ISSN: 2473-4901
v12 n6648

Teaching Case

NASA Moon Data for SQL Analytics: An Adoption-Ready Dataset for GROUP BY, HAVING, and Window Functions

James R. Wolf
jrwolf@ilstu.edu
School of Information Technology
Illinois State University
Normal, IL 61790, USA

Hook

Teach SQL with data that students can sanity-check.

Abstract

This teaching case presents an adoption-ready SQL activity sequence built around NASA's "Moon Phase and Libration" data. The core idea is simple: students work with a realistic dataset that they can also sanity-check using familiar physical expectations, making many SQL mistakes easier to detect without relying entirely on answer keys or instructor intervention. Using a single-table schema with 8,760 hourly observations for one calendar year, the case supports introductory instruction in GROUP BY, HAVING, temporal grouping, and max-event queries, while also offering optional extensions for window functions, multidimensional aggregation, and plan-aware query design. The paper includes a Quick Start guide, minimal schema, validation queries, modular classroom exercises, and ready-to-use assignment materials that instructors can adopt with little preparation. The core sequence targets a 200-level introductory database course; optional extensions scale to advanced undergraduate and graduate courses. Pedagogically, the case is designed as a small-teaching intervention: it preserves the realism of authentic data while reducing the domain-decoding burden that often competes with SQL reasoning for student attention. By pairing technical rigor with an intuitively understandable domain, the activity is designed to support set-based reasoning, habits of plausibility checking, and more confident engagement with core database concepts.

Keywords: SQL pedagogy; cognitive load theory; real-world datasets; metacognition; window functions; student engagement

NASA Moon Data for SQL Analytics: An Adoption-Ready Dataset for GROUP BY, HAVING, and Window Functions

James R. Wolf

1. Introduction

Novice SQL learners often face a verification problem. When an aggregate query returns plausible results, they have no intuition about whether the results are correct or not (Carr et al., 2023; Naha et al., 2025; Sooriamurthi et al., 2025). In many classroom datasets, the domain offers no connections to the students' lived experiences, so verification requires hand calculations or a dependence on answer keys and instructor authority (Miao et al., 2019; Naha et al., 2025; Taipalus, 2023). This teaching note provides an adoption-ready alternative: a lunar ephemeris dataset whose physical constraints make many common SQL mistakes visibly wrong (e.g., results that imply impossible lunar behavior). The goal is not to add scientific complexity, but to decouple technical difficulty from domain difficulty so students can focus cognitive effort on SQL logic rather than interpreting business rules or learning a new domain.

For readers less familiar with the domain, four everyday facts are all the modules require: the Moon's average distance from Earth is about 384,400 km; because its orbit is elliptical, that distance varies between roughly 356,000 km (perigee) and 407,000 km (apogee); the closer the Moon, the larger it appears in the sky (apparent diameter); and its phases repeat on a cycle of about 29.5 days. NASA's Scientific Visualization Studio "Moon Phase and Libration" page (see Section 2.1.1) provides images and videos that instructors can use to ground these facts visually.

To support immediate classroom use, this note includes a minimal schema, a short set of validation queries to confirm the dataset loaded correctly, and a modular activity sequence that targets aggregation, grouping, and debugging through plausibility checks. The modules are designed as "small teaching" (Lang, 2021) inserts that can be dropped into an existing unit rather than requiring a course redesign. Optional extensions provide a path to stretch the same dataset into more advanced topics once students are ready. The core sequence targets a 200-level introductory database course; the extensions scale to advanced undergraduate and graduate work.

This instructional choice is also timely. Artemis II launched on April 1, 2026, as NASA's first crewed lunar flyby in 50 years, and within days, its crew surpassed the Apollo 13 record for the farthest human spaceflight from Earth (Dooren, 2026). In that context, lunar data offer more than a familiar scientific domain: they connect classroom SQL work to an object newly visible in public discourse while retaining the physical regularities that make student errors easier to detect.

To respect the instructor's limited preparation time, this paper deliberately inverts the standard academic structure. It presents a 'Quick Start' guide before detailing the theoretical rationale, which is grounded in Cognitive Load Theory (Bunch, 2009; Chandler & Sweller, 1996; Sweller, 2024) and the Expertise Reversal Effect (Akhuseyinoglu et al., 2022; Kalyuga, 2007; Sweller et al., 2011).

This placement allows instructors to first verify the logistical feasibility of the NASA Lunar Database, ensuring that the subsequent theoretical discussion is grounded in a concrete understanding of the dataset's simplicity. The detailed pedagogical framework and literature review follow immediately after. The next section provides a Quick Start checklist to help instructors implement the activity with minimal preparation.

This paper makes three contributions. First, it provides an adoption-ready, real-data SQL teaching case whose domain permits student self-verification: stable physical regularities make many semantic errors visibly wrong. Second, it supplies a complete instructional package. This package includes a modular activity sequence with verification cues, solution sketches, expected outputs, ready-to-use assignments, and assessment rubrics. All of which can be adopted without redesigning a course. Third,

it articulates a transferable design pattern: choosing datasets whose well-known regularities allow students to check the plausibility of query results without relying on an answer key.

Quick Start

Prep (15-30 minutes, instructor):

1. Download one year of NASA SVS "Moon Phase and Libration" data (JSON or text). Here is the link for 2026: https://svs.gsfc.nasa.gov/5587
2. Load into a single table named moon26 using the minimal schema (fields: DATETIME, PHASE, AGE, DIAM, DIST). The companion file moon26_load.sql provides ready-to-run Oracle 19c DDL and 8,760 INSERT statements for the 2026 dataset. Both moon26_load.sql and moon26_queries.sql are included with the supplemental materials.
3. Validate load (recommended before students see the table):
   - Row count should be 8,760 (non-leap year) or 8,784 (leap year).
   - Check plausible ranges via MIN/MAX for dist and diam.

In-class core sequence (50-75 minutes total; assign as one lab or split across 1-2 meetings):

- Module 0 (2-5 min): Prediction hook. Students commit to a prediction (no grading for correctness).
- Module 1: Monthly average distance (GROUP BY time key).
- Module 2: Temporal granularity variations (day / quarter).
- Module 3: Extreme months using HAVING (MAX/MIN thresholds).
- Module 4: "Max event" timestamp task (largest apparent diameter).
- Module 5 (5 min or homework): Reflection + quick assessment (verification cue + debugging move).

Optional (advanced / graduate):

- Add an index on datetime only if using the performance extension.

2. The NASA Lunar Database: Design and Properties

2.1. Dataset Overview and Sources

2.1.1 Acquisition (Download)

NASA's Scientific Visualization Studio (SVS) publishes a "Moon Phase and Libration" page for each year (Wright, 2026). Each page provides a full calendar year of hourly observations (typically 8,760 rows for a non-leap year; 8,784 for a leap year), available in JSON and text formats. Files commonly follow the naming pattern `mooninfo_20xx.json` and `mooninfo_20xx.txt`. Instructors can use any single year (e.g., 2026 or earlier) without changing the logic of the exercises. Primary source page (year 2026): https://svs.gsfc.nasa.gov/5587 (Moon Phase and Libration, 2026).

