

0.5 Query Writing Intermediate

you-said

University Schema and Data.pdf
PDF

Write an SQL query to "Display the list of all course sections offered in Spring 2018, along with the ID and name of each instructor teaching the section. If a section has more than one instructor, that section should appear as many times in the result as it has instructors. If a section does not have any instructor, it should still appear in the result with the instructor name set to "-". For this University Schema and Data

ChatGPT said:

Here is an SQL query to display the list of all course sections offered in Spring 2018 along with the instructor details. If a section has multiple instructors, it will appear multiple times, and if no instructor is assigned, the name will be displayed as "-":

```

[] sqlCopy codeSELECT s.course_id, s.sec_id, s.semester, s.year, COALESCE(i.ID,
'-') AS instructor_id, COALESCE(i.name, '-') AS instructor_name FROM section s
LEFT JOIN teaches t ON s.course_id = t.course_id AND s.sec_id = t.sec_id AND
s.semester = t.semester AND s.year = t.year LEFT JOIN instructor i ON t.ID = i.ID
WHERE s.semester = 'Spring' AND s.year = 2018;

```

explanation

Explanation:

- LEFT JOIN ensures that sections without an instructor still appear in the results.
- COALESCE(i.ID, '-') and COALESCE(i.name, '-') handle sections with no instructor by showing "-" in place of NULL.
- The query filters by s.semester = 'Spring' and s.year = 2018 to retrieve the relevant sections.

you-said-1

What is the difference between above query and this: select course_id, sec_id, ID, decode(name, null, '-', name) as name from (section natural left outer join teaches) natural left outer join instructor where semester= 'Spring' and year= 2018;

ChatGPT said:

There are a few key differences between the first SQL query and the new one you provided:

join-methods

1. Join Methods:1. Join Methods:

- **First Query:**

- Uses explicit `LEFT JOIN` clauses to join `section`, `teaches`, and `instructor` tables.
- The join conditions are explicitly defined, specifying the exact columns to match on for each `LEFT JOIN`.
- This method offers more control over how the joins are executed and allows for handling specific conditions or adjustments in the join logic.

- **Second Query:**

- Uses `NATURAL LEFT OUTER JOIN`, which implicitly joins tables based on columns that share the same name in both tables.
- The `NATURAL JOIN` doesn't require specifying the join condition explicitly, and it automatically matches columns with identical names.
- This can sometimes lead to unintended joins if columns with the same name exist but are not logically related.

handling-of-missing-data-instructor-name

2. Handling of Missing Data (Instructor Name):2. Handling of Missing Data (Instructor Name):

- **First Query:**

- Uses `COALESCE(i.ID, '-')` and `COALESCE(i.name, '-')` to handle `NULL` values, explicitly converting missing data to `"-"`.
- This is more flexible as it can handle both `ID` and `name` with a custom value for each.

- **Second Query:**

- Uses `DECODE(name, null, '-', name)` to replace NULL instructor names with "-".
- `DECODE` is more limited compared to `COALESCE`, as it is specific to one column (`name`) and only substitutes for NULL values in that column.

clarity-and-readability

3. Clarity and Readability:3. Clarity and Readability:

- **First Query:**

- The joins and `COALESCE` function are more explicit and easier to understand for someone reading the query, as they can clearly see the relationships and how NULL values are handled.

- **Second Query:**

- While shorter, the use of `NATURAL LEFT OUTER JOIN` and `DECODE` can obscure exactly which columns are being joined and how missing data is handled, making the logic less transparent.

join-flexibility

4. Join Flexibility:4. Join Flexibility:

- **First Query:**

- Since you explicitly define how to join the tables, it allows more customization, for example, adding more complex join conditions or handling ambiguous column names more easily.

- **Second Query:**

- The use of `NATURAL JOIN` limits flexibility, especially when there are multiple columns with the same name that should not be joined. It is more suitable for situations where the schema is simple and there are no column name conflicts.

conclusion

Conclusion:

- The **first query** offers greater control, clarity, and flexibility, especially in complex schemas where column names might overlap or require more specific handling.
- The **second query** is more concise but potentially less robust due to the automatic nature of `NATURAL JOIN`, and it lacks the flexibility to handle complex cases.

you-said-2

University Schema and Data.pdf
PDF

Write an SQL query to "Display the list of all departments, with the total number of instructors in each department, without using subqueries. Make sure to show departments that have no instructors, and list those departments with an instructor count of zero." Use the schema and data provided in the pdf.

section

Here's an SQL query that lists all departments along with the total number of instructors in each department. Departments with no instructors will be shown with an instructor count of zero, without using subqueries:

```
sqlCopy codeSELECT d.dept_name, COUNT(i.ID) AS instructor_count FROM
department d LEFT JOIN instructor i ON d.dept_name = i.dept_name GROUP BY
d.dept_name;
```

explanation-1

Explanation:

