Competency questions description and justification

1. What is the average grading for each group from last month?

Scenario: This query helps track group-level academic performance to identify trends or issues and prioritize support where needed.

Justification: Partitioning the Grading table by group_id and dynamic partitioning ensures efficient aggregation of group-level data.

2. Who are the 5 students with the highest average percentage points?

Scenario: Enables highlighting top-performing students for awards, scholarships, or other recognition.

Justification: Bucketing by student_id optimizes joins and sorting operations on large datasets, crucial for this type of query.

3. Who are the 10 students with the lowest average percentage points?

Scenario: Facilitates identifying students needing extra academic support or intervention.

Justification: Using the ORC format in the Grading table ensures efficient storage and fast query response times.

4. Which teachers have the best average points obtained by their students in the last month?

Scenario: Evaluates teacher effectiveness and informs performance reviews or teaching method adjustments.

Justification: The Teachers table uses the MAP complex type to store multilingual surnames, allowing personalization and inclusion in different linguistic contexts.

5. What is the average percentage of points for each student?

Scenario: Tracks individual student performance over time for personalized academic counseling.

Justification: The percentage_of_points column in the Grading table is an ARRAY type, enabling flexible analysis of grading data.

6. Which group has the best average points?

Scenario: Highlights high-performing groups and uses their practices as benchmarks for others.

Justification: The partitioned structure of the Grading table accelerates group-level analysis.

7. What are the average percentage points for each academic year (course_year)?

Scenario: Helps compare performance across years to measure curriculum effectiveness or difficulty levels.

Justification: Static partitioning in the Courses table by course_year provides efficient data organization for this query.

8. What is the distribution of students across academic years?

Scenario: Informs strategic decisions regarding class sizes, resource allocation, and future enrollment planning.

Justification: Storing Students data in Parquet format ensures optimized access for such aggregate queries.

9. How has the average grading of groups changed over the last year?

Scenario: Tracks group progress and evaluates the long-term impact of academic policies or interventions.

Justification: External Dates table with partitioned data allows efficient time-series analysis for performance trends.

10. Which teachers have taught the most courses, and how does their students' performance compare?

Scenario: Identifies high-performing and overloaded teachers for appropriate workload balancing.

Justification: External Teachers and Courses tables ensure flexibility in data access while integrating seamlessly with the Grading table.

Warehouse design description and justification

1. Partitioning for Efficient Subset Queries

- **Table:** Grading
    - **Justification:** Partitioning the Grading table by group_id allows efficient queries at the group level, such as calculating average grades for a specific group. Dynamic partitioning enables seamless insertion of data into appropriate partitions without manual intervention.
- **Table:** Courses
    - **Justification:** Static partitioning by course_year helps retrieve course data for a specific academic year efficiently, crucial for analyzing trends across years.

2. **Bucketing for Optimized Joins and Aggregations**

- **Table:** Grading
    - **Justification:** Bucketing by student_id optimizes queries like identifying top-performing or low-performing students. Buckets distribute data into smaller, sorted partitions, enabling faster lookups and joins when filtering by student_id.

3. **Efficient Storage Formats for Query Performance**

- **Table:** Grading
    - **Justification:** Stored in **ORC** format to handle large volumes of grading data with better compression and faster query execution. This is critical for analytical queries requiring scanning or aggregating large datasets.
- **Table:** Students
    - **Justification:** Stored in **Parquet** format for optimized reading and writing performance. This is particularly helpful for queries requiring detailed student-level data.

4. **Complex Types for Rich Data Representation**

- **Table:** Grading
    - **Justification:** The percentage_of_points column is an **ARRAY** type, allowing the storage of detailed breakdowns of grading components (e.g., quizzes, assignments). This avoids creating multiple columns for each grading category, keeping the schema compact.
- **Table:** Teachers
    - **Justification:** The surnames column is a **MAP** type to store teacher surnames in different languages. This enhances inclusivity and usability in multilingual environments.

5. **Internal Tables for Frequently Accessed Data**
    - **Tables**: Grading, Courses

- **Justification:** Internal tables ensure that frequently accessed and processed data (e.g., grades and courses) are optimized for performance and managed entirely within the warehouse.

6. **External Tables for Flexibility and Integration**

- **Tables:** Students, Teachers, Dates, Groups

- **Justification:** External tables allow seamless integration with external systems, ensuring that data remains accessible for other applications or systems outside the data warehouse. For example, the Dates table is external and partitioned to enable efficient time-series queries.

7. **Support for Analytical Queries with Aggregation**

- **Table:** Grading

- **Justification:** Aggregating grades for competency questions such as "Which group has the best average points?" or "What are the average percentage points per academic year?" benefits from ORC format and partitioning.

8. **Scalable Time-Based Analysis**

- **Table:** Dates

- **Justification:** The Dates table supports scenarios requiring time-series analysis, such as calculating average grades from the last month or tracking yearly group progress. The external storage location allows easy updates and integration with other datasets.

9. **Scenario-Based Schema Design**

- **Table:** Courses

- **Justification:** Storing courses partitioned by course_year aligns with queries analyzing performance trends across academic years. This design ensures only relevant partitions are scanned, speeding up queries like "What are the average grades for courses in the first year?"

10. **Support for Multidimensional Analytics**

- **Table:** Grading

- **Justification:** The use of both partitioning and bucketing ensures the table can support multidimensional queries (e.g., by group, student, or course). Combining ARRAY types for grading details and partitioned storage provides robust capabilities for slicing data across different dimensions.