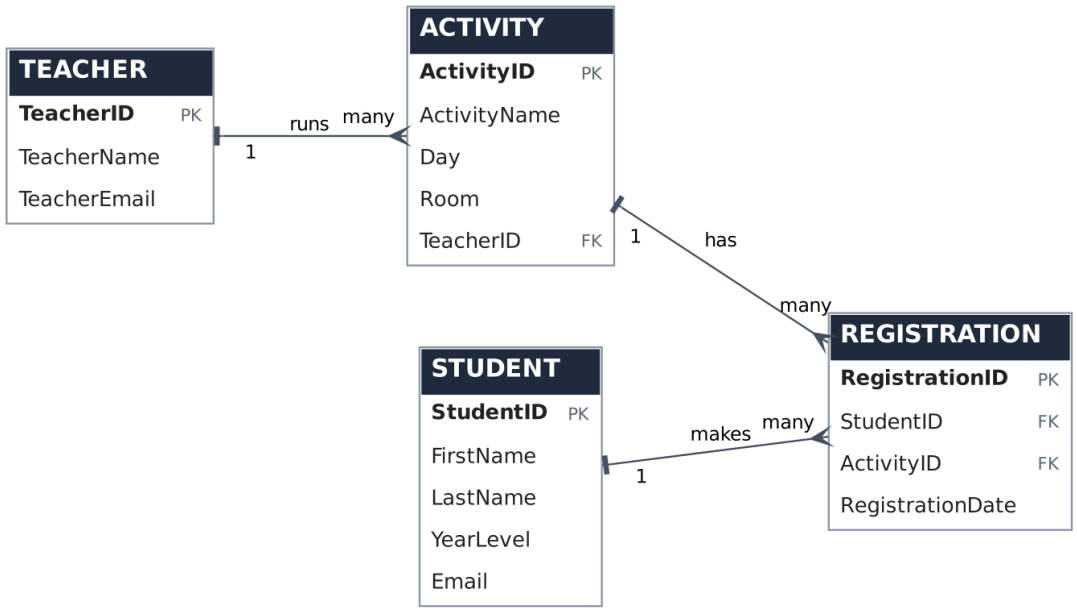
VCE APPLIED COMPUTING · UNIT 1 OUTCOME 1 · RELATIONAL DATABASES · LESSONS 3–4

Querying Ridgeview with SQL

Predict · Run · Investigate · Modify · Make. Read and predict code before you write your own.

Set up

1. Open **ridgeview.db** in DB Browser for SQLite (free from sqlitebrowser.org). If you only have the .sql file: File > Import > Database from SQL file.
2. Check Edit > Preferences > Database: **Foreign keys** should be ticked. Without it, SQLite does not check foreign keys.
3. Use the **Execute SQL** tab. Do not click *Write Changes* today.



The database built from our Lesson 2 ERD.

Predict and Run

Write your prediction **before** you run each query. A wrong prediction is useful: it shows you what to investigate.

```
-- Query 1
SELECT ActivityName, Room
FROM Activity
WHERE Room = 'D04';
```

My prediction (how many rows? which ones?)	What actually happened

```
-- Query 2
SELECT Student.FirstName, Student.LastName
FROM Student, Registration
WHERE Student.StudentID = Registration.StudentID
  AND Registration.ActivityID = 'A02';
```

My prediction	What actually happened

Investigate

1. In Query 2, which line links the two tables? Explain it using the words *primary key* and *foreign key*.
2. Predict, then run: `SELECT COUNT(*) FROM Student, Registration;` (no linking line). Why is the answer 800? What does this tell you about forgetting the link?
3. Run the query below. Why does Dan O'Leary appear twice? Add `DISTINCT` after SELECT and run it again.

```
SELECT Student.FirstName, Student.LastName
FROM Student, Registration, Activity, Teacher
WHERE Student.StudentID = Registration.StudentID
  AND Registration.ActivityID = Activity.ActivityID
  AND Activity.TeacherID = Teacher.TeacherID
  AND Teacher.TeacherName = 'L. Okafor';
```

Modify

M. Costa needs a roll for **Stage Band** with each student's email. Change Query 2 so that it finds students by *activity name*, not ActivityID. (Hint: you need a third table and a second linking line.)

Make: answer the office's questions

Plan each query first with a **query design** (a design tool named in the Study Design), then write and test the SQL.