2.1.2 Load (Instructor Setup)

To minimize adoption friction, the teaching note assumes the dataset is loaded into one table named `moon26`. The 2026 Oracle 19c load file (moon26_load.sql) contains the DDL and 8,760 hourly INSERT statements; the companion file moon26_queries.sql contains all instructional queries. The import pipeline may vary (Oracle, PostgreSQL, MySQL), as long as the five instructional fields in Section 2.1.3 are available. Both companion files (moon26_load.sql and moon26_queries.sql) are included with the supplemental materials for this paper.

Recommended lowest-friction workflow:

1. Download the year's file from NASA SVS (JSON or text).
2. Convert to CSV if needed using a small script or tool of your choice. (If you want zero scripting, use the text file and a fixed-width import workflow.)

3. Run `moon26_load.sql` (Oracle) or import the NASA SVS file into a table named moon26 using the minimal schema below.
4. Run the validation queries in Section 2.1.4 to confirm row count and plausible value ranges.

Optional performance setup (for the advanced extension): add an index on datetime so students can compare sargable predicates to function-wrapped predicates using `EXPLAIN PLAN` or equivalent tooling. This stays optional so the introductory sequence remains lightweight.

2.1.3 Minimal Instructional Schema (Single Table)

This teaching note uses a deliberately small subset of the ephemeris fields—chosen to support aggregation, temporal grouping, and built-in plausibility checks without requiring coordinate systems or astronomical background.

Table name: `moon26`

Column	Type (suggested)	Meaning / units
datetime	TIMESTAMP (UTC)	Observation timestamp (hourly).
phase	NUMBER	Percent illumination (0–100).
age	NUMBER	Days since new moon.
dist	NUMBER	Earth–Moon distance (km).
diam	NUMBER	Apparent diameter (arcseconds).

Oracle DDL (example)

```
CREATE TABLE moon26 (
  DATETIME  TIMESTAMP(0)     NOT NULL,
  PHASE     NUMBER           NULL,   -- percent, 0-100
  AGE       NUMBER           NULL,   -- days since new moon
  DIAM      NUMBER           NULL,   -- arcseconds
  DIST      NUMBER           NULL    -- km
);
```

Sample rows (first five hourly observations of 2026):

DATETIME	PHASE	AGE	DIAM	DIST
2026-01-01 00:00:00	91.40	11.928	1985.1	361045.0
2026-01-01 01:00:00	91.68	11.970	1985.5	360985.0
2026-01-01 02:00:00	91.95	12.012	1985.8	360927.0
2026-01-01 03:00:00	92.22	12.053	1986.1	360873.0
2026-01-01 04:00:00	92.49	12.095	1986.4	360820.0

Plausibility anchors used for verification: students can sanity-check aggregates against stable physical expectations (e.g., the Moon's mean distance is about 384,400 km; an elliptical orbit yields a substantial but bounded range).

2.1.4 Quick Validation Queries (Recommended)

These checks help confirm the load worked before students ever see the table:

```
-- 1) Expected row count: 8,760 (non-leap-year) or 8,784 (leap-year)
SELECT COUNT(*) AS n_rows FROM moon26;

-- 2) Basic plausibility ranges (should be reasonable, not
extreme/empty)
SELECT MIN(dist) AS min_dist_km,
       MAX(dist) AS max_dist_km,
       MIN(diam) AS min_diam_arcsec,
       MAX(diam) AS max_diam_arcsec
FROM moon26;
```

3. Related Work and Rationale

SQL instructors face a persistent realism dilemma. Simplified databases can clarify core ideas and make results easy to inspect, but they may underprepare students for the complexity of professional data environments; more authentic datasets can improve perceived relevance while increasing domain-decoding burden (Jukic & Gray, 2008; Wagner et al., 2003; Yue, 2013). This matters because students routinely report that computing coursework feels disconnected from real-world contexts, and relevance-oriented interventions tend to improve attitudes more reliably than "fun" alone (Bellino et al., 2021; Biggers et al., 2008; Hsu et al., 2018). The instructional problem, then, is not simply choosing "realistic" data, but selecting a dataset whose context does not compete with SQL reasoning for limited attention.

A second constraint is that many of the concepts students most need to master are cognitively demanding even before domain complexity is added. Research consistently identifies aggregation and grouping—especially `GROUP BY` and `HAVING`—as major stumbling blocks for novices, alongside multi-relation reasoning and nested queries (Ahadi et al., 2016; Miedema et al., 2023; Taipalus et al., 2018). These difficulties are often tied to SQL's declarative nature: students import procedural habits and then mis-specify set-level logic (Celko, 2008; Sadiq et al., 2004; Taipalus, 2023). In practice, many consequential student errors are semantic rather than syntactic—the query runs but answers the wrong question—such as confusing row-level filters with group-level conditions or misplacing aggregate predicates in `WHERE` instead of `HAVING` (Brass & Goldberg, 2006; Welty, 1985). Because these errors can produce numerically "plausible" outputs, students often struggle to diagnose them without strong verification cues.

Cognitive Load Theory suggests a straightforward design implication: when the domain is familiar and already organized in long-term memory, students spend less working-memory capacity decoding context and more capacity on the intended SQL reasoning (Miedema et al., 2023; Sweller, 2024). This is especially relevant for aggregation tasks, where students must simultaneously track grouping grain, attribute meaning, and the difference between row- and group-level logic—conditions that can amplify working-memory strain and increase error rates (Smelcer, 1995; Mason et al., 2016). A familiar domain can therefore function as a "cognitive stabilizer": it reduces domain comprehension overhead and makes it easier to notice when outputs violate expectations, accelerating debugging and supporting the shift to set-based thinking.

Familiarity must, however, be handled with care: overly sanitized "toy" contexts can invite shallow success and poor transfer (Kalyuga et al., 2003; Yue, 2013). The approach in this teaching note is designed to bridge these concerns through perceived realism plus verifiability: a high-fidelity dataset grounded in a universally familiar physical domain preserves authentic structure (timestamps, continuous measures, nontrivial aggregation demands) while providing internal plausibility checks that help students detect semantic errors without relying on answer keys or instructor authority (Lang, 2021; Miedema et al., 2023). This combination motivates the NASA lunar dataset as a practical compromise: the domain remains easy to interpret, while the SQL tasks scale from introductory

grouping to advanced analytic work. Section 4 details how the design manages the resulting Expertise Reversal risk.

4. Design Principles: Managing Expertise Reversal

4.1 Decoupling Domain and Technical Complexity

The NASA Lunar Database addresses this by decoupling domain complexity from technical complexity: the domain stays stable while the SQL demands escalate. For novices, a familiar domain reduces extraneous cognitive load and lets students focus on core SQL concepts. As students advance, instructors can increase intrinsic load by assigning tasks that demand deeper technical reasoning rather than greater domain complexity. For example, instruction can progress from monthly averages to analyses such as identifying months with the highest distance variance. It can also include detecting discontinuities or outliers in distance—tasks that naturally require window functions. At that point, students evaluate both correctness and efficiency, using techniques such as indexing, predicate rewriting, and EXPLAIN PLAN. This progression keeps tasks aligned with students' developing expertise, mitigating expertise reversal while sustaining engagement through perceived relevance.

