Databases & Web Programming

Sample Paper 1 — Mark Scheme & Model Answers

Model answers below are not the only correct ones — alternative phrasings and equivalent SQL/JDL that satisfy the requirements receive full marks. Green italic notes are study tips, not part of the answer.

Question 1 — Databases (30 marks)

1(a)(i) — Anomalies [4 marks]

Any two of the following, with the scenario applied. **1 mark** for naming, **1 mark** for the scenario link.

• **Update anomaly:** the same customer phone number is stored on many rows. If only some rows are updated, the database holds contradictory phone numbers for one customer.
• **Deletion anomaly:** deleting the last order belonging to a customer also removes the only record of that customer's contact details.
• **Insertion anomaly:** a new customer cannot be added until they place an order (no row to put them on).

1(a)(ii) — Constraints [6 marks]

Three of the below, each **2 marks** (1 for the constraint, 1 for the column + reason):

• **PRIMARY KEY** on `customer_id` — uniquely identifies each customer.
• **NOT NULL** on `phone_number` — every customer must be contactable.
• **FOREIGN KEY** on `order.customer_id` — links each order to a real customer; prevents orphan orders.
• **CHECK** on `unit_price >= 0` — prevents negative prices.
• **DEFAULT 0** on `quantity` — sensible value if not supplied.

■ *Examiner tip: the comma-separated product cell also tells you the table isn't even in 1NF. Mentioning that is a free bonus.*

1(b)(i) — 3NF Decomposition [8 marks]

Four tables. **2 marks** per correct table, including PK/FK marking.

```
Student(StudentID PK, StudentName)
Module(ModuleCode PK, ModuleTitle, LecturerID FK)
Lecturer(LecturerID PK, LecturerName, LecturerOffice)
Enrolment(StudentID PK FK, ModuleCode PK FK, Grade)
```

■ *Equivalent: keep* `Lecturer` *name on* `Module` *and have a separate* `LecturerOffice` *table — as long as the transitive dependency Lecturer→Office is broken.*

1(b)(ii) — Transitive dependency [2 marks]

`StudentID,ModuleCode` → `Lecturer` → `LecturerOffice`. `LecturerOffice` depends on a non-key attribute (Lecturer), not on the table's primary key. That's a transitive dependency, which 3NF forbids.

1(c)(i) — CREATE TABLE [4 marks]

```
CREATE TABLE Streams (
    StreamID         INT PRIMARY KEY,
    UserID           INT NOT NULL,
    ShowTitle        VARCHAR(150) NOT NULL,
    MinutesWatched   INT DEFAULT 0,
    StreamDate       DATE NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
```

```
);
```

Marking: 1 PK, 1 FK, 1 DEFAULT 0, 1 overall syntax (commas, semicolon, NOT NULL where appropriate).

1(c)(ii) — JOIN + GROUP BY + HAVING [6 marks]

```
SELECT u.FullName,
       SUM(s.MinutesWatched) AS TotalMinutes
FROM   Users   u
JOIN   Streams s ON s.UserID = u.UserID
GROUP BY u.UserID, u.FullName
HAVING SUM(s.MinutesWatched) > 500
ORDER BY TotalMinutes DESC;
```

Marking: JOIN (1), GROUP BY (1), aggregate SUM (1), HAVING filter (1), ORDER BY DESC (1), correct alias / column selection (1).

Question 2 — WP Principles (30 marks)

2(a) — REST violations [9 marks]

Three violations, **3 marks** each (issue named + fixed endpoint):

1. Verbs in URLs. URLs should be *nouns*; the HTTP method is the verb. `getBookById` repeats GET.
Fix: `GET /api/books/42`

2. Wrong HTTP method. Deleting a resource must use DELETE, not POST.
Fix: `DELETE /api/books/42`

3. Non-plural / inconsistent resource name. Resources should be plural nouns; `books_all` mixes snake_case and is redundant (the collection endpoint already implies 'all books').
Fix: `GET /api/books`

■ *Other valid issues a student might raise: identifier as query param vs path variable (use a path variable for a unique resource), missing API versioning (`/api/v1/...`), or kebab-case vs snake_case inconsistency.*

2(b)(i) — REST vs CGI [4 marks]

REST is **stateless**: each request carries all the information the server needs, so any server in a load-balanced fleet can handle it (1 mark) and the system scales horizontally (1 mark). CGI-bin spawns a new **process per request**, which is heavy and doesn't scale under load (1 mark). REST also separates frontend and backend, allowing them to be deployed and scaled independently (1 mark).

