

Databases

Read online at pastopic.com/cambridge/computer-science-0478/notes/databases

Last checked 30 September 2026

Read first: [Validation and verification](#)

© 2026 Pastopic. Free for personal study. Please share the link, not the file.

What you need to know

By the end of this topic you should be able to:

1. **Set up a one-table database from some data you are shown.** Decide what columns (fields) it needs and give them sensible names, recognise that each row is one record, and say what checks (validation) should guard the data typed into a field.
2. **Pick a sensible data type for each field** from this list: integer, real, Boolean, character, text (also called alphanumeric) and date/time.
3. **Say why a table needs a primary key**, and decide which field of the table in front of you should be the key, backing your choice with a reason.
4. **Write down what an SQL statement would display, and fill the gaps in part-written ones**, for statements that search a single table. Only these keywords are needed: SELECT, FROM, WHERE, ORDER BY (with ASC or DESC), AND, OR, SUM and COUNT. You must be able to follow what each statement does to the table printed in the question.

You do not need tables linked to each other, foreign keys, or SQL that changes data (adding, editing or deleting records). Older papers (2021–2022) used a "query-by-example" grid instead of SQL; that grid is **no longer examined**, but the same thinking now goes into SQL. Validation checks are covered in depth, with pseudocode, in Validation and verification (linked under "Read first"); this note shows how to choose them for the fields of a table.

Understand it

Tables, fields and records

A database is an organised store of data that a computer can search and sort quickly. At IGCSE every database has just **one table**. Think of a climbing centre that keeps a list of its routes:

field (one column)							field names	
RouteID	Colour	Grade	LengthM	Holds	Indoor	SetOn		
R101	Red	C	12.5	24	TRUE	03/02/2026		
R102	Blue	A	8.0	15	TRUE	10/02/2026		
R103	Green	D	15.5	31	FALSE	14/03/2026		
R104	Red	B	9.5	18	TRUE	21/03/2026		
R105	Yellow	E	18.0	40	FALSE	02/04/2026		
primary key (no value repeats)			record (one row)					

- Each **column** is a **field**: one kind of fact that is stored about every item, such as its length. The top row gives each field its name.
- Each **row** is a **record**: all the facts about one item, here one route.
- ROUTES has **7 fields and 5 records**. The row of field names is not a record, so never count it.

Field names are written as one word with no spaces, often with capitals to show where words join (LengthM, SetOn). A unit can go in the name so nobody has to guess it: LengthM is in metres.

Choosing data types

Every field has one **data type**, which tells the database what kind of value it will hold. The right type lets the database check entries, sort them properly and do arithmetic with them.

Data type	Holds	Fields in ROUTES
integer	whole numbers you might count or add up	Holds (24, 15, ...)
real	numbers that can have a decimal part: measurements, money	LengthM (12.5, 8.0, ...)
Boolean	exactly one of two values: TRUE/FALSE, Yes/No, Y/N	Indoor
character	a single letter, digit or symbol	Grade (A to E)
text / alphanumeric	any mix of letters, digits, spaces and symbols	RouteID, Colour
date/time	a calendar date, a clock time, or both	SetOn

Three things catch people out:

- A code made of digits is still text** if you would never do sums with it and it may start with 0, such as a phone number. R101 has a letter, so it is text without any doubt.
- Prices are usually real**, because they have cents or pence even when a few values happen to be whole. The exception: if every price in the table is a whole number shown without decimals (such as a cinema seat price stored as 8 or 12), integer is the type mark schemes expect. Go by the data you are shown.

- **Yes/No, Y/N and TRUE/FALSE are all Boolean.** It is the "only two possible values" that matters, not how they are written.

Validation

When the table is set up, each field can be given a rule that the database applies to every value typed in, so impossible values are refused before they are stored. Part of defining a table is deciding which rule suits each field. Match the rule to what could go wrong with that field:

Check	What it tests	A field in ROUTES it suits
range check	a number (or date) lies between a lowest and a highest value	LengthM: more than 0 and no more than 30
length check	the number of characters is right	Grade: exactly one character
type check	the value is the right kind of data, such as a real date or a number	SetOn: only real dates get in
presence check	the field has not been left empty	RouteID: every route must have one
format check	the characters follow a fixed pattern	RouteID: the letter R then three digits
check digit	a code's last digit, worked out from its other digits, still matches	an ID number that is often keyed in by hand

A field can have more than one check: RouteID needs both presence and format. Validation stops impossible data; it cannot stop a sensible-looking wrong value, such as 12.5 typed as 15.2.

