

2024 CpSc Q14

Section: Database Design and Development

Topic: SQL — Joins, GROUP BY, Aggregates, Subqueries

Question Summary

- (a) Select bookings for pupils with instructors where `hourlyRate > 35`, joining `Instructor`, `Pupil` and `Booking` appropriately.
- (b) (i) Produce a totals report grouped by town and sorted by town and count.
- (b) (ii) Explain why `GROUP BY town` is required.
- (c) Find the average hourly rate for instructors whose day off is Saturday or Sunday; include an alias.
- (d) Find the instructor(s) with the minimum (cheapest) hourly rate using a subquery (or a saved query used within another).

Worked Solution

(a) Join the three tables on their keys and apply the rate filter. Using equi-joins:

```
SELECT *
```

```
FROM Instructor, Pupil, Booking
```

```
WHERE hourlyRate > 35
```

```
AND Instructor.instructorID = Booking.instructorID
```

```
AND Booking.pupilRef = Pupil.pupilRef;
```

Note: A NATURAL JOIN or explicit INNER JOIN ... ON ... gets equivalent credit.

(b)(i) 'Totals' implies grouping and counting per town with a doubly sorted output:

```
SELECT town, COUNT(*) AS [Number Per Town]
FROM Pupil
GROUP BY town
ORDER BY town ASC, [Number Per Town] ASC;
```

(b)(ii) GROUP BY town is required because the SELECT includes a non-aggregate field (town) and we need one output row per town; grouping produces one result for each town.

(c) Compute the average hourly rate, using an alias, and restrict to weekend days off:

```
SELECT AVG(hourlyRate) AS [Average Hourly Rate]
FROM Instructor
WHERE dayOff = 'Saturday' OR dayOff = 'Sunday';
Alternative pattern: WHERE dayOff LIKE 'S%'.
```

(d) Use a subquery to locate the minimum hourlyRate, then select the instructor(s) who match it:

```
SELECT *
FROM Instructor
WHERE hourlyRate = (
    SELECT MIN(hourlyRate) FROM Instructor
);
```

Equivalent: first query to find MIN, then reference that query (or a saved view) in a second query.

Final Answer

Final Answer

(a) Equi-join (or natural/INNER JOIN) across

Instructor-Booking-Pupil with condition `hourlyRate > 35`.

(b)(i) `SELECT town, COUNT(*) AS [Number Per Town] FROM Pupil GROUP BY town ORDER BY town, [Number Per Town];`

(b)(ii) `GROUP BY town` ensures one result row per town since town is non-aggregate.

(c) `SELECT AVG(hourlyRate) AS [Average Hourly Rate] FROM Instructor WHERE dayOff='Saturday' OR dayOff='Sunday';`

(d) `SELECT * FROM Instructor WHERE hourlyRate = (SELECT MIN(hourlyRate) FROM Instructor);`

Revision Tips

- When non-aggregate fields appear in `SELECT` with aggregates, they must be listed in `GROUP BY`.
- Alias calculated columns for easier `ORDER BY`, e.g. `[Number Per Town]`.
- Prefer explicit `JOIN ... ON ...` for clarity, but older equi-join notation is acceptable.
- Subqueries are ideal for 'cheapest', 'highest', or 'latest' queries that compare against an aggregate.

Exam Alignment

Exam Alignment

Matches SQA 2024 MI for Q14: (a) correct tables and joins with `hourlyRate > 35`; (b)(i) totals with doubly sorted output; (b)(ii) `GROUP BY` justification; (c) `AVG` on `hourlyRate` with alias and weekend criteria; (d) `MIN` via subquery or saved query.