ANSWERS

A11.4 SQL: defining tables and joining them

Databases and big data · A level · OCR H446 1.3.2, AQA 7517 4.10.4, Eduqas A500QS 2.5 · about 60 min

What this lesson is about

CREATE TABLE with types and keys, ALTER and DROP, then SELECT with INNER JOIN, wildcards, aggregates, GROUP BY and nested queries.

- CREATE TABLE
- ALTER TABLE and DROP TABLE
- The club database
- Searching: LIKE, BETWEEN and IN
- INNER JOIN
- Aggregates and GROUP BY
- Nested SELECT

Questions

6 marks in all

1. Which SQL statement defines a new table? [1 mark]

- ☐ **A** CREATE TABLE
- ☐ **B** INSERT INTO
- ☐ **C** ALTER TABLE
- ☐ **D** SELECT INTO

> **Answer:** A. CREATE TABLE is data definition; INSERT adds records to a table that already exists.

2. Which condition matches names that start with D and have exactly three letters? [1 mark]

- ☐ **A** name LIKE 'D__'
- ☐ **B** name LIKE 'D%'
- ☐ **C** name LIKE '%D%'
- ☐ **D** name = 'D__'

> **Answer:** A. _ matches exactly one character, so D followed by two of them is three letters; % would match any length.

3. What does this print? [1 mark]

```
import sqlite3
db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE robots (robot_id INTEGER PRIMARY KEY, name TEXT)")
db.execute("CREATE TABLE runs (run_id INTEGER PRIMARY KEY, robot_id INTEGER, cm REAL)")
db.executemany("INSERT INTO robots VALUES (?, ?)", [(1, "Ada"), (2, "Bolt"), (3, "Cog")])
db.executemany("INSERT INTO runs VALUES (?, ?, ?)", [(1, 1, 42.0), (2, 2, 38.0), (3, 1,
80.0)])
for row in db.execute("""SELECT robots.name, COUNT(*), MAX(runs.cm)
                         FROM runs JOIN robots ON runs.robot_id = robots.robot_id
                         GROUP BY robots.name ORDER BY robots.name"""):
    print(*row)
```

Answer:

```
Ada 2 80.0
Bolt 1 38.0
```

The inner join groups the runs by robot. Cog has no runs, so an inner join leaves it out.

4. What does this print? [1 mark]

```
import sqlite3
db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE runs (run_id INTEGER PRIMARY KEY, task TEXT, cm REAL)")
db.executemany("INSERT INTO runs VALUES (?, ?, ?)", [(1, "wall", 42.0), (2, "square", 80.0),
(3, "wall", 45.5), (4, "square", 81.0)])
print(db.execute("SELECT run_id FROM runs WHERE cm = (SELECT MAX(cm) FROM runs WHERE task =
'wall')").fetchall())
```

Answer:

```
[(3,)]
```

The inner query finds the longest wall run, 45.5, and the outer query finds the run with that distance.

5. What is the difference between WHERE and HAVING? [1 mark]

- ☐ **A** WHERE filters records before grouping; HAVING filters the groups after GROUP BY
- ☐ **B** They are the same
- ☐ **C** HAVING filters records; WHERE filters groups
- ☐ **D** HAVING can only be used with ORDER BY

Answer: A. HAVING tests an aggregate, such as COUNT(*) >= 2, for each group.

6. What does DROP TABLE Booking do? [1 mark]

- ☐ **A** Deletes the Booking table and all its records
- ☐ **B** Deletes all records but keeps the empty table
- ☐ **C** Deletes the last record
- ☐ **D** Removes the primary key

Answer: A. DROP removes the structure itself; DELETE FROM removes records.

The task: build it, then join it

Build this design and query it. The lists give the records: `teams` holds `(team_id, team_name)`, `robots` holds `(robot_id, name, team_id)` and `runs` holds `(run_id, robot_id, task, cm)`, where ids are whole numbers, names and tasks are text and cm is a float. - Team (<u>team_id</u>, team_name) - Robot (<u>robot_id</u>, name, team_id*) - Run (<u>run_id</u>, robot_id*, task, cm) 1. `CREATE TABLE` all three, with a primary key on each and both foreign keys declared with `REFERENCES`. 2. Insert the records. 3. With **one** `SELECT` that joins all three tables with two `JOIN ... ON` clauses, uses `GROUP BY`, `COUNT` and `MAX`, and sorts by team name, print one line per team: `<team_name>: <number of runs> runs, longest <longest cm> cm`, for example `Owls: 2 runs, longest 76.5 cm`. The robot does not move.

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

import sqlite3

teams = [(1, "Hawks"), (2, "Owls")]
robots = [(1, "Ada", 1), (2, "Bolt", 1), (3, "Cog", 2), (4, "Dot", 2)]
runs = [(1, 1, "wall", 42.0), (2, 1, "square", 80.0), (3, 2, "wall", 38.0), (4, 1, "wall",
45.5),
        (5, 3, "square", 76.5), (6, 2, "square", 81.0), (7, 4, "wall", 51.0)]

db = sqlite3.connect(":memory:")
```

The hint students can ask for: Create the parent tables before the table that refers to them. The query has to travel from runs to robots to teams, so it needs two joins, each matching a foreign key to the primary key it refers to. Then group by team.

A solution

```
# the two lines every program starts with: the commands, then the robot
from bugbot import *
connect()

import sqlite3

teams = [(1, "Hawks"), (2, "Owls")]
robots = [(1, "Ada", 1), (2, "Bolt", 1), (3, "Cog", 2), (4, "Dot", 2)]
runs = [(1, 1, "wall", 42.0), (2, 1, "square", 80.0), (3, 2, "wall", 38.0), (4, 1, "wall",
45.5),
        (5, 3, "square", 76.5), (6, 2, "square", 81.0), (7, 4, "wall", 51.0)]

db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE teams (team_id INTEGER PRIMARY KEY, team_name VARCHAR(20) NOT
NULL)")
db.execute("""CREATE TABLE robots (robot_id INTEGER PRIMARY KEY, name VARCHAR(20) NOT NULL,
              team_id INTEGER REFERENCES teams(team_id))""")
db.execute("""CREATE TABLE runs (run_id INTEGER PRIMARY KEY, robot_id INTEGER REFERENCES
robots(robot_id),
              task VARCHAR(10), cm REAL)""")
db.executemany("INSERT INTO teams VALUES (?, ?)", teams)
db.executemany("INSERT INTO robots VALUES (?, ?, ?)", robots)
db.executemany("INSERT INTO runs VALUES (?, ?, ?, ?)", runs)

query = """SELECT teams.team_name, COUNT(*), MAX(runs.cm)
```

```
            FROM runs
            JOIN robots ON runs.robot_id = robots.robot_id
            JOIN teams ON robots.team_id = teams.team_id
            GROUP BY teams.team_name
            ORDER BY teams.team_name"""
for name, n, longest in db.execute(query):
    print(f"{name}: {n} runs, longest {longest} cm")
```

Any program that meets the task's checks is marked correct in the simulator; this is one way, not the only way.