The primary key

Every table needs one field that can be used to point at a single record without any doubt. That field is the **primary key**: its value is different in every record, so giving one value finds one record and no other.

In ROUTES, Colour could never be the key, because two routes are Red. RouteID is the natural choice: the centre gives each route its own code, and no code is ever used twice. Notice that Grade and SetOn also happen to have no repeats in these five rows, but nothing stops two routes having the same grade or being set on the same day, so they are poor keys. Good keys are **codes made up for the purpose** (IDs, reference numbers). Names are poor keys: two people or things can share a name, and the same name can be typed in slightly different ways.

Asking questions with SQL

Structured query language (SQL) is how you ask a database for data. A statement is made of clauses, each starting with a keyword, and ends with a semicolon:

```
SELECT RouteID, LengthM
FROM ROUTES
WHERE Indoor = TRUE
ORDER BY LengthM DESC;
```

- **SELECT** names the fields to show, in the order you want the columns, separated by commas.
- **FROM** names the table.
- **WHERE** gives the condition a record must meet to be shown. Leave it out and every record is shown.
- **ORDER BY** sorts the results by a field: **ASC** for smallest first (A to Z, earliest date first), **DESC** for largest first. With neither, ascending is used. The sort field does **not** have to be one of the fields shown.

```
SELECT RouteID, LengthM
FROM ROUTES
WHERE Indoor = TRUE
ORDER BY LengthM DESC;
```

RouteID	LengthM	Indoor
R101	12.5	TRUE
R102	8.0	TRUE
R103	15.5	FALSE
R104	9.5	TRUE
R105	18.0	FALSE

1 WHERE keeps the shaded rows

output (no headings)

R101 12.5
R104 9.5
R102 8.0

2 ORDER BY: largest first

3 SELECT: only these
two fields

The database works through it in this order: take the table named in **FROM**, keep the rows that pass **WHERE**, sort them with **ORDER BY**, then show only the **SELECT** columns. With no **ORDER BY**, the rows come out in the order they appear in the table. A statement like this, or a few of them saved together to be run again, is often called an **SQL script**.

Writing conditions

A condition compares a field with a value using **=**, **<>** (not equal to), **<**, **>**, **<=** or **>=**.

- **Text values go in quotation marks:** `WHERE Colour = "Red"`. Single quotes (`'Red'`) are fine too.
- **Numbers do not:** `WHERE Holds > 20`.
- **Boolean fields** are tested with `= TRUE` or `= FALSE` (or `= Yes`, `= "Y"`, matching how the table shows them). Writing just the field name, `WHERE Indoor`, means the same as `WHERE Indoor = TRUE`.

A condition can also compare **two fields of the same record**. In a shop's stock table, `WHERE NumberInStock < ReorderLevel` keeps the items that are running low. Neither side has quotation marks, because both are field names, not values.

Join two conditions with **AND** (a record must pass **both**) or **OR** (passing **either one** is enough). On ROUTES, **WHERE Colour = "Red" AND Holds > 20** keeps only R101, but **WHERE Colour = "Red" OR Holds > 20** keeps R101, R103, R104 and R105. **AND** gives fewer rows; **OR** gives more.

Adding up and counting

- **SUM(field)** adds the values in a numeric field over the records that pass the condition. **SELECT SUM(Holds) FROM ROUTES;** returns 128, the holds on all five routes, and **SELECT SUM(Holds) FROM ROUTES WHERE Indoor = TRUE;** returns 57.
- **COUNT(field)** gives **how many records** pass the condition, not a total. **SELECT COUNT(RouteID) FROM ROUTES WHERE Indoor = FALSE;** returns 2, because two routes are outdoors.

Each gives a single number as its output.

Key facts and formulas

There is no formula list for this topic. Everything below is "Learn it".

Formula	Status
Field = column; record = row; the heading row is not a record	Learn it
Primary key: a field whose value is different for every record	Learn it
Data types: integer, real, Boolean, character, text/alphanumeric, date/time	Learn it
SELECT fields to show FROM table WHERE condition ORDER BY field ASC/DESC ;	Learn it
Comparisons: =, <> (not equal), <, >, <=, >=	Learn it
AND : both conditions true; OR : at least one true	Learn it
SUM(field) : total of that field; COUNT(field) : number of records	Learn it
Text values in quotes; numbers and TRUE/FALSE without	Learn it

How to do it

In the Paper 2 questions we have tagged for this topic (37 questions, 2021–2025), the types came up this often. 33 of the 37 questions are made of several parts of different types, so the counts add up to more than 37.