4.2 Intuitive Verification Across the Curriculum

The familiar domain of the lunar data transforms the dataset from a mere input source into a feedback mechanism. This benefit applies equally to novices struggling with basic syntax and experts wrestling with complex logic.

For novices, the dataset offers an immediate, passive defense against common logic errors. Consider a student attempting to join the lunar data with a calendar table but accidentally creating a Cartesian product (cross join) or summing distances rather than averaging them. In a generic "Order Entry" database, a result of "15,000,000" might look plausible to a student who lacks context for the company's sales volume. However, when a student sees a calculated Earth–Moon distance of 15 million kilometers, the feedback is immediate and internal. They do not need to run a separate debugging protocol; the physical impossibility of the result triggers an instant "stop" signal. This allows the student to self-correct the SQL logic in the moment, reducing frustration and keeping the focus on syntax rather than business rule decoding.

For advanced students, this same familiarity facilitates high-level verification. When writing sophisticated queries using Window Functions (e.g., `LAG`, `LEAD`) or multi-dimensional aggregation (`CUBE`, `ROLLUP`), the primary cognitive burden shifts to the abstract logic of the query optimizer. Here, the lunar cycle allows students to perform "sanity checks" on trend analysis—for instance, verifying that a calculated "moving average" of distance creates a smooth curve consistent with an elliptical orbit.

4.3 Building Professional Realism Without Overwhelming Novices

The dataset also mitigates the risk of context-dependency—the inability to transfer skills to messy, real-world environments (Bransford & Schwartz, 1999; Perkins & Salomon, 1989; Taipalus, 2020). Unlike sanitized textbook examples, the lunar data retains the characteristics of a "data demand agnostic" professional database: it contains continuous numeric values, precise timestamps, and requires optimization strategies like indexing to handle performance efficiently. By revisiting the same dataset with increasingly difficult tasks—from simple GROUP BY operations to analyzing execution plans for analytic functions—instruction moves from "teaching the data" to "using the data," effectively preparing graduates for the complexity of professional database systems without sacrificing the engagement benefits of perceived realism. What follows is a synthesis of these principles: a summary of how the Moon dataset bridges the pedagogical tensions identified in the literature review and a reflection on implications for database instruction beyond this specific application.

4.4 A "Small Teaching" Intervention

Finally, the adoption of the NASA Lunar Database represents a "Small Teaching" intervention (Lang, 2021)—an instructional change that yields pedagogical benefits without requiring a comprehensive course redesign.

Instructors often hesitate to introduce real-world data because of the perceived "instructional burden" associated with teaching the context of a new, messy dataset. However, replacing a generic sales database with lunar data is a low-stakes logistical swap that reduces extraneous cognitive load. Because the domain is universally familiar, the "domain decoding" tax is removed, freeing up students' working memory.

This allows the instructor to retain their existing syllabus structure—teaching the same progression of `GROUP BY`, `HAVING`, and *window functions*, while significantly increasing the "germane load" devoted to actual SQL reasoning. The intervention is not a change in *what* is taught, but simply a change in the *materials* used to teach it.

4.5. Technical Correctness

Researchers group SQL mistakes into four types: syntax errors, semantic errors, logical errors, and complications (Ahadi et al., 2016; Miedema et al., 2023; Taipalus et al., 2018). **Syntax errors** break SQL's rules, so the database cannot run the query (Ahadi et al., 2015, 2016; Taipalus et al., 2018). **Semantic errors** run without crashing, but the query answers the wrong question for the task (Brass & Goldberg, 2006; Taipalus et al., 2018). **Logical errors** run without crashing, but the query fails for a specific case in the task (Taipalus et al., 2018). **Complications** return the right table, but the query uses extra steps or extra clauses it does not need (Brass & Goldberg, 2006; Taipalus et al., 2018). Taipalus et al. (2018) call a clean, simple, correct query *exemplary*.

This taxonomy maps naturally onto grading: exemplary queries earn full credit; complications carry only minor deductions; semantic and logical errors earn partial credit scaled by how much the results change; and syntax errors lose the most points because the query does not run. Appendix D provides the full rubric, together with an instructor note on grading LLM-assisted answers.

5. Activity Set for Introductory SQL (3-6 Exercises)

The following modular sequence operationalizes the design principles outlined in Section 4. Specifically, it addresses the risk of Expertise Reversal by holding the domain constant (the Moon) while progressively increasing the technical cognitive load, moving from basic aggregation to complex filtering. Crucially, each module integrates the 'Verification' principle by including a specific 'Verification Cue.' These cues mimic the professional habit of sanity-checking data, training students to rely on physical plausibility rather than an answer key.

This activity set is designed as a modular sequence. Instructors may assign the core modules as a single lab or spread them across 1–2 class meetings, with optional extensions as plug-ins.

Module 0 — Prediction Hook (2–5 minutes)

Goal: Activate prior knowledge and create an information gap that the SQL queries will resolve.
Student prompt (pick 1–2):

- Do you think the Moon is closer to Earth during a Full Moon or a New Moon—or does it not matter?
- We know the average distance between the Moon and Earth is about 384,400 km. How much does it vary in a single month: a few hundred km, or tens of thousands?

Verification cue: You are not grading correctness here; you are creating a commitment students can later confirm or revise.
Instructor note: This prediction hook is a deliberate attention strategy. When students have a personal stake in the query's output, even if just a guess, the subsequent debugging process feels more like scientific discovery rather than a battle with syntax.

Module 1 — Monthly Average Distance (GROUP BY on time)

Goal: Practice GROUP BY on a time-derived key; interpret an aggregate output in a physically meaningful way.
Student prompt: For each month in 2026, what was the Moon's average distance from Earth?
Verification cue: Monthly averages should cluster around ~384,400 km and should not show wild month-to-month swings.
Common failure modes (what to watch for):

- Grouping by a month name only (e.g., 'January') can misorder results or collapse across years; prefer a sortable month key.
- Confusing row-level filters (`WHERE`) with group-level filters (`HAVING`)—save HAVING for Module 3.

Solution sketch (Oracle-friendly):

```
SELECT TRUNC(datetime, 'MM') AS month_start,
       ROUND(AVG(dist), 1) AS avg_distance_km
FROM moon26
GROUP BY TRUNC(datetime, 'MM')
ORDER BY month_start;
```

Expected output (first 3 of 12 rows): 2026-01 → 382,888.4 km; 2026-02 → 384,600.1 km; 2026-03 → 384,271.0 km. All twelve monthly averages fall between 382,522.7 km (October) and 386,960.3 km (May).

Instructor note: You may show `TO_CHAR(datetime,'Month')` as an output readability variant, but teach `TRUNC(datetime,'MM')` (or `TO_CHAR(datetime,'YYYY-MM')`) as the robust grouping key.

