

Computing III: Build, Query and Explain a Database Web App

GAC017 public-outcome practical teaching; full centre guide governs assessed depth.

A working app must answer a clear question

- Build a database, a backend and a reader interface.
- Explain what the project can and cannot establish.
- Use the current centre brief for assessed work.

Browser, server and database have different jobs

Component	Job in the course example
Browser	Collect the minimum and display returned course totals.
JavaScript server	Validate the request, bind its value and run the query.
SQLite database	Store related records and calculate totals inside the server process.

Routes and validation belong on the server

Server decision	Concrete behaviour
Route / method	GET /api/courses handles this read-only operation.
Input rule	Accept one whole minimum from 0 to 99; a missing minimum means 0.
Query / response	Use fixed SQL with a bound value; return an honest status and JSON.

Use the evidence

Direct callers can skip the browser's number control, so the server must enforce its own rule.

A third table represents the many-to-many relationship

Table	Key and purpose
students	id identifies one student; store the name once.
courses	id identifies one class; store title and capacity once.
enrolments	The pair (student_id, course_id) links existing records and must be unique.

Use the evidence

Each student can join several classes, and each class can contain several students. The third table represents those pairs.

SQL filtering and grouping answer different questions

SQL operation	Question it answers
JOIN	Which enrolment records relate to each class ID?
WHERE c.id = 20	Which rows belong to class 20, before calculating totals?
GROUP BY / COUNT	How many matched enrolment IDs belong to each class?
HAVING COUNT(...) >= 2	Which grouped classes meet the minimum total?

Bound values and empty results are normal API behaviour

Prepared query

The server owns the SQL structure and its value placeholder.

`courses.all(minimum)` supplies the accepted number separately.

Valid empty result

Minimum 4 matches no original class.

Return a successful empty list with 200; the input and query still succeeded.

Data analysis needs a question, checks and limits

Ask which classes currently have spare places; use one class ID as the analysis unit.



Check missing values, IDs, repeated enrolment pairs and non-negative capacities.



Calculate capacity minus enrolments for each class.



Explain the descriptive result and its limits before recommending a next action.

Trace a minimum-two request through all three parts

The browser sends GET /api/courses?min=2.



The server validates 2 and binds it to the minimum placeholder.



SQLite returns course 10: Coding, capacity 3, enrolled 3.



The browser shows the returned class and calculates 0 places left.

Run the complete example through its own server

- Keep server.mjs, courses.sql and index.html together.
- Use the printed loopback address.
- Restarting resets the in-memory practice database.

```
node server.mjs  
# If the default port is occupied:  
GAC_DEMO_PORT=0 node server.mjs
```

One enrolment row links two existing records

Fictional enrolment pairs	What they establish
(1,10), (2,10), (3,10)	Class 10 has three enrolments.
(1,20)	Student 1 also belongs to class 20.
(4,30)	Class 30 has one enrolment.
No pair with course 40	Music has no enrolments; the class record still exists.

Use the evidence

Change Mei's name in the student record, not in every enrolment. Names and class facts belong in their parent tables.

Create parent tables with simple constraints

- Primary keys identify records.
- NOT NULL requires a stored value.
- CHECK rejects a negative capacity.

```
CREATE TABLE courses (  
  id INTEGER PRIMARY KEY,  
  title TEXT NOT NULL,  
  capacity INTEGER NOT NULL CHECK (capacity >= 0)  
);
```

Enable references and make each enrolment pair unique

- Foreign keys require existing parent records.
- Enable foreign-key checking for this SQLite connection.
- The pair is unique; each individual ID may repeat.

```
PRAGMA foreign_keys = ON;  
CREATE TABLE enrolments (  
  student_id INTEGER NOT NULL REFERENCES students(id),  
  course_id INTEGER NOT NULL REFERENCES courses(id),  
  PRIMARY KEY (student_id, course_id)  
);
```

Capacity data does not enforce a booking limit

Database constraint

CHECK (capacity ≥ 0) rejects a negative capacity.

The paired primary key rejects the same enrolment twice.

Booking operation

Those rules do not compare the number of enrolments with capacity.

A real booking needs an explicit capacity check and safe handling of simultaneous requests. This example is read-only.

Your turn: preserve class identity

Your turn

Classes 10 and 20 are both called Coding, with three and one enrolments. What would grouping only by title count? Explain which grouping fields preserve the intended class-level question.

Attempt the complete task before advancing. Show the method and interpret the result.

Group by class identity, not a shared title

Class identity	Title	Enrolments
10	Coding	3
20	Coding	1

Use the evidence

Grouping only by title combines the two classes into 4. Group by ID, title and capacity to answer a class-level question.

Keep the empty class with a left join

- Start with every course.
- Match its enrolment rows where they exist.
- Keep Music even when no enrolment matches.

```
SELECT c.id, c.title, c.capacity,  
       COUNT(e.student_id) AS enrolled  
FROM courses c  
LEFT JOIN enrolments e ON c.id = e.course_id  
GROUP BY c.id, c.title, c.capacity  
ORDER BY enrolled DESC, c.id;
```

Your turn: count an empty class

Your turn

Music has a course record but no matching enrolment rows. Predict `COUNT(e.student_id)` and `COUNT(*)` after `LEFT JOIN`. Explain which gives its enrolment total.

Attempt the complete task before advancing. Show the method and interpret the result.

COUNT of a matched ID differs from COUNT of every row

COUNT(e.student_id)

Music's preserved course row has no matched student ID.

Counting matched IDs gives 0 enrolments.

COUNT(*)

The LEFT JOIN still preserves one row for Music.

