CAIE Computer Science IGCSE

9 - Databases

Advanced Notes

Database terminology

A database is an organised and structured collection of data that can be easily stored, searched, and updated. A table holds related data in rows and columns, like a spreadsheet.

Fields are specific data points (like name or price), records are complete sets of related fields for a single entry (like one customer's information), and validation is the process of ensuring data within fields and records adheres to predefined rules to maintain data integrity.

Data types

There are many data types that can be used to store information within databases. Each field will use a specific data type.

Data Type	Description	Possible use cases	Example(s)
Text / Alphanumeric	Stores letters, numbers, and symbols, but not used for calculations.	Storing names, addresses and emails.	`"John Smith"`, `"AB123"`
Character	Stores a single letter, digit, or symbol.	Storing a sex field ('M' / 'F'), a single exam grade ('A', 'B'), or a single character code.	`'A'`, `'7'`, `'#'`
Boolean	Stores one of two values only: TRUE/FALSE, which can be used to mean Yes/No.	Representing Yes/No answers (e.g. "Subscribed to newsletter?")	`TRUE`, `FALSE`
Integer	Whole numbers only (positive or negative), no decimal part.	Storing quantities (e.g. stock levels, number of tickets), ages, or ID numbers.	`42`, `-7`, `1500`
Real	Numbers that may include a decimal/fractional part.	Storing values that may include decimals such as weights or measurements.	`3.14`, `-0.5`, `99.99`
Date/Time	Stores calendar dates and/or times in a standard format.	Storing dates of birth, booking dates, order timestamps, or deadlines.	`24/08/2025`, `14:35`, `24/08/2025 14:35`

Primary keys

A primary key is a field that provides a unique identifier for every record in a database table. Primary keys are important because they help to ensure that every record is unique - this is especially useful when linking several database tables together (although you don't need to know details about how this works).

To select a primary key, look for the field that uniquely identifies every record, and that it wouldn't make sense to have duplicated across records.

In this table, which stores flight information, FlightNo would be the most appropriate primary key. This is because each record (flight) can be uniquely identified by its flight number, and there's no reason that multiple flights would ever have the same flight number.

Table: Flights

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

The PilotNo field wouldn't be appropriate as a primary key, because it wouldn't necessarily identify each flight individually if several flights have the same pilot. For example, the flights to Munich and Athens are both flown by pilot 20547. Additionally, the destination couldn't be used as a unique identifier, since there'll likely be more than one flight to each destination over time.

If there isn't a field that provides a unique identifier for every record, then it may be the case that none of the records can be used as a primary key. In this case, a new field could be added to uniquely identify each record, such as an ID number that relates to the record.

Structured Query Language (SQL)

SQL is a language used to search for, manage, and manipulate data in a relational database.

The SELECT command

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

```
SELECT <field> FROM <table> WHERE <condition> AND <condition> OR
<condition> ORDER BY <field> <ASCENDING/DESCENDING>
```

To select several fields, separate their names with commas. Note that you can use the AND and OR keywords as many times as needed (if at all), to specify additional conditions that must be met. 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 FlightNo FROM Flights WHERE PilotNo = '13584' AND Destination
= 'Swansea'
>> ESY1225

SELECT FlightNo, Destination FROM Flights WHERE PilotNo = '13584'
AND Destination = 'Glasgow'
>> ESY8876, Glasgow

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

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.

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

In SQL, wildcards are usually notated with an asterisk. For example, to select all details of a Car from the above table with a price greater than £4000, we would write this command:

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

The SUM and COUNT commands

SUM and COUNT are aggregate functions in SQL. They are used when you want to calculate values across multiple rows in a database table.

- SUM adds up all the values in a numeric field.
- COUNT counts the number of rows that meet a condition (or all rows if no condition is given).

The commands take the following form:

```
SELECT SUM(<field>) FROM <table> WHERE <condition>

SELECT COUNT(<field>) FROM <table> WHERE <condition>
```

Like with SELECT, the WHERE clause is optional, and conditions can be joined using AND and OR.

Let’s use the following table as an example.

Table: Tickets		
TicketNo	**FlightNo**	**Price**
T001	ESY8876	85
T002	ESY8876	90
T003	ESY1225	70
T004	BRI1101	120
T005	BRI1101	130

Find the total cost of all tickets in the table:

```
SELECT SUM(Price) FROM Tickets
>> 495
```

To find the total cost of all tickets for flight BRI1101:

```
SELECT SUM(Price) FROM Tickets WHERE FlightNo = 'BRI1101'
>> 250
```

Count how many tickets are for flight ESY8876:

```
SELECT COUNT(TicketNo) FROM Tickets WHERE FlightNo = 'ESY8876'
>> 2
```