- `LEFT JOIN` ensures that departments with no instructors are included in the result.
- `COUNT(i.ID)` counts the number of instructors in each department. For departments without instructors, `COUNT(i.ID)` will return 0 because of the `LEFT JOIN`.
- `GROUP BY d.dept_name` groups the results by department name to ensure each department appears once in the result set.

you-said-3

University Schema and Data.pdf
PDF

Write a query to define the view `student_grades` (ID, GPA) giving the grade-point average of each student and relation grade points(grade, points) to get the numeric points associated with a letter grade. Make sure your view definition correctly handles the case of null values for the grade attribute of the `takes` relation.

section-1

Here's an SQL query to create a view `student_grades` that calculates the grade-point average (GPA) for each student. This view handles `NULL` values in the `grade` column of the `takes` table by excluding them from the GPA calculation:

```
[] sqlCopy codeCREATE VIEW student_grades AS SELECT  s.ID, COALESCE(AVG(gp.points),
0) AS GPA FROM    student s LEFT JOIN    takes t ON s.ID = t.ID LEFT JOIN
grade_points gp ON t.grade = gp.grade WHERE    t.grade IS NOT NULL -- Exclude
null grades from GPA calculation GROUP BY    s.ID;
```

explanation-2

Explanation:

- `LEFT JOIN on takes and grade_points` ensures that all students are included, even if they have not taken any courses or received any grades.
- `COALESCE(AVG(gp.points), 0)` is used to return a GPA of 0 for students who have no grades (i.e., when the result of the `AVG` would be `NULL`).
- The `WHERE t.grade IS NOT NULL` clause ensures that only valid grades are used for the GPA calculation.

you-said-4

Use the schema and data in PDF file to define the view
ChatGPT said:

To define the `student_grades` view using the specific schema and data provided in your PDF, here's the SQL query:

```
[] sqlCopy codeCREATE VIEW student_grades AS SELECT  s.ID, COALESCE(AVG(gp.points),
0) AS GPA FROM    student s LEFT JOIN    takes t ON s.ID = t.ID LEFT JOIN
grade_points gp ON t.grade = gp.grade WHERE    t.grade IS NOT NULL -- Exclude
null grades from GPA calculation GROUP BY    s.ID;
```

breakdown-based-on-your-schema

Breakdown based on your schema:

1. **student**: The base table to get student IDs (**s.ID**).
2. **takes**: Contains the grades that students received in courses, joined on **s.ID = t.ID**.
3. **grade_points**: Maps letter grades (e.g., 'A', 'B+') to numeric grade points, joined on **t.grade = gp.grade**.
4. **Handling NULL Grades**: The **WHERE t.grade IS NOT NULL** clause ensures that only students with valid (non-NULL) grades are considered for the GPA calculation.
5. **COALESCE(AVG(gp.points), 0)**: Returns a GPA of 0 for students who have no valid grades. The **AVG** function computes the grade-point average for each student.

This view will give the GPA for each student using the letter grade to points conversion in the **grade_points** table, while excluding courses with null grades(University Schema and D...)(University Schema and D...).

you-said-5

You are not taking into account the credits of each course

ChatGPT said:

You're correct. The GPA calculation should account for the number of credits a course carries. To incorporate this, the total grade points earned in each course should be weighted by the course's credits, and the GPA should be the sum of these weighted points divided by the total number of credits.

Here's the corrected SQL query for defining the **student_grades** view:

```

[] sqlCopy codeCREATE VIEW student_grades AS SELECT  s.ID, COALESCE(SUM(gp.points
* c.credits) / SUM(c.credits), 0) AS GPA FROM  student s LEFT JOIN  takes t ON s.ID
= t.ID LEFT JOIN  section sec ON t.course_id = sec.course_id -- To get the course_id for
each section LEFT JOIN  course c ON sec.course_id = c.course_id -- To get the number of
credits for each course LEFT JOIN  grade_points gp ON t.grade = gp.grade -- To get the
points associated with the grade WHERE  t.grade IS NOT NULL -- Exclude null grades
from GPA calculation GROUP BY  s.ID;

```

key-changes

Key Changes:

1. **Weighted by Credits**: The GPA calculation now weights the grade points by the number of credits for each course (**gp.points * c.credits**).

2. **Total Credits:** The total grade points are divided by the total number of credits for the courses the student has taken (`SUM(c.credits)`).
3. **Join with course table:** The `course` table provides the number of credits for each course, and this is joined via the `section` and `takes` tables.

This approach ensures that courses with more credits have a greater impact on the student's GPA, as is typically the case in real-world GPA calculations(University Schema and D...)(University Schema and D...).

You've hit the Free plan limit for GPT-4o.

Responses will use another model until your limit resets after 11:08 PM.

Get Plus
prompt-textarea

composer-background