Question type	Questions
Choose the primary key and give a reason	21
Choose data types for fields (or fields for data types)	20
Write or complete an SQL query	17

Question type	Questions
Complete a query-by-example grid, 2021–2022 only	14
Give the output of a query	11
Count the fields and records	10
Explain SQL keywords, statements or database terms	5
Name the fields for described data	3

1. Choose the primary key and give a reason (21 questions)

1. Look for a field made up to identify things: a code, ID or number field, usually the first column.
2. Scan down it to be sure no value appears twice.
3. Write the field name **exactly** as printed.
4. Give the reason in terms of uniqueness: every value in that field is different, so each value picks out one record only.

Variations you will meet:

- **"State the purpose of a primary key."** Say what it does for the table, not which field it is: it lets each record be told apart from every other by one value that never repeats.
- **"Why is field X not suitable?"** Find a value in X that repeats and say so (quote it if you can). For a name field, you can also say that a name typed slightly differently would no longer match.
- **"Is any field suitable?"** Sometimes the answer is **no**. In a table of paint tins with fields for shade, finish and tin size, any value could turn up in two records. Say that no field can be relied on to differ from record to record, so none of them could act as the key.
- **"Why is a new ID field better?"** The database can make sure no code is ever handed out twice, and because every code follows one short pattern, a format check can test each new entry.

2. Choose data types for fields (or fields for data types) (20 questions)

1. Look at the example data (or the description) for each field.
2. Apply the table in "Understand it": whole numbers → integer; decimals, money, measurements → real; two possible values → Boolean; one symbol → character; codes, names and descriptions → text/alphanumeric; dates and times → date/time.
3. Use **only** the type names the question lists. If it says "alphanumeric", write that rather than "text"; never write "string" or "number".

Variations you will meet:

- **"Give one field for each data type."** Go the other way: find a Boolean field (Yes/No, TRUE/FALSE), an integer (a count, a year), a real (a price, a length with decimals) and a text field (a name). If no field may be used twice, pick a different one for each type.

- **No type may be used twice.** Then one of the fields must be the less obvious type, so a one-letter code such as a size S, M or L becomes character, not text.
- **"Why is this field Boolean?"** Because it can only ever hold one of two values. Don't just repeat the word Boolean as your reason.
- Mark schemes allow sensible alternatives, such as an ID as text or integer, or Yes/No stored as text, but the expected answer is the safest.

3. Write or complete an SQL query (17 questions)

1. **SELECT:** list only the fields the question asks for, spelt exactly as in the table, separated by commas. Extra fields lose the mark.
2. **FROM:** the table name, spelt exactly.
3. **WHERE:** field, operator, value. Text in quotes; numbers bare; Booleans = `TRUE/= FALSE` (or however the table writes them).
4. **ORDER BY** only if the question asks for a sort: "in order of X" or "smallest first" means `ASC` (or nothing); "largest first", "most recent first" or "descending" means `DESC`.
5. End with `;`.

Translate words into operators with care: "at least 10" is `>= 10`, "more than 10" is `> 10`, "no more than 10" is `<= 10`, "not yet paid" is `= FALSE` or `= "No"`.

Variations you will meet:

- **Two conditions:** "vegan items under 2.50" needs `AND`; "items that are red or longer than 15 m" needs `OR`.
- **A total:** `SELECT SUM( field )` with the brackets round the field to be added, then `FROM` and any `WHERE`.
- **"How many ...":** `SELECT COUNT( field )` with a `WHERE` that picks the records to count.
- **"Explain how a different field could give the same result."** Look for two fields that carry the same information, such as a Yes/No field `Empty` and a count field `BikesDocked` in a table of bike-share docks: `WHERE Empty = TRUE` and `WHERE BikesDocked = 0` pick the same rows. Name the other field and the value you would test it against.
- **Part of the statement is printed** (for example `SELECT` already filled in). Copy nothing twice and fill only the gaps.

4. Complete a query-by-example grid, 2021–2022 only (14 questions)

These questions came from the old syllabus, which asked students to fill a grid instead of writing SQL. The grid is not in the current syllabus, so you won't be asked to fill one. The thinking is exactly the same as writing SQL, so the old questions are still good practice if you answer them in SQL instead: the grid's shown fields become the `SELECT` list, its table row becomes `FROM`, its criteria become `WHERE` (joined with `AND` when they sit in the same row) and its sort row becomes `ORDER BY`.

