ANSWERS

A11.2 Relational databases and keys

Databases and big data · A level · OCR H446 1.3.2, AQA 7517 4.10.2, Eduqas A500QS 2.4 · about 55 min

What this lesson is about

Flat files against relational databases, and every kind of key: primary, composite, foreign and secondary, with indexes and how records are found.

- Flat files and relational databases
- The vocabulary
- Keys
- Indexes
- How records are found in a file

Questions

6 marks in all

1. What is a composite primary key? [1 mark]

- ☐ **A** A primary key made of two or more attributes that together are unique
- ☐ **B** A primary key that is also a foreign key
- ☐ **C** A key that is indexed for searching
- ☐ **D** A key made by joining two tables

> **Answer:** A. No single attribute is unique, but the combination is, such as (RobotID, AttemptNo).

2. Which describes a secondary key? [1 mark]

- ☐ **A** An attribute other than the primary key that is indexed so records can be searched or sorted by it quickly
- ☐ **B** The second attribute of a composite key
- ☐ **C** A backup copy of the primary key
- ☐ **D** A foreign key in a link table

> **Answer:** A. A secondary key is indexed for fast searching and need not be unique.

3. Which are disadvantages of a flat file database compared with a relational database? [1 mark]

Tick every answer that is true.

- ☐ **A** The same facts are stored many times (data redundancy)
- ☐ **B** A missed edit can leave the data inconsistent
- ☐ **C** It needs a DBMS to be set up
- ☐ **D** It cannot be opened by a spreadsheet

> **Answer:** A, B. Repeating facts wastes space and invites inconsistency. A flat file is simpler to set up, not harder.

4. Adding an index on the task field of a runs table will usually... [1 mark]

- ☐ **A** make searches by task faster, but take more storage and slow down inserts, updates and deletes
- ☐ **B** make every operation on the table faster
- ☐ **C** stop two records having the same task
- ☐ **D** remove the need for a primary key

> **Answer:** A. The index must be kept up to date on every change, which costs time and space.

5. Records are stored at an address calculated from their key. Which file organisation is this? [1 mark]

- ☐ **A** Direct (hashed) access
- ☐ **B** Serial
- ☐ **C** Sequential
- ☐ **D** Indexed sequential

> **Answer:** A. A hash function turns the key into an address, so one record can be read without searching.

6. What does this print? [1 mark]

```
SLOTS = 7
table = [None] * SLOTS
for key in [9, 16, 4]:
    address = key % SLOTS
    while table[address] is not None:
        address = (address + 1) % SLOTS
    table[address] = key
print(table)
```

> **Answer:**
>
> ```
> [None, None, 9, 16, 4, None, None]
> ```
>
> 9 and 16 both hash to 2, so 16 moves on to 3; 4 hashes to 4.

The task: a composite key

A robot may make several attempts at a task, numbered from 1 for each robot. The list `attempts` holds records `(robot_id, attempt, cm)`: two whole numbers and a float. 1. Create a table `attempts` with the fields `robot_id`, `attempt` and `cm`, and the composite primary key (`robot_id`, `attempt`). 2. Insert every record, in order. For each one print `added robot <robot_id> attempt <attempt>`, or, if the database refuses it with `sqlite3.IntegrityError`, print `rejected robot <robot_id> attempt <attempt>: already logged`. 3. Use `SELECT COUNT(*)` to print the number of records now in the table as `records: <n>`. Seven lines in all. The robot does not move.

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

import sqlite3

attempts = [(1, 1, 42.0), (1, 2, 45.5), (2, 1, 38.0), (1, 2, 44.0), (3, 1, 76.5), (2, 1,
39.0)]
db = sqlite3.connect(":memory:")
```

The hint students can ask for: Neither field is unique on its own, but the pair is. Declare that pair as the key when you create the table, then try every insert and let the database tell you which ones break the key.

A solution

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

import sqlite3

attempts = [(1, 1, 42.0), (1, 2, 45.5), (2, 1, 38.0), (1, 2, 44.0), (3, 1, 76.5), (2, 1,
39.0)]
db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE attempts (robot_id INTEGER, attempt INTEGER, cm REAL, PRIMARY KEY
(robot_id, attempt))")

for robot_id, attempt, cm in attempts:
    try:
        db.execute("INSERT INTO attempts VALUES (?, ?, ?)", (robot_id, attempt, cm))
        print(f"added robot {robot_id} attempt {attempt}")
    except sqlite3.IntegrityError:
        print(f"rejected robot {robot_id} attempt {attempt}: already logged")

count = db.execute("SELECT COUNT(*) FROM attempts").fetchone()[0]
print("records:", count)
```

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