Office question	Tables needed	Fields shown	Condition (WHERE)	Tested? Number of rows
Example: Which activities run on Thursday?	Activity	ActivityName, Room	Day = 'Thursday'	Yes, 2 rows
1. Which activities are in room D04, and who teaches them?				
2. Which students are in Years 11 and 12?				
3. **Challenge:** How many students registered for each activity? (Research `COUNT` and `GROUP BY`.) Save the result for Lesson 5.				

Stretch. The same link can be written as `FROM Student JOIN Registration ON Student.StudentID = Registration.StudentID`. Rewrite Query 2 this way. Do the results match? Which version makes the link easier to see?

Test the rules

Our design has rules (validation). Predict whether each statement will be **accepted** or **rejected**, then run it. Record the results in your Lesson 5 test table.

```
INSERT INTO Registration (StudentID, ActivityID, RegistrationDate)
  VALUES ('S1419', 'A05', '2026-02-20');   -- Tomasz signs up for Chess again
INSERT INTO Student VALUES ('S1700', 'Test', 'Student', 13, 't.student@rvsc.example');
INSERT INTO Registration (StudentID, ActivityID, RegistrationDate)
  VALUES ('S9999', 'A01', '2026-02-20');   -- a student who does not exist
```

Which spreadsheet problem from Lesson 1 does each rule now prevent?

Extension: a Python interface

`r4-python-bridge.py` uses the standard sqlite3 steps: connect, create a cursor, execute, fetch, close. It prints the roll for any activity you type in, then saves the registration counts to **activity_counts.csv** for Lesson 5.

Its query uses a `?` placeholder instead of an f-string. Run `r4-injection-demo.py`, which builds the same query with an f-string, and type `x' OR '1'='1` as the activity name. How many rows come back, and why? This is SQL injection, and it is why we never put user input straight into SQL (Unit 2 Outcome 2).

Teacher answers (tested on the supplied database)

Task	Expected result
Query 1	2 rows: Robotics Club D04; Coding Club D04.
Query 2	7 rows: Amira Haddad, Jesse Whitcombe, Priya Raman, Lachlan Brooks, Noah Williams, Hamish Fraser, Oscar Tran.
Investigate 2	800 = 20 students × 40 registrations: every student paired with every sign-up (a Cartesian product). Missing join conditions are a documented novice SQL error (Miedema et al., 2021).
Investigate 3	Dan is in two of L. Okafor's activities (Debating, Chess Club). DISTINCT removes repeated rows; ask whether that is what the office wants.
Modify	FROM Student, Registration, Activity with two linking lines; AND Activity.ActivityName = 'Stage Band'. 5 rows: Jesse Whitcombe, Sofia Rossi, Isla Murphy, Ruby Clarke, Leila Karimi.
Make 1	Link Activity and Teacher; WHERE Room = 'D04'. Robotics Club (Tuesday, T. Nguyen); Coding Club (Thursday, T. Nguyen).
Make 2	`WHERE YearLevel >= 11` or `IN (11, 12)`: 7 students.
Make 3	GROUP BY ActivityName: Robotics Club 7; Choir, Debating, Junior Science Club 6; Chess Club, Coding Club, Stage Band 5.
Stretch	JOIN … ON returns the same 7 rows.
Tests	All three rejected: UNIQUE (StudentID, ActivityID) blocks the duplicate sign-up; CHECK (YearLevel BETWEEN 7 AND 12) blocks Year 13; the foreign key blocks S9999. These match the Lesson 1 duplicate, the 'YearLevel 71' value and orphan records.
Extension	The safe version returns 5 rows for Stage Band. With the f-string version, `x' OR '1'='1` returns 5,600 rows (20 × 40 × 7): the OR turns the WHERE clause true for every row of the three tables.

Scope. The unit sets SELECT, FROM and WHERE across related tables as the Unit 1 target, with tables linked in the WHERE clause. COUNT, GROUP BY and JOIN appear only as challenges. Data Analytics Unit 3 Outcome 1 returns to SQL with larger repositories.

Resource by Arvin Hay, part of the Ridgeview database unit: ridgeview.db, the schema script, two Python files and this worksheet. PRIMM structure from Sentance, Waite and Kallia (2019). DB Browser for SQLite: https://sqlitebrowser.org/