5. Give the output of a query (11 questions)

1. **WHERE first.** Go down the table and mark every record that passes the condition. With **AND** it must pass both parts; with **OR**, either.
2. **ORDER BY next.** Put the marked records in order of the sort field (smallest first unless **DESC**), even if that field isn't shown.
3. **SELECT last.** For each record write only the listed fields, in the order **SELECT** lists them, one record per line.
4. Copy every value exactly as printed (spelling, decimals, capitals). Add no commas between columns, no quotation marks and no headings.

For example, on **ROUTES**, `SELECT RouteID FROM ROUTES WHERE Grade <> "A" ORDER BY Holds ASC`; skips R102 (grade A), then sorts by the hidden field **Holds** (18, 24, 31, 40), giving R104, R101, R103, R105, one per line.

Variations you will meet:

- `<>` means "not equal to": on **ROUTES**, `WHERE Grade <> "A"` keeps every route except R102.
- **Long tables** (often 15 rows or more): tick rows as you go so you don't miss one.
- **A SUM or COUNT statement:** the output is one number.

6. Count the fields and records (10 questions)

1. **Fields** = number of columns. Count the headings carefully, including very short ones (a single letter or a two-letter heading is still a field).
2. **Records** = number of data rows. **Don't count the heading row.**
3. Put each number next to the right word: swapping them loses both marks.

ROUTES has 7 fields and 5 records. If the fields are given as a list rather than a table, count the list.

7. Explain SQL keywords, statements or database terms (5 questions)

These come in three shapes: explain what a whole statement does, match each keyword to a description, or fill the gaps in a short paragraph of database words.

Explaining a whole statement. Read it in the order the database runs it, not the order it is written:

1. Begin at **FROM**: which table is being searched?
2. Look at **WHERE**: which records survive? If there is no **WHERE**, say "all records".
3. Look at what **SELECT** asks for. A list of fields gives a list; `SUM( field )` gives one total; `COUNT( field )` gives one number of records.
4. Write one sentence joining the result to the records: `SELECT SUM(Holds) FROM ROUTES WHERE Indoor = TRUE`; "adds up the holds on the indoor routes only, giving 57". Naming the result without saying which records it covers throws away a mark.

Matching keywords. Ask what each keyword changes about the result:

If the description is about ...	the keyword is
a single number of how many records passed	COUNT
a single total of one numeric field	SUM
the order the results come out in	ORDER BY (ASC or DESC)
the test a record must pass	WHERE (with AND / OR for two tests)
where the data comes from	FROM
which columns appear	SELECT

Some keywords in the list are spares that match nothing, so cross each one off as you use it.

Paragraph of terms. Picture the table as a grid. Going across, one line is a **record**, also called a **row**; going down, one line is a **field**, also called a **column**. The field whose entries never repeat is the **primary key**. Instructions written in SQL to search the table are **SQL scripts** (or statements). Several of the offered words are there only to mislead, such as "variable" or "program", which belong to programming, not databases.

8. Name the fields for described data (3 questions)

1. For each piece of data described, make a short name with no spaces that says what it holds (`SeatsLeft`, `notField 3`).
2. Give each its data type, using the rules in section 2.
3. Choose the field that suits being the primary key, usually the code or ID.

A field is unnecessary if its value can always be worked out from another field (for example, a Yes/No field `Adult` in a table that already has each person's `Age`).

Worked examples

The bold codes show where marks are earned, as in Cambridge mark schemes. The number is how many marks.

- **B1** mark for a correct result on its own, with no method needed.

Example 1

A bike repair shop stores its jobs in a database table called `REPAIRS`.

JobRef	Customer	BikeType	Gears	Cost	Paid	Booked
J2041	K Osei	Road	22	45.50	TRUE	04/05/2026
J2042	M Duarte	Mountain	24	72.00	FALSE	04/05/2026
J2043	K Osei	Hybrid	21	18.25	TRUE	05/05/2026
J2044	T Nakamura	Road	18	30.00	FALSE	06/05/2026
J2045	L Brennan	Mountain	12	95.75	TRUE	06/05/2026
J2046	S Ivanova	BMX	1	22.40	TRUE	07/05/2026
J2047	M Duarte	Road	16	12.00	FALSE	08/05/2026
J2048	A Haddad	Hybrid	21	27.60	TRUE	08/05/2026

(a) State the number of fields and the number of records in REPAIRS. [2]

(b) Identify the most suitable field for the primary key, and state why Booked would not be suitable. [2]

(c) Give the most appropriate data type for each of these fields: JobRef, Gears, Cost, Paid, Booked. [3]

(d) Suggest a suitable validation check for the Cost field. [2]

(a) Fields: 7 **B1**. Records: 8 **B1** (the heading row is not a record).