2(b)(ii) — Derived query method [8 marks]

(1) **Returns** a list of `Book` entities whose `author` field equals the given string AND whose `publishedYear` is strictly greater than the given year (3 marks).
(2) Spring Data JPA **parses the method name**: tokens like `findBy`, `And`, `GreaterThan` map to a JPQL/SQL query, and the interface is implemented at runtime by a generated proxy. No hand-written SQL is needed (3 marks).
(3) Use `@Table(name = "library_book")` on the `Book` class (in addition to `@Entity`) (2 marks).

2(c)(i) — AuthN vs AuthZ [4 marks]

Authentication — verifying the user's identity ("are you really Sarah?"), typically via username + password producing a token (2 marks).
Authorization — deciding what an already-authenticated user is allowed to do ("Sarah is a MEMBER, so she cannot delete books") (2 marks).

2(c)(ii) — 401 vs 403 [5 marks]

When the member is rejected because they lack the LIBRARIAN role, the server returns **403 Forbidden** (2 marks) — the request is authenticated but not authorised. When the request has no token at all, the server returns **401 Unauthorized** (2 marks) — the user has not authenticated. The key difference: 401 means "we don't know who you are"; 403 means "we know who you are and you still can't do this" (1 mark).

Question 3 — WP Practice (40 marks)

3(a) — JDL Modelling [15 marks]

```
entity Album {
    title String required maxlength(100),
    releaseDate LocalDate,
    coverArt ImageBlob
}

entity Track {
    title String required,
    durationSeconds Integer min(1),
    plays Long
}

relationship OneToMany {
    Album{tracks} to Track{album}
}

relationship ManyToOne {
    Album{uploader} to User with builtInEntity
}
```

Marking: Album entity correct (3), Track entity correct (3), required + maxlength on title (2), min on durationSeconds (1), correct types incl. ImageBlob/Long/LocalDate (2), OneToMany Album-Track (2), ManyToOne to built-in User (2).

■ *Common slip: writing* `relationship OneToMany { Album to Track }` *without the navigation names. Examiners usually accept it but lose 1 mark for not naming the side that holds the relationship.*

3(b)(i) — PATCH cURL [5 marks]

```
curl -X PATCH http://localhost:8080/api/tracks/88 \
     -H "Content-Type: application/merge-patch+json" \
     -H "Authorization: Bearer <TOKEN>" \
     -d '{ "plays": 1500 }'
```

Marking: `-X PATCH` (1), correct URL with id (1), Content-Type header for JSON / merge-patch (1), Authorization Bearer header (1), JSON body with the right field (1). JHipster uses `application/merge-patch+json`; plain `application/json` is also accepted.

3(b)(ii) — 404 vs 401 [5 marks]

Missing track → **404 Not Found** (2): the resource doesn't exist.
Unauthenticated → **401 Unauthorized** (2): no valid credentials supplied.
Difference (1): 404 is about whether the *resource* exists; 401 is about whether the *caller* is identified.

3(c)(i) — @RequestMapping for v2 [5 marks]

```
@RestController
@RequestMapping("/api/v2")
public class TrackResourceV2 {
    @GetMapping("/tracks/{id}")
    public ResponseEntity<TrackDTO> getTrack(@PathVariable Long id) { ... }
}
```

Marking: class-level `@RequestMapping("/api/v2")` (3), renaming the class / keeping v1 untouched (1), correct path variable usage (1). An equally valid answer is method-level mapping `@GetMapping("/v2/tracks/{id}")`.

3(c)(ii) — Deprecation strategy [5 marks]

Name (1 mark): deprecation period (also accepted: sunset period, grace period).
Two strategies, 2 marks each:

• **Deprecation HTTP headers** — v1 responses include `Deprecation: true` and `Sunset: <date>`, plus a `Link` header pointing to v2 docs.
• **Direct developer comms** — email registered API consumers, post on the developer portal, surface a banner in the dashboard, log a server-side warning every time v1 is hit.
• **Phased throttling / soft errors** — reduce v1's rate limits as the sunset date approaches; return synthetic 5xx on selected dates as a “brownout”.

3(d)(i) — Owner-only delete [5 marks]

```
@DeleteMapping("/albums/{id}")
public ResponseEntity<Void> deleteAlbum(@PathVariable Long id) {
    String login = SecurityUtils.getCurrentUserLogin()
        .orElseThrow(() -> new AccessDeniedException("Not authenticated"));
    Album album = albumRepository.findById(id)
        .orElseThrow(() -> new ResourceNotFoundException("Album not found"));
    if (!album.getUploader().getLogin().equals(login)) {
        return ResponseEntity.status(HttpStatus.FORBIDDEN).build();
    }
    albumRepository.deleteById(id);
    return ResponseEntity.noContent().build();
}
```

Marking: getting current user login via `SecurityUtils` (2), loading the Album from repository (1), ownership check (1), returning 403 on mismatch (1). 204 No Content on success is a nice extra.

— End of Mark Scheme —