ANSWERS

A11.6 Transactions and ACID

Databases and big data · A level · OCR H446 1.3.2, AQA 7517 4.10.5, Eduqas A500QS 2.5 · about 55 min

What this lesson is about

A transaction is all or nothing: atomicity, consistency, isolation and durability, commit and rollback, and redundancy.

- What a transaction is
- Half a change
- ACID
- Consistency: rules the database enforces
- A transaction around a physical action
- Record locking and redundancy

Questions

5 marks in all

1. Which ACID property means a transaction happens completely or not at all? [1 mark]

- ☐ **A** Atomicity
- ☐ **B** Consistency
- ☐ **C** Isolation
- ☐ **D** Durability

> **Answer:** A. Atomicity: if any part fails, every change in the transaction is rolled back.

2. After COMMIT, a power cut hits the server. Which property guarantees the change is still there? [1 mark]

- ☐ **A** Durability
- ☐ **B** Atomicity
- ☐ **C** Isolation
- ☐ **D** Consistency

> **Answer:** A. Durability: committed changes are already in non-volatile storage and the log.

3. Two transactions run at the same time, and neither sees the other's uncommitted changes. Which property is this? [1 mark]

- ☐ **A** Isolation
- ☐ **B** Durability
- ☐ **C** Atomicity
- ☐ **D** Redundancy

> **Answer:** A. Isolation makes concurrent transactions give the same result as running them one after another.

4. What does this print? [1 mark]

```
import sqlite3
db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE zones (name TEXT PRIMARY KEY, balls INTEGER CHECK (balls >= 0))")
db.executemany("INSERT INTO zones VALUES (?, ?)", [("red", 1), ("blue", 4)])
db.commit()
try:
    with db:
        db.execute("UPDATE zones SET balls = balls + 3 WHERE name = 'blue'")
        db.execute("UPDATE zones SET balls = balls - 3 WHERE name = 'red'")
except sqlite3.IntegrityError:
    print("rolled back")
print(db.execute("SELECT * FROM zones ORDER BY name").fetchall())
```

Answer:

```
rolled back
[('blue', 4), ('red', 1)]
```

Red would go below 0, so the CHECK fails and the transaction rolls back, undoing blue's +3 as well.

5. In OCR's transaction processing topic, what does redundancy refer to? [1 mark]

- ☐ **A** Keeping extra copies, such as mirrored disks or replica servers, so a failure loses no data or service
- ☐ **B** Storing the same fact many times in one table
- ☐ **C** Deleting data that is no longer needed
- ☐ **D** Records that have no primary key

Answer: A. This is deliberate duplication for resilience, not the data redundancy that normalisation removes.

The task: all or nothing

The table `zones` holds each zone's `name` (text, the primary key) and its number of `balls` (a whole number), with the rule `CHECK (balls >= 0)`. It starts as red 3, blue 2, green 0. `moves` is a list of `(src, dst, n)`: two zone names and a whole number of balls. 1. Write `move(src, dst, n)` that, as **one transaction**, first adds `n` to `dst` with `UPDATE zones SET balls = balls + ?` and then takes `n` from `src`. If the database refuses the change with `sqlite3.IntegrityError`, the whole move must be undone: neither zone changes. Print `moved <n> <src> -> <dst>` if it worked, or `refused <n> <src> -> <dst>` if not. Do not test the numbers with an `if` of your own: let the `CHECK` rule refuse. 2. Call `move` for each item of `moves`, in order. 3. Print every zone in name order as `<name> <balls>`, then `total <sum>` using `SUM`. Eight 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

db = sqlite3.connect(":memory:")
db.execute("CREATE TABLE zones (name TEXT PRIMARY KEY, balls INTEGER NOT NULL CHECK (balls
>= 0))")
db.executemany("INSERT INTO zones VALUES (?, ?)", [("red", 3), ("blue", 2), ("green", 0)])
db.commit()

moves = [("red", "blue", 2), ("blue", "green", 5), ("blue", "green", 3), ("red", "green",
2)]
```

The hint students can ask for: Do the increase first and the decrease second, as the task says, so a refused move has already changed one row by the time it fails. The only way the totals stay right is if that half-done change is undone.

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 zones (name TEXT PRIMARY KEY, balls INTEGER NOT NULL CHECK (balls
>= 0))")
db.executemany("INSERT INTO zones VALUES (?, ?)", [("red", 3), ("blue", 2), ("green", 0)])
db.commit()

moves = [("red", "blue", 2), ("blue", "green", 5), ("blue", "green", 3), ("red", "green",
2)]

def move(src, dst, n):
    try:
        with db:
            db.execute("UPDATE zones SET balls = balls + ? WHERE name = ?", (n, dst))
            db.execute("UPDATE zones SET balls = balls - ? WHERE name = ?", (n, src))
        print(f"moved {n} {src} -> {dst}")
    except sqlite3.IntegrityError:
        print(f"refused {n} {src} -> {dst}")
```

```
for src, dst, n in moves:
    move(src, dst, n)

for name, balls in db.execute("SELECT name, balls FROM zones ORDER BY name"):
    print(name, balls)
print("total", db.execute("SELECT SUM(balls) FROM zones").fetchone()[0])
```

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