ANSWERS

A11.5 SQL: changing data and referential integrity

Databases and big data · A level · OCR H446 1.3.2, AQA 7517 4.10.2, Eduqas A500QS 2.5 · about 55 min

What this lesson is about

INSERT, UPDATE and DELETE at A level, and how a database refuses orphan records. The robot logs its own moves.

- INSERT
- UPDATE
- DELETE
- Referential integrity
- The robot logs its own runs

Questions

5 marks in all

1. What does referential integrity mean? [1 mark]

- ☐ **A** Every foreign key value matches a primary key value in the table it refers to
- ☐ **B** Every table has a primary key
- ☐ **C** No two records are the same
- ☐ **D** Every field has a value

> **Answer:** A. A foreign key that matches nothing is an orphan, which referential integrity forbids.

2. Runs refer to robots with ON DELETE CASCADE. What happens when robot 2 is deleted? [1 mark]

- ☐ **A** Robot 2's runs are deleted too
- ☐ **B** The delete is refused
- ☐ **C** Robot 2's runs keep robot_id 2
- ☐ **D** Robot 2's runs get robot_id null

> **Answer:** A. Cascade carries the delete on to the child records; restrict would refuse it.

3. What does this print? [1 mark]

```
import sqlite3
db = sqlite3.connect(":memory:")
db.execute("PRAGMA foreign_keys = ON")
db.execute("CREATE TABLE robots (robot_id INTEGER PRIMARY KEY)")
db.execute("CREATE TABLE runs (run_id INTEGER PRIMARY KEY, robot_id INTEGER REFERENCES
robots(robot_id))")
db.execute("INSERT INTO robots VALUES (1)")
db.execute("INSERT INTO runs (robot_id) VALUES (1)")
try:
    db.execute("INSERT INTO runs (robot_id) VALUES (5)")
    print("added")
except sqlite3.IntegrityError:
    print("refused")
print(db.execute("SELECT COUNT(*) FROM runs").fetchone()[0])
```

Answer:

```
refused
1
```

Robot 5 does not exist, so the second run is refused and only one run is stored.

4. What does this print? [1 mark]

```
import sqlite3
db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE runs (run_id INTEGER PRIMARY KEY, robot_id INTEGER, cm REAL)")
db.executemany("INSERT INTO runs VALUES (?, ?, ?)", [(1, 1, 40.0), (2, 2, 38.0), (3, 1,
50.0)])
db.execute("UPDATE runs SET cm = cm + 5 WHERE robot_id = 1")
db.execute("DELETE FROM runs WHERE cm < 45")
print(db.execute("SELECT run_id, cm FROM runs").fetchall())
```

Answer:

```
[(1, 45.0), (3, 55.0)]
```

Robot 1's runs become 45.0 and 55.0; then run 2, at 38.0, is the only one under 45.

5. Why should a program insert values with ? placeholders rather than joining them into the SQL string? [1 mark]

- ☐ **A** The values are passed separately, so typed text cannot change the SQL (SQL injection)
- ☐ **B** Placeholders make the query run without a database
- ☐ **C** Placeholders are required for numbers
- ☐ **D** Joining strings is not allowed in Python

Answer: A. With placeholders, a value is always treated as data, never as part of the statement.

The task: log the legs, keep the links

The starter builds two tables, `robots` and `runs`. `runs.robot_id` refers to `robots(robot_id)` with `ON DELETE CASCADE`. Robot 1 is Ada (the robot on the mat) and robot 2 is Bolt, who already has run 1. `legs` is a list of `(move, cm)`, where move is `"forward"`, `"right"` or `"backward"` and cm is a whole number of centimetres. 1. Switch foreign key checking on. 2. For each leg in order, drive it at speed 40, measure how far the robot really moved from `position()` before and after (the straight-line distance, rounded to a whole number), and `INSERT` it as a run for robot 1 with a field list and `?` placeholders, letting the database number it. 3. Try to insert a run for robot 9. When the database refuses with `sqlite3.IntegrityError`, print `refused: robot 9 does not exist`. 4. `UPDATE` the runs whose move is `right` so their move is `strafe`. 5. `DELETE` Bolt from `robots`. 6. With a query that joins `runs` to `robots`, print every run in run order as `run <run_id> <name> <move> <cm>`, then print `runs: <n>`, the number of records left in `runs`. Five lines in all, starting with the refusal.

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

import sqlite3

db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE robots (robot_id INTEGER PRIMARY KEY, name TEXT NOT NULL)")
db.execute("""CREATE TABLE runs (run_id INTEGER PRIMARY KEY,
              robot_id INTEGER NOT NULL REFERENCES robots(robot_id) ON DELETE CASCADE,
              move TEXT, cm INTEGER)""")
db.executemany("INSERT INTO robots VALUES (?, ?)", [(1, "Ada"), (2, "Bolt")])
db.execute("INSERT INTO runs VALUES (1, 2, 'forward', 30)")
db.commit()

legs = [("forward", 20), ("right", 15), ("backward", 10)]
```

The hint students can ask for: Switch the checking on before anything else touches the database. Measure each leg from position() before and after it. The refusal and the vanishing of Bolt's run should both come from the database, not from if statements of your own.

A solution

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

import sqlite3

db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE robots (robot_id INTEGER PRIMARY KEY, name TEXT NOT NULL)")
db.execute("""CREATE TABLE runs (run_id INTEGER PRIMARY KEY,
              robot_id INTEGER NOT NULL REFERENCES robots(robot_id) ON DELETE CASCADE,
              move TEXT, cm INTEGER)""")
db.executemany("INSERT INTO robots VALUES (?, ?)", [(1, "Ada"), (2, "Bolt")])
db.execute("INSERT INTO runs VALUES (1, 2, 'forward', 30)")
db.commit()

legs = [("forward", 20), ("right", 15), ("backward", 10)]
db.execute("PRAGMA foreign_keys = ON")
moves = {"forward": forward, "right": right, "backward": backward}
```

```
for move, cm in legs:
    x0, y0 = position()
    moves[move](40, distance=cm)
    x1, y1 = position()
    moved = round(((x1 - x0) ** 2 + (y1 - y0) ** 2) ** 0.5)
    db.execute("INSERT INTO runs (robot_id, move, cm) VALUES (?, ?, ?)", (1, move, moved))

try:
    db.execute("INSERT INTO runs (robot_id, move, cm) VALUES (?, ?, ?)", (9, "forward", 5))
except sqlite3.IntegrityError:
    print("refused: robot 9 does not exist")

db.execute("UPDATE runs SET move = 'strafe' WHERE move = 'right'")
db.execute("DELETE FROM robots WHERE robot_id = 2")

for run_id, name, move, cm in db.execute("""SELECT runs.run_id, robots.name, runs.move,
runs.cm
                                            FROM runs JOIN robots ON runs.robot_id =
robots.robot_id
                                            ORDER BY runs.run_id"""):
    print("run", run_id, name, move, cm)
print("runs:", db.execute("SELECT COUNT(*) FROM runs").fetchone()[0])
```

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