDER BY FlightNo DESC
>> Glasgow, Swansea
```

The UPDATE command

This command is used in databases for modifying the attributes of an existing entity and takes the form:

```
UPDATE <table> SET <attribute> = <value> WHERE <attribute> = <value>
```

Table: Students			
StudentNo 🔑	**Name**	**Email**	**Year**
55685	Aaron Aaronson	a.a.aaronson@outlook.com	1
55887	Beth Hunter	elisabeth.h@gmail.com	2
55622	Sam Cooper	samc00per@hotmail.com	1

```
UPDATE Students SET Email = 'beth24@yahoo.co.uk' WHERE StudentNo = 55887
UPDATE Students SET Name = Samuel Cooper WHERE StudentNo = 55622
```

Once the two UPDATE commands above have been carried out on the table above, the table looks like this:

Table: Students			
StudentNo 🔑	**Name**	**Email**	**Year**
55685	Aaron Aaronson	a.a.aaronson@outlook.com	1
55887	Beth Hunter	beth24@yahoo.co.uk	2
55622	Samuel Cooper	samc00per@hotmail.com	1

UPDATE commands usually use the table's primary key to identify which entities to update but can use more general conditions which would update all of the entities that meet the condition.

```
UPDATE Students SET Year = 2 WHERE StudentNO < 55700
```

Table: Students			
StudentNo 🔑	**Name**	**Email**	**Year**
55685	Aaron Aaronson	a.a.aaronson@outlook.com	2
55887	Beth Hunter	beth24@yahoo.co.uk	2
55622	Samuel Cooper	samc00per@hotmail.com	2

The DELETE command

As you might expect, the DELETE command is used for removing entities from a database. The commands take the following form:

```
DELETE FROM <table> WHERE <condition>
```

Table: Cars				
Model 🔑	**Manufacturer** 🔑	**Price**	**Year**	**Sold**
Polo	Volkswagen	4995	2010	TRUE
i10	Hyundai	5225	2013	FALSE
Fiesta	Ford	3995	2009	TRUE

```
DELETE FROM Cars WHERE Sold = TRUE
```

Table: Cars				
Model 🔑	**Manufacturer** 🔑	**Price**	**Year**	**Sold**
i10	Hyundai	5225	2013	FALSE

The INSERT command

When using SQL to add new records to an existing table, the INSERT command is used. The command usually takes the form

```
INSERT INTO <table> (<column1>, <column2>, …) VALUES (<value1>, <value2>, …)
```

but can be simplified to

```
INSERT INTO <table> VALUES (<value1>, <value2>, …)
```

when all of the columns in the table are being used in the correct order.

For example, executing the following commands would add two new records to the `Cars` table.

```
INSERT INTO Cars VALUES (“KA”, “Ford”, 3999, 2010, FALSE)
INSERT INTO Cars (Model, Year, Manufacturer) VALUES (“E-Type”, 1970, “Jaguar”)
```

The first command inserts values into all columns in the correct order. The second command inserts only some values in the wrong order, so must list columns.

Wildcards

Wildcards can be used in SQL commands to specify any possible value. For example, rather than selecting a specific attribute in a SELECT command, a wildcard could be used to return all attributes.

In SQL, wildcards are usually notated with an asterix. For example, using the original `Cars` table from before the delete command:

```
SELECT * FROM Cars WHERE Price > 4000
>> [Polo, Volkswagen, 4995, 2010, TRUE], [Hyundai, 5225, 2013, FALSE]
```

Defining a table with SQL

SQL can be used to make new database tables with the CREATE command. This command specifies the name of the new table, its attributes and their data types. Also specified are entity identifiers like primary and secondary keys.

For example, the following command could be used to make a table called `Artists` with the attributes Title, Artist and Date with a composite primary key composed of the attributes Title and Artist.

```
CREATE TABLE Artworks (Title VARCHAR(225), Artist VARCHAR(255),
Date YEAR, PRIMARY KEY (Title, Artist))
```

This creates an empty table. New records can be added using the INSERT command.

Table: Artworks		
Title 🔑	**Artist** 🔑	**Date**
The Night Watch	Rembrandt	1642
The Persistence of Memory	Salvador Dali	1931
The Great Wave off Kanagawa	Hokusai	1830

SQL Data Types

Data types for attributes are specified when using the CREATE command. The data types supported by SQL are listed in the table below.

Data type	SQL	Description
Fixed length string	`CHAR(size)`	A string with the number of characters specified by size
Variable length string	`VARCHAR(size)`	A string with any number of characters up to the number specified by size
Integer	`INT(size)`	A whole number stored using the number of bits specified by size
Date	`DATE`	A date in the format YYYY-MM-DD
Date and time	`DATETIME`	A date and time combined in the format YYYY-MM-DD HH:MM:SS
Time	`TIME`	A time in the format HH:MM:SS
Year	`YEAR`	A year in one of the two formats YY or YYYY

Client server databases

A client server database system provides simultaneous access to a database for multiple clients. For example, social media websites store information on databases that are continuously being accessed and modified by different users simultaneously.

Issues rarely arise when two users are requesting access to different, unrelated fields in a database. However, when different users attempt to access the same field at the same time, a problem known as concurrent access occurs.

Concurrent access can result in database updates being lost if two users edit a record at the same time and can be managed with the use of record locks, serialisation, timestamp ordering and commitment ordering.