Module 2 — Temporal Granularity Variations (day, quarter)

Goal: Show that the same dataset supports multiple groupings (day/month/quarter) and that the grouping choice changes the story.
Student prompt (choose one):

- Compute the Moon's average distance by day for one month of your choice; identify the day with the smallest average distance.
- Compute average distance by quarter (Q1–Q4).

Verification cue: Daily aggregates should show smooth trends; quarter-level averages should be even more stable.
Common failure modes:

- Students forget to adjust ORDER BY to match the grouping key (e.g., string months sort alphabetically).

Solution sketch (quarter example):

```
SELECT TO_CHAR(datetime, 'YYYY') AS yr,
       TO_CHAR(datetime, 'Q') AS qtr,
       ROUND(AVG(dist), 1) AS avg_distance_km
FROM moon26
GROUP BY TO_CHAR(datetime, 'YYYY'), TO_CHAR(datetime, 'Q')
ORDER BY yr, qtr;
```

Expected output (all 4 rows): Q1 → 383,897.2 km; Q2 → 386,349.9 km; Q3 → 384,684.1 km; Q4 → 383,620.8 km.

Module 3 — Extreme Months (HAVING with MAX/MIN thresholds)

Goal: Introduce HAVING as post-aggregation filtering; differentiate it from WHERE.
Student prompts (pick 1–2):

- Which months had a maximum distance greater than 406,400 km?

- List the months where the Moon's closest approach was under 364,000 km.

Verification cue: Results should be plausible but not ubiquitous; only some months should cross a strict threshold.

Common failure modes:

- Trying to put MAX(dist) > … in WHERE instead of HAVING.
- Grouping by month name only, causing ordering or grouping errors.

Solution sketch (show one; assign the other):

```
SELECT TRUNC(datetime, 'MM') AS month_start,
       MAX(dist) AS farthest_km
FROM moon26
GROUP BY TRUNC(datetime, 'MM')
HAVING MAX(dist) > 406400
ORDER BY farthest_km DESC;
```

Expected output (2026): one row — month_start 2026-12-01, farthest 406,436 km. The companion prompt (closest approach under 364,000 km) returns eight months.

Instructor note: Encourage students to order by the aggregate itself (e.g., farthest_km) to reinforce that aggregates produce new values the query can sort on.

Instructor note (year-specific thresholds): The thresholds used here (406,400 km and 364,000 km) are tuned to the 2026 data, in which only December exceeds 406,400 km and eight months dip under 364,000 km. For other years, run the validation queries in Section 2.1.4 to obtain that year's minimum and maximum distances, then choose thresholds a few thousand kilometers inside the observed extremes so that the HAVING clause is selective but not empty.

Module 4 — Timestamp Verification Task (largest apparent diameter)

Goal: Practice "find the max event" queries and use the result as a concrete sanity-check moment.

Student prompt: Find the exact date/time when the Moon's apparent diameter was largest in 2026; report that timestamp, diameter, and distance. (Instructor note: the 2026 data has five tied hourly rows on 2026-12-24 with diam = 2009.6" — a useful tie-handling discussion.)

Verification cue: Once students identify the timestamp, they can verify the diameter value is maximal and that the associated distance is consistent with "closer = larger apparent diameter."

Common failure modes:

- Pulling the right maximum but returning multiple rows (ties) without noticing.
- Filtering with formatted strings too early (works as a stepping stone, but is not ideal long-term).

Solution sketch (clean, teachable version):

```
SELECT datetime, diam, dist
FROM moon26
WHERE diam = (SELECT MAX(diam) FROM moon26);   -- 2026: returns 5 tied rows on
2026-12-24
```

Expected output: five tied rows, 2026-12-24 07:00–11:00, diam = 2,009.6 arcsec, dist between 356,648 and 356,658 km.

Instructor note: If you demonstrate a less elegant first-pass approach, explicitly label it as a deliberate stepping stone before introducing better techniques.

Module 5 — Reflection + Quick Assessment (5 minutes or homework)

Goal: Make verification labor visible and discussable; reinforce the "stop signal" idea.

Student deliverable (short):

- Name one result that initially looked wrong (or surprising). What did you check, and what change fixed it?

Instructor grading lens (lightweight): Grade primarily on whether the student identified a plausible verification cue and described a debugging move (not on rhetorical polish). For adoption support, Appendix D provides a lightweight grading rubric and feedback shorthand aligned to common SQL error categories (syntax, semantic, logical, and unnecessary complication).

6. Extensions (Optional): Advanced / Graduate Plug-ins

This section is optional. Instructors can adopt the dataset and the introductory activity set (Section 5) without using any of the extensions below. These plug-ins are included to demonstrate curricular scalability: the same familiar time-series dataset can support advanced SQL topics once students are ready for analytic depth. The full activity descriptions are in the companion document Optional_Extensions.docx, included in the supplemental materials; the appendices to this paper (A–D) contain the dataset summary, the two core assignments, and the grading rubric.

6.1 Scaling Up: Advanced Extensions

While this teaching note focuses on the introductory sequence to ensure immediate adoptability, the NASA Lunar Database is sufficiently rich to support advanced coursework. The dataset's continuous variables and time-series nature make it an ideal playground for analytic window functions (**`LAG/LEAD`**), multidimensional aggregation (**`ROLLUP/CUBE`**), and performance tuning. To preserve the "Small Teaching" focus of the main text, these optional advanced modules are detailed in **`the companion document (Optional_Extensions.docx)`**. Instructors wishing to scale the dataset from an introductory "**`SELECT`**" lab to a graduate-level "Optimization" module will find full activity descriptions and verification cues there.

7. Conclusion

This paper demonstrates that NASA lunar data resolves a persistent dataset dilemma in SQL instruction: the trade-off between simplified datasets that teach cleanly and realistic datasets that overwhelm novices. By pairing a universally familiar physical domain with rigorous SQL tasks, the dataset turns students' everyday intuitions into built-in plausibility checks, strengthening set-based reasoning without adding domain-decoding burden.

In introductory instruction, students use stable lunar regularities as stop signals: implausible distances, phases, or diameter–distance relationships make semantic mistakes hard to ignore. As a course advances, the same table scales to analytic window functions, multidimensional aggregation (ROLLUP/CUBE), and plan-aware query design—a spiral progression in which the domain stays constant while technical demands increase.

Ultimately, this approach offers instructors a path to professional rigor that does not require a complete curriculum overhaul. By treating the dataset choice as a 'Small Teaching' opportunity, educators can swap out context-heavy financial schemas for the intuitively verifying context of the Moon. This simple material substitution lowers the barrier to entry for complex SQL topics, ensuring that students spend their limited cognitive resources not on decoding a business logic they have never seen, but on mastering the set-based reasoning they will need for the rest of their careers.

The beauty of this intervention lies in what it *doesn't* ask of the instructor. It is not a change in the syllabus or the learning objectives. It is merely a change in the materials used to teach them.
Supplemental materials. The submission includes: moon26.csv (one year of hourly observations); moon26_load.sql (DDL and INSERT statements); moon26_queries.sql (every instructional query with expected output); Optional_Extensions.docx (advanced and graduate modules); student and CMS-ready versions of Assignments 1 and 2 (also reproduced in Appendices B and C); and instructor answer keys for both assignments.