Counting every joined row gives 1, which is wrong for Music's enrolment total.

Use HAVING to apply a threshold to course totals

- Minimum 1 keeps classes 10, 20 and 30.
- Minimum 3 keeps class 10.
- The threshold includes equality.

```
GROUP BY c.id, c.title, c.capacity  
HAVING COUNT(e.student_id) >= ?  
ORDER BY enrolled DESC, c.id;
```

Enrolment totals do not count unique students

Count	Original dataset
Enrolment pairs	5: three in class 10, one in 20 and one in 30.
Students	4 distinct student records.
Reason for the difference	Mei belongs to two classes and appears in two enrolment pairs.

Use the evidence

Label the unit you count. Five enrolments cannot be reported as five unique students.

Validate one whole number on the server

- Accept 0 through 99; absent min defaults to 0.
- Reject negatives, decimals, SQL text and repeated values.
- Do not rely on the browser number field.

```
const values = url.searchParams.getAll('min');
const raw = values.length === 0 ? '0' : values[0];
if (values.length > 1 || !/^(0|[1-9][0-9]?)$/.test(raw)) {
  json(response, 400, { error: 'Use one whole number, 0 to 99.' });
  return;
}
const minimum = Number(raw);
```

Prepare structure, then bind the accepted value

- The query contains a value placeholder.
- `courses.all(minimum)` supplies the number separately.
- Binding this value does not secure every operation.

```
const courses = database.prepare(`
  SELECT c.id, COUNT(e.student_id) AS enrolled
  FROM courses c LEFT JOIN enrolments e ON c.id = e.course_id
  GROUP BY c.id
  HAVING COUNT(e.student_id) >= ?
`);
const result = courses.all(minimum);
```

Your turn: empty success or bad input

Your turn

Predict the status and meaning for GET `/api/courses?min=4` and GET `/api/courses?min=-1` using the original data. Why do they require different responses?

Attempt the complete task before advancing. Show the method and interpret the result.

Use status codes that explain the request outcome

Request	Status	Meaning
GET ...?min=2	200	Course 10 only
GET ...?min=4	200	Valid query; empty list
GET ...?min=-1	400	Invalid minimum
GET /missing	404	No matching route
POST /api/courses	405	Method not accepted

Return only the data needed by the reader

Needed response data

Return course identities, titles, capacities and enrolment totals.

The browser can calculate places left from those totals.

Unneeded personal data

Student names stay out of this public response.

A real application also needs appropriate permissions and access rules. The practice records are invented.

A successful fetch still needs an HTTP status check

- Read the JSON response.
- Check `response.ok` before using successful data.
- An HTTP error may arrive through a completed request.

```
const response = await fetch(
  `/api/courses?min=${encodeURIComponent(minimum.value)}`
);
const data = await response.json();
if (!response.ok) throw new Error(data.error || 'Request failed.');
```

Insert course values as text inside created cells

- Create each row and cell as a page element.
- Set `textContent` for each returned value.
- Calculate places left from capacity minus enrolled.

```
const row = document.createElement('tr');
for (const value of [course.id, course.title, course.enrolled,
                    course.capacity - course.enrolled]) {
  const cell = document.createElement('td');
  cell.textContent = value;
  row.append(cell);
}
rows.append(row);
```

Clear old data and show loading, empty or failed states

Page state	What the reader sees
Loading	Clear old rows, show a loading message and disable the load button.
Success / empty	Display current rows, or explain that no classes match the valid minimum.
Failure	Show a failure message and enable retry; old results must not look current.

Ignore a late result from an older request

- Number each request as it starts.
- Only the latest number may update the result.
- Only its completion re-enables the load button.

```
const requestId = ++latestRequest;  
// After receiving the response:  
if (requestId !== latestRequest) return;  
// In finally:  
if (requestId === latestRequest) load.disabled = false;
```

Check records and name the unit of analysis

Check	Reason
Class ID as analysis unit	Keep Coding 10 and Coding 20 separate.
Keys / missing records	Check valid IDs and missing values; enabled foreign keys reject absent parents.
Repeated pairs / capacity	Reject duplicate enrolments and negative capacities.
Time and limits	Date real records; enrolments change. These invented counts cannot establish causes.

Recommend a check supported by the current totals

ID / class	Capacity	Enrolled	Places left
10 Coding	3	3	0
20 Coding	2	1	1
30 Art	2	1	1
40 Music	3	0	3

Use the evidence

Places left = capacity — enrolled. Check current availability in classes 20, 30 and 40 before offering places; these are invented records.

A chart and report should preserve identity and limits

A supported short report

Question and method: identify spare places by class ID; calculate capacity minus enrolments after checking the records.

Result: classes 20, 30 and 40 have spare places. Music 40 has three, the other two one each.

Limit and next action: four invented classes at one time cannot establish quality or future demand. Verify current availability before offering places.

Notice

A chart should retain both Coding IDs and Music's zero enrolments. Include the query or calculation for reproducibility.

Predict query and dataset changes before running them

Change to original data	Prediction to explain
Minimum 1 / minimum 3	IDs 10, 20, 30 / ID 10 only; equality is included.
Add empty Drama 50, capacity 2 Add enrolment (2,20)	At minimum 0: zero enrolments and two places left. Class 20 reaches two; minimum 2 now returns 10 and 20.
Duplicate (1,10) / missing (99,10)	Paired primary key / enabled foreign key rejects the invalid record.

Check yourself

- Explain identity, joins, counts, constraints and validation.
- Separate empty success from invalid input.
- Give a supported recommendation and an explicit limit.