(b) JobRef **B1**. Booked has repeated values (04/05/2026 appears twice), so a date does not pick out one job **B1**.

Cost happens to have no repeats here, but two jobs could easily cost the same, so it is not a sensible key.

(c)

Field	Data type
JobRef	text (alphanumeric)
Gears	integer
Cost	real
Paid	Boolean
Booked	date/time

B3 for all five correct, **B2** for three or four, **B1** for one or two.

(d) A range check **B1**, so that Cost must be greater than 0 and no more than, say, 200 **B1**. (A type check that only numbers are accepted would also earn the first mark.)

Example 2

A hill-walking club keeps this table, PEAKS.

HillID	PeakName	Area	HeightM	Climbed
H01	Stob Lairig	North	987	TRUE
H02	Meall Fionn	South	1024	FALSE
H03	Carn Righ	North	1109	TRUE
H04	Sgurr Ban	West	956	TRUE
H05	Creag Odhar	South	912	FALSE
H06	Beinn Tarsa	West	1061	FALSE
H07	Druim Glas	North	934	FALSE

(a) Give the output from this SQL statement. [3]

```
SELECT PeakName, HeightM
FROM PEAKS
WHERE Area = "North" OR HeightM > 1000
ORDER BY HeightM DESC;
```

(b) Give the output from each of these SQL statements. [2]

Statement 1: `SELECT COUNT(HillID) FROM PEAKS WHERE Climbed = FALSE;`

Statement 2: `SELECT SUM(HeightM) FROM PEAKS WHERE Climbed = TRUE;`

(a) WHERE: a peak passes if it is in the North **or** higher than 1000 m. That is H01, H07 and H03 (North) and H02 and H06 (over 1000); H03 passes both but appears once. H04 and H05 fail both.

ORDER BY HeightM DESC puts the highest first:

```
Carn Righ 1109
Beinn Tarsa 1061
Meall Fionn 1024
Stob Lairig 987
Druim Glas 934
```

B1 for the right five peaks with correct data, **B1** for exactly two columns in the right order with no extra punctuation, **B1** for the rows in the correct order.

(If the condition had used AND, only Carn Righ would pass.)

(b) Statement 1: four peaks have Climbed = FALSE (H02, H05, H06, H07), so the output is **4 B1**.

Statement 2: the climbed peaks are H01, H03 and H04, so the output is $987 + 1109 + 956 = 3052$ **B1**.

Example 3

A bakery has a database table BAKES with these fields:

- ItemCode: a code such as B04
- ItemName: the name of the item, such as Seed roll
- Vegan: whether the item is vegan, TRUE or FALSE
- Price: the price in dollars, such as 2.45
- Made: how many of the item were baked today
- Sold: how many of the item have been sold today

(a) Complete the SQL statement to show only the name and price of the vegan items that cost less than \$2.50, cheapest first. **[4]**

(b) Write an SQL statement to give the total number of items sold today. **[2]**

(c) Explain what this SQL statement does. **[2]**

```
SELECT COUNT(ItemCode) FROM BAKES WHERE Made = Sold;
```

(a)

```
SELECT ItemName, Price
FROM BAKES
WHERE Vegan = TRUE AND Price < 2.50
ORDER BY Price ASC;
```

B1 for exactly ItemName, Price after SELECT; **B1** for FROM BAKES; **B1** for Vegan = TRUE AND Price < 2.50 (WHERE Vegan AND Price < 2.50 is equally correct); **B1** for ORDER BY Price ASC (or just ORDER BY Price).

(b) SELECT SUM(Sold) **B1** FROM BAKES; **B1**. There is no WHERE, because every item counts.

(c) It counts how many records there are **B1** where the number made equals the number sold, that is, how many items have sold out today **B1**.