References

Ahadi, A., Prior, J., Behbood, V., & Lister, R. (2015). A quantitative study of the relative difficulty for novices of writing seven different types of SQL queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 201–206). Association for Computing Machinery. https://doi.org/10.1145/2729094.2742620

Ahadi, A., Prior, J., Behbood, V., & Lister, R. (2016). Students' semantic mistakes in writing seven different types of SQL queries. In *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 272–277). Association for Computing Machinery. https://doi.org/10.1145/2899415.2899464

Akhuseyinoglu, K., Hardt, R., Barria-Pineda, J., Brusilovsky, P., Pollari-Malmi, K., Sirkiä, T., & Malmi, L. (2022). A study of worked examples for SQL programming. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2022)* (Vol. 1, pp. 82–88). Association for Computing Machinery. https://doi.org/10.1145/3502718.3524813

Bellino, A., Herskovic, V., Hund, M., & Muñoz-Gama, J. (2021). A real-world approach to motivate students on the first class of a computer science course. *ACM Transactions on Computing Education*, *21*(1), 1–23. https://doi.org/10.1145/3445982

Biggers, M., Brauer, A., & Yilmaz, T. (2008). Student perceptions of computer science: A retention study comparing graduating seniors with CS leavers. *ACM SIGCSE Bulletin*, *40*(1), 402–406. https://doi.org/10.1145/1352322.1352274

Bransford, J. D., & Schwartz, D. L. (1999). Rethinking transfer: A simple proposal with multiple implications. *Review of Research in Education*, *24*(1), 61–100. https://doi.org/10.3102/0091732X024001061

Brass, S., & Goldberg, C. (2006). Semantic errors in SQL queries: A quite complete list. *Journal of Systems and Software*, *79*(5), 630–644. https://doi.org/10.1016/j.jss.2005.06.028

Bunch, J. M. (2009). An approach to reducing cognitive load in the teaching of introductory database concepts. *Journal of Information Systems Education*, *20*(3), 269–275.

Carr, N., Shawon, F. R., & Jamil, H. M. (2023). An experiment on leveraging ChatGPT for online teaching and assessment of database students. In *2023 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)* (pp. 1–8). IEEE. https://doi.org/10.1109/TALE56641.2023.10398239

Celko, J. (2008). *Joe Celko's thinking in sets: Auxiliary, temporal, and virtual tables in SQL*. Morgan Kaufmann.

Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied Cognitive Psychology*, *10*(2), 151–170. https://doi.org/10.1002/(SICI)1099-0720(199604)10:2<151::AID-ACP380>3.0.CO;2-U

Dooren, J. M. (2026, April 10). *NASA welcomes record-setting Artemis II moonfarers back to Earth*. NASA. https://www.nasa.gov/news-release/nasa-welcomes-record-setting-artemis-ii-moonfarers-back-to-earth/

Hsu, T.-C., Chang, S.-C., & Hung, Y.-T. (2018). How to learn and how to teach computational thinking: Suggestions based on a review of the literature. *Computers & Education*, *126*, 296–310. https://doi.org/10.1016/j.compedu.2018.07.004

Jukic, N., & Gray, P. (2008). Using real data to invigorate student learning. *ACM SIGCSE Bulletin*, *40*(2), 6–10. https://doi.org/10.1145/1383602.1383604

Kalyuga, S. (2007). Expertise reversal effect and its implications for learner-tailored instruction. *Educational Psychology Review*, *19*(4), 509–539. https://doi.org/10.1007/s10648-007-9054-3

Kalyuga, S., Ayres, P., Chandler, P., & Sweller, J. (2003). The expertise reversal effect. *Educational Psychologist*, *38*(1), 23–31. https://doi.org/10.1207/S15326985EP3801_4

Lang, J. M. (2021). *Small teaching: Everyday lessons from the science of learning* (2nd ed.). Jossey-Bass.

Mason, R., Seton, C., & Cooper, G. (2016). Applying cognitive load theory to the redesign of a conventional database systems course. *Computer Science Education*, *26*(1), 68–87. https://doi.org/10.1080/08993408.2016.1160597

Miao, Z., Roy, S., & Yang, J. (2019). Explaining Wrong Queries Using Small Examples. *Proceedings of the 2019 International Conference on Management of Data*, 503–520. https://doi.org/10.1145/3299869.3319866

Miedema, D., Taipalus, T., & Aivaloglou, E. (2023). Students' Perceptions on Engaging Database Domains and Structures. *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 122–128. https://doi.org/10.1145/3545945.3569727

Naha, K., Rubaiat, S. Y., & Jamil, H. M. (2025). Are large language models smart enough for SQL tutoring and assessment? In *2025 IEEE International Conference on Advanced Learning Technologies (ICALT)* (pp. 357–359). IEEE. https://doi.org/10.1109/ICALT64023.2025.00109

Perkins, D. N., & Salomon, G. (1989). Are cognitive skills context-bound? *Educational Researcher*, *18*(1), 16–25. https://doi.org/10.3102/0013189X018001016

Sadiq, S., Orlowska, M., Sadiq, W., & Lin, J. (2004). SQLator: An online SQL learning workbench. *Proceedings of the 9th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education*, 223–227. https://doi.org/10.1145/1007996.1008055

Smelcer, J. B. (1995). User errors in database query composition. *International Journal of Human–Computer Studies*, *42*(4), 353–381. https://doi.org/10.1006/ijhc.1995.1017

Sooriamurthi, R., Tu, X., & Pensky, A. E. C. (2025). A generative AI tool to foster and assess authentic learning: A case study in teaching SQL. In *Proceedings of the 30th ACM Conference on Innovation and Technology in Computer Science Education* (Vol. 1, pp. 465–471). Association for Computing Machinery. https://doi.org/10.1145/3724363.3729022

Sweller, J. (2011). Cognitive load theory. In J. P. Mestre & B. H. Ross (Eds.), *Psychology of learning and motivation* (Vol. 55, pp. 37–76). Elsevier. https://doi.org/10.1016/B978-0-12-387691-1.00002-8

Sweller, J. (2024). Cognitive load theory and individual differences. *Learning and Individual Differences*, *110*, 102423. https://doi.org/10.1016/j.lindif.2024.102423

Sweller, J., Ayres, P., & Kalyuga, S. (2011). The expertise reversal effect. In *Cognitive load theory* (pp. 155–170). Springer. https://doi.org/10.1007/978-1-4419-8126-4_12

Taipalus, T. (2020). The effects of database complexity on SQL query formulation. *Journal of Systems and Software*, *165*, 110576. https://doi.org/10.1016/j.jss.2020.110576

Taipalus, T. (2023). SQL: A Trojan horse hiding a decathlon of complexities. In *Proceedings of the 2nd International Workshop on Data Systems Education: Bridging education practice with education research* (pp. 9-13).

Taipalus, T., Siponen, M., & Vartiainen, T. (2018). Errors and Complications in SQL Query Formulation. *ACM Transactions on Computing Education*, *18*(3), 1–29. https://doi.org/10.1145/3231712

Wagner, P. J., Shoop, E., & Carlis, J. V. (2003). Using scientific data to teach a database systems course. In *Proceedings of the 34th SIGCSE Technical Symposium on Computer Science Education* (pp. 224–228). Association for Computing Machinery. https://doi.org/10.1145/611892.611975

Welty, C. (1985). Correcting user errors in SQL. *International Journal of Man–Machine Studies*, *22*(4), 463–477. https://doi.org/10.1016/S0020-7373(85)80045-7

Wright, E. (2026). *Moon phase and libration, 2026* [Data set]. NASA Scientific Visualization Studio. https://svs.gsfc.nasa.gov/5587/

Yue, K.-B. (2013). Using a semi-realistic database to support a database course. *Journal of Information Systems Education*, *24*(4), 327–336.

APPENDIX A. NASA MOON PHASE AND LIBRATION DATA (INSTRUCTIONAL SUMMARY)

This teaching note draws on NASA's Scientific Visualization Studio (SVS) "Moon Phase and Libration" data for a single calendar year, typically reported at hourly intervals (UTC). For instruction, the activity set uses a deliberately small subset of fields that support temporal grouping, aggregation, and built-in plausibility checks without requiring an astronomical background.
Instructional fields retained (used throughout this note):

Note: The SVS source files contain additional ephemeris and coordinate fields. This teaching note intentionally omits them to reduce domain-decoding overhead while preserving realistic timestamps and continuous measures for SQL analytics.

Column	Meaning / units	Why it matters in SQL tasks
datetime	Observation timestamp (UTC, hourly)	Grouping keys; range predicates; ordering for analytic functions
phase	Percent illumination (0-100)	Optional binning/filters; plausibility checks
age	Days since new moon	Optional grouping/interpretation; cycle reasoning
dist	Earth-Moon distance (km)	Aggregation targets; threshold checks via HAVING; plausibility anchors
diam	Apparent diameter (arcseconds)	Max-event tasks; distance-diameter verification cue

ISSN: 2473-4901
v12 n6648

APPENDIX B. MOON ASSIGNMENT 1

Submission Instructions

Submit a single plain-text file (.txt) containing all your SQL statements and their output. Use the following SQL Developer commands to capture everything automatically:

```
-- At the top of your script file:
SET ECHO ON
SPOOL assignment1_YOURNAME.txt

-- ... all your queries go here ...

SPOOL OFF
```

SQL Developer Capture Workflow

SET ECHO ON causes SQL Developer to print each SQL statement before its output, so the grader can see both the query and the results in a single file.

SPOOL writes everything displayed on screen — including echoed SQL and query output — to the specified file. Run SPOOL OFF at the end to close the file.

Name your file: assignment1_YOURNAME.txt (replace YOURNAME with your last name).

Do not submit screenshots, Word documents, or PDF exports of SQL output.

File Format Requirements

Include at the top of your script (after SPOOL): your name, course, section, and the date.

Example:

```
-- Name:    Jane Smith
-- Course:  IS 385 — Database Management
-- Date:    2026-04-15
```

Each task must appear in the submission file in order.

For each task, include: (1) the SQL statement(s), (2) the query output, and (3) a brief verification note as a SQL comment (--).

ISSN: 2473-4901
v12 n6648

Dataset: moon26

The table moon26 contains 8,760 hourly observations of the Moon for the full year 2026. It has been loaded into the course database. Each row represents one measurement taken on the hour.

Lunar background (everything you need to know): the Moon's average distance from Earth is about 384,400 km; its orbit is elliptical, so the distance varies between roughly 356,000 km (perigee) and 407,000 km (apogee); the closer the Moon, the larger it appears in the sky (apparent diameter); and the phases repeat about every 29.5 days. Images and videos: NASA's "Moon Phase and Libration" page, https://svs.gsfc.nasa.gov/5587/.

Column	Type	Meaning / Units
DATETIME	TIMESTAMP(0)	Observation timestamp (UTC, hourly)
PHASE	NUMBER	Percent illumination (0–100)
AGE	NUMBER	Days since new moon
DIAM	NUMBER	Apparent diameter (arcseconds)
DIST	NUMBER	Earth–Moon distance (km)

ℹ Before You Begin

You do not need to create or load the table — it is already in your schema.

Run this validation query first to confirm access:

```
SELECT COUNT(*) FROM moon26;   -- should return 8760
```

TASK 0 — PREDICTION (2–5 MIN)

Before writing any SQL, answer the following questions as comments in your submission file. You will verify your answers with your queries.

1. Do you think the Moon is closer to Earth during a Full Moon or a New Moon, or does it not matter? Explain briefly.
2. The average Earth–Moon distance is about 384,400 km. How much do you think the distance varies within a single month: a few hundred km, or tens of thousands?

✓ What to submit

Write your answers as SQL comments (--) in your .txt file before your first query.

Example: -- Task 0, Q1: I think the Moon is closer during Full Moon because...

ISSN: 2473-4901
v12 n6648

TASK 1 — MONTHLY AVERAGE DISTANCE (GROUP BY)

Goal: Practice GROUP BY on a time-derived key and interpret aggregate output in a physically meaningful way.

Prompt: For each month in 2026, what was the Moon's average distance from Earth? List months in chronological order and round the average to one decimal place.

’ Self-check: Monthly averages should cluster around ~384,400 km with no wild jumps.
Use TRUNC(datetime, 'MM') as your grouping key — it produces a sortable date value.
Do not use TO_CHAR(datetime,'Month') as a grouping key; it sorts alphabetically.

✓ What to submit
1. The SQL query.
2. The full query output (all 12 rows).
3. A brief verification note as a comment: how did you confirm the results look right?

TASK 2 — TEMPORAL GRANULARITY (CHOOSE ONE)

Goal: Show that changing the grouping key changes what the data reveal.

Choose one of the following prompts:

- (a) Compute the Moon's average distance by day for one month of your choice. Identify the day with the smallest average distance.
- (b) Compute average distance by quarter (Q1–Q4 of 2026).

’ Daily aggregates should show smooth, gradual trends. Quarter-level averages should be even more stable and close to the yearly mean of ~384,400 km.
Hint for quarters: TO_CHAR(datetime, 'Q') returns '1', '2', '3', or '4'.

✓ What to submit
1. State which option (a or b) you chose.
2. The SQL query.
3. The full query output.
4. A brief verification note as a comment.

ISSN: 2473-4901
v12 n6648

TASK 3 — EXTREME MONTHS (HAVING)

Goal: Use HAVING to filter groups by an aggregate condition; distinguish it from WHERE.

Answer both prompts:

3. Which months in 2026 had a maximum Earth–Moon distance greater than 406,400 km?
4. Which months had a minimum Earth–Moon distance under 364,000 km?

’ Use HAVING (not WHERE) when your filter involves an aggregate function.
Only some months should satisfy each threshold — if every month appears, re-check your aggregate.

✓ What to submit

For each of the two prompts:

1. The SQL query.
2. The full query output.

3. One combined verification note explaining how you confirmed both result sets look right.

TASK 4 — LARGEST APPARENT DIAMETER

Goal: Retrieve the exact row(s) at a global maximum; use the result as a sanity check.

Prompt: Find the exact date and time when the Moon’s apparent diameter (DIAM) was largest in 2026. Report that timestamp, the diameter value, and the distance.

’ Self-check: Once you have the row(s), verify two things:
(1) Is DIAM the highest value in the whole table?
(2) Is DIST on that row consistent with “closer = larger apparent diameter”?

✓ What to submit

1. The SQL query.
2. The full query output.
3. Note how many rows were returned and whether that was expected.
4. A brief verification note (2–3 sentences) confirming the physical consistency of the result.

ISSN: 2473-4901
v12 n6648

TASK 5 — REFLECTION

Goal: Move the student from passive syntax execution to active data verification. By asking them to articulate their 'stop signals.' The goal is to instill the habits of plausibility checking that separate novices from database professionals.

Deliverable: In 3–5 sentences (as a SQL comment block in your submission file), describe one result that initially looked wrong or surprising. What did you check, and what change fixed it or confirmed it was actually correct?

✓ **What to submit**

A SQL comment block (--) with your 3–5 sentence reflection, placed after your Task 4 output.
Example format:

```
-- Task 5 Reflection:
-- My Module 3 query initially returned all 12 months. I realized I had put MAX(dist) > 406400
-- in the WHERE clause instead of HAVING. Moving it to HAVING fixed the result.
```

ISSN: 2473-4901
v12 n6648

Grading Rubric (5 points per task)

The same rubric applies to every query task (Tasks 1–4). Task 5 is graded on the Verification Note criterion only.

Criterion	What is checked	Points
A. Executable (Syntax)	Query runs without database errors	0–1
B. Correct result (Semantic/Logical)	Output answers the prompt — right grouping key, correct aggregate, expected row count	0–2
C. Reasonable design	Avoids unnecessary complexity; no redundant predicates; sargable datetime expressions preferred	0–1
D. Verification note	Comment explains what plausibility check was used and what it revealed	0–1

Total: 5 points per query task. Task 0 (prediction) and Task 5 (reflection) are graded holistically on effort and reasoning, not on reaching a specific numeric answer.

ISSN: 2473-4901
v12 n6648

APPENDIX C. MOON ASSIGNMENT 2

Submission Instructions

Submit a single plain-text file (.txt) containing all your SQL statements and their output. Use the following SQL Developer commands to capture everything automatically:

```
-- At the top of your script file:
SET ECHO ON
SPOOL assignment2_YOURNAME.txt

-- ... all your queries go here ...

SPOOL OFF
```

ISSN: 2473-4901
v12 n6648

Extensions Overview

Complete the extension(s) assigned by your instructor. Each is independent — you do not need to finish all three.

Extension	Topic	Key SQL Concepts
A	Multidimensional Aggregation	GROUP BY ROLLUP / CUBE, GROUPING()
B	Window Functions & Time-Series	LAG / LEAD, RANK / DENSE_RANK, moving average
C	Performance & Plan-Aware Design	Sargable predicates, EXPLAIN PLAN, index access

ℹ Before You Begin
Dataset reminder: moon26 has 8,760 rows (one per hour, full year 2026).
Columns: DATETIME (TIMESTAMP), PHASE, AGE, DIAM, DIST (all NUMBER).
For Extension C, an index on DATETIME may be required — ask your instructor if:
CREATE INDEX idx_moon26_datetime ON moon26 (DATETIME); has been created.

EXTENSION A — MULTIDIMENSIONAL AGGREGATION (ROLLUP / CUBE)

Best fit: Advanced SQL, data warehousing / BI, or OLAP reporting modules.

Task A.1 — Monthly Averages with ROLLUP

Write a single query that returns monthly averages of Earth–Moon distance (DIST) plus a grand total in one result set using ROLLUP. Label each row so it is clear whether it is a detail row or a subtotal row (use GROUPING() or a CASE expression).

’ Self-check: The grand-total average must match SELECT ROUND(AVG(dist),1) FROM moon26. Expected: approximately 384,400 km. A 2-column ROLLUP also adds a yearly subtotal row.

✓ What to submit
1. The SQL query.
2. The full output (detail rows + subtotal row(s)).
3. The result of SELECT ROUND(AVG(dist),1) FROM moon26 as a cross-check.
4. A comment confirming the grand total matches.

Task A.2 — Two-Dimensional CUBE (Optional / as assigned)

Derive a phase_category column from PHASE (New = PHASE < 10 or PHASE > 90; Full = PHASE BETWEEN 45 AND 55; Quarter = everything else). Write a CUBE query crossing time period and phase_category to produce subtotals for every combination.

ISSN: 2473-4901
v12 n6648

’ A CUBE on two dimensions with n_1 and n_2 groups produces $(n_1 + 1) \times (n_2 + 1)$ rows.
All phase-category averages of DIST should be close to each other (±3,000 km of ~384,400 km).
Phase does not determine orbital distance — the orbit is elliptical, not phase-locked.

✓ **What to submit**
1. The SQL query (including the phase_category derivation).
2. The full output.
3. A comment (2–3 sentences): what do the phase-category subtotals reveal about the relationship between lunar phase and Earth–Moon distance?

EXTENSION B — WINDOW FUNCTIONS & TIME-SERIES ANALYSIS (LAG / LEAD)

Best fit: Analytic SQL or data analysis modules.

Task B.1 — Hour-to-Hour Change in Distance

Write a query that computes, for each row: DATETIME, DIST, the previous hour’s distance (using LAG), and the difference (current − previous) labeled delta_dist. The first row will have a NULL delta.

’ Typical hour-to-hour delta: a few km. Values larger than ~100 km/hr signal an ORDER BY error.

✓ **What to submit**
1. The SQL query.
2. The first 10 rows of output (FETCH FIRST 10 ROWS ONLY) to show structure.
3. A comment stating the approximate range of delta_dist values you observed.

Task B.2 — Ranking the Largest Changes

Using a CTE or subquery based on Task B.1, rank all rows by the absolute value of delta_dist in descending order. Return the top 10 rows with DATETIME, DIST, delta_dist, and rank. Use both RANK() and DENSE_RANK() and note any difference in the output.

’ Large changes should cluster near perigee or apogee transitions, not scatter randomly.
If RANK() and DENSE_RANK() produce different counts in the top 10, explain why in your note.

✓ **What to submit**
1. The SQL query.
2. The top-10 output showing both RANK and DENSE_RANK columns.
3. A comment (2–3 sentences): what date is rank 1, and is the magnitude physically plausible?

ISSN: 2473-4901
v12 n6648

Task B.3 — Moving Average (Optional / as assigned)

Add a 24-hour trailing moving average of DIST using a window frame. Return DATETIME, DIST (raw), and the moving average (labeled dist_ma24). Retrieve one full month of rows.

’ Trailing 24-hour window syntax: ROWS BETWEEN 23 PRECEDING AND CURRENT ROW
The moving average should be visibly smoother than the raw hourly DIST column.

✓ **What to submit**

1. The SQL query.
2. The first 30 rows of output.
3. A comment (2–3 sentences) describing how the smoothed trend differs from the raw values.

ISSN: 2473-4901
v12 n6648

EXTENSION C — PERFORMANCE & PLAN-AWARE QUERY DESIGN

Best fit: Query optimization, indexing, or execution plan units.

Task C.1 — Non-Sargable Query and Plan

Run the following query and capture its EXPLAIN PLAN output:

```
EXPLAIN PLAN FOR
  SELECT * FROM moon26
  WHERE  TO_CHAR(datetime, 'YYYY-MM') = '2026-03';

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

ℹ Note on EXPLAIN PLAN

The EXPLAIN PLAN statement itself produces no rows — it only stores the plan.
DBMS_XPLAN.DISPLAY reads and formats the stored plan.
Both statements must appear in your SPOOL file.

' Before looking at the plan: predict whether Oracle can use an index on DATETIME for this query.
Write your prediction as a comment before the EXPLAIN PLAN statement.

✓ What to submit

1. Your prediction as a comment (1–2 sentences).
2. The EXPLAIN PLAN FOR statement and the SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY) output.
3. The COUNT(*) for this query: SELECT COUNT(*) FROM moon26 WHERE TO_CHAR(datetime,'YYYY-MM')='2026-03';

Task C.2 — Sargable Rewrite and Plan

Rewrite the same filter using a sargable range predicate on the bare DATETIME column (no function wrapping). Capture the EXPLAIN PLAN for the rewrite.

' Oracle TIMESTAMP literal syntax: TIMESTAMP '2026-03-01 00:00:00'
Form: WHERE datetime >= TIMESTAMP '2026-03-01 00:00:00' AND datetime < TIMESTAMP '2026-04-01 00:00:00'

✓ What to submit

1. The sargable SQL query.
2. The EXPLAIN PLAN FOR statement and DBMS_XPLAN.DISPLAY output.
3. The COUNT(*) for the rewritten query, confirming it matches Task C.1.

ISSN: 2473-4901
v12 n6648

Task C.3 — Written Analysis

Answer the following questions as a comment block in your submission file:

1. Confirm the row counts from Tasks C.1 and C.2 are identical. State the count.
2. What access method appears in each plan? (e.g., TABLE ACCESS FULL vs. INDEX RANGE SCAN)
3. If Oracle chose a full scan on the sargable version, explain why the optimizer might do this on a small development table, and how you could force index use for demonstration purposes.
4. In a production table with millions of rows, which version would you deploy? Justify in 2–3 sentences.

✓ What to submit

A comment block answering all four questions, labeled Q1–Q4, placed after your C.2 plan output.

ISSN: 2473-4901
v12 n6648

Grading Rubric (5 points per task)

The rubric applies to every task. For tasks that ask for written analysis (B.2 comment, C.3), criterion B (Correct result) is assessed on the accuracy and specificity of your explanation.

Criterion	What is checked	Points
A. Executable (Syntax)	Query runs without errors; EXPLAIN PLAN executes cleanly	0–1
B. Correct result	Output answers the prompt; expected row counts match; analysis is accurate	0–2
C. Reasonable design	No unnecessary complexity; sargable predicates used where applicable	0–1
D. Verification note	Comment confirms plausibility or cross-checks the result numerically	0–1

APPENDIX D. ASSESSMENT AND FEEDBACK

This teaching note emphasizes technical correctness while keeping grading lightweight. In practice, most novice SQL errors fall into four categories: syntax errors, semantic errors, logical errors, and complications. Each category calls for a different kind of feedback. The Moon dataset supports faster grading because physically implausible outputs (for example, impossible distances or erratic trends) act as visible stop signals that make semantic and logical mistakes easier to detect without an answer key.

A lightweight rubric (per query)

Use the same rubric for each module to keep feedback consistent and reduce instructor workload.

Criterion	What you're checking	Suggested scoring
A. Executable (Syntax)	Query runs without database errors	0/1
B. Result is correct (Semantic/Logical)	Output answers the prompt as stated (right grouping key, correct filter placement, correct aggregates)	0/2
C. Reasonable design (Complications)	Avoids unnecessary complexity (gratuitous DISTINCT, redundant subqueries, needless GROUP BY columns, etc.)	0/1
D. Verification note (Process)	1–3 sentences: what plausibility check they used and what they changed when something looked wrong	0/1

Total: 5 points per query (or scale to your course norms).

How to apply the four error categories (feedback shorthand)

Syntax errors (does not run): Feedback goal: get students to a runnable baseline quickly.

- Typical feedback: "Fix the error so the query executes; then re-check the grouping key."

Semantic errors (runs, but meaning is misaligned): Feedback goal: reconnect the SQL structure to the question's intent.

- Typical signals: grouping on the wrong time grain; mixing row filtering with group filtering; returning the wrong columns for the stated question.
- Typical feedback: "Your query returns a table, but it is not answering the question at the requested grain yet; revise the grouping key and aggregate accordingly."

Logical errors (runs, but wrong for this prompt/data): Feedback goal: correct a specific mistake in logic or constraint handling.

- Typical signals: incorrect threshold direction; wrong ordering; selecting a max or min but returning non-max rows; off-by-one time windows.
- Typical feedback: "You are close, but the logic is reversed or incomplete; check the predicate and the aggregate used."

Complications (correct result, unnecessary complexity): Feedback goal: teach elegance and maintainability after correctness is secured.

- Typical signals: redundant subqueries; extra GROUP BY columns that do not change the result; formatting expressions used in predicates; function-wrapped filters that inhibit indexing (if you teach that).
- Typical feedback: "Correct result; now simplify. Remove redundant steps and prefer sargable predicates on datetime when appropriate."

Minimal instructor spot checks by module (fast grading)

These checks let you evaluate correctness quickly without recomputing the full solution.
Monthly average distance (Module 1): Should return 12 rows for a single year; values should cluster near the known mean with no wild month-to-month jumps.
HAVING extremes (Module 3): Should return a small subset of months, not all months; students should use HAVING (not WHERE) for aggregate thresholds.
Largest apparent diameter timestamp (Module 4): Should return the row(s) at the global maximum diameter; distance should be consistent with closer corresponds to larger apparent diameter.

Recommended feedback practice

- Prioritize correctness over optimization in the introductory sequence; treat complications as a small deduction or a revision prompt.
- Require the Verification Note even when the numeric answer is wrong; it rewards scientific reasoning and makes debugging visible.
- If you allow revisions, ask students to resubmit only: (a) corrected SQL and (b) one sentence stating what the verification cue revealed.

Instructor note (applying the error taxonomy to grading): Exemplary queries earn full credit, and complications lose only a few points. Semantic and logical errors earn partial credit, with the deduction scaled by how much the results change. Syntax errors lose the most points, since the query does not run.
Instructor note (LLM-generated answers): LLMs typically add a WHERE year = 2026 predicate whenever a question mentions a year. Because the table contains a single year of data, this predicate is a complication—technically harmless but unnecessary. Instructors may choose to grade this complication more