Common mistakes

- **Swapping fields and records.** Examiners regularly see the two numbers the wrong way round, especially for long tables where the rows far outnumber the columns. Fields are the columns; records are the rows.
- **Counting the heading row as a record.** It only holds the field names.
- **Adding punctuation to an output.** Commas between columns, quotation marks round text, or column headings all cost marks. Write the values alone, exactly as the table shows them.
- **Getting the row order wrong.** With ORDER BY, the first line of your output must be the smallest value of the sort field (or the largest with DESC), even when that field isn't displayed. Unsorted outputs follow the table's own row order.
- **Leaving quotation marks off text values.** WHERE Colour = Red loses the criterion mark; write = "Red". Numbers need no quotes.

- **Selecting extra fields**, or misspelling a field or table name. Copy names letter for letter from the table: `SeatsFree`, not `Seats free`.
- **Using a data type that isn't in the list**. "String", "number" and "currency" are not accepted when the question gives integer, real, text and so on. When the question says no type may be repeated, don't give the same one twice.
- **Mixing up integer and real**. Money and measurements with decimal places are real; counts are integers.
- **Giving the type as its own reason**. "It is Boolean because it is a Boolean" earns nothing; "it can only be yes or no" does.
- **Ignoring what you worked out before**. If an earlier part of the same question settles which field or value matters (for example, which field is Boolean), your WHERE condition should use it.
- **Naming a key field without checking it**. Scan the column first: a field with a repeated value can never be the primary key.

How to get the marks

- **Output questions** lose marks for rows in the wrong order, a missing or extra row, a wrong or extra column, or a misspelt value. Line your values up in columns, one record per line.
- **Completing SQL**: each clause earns its own mark, so a slip in the condition doesn't cost you the others. Make sure the easy lines are perfect.
- **"Give the name of the field"** needs the name exactly as printed, nothing more.
- **"State the reason" for a key** needs one clear point about uniqueness: every value in the field is different. For a field that fails, point to its repeated values.
- **Data-type tables** give part marks when only some boxes are right, so fill every box even if you are unsure of one.
- **"Explain the purpose of" a statement** needs what it calculates and which records it covers, such as "the number of classes that run online".
- **"Identify" or "State"** needs a short answer: one field, one type, one number. **"Explain"** needs a reason as well.
- **Boolean criteria** can be written `= TRUE`, `= Yes` or with the field name alone; use the values the table shows.

Quick check

A dance school stores its classes in the table CLASSES.

ClassID	Style	Level	Day	Places	Fee	Online
D01	Ballet	B	Mon	12	8.50	FALSE
D02	Salsa	I	Tue	20	7.00	TRUE
D03	Tap	B	Wed	10	6.50	FALSE

ClassID	Style	Level	Day	Places	Fee	Online
D04	Ballet	A	Thu	8	11.00	FALSE
D05	Hip hop	I	Fri	25	7.50	TRUE
D06	Jazz	A	Sat	14	9.25	TRUE

1. State the number of fields and records in CLASSES, and explain why Style is not a suitable primary key.
2. Give the most appropriate data type for Level, Places, Fee and Online, using a different type for each.
3. Give the output from `SELECT Style, Fee FROM CLASSES WHERE Online = TRUE OR Places < 11 ORDER BY Fee DESC;`
4. Write an SQL statement to show only the class ID and day of the beginner (B) classes that are not online, and give its output.
5. What does `SELECT SUM(Places) FROM CLASSES WHERE Level = "A";` return, and what does the number mean?

Answers

1. 7 fields and 6 records. Style repeats (Ballet appears twice), so it cannot pick out one class; ClassID should be the key.
2. Level: character (one letter). Places: integer. Fee: real. Online: Boolean.
3. The rows passing are D02, D05 and D06 (online) and D03 and D04 (fewer than 11 places). Sorted by Fee, highest first:


```
Ballet 11.00  
Jazz 9.25  
Hip hop 7.50  
Salsa 7.00  
Tap 6.50
```
4. `SELECT ClassID, Day FROM CLASSES WHERE Level = "B" AND Online = FALSE;` Output:
D01 Mon and D03 Wed, one per line.
5. It returns $8 + 14 = 22$: the total number of places in the advanced (A) classes.

Past questions to try

- [0478/21/O/N/25 Q1](#): matching SQL keywords to what they do.
- [0478/23/M/J/25 Q5](#): data types for eight fields, then a Boolean condition in SQL.
- [0478/21/M/J/24 Q9](#): deciding that no field can be the key, then a SUM with a condition.
- [0478/23/O/N/23 Q9](#): why a name makes a poor key, an output, and a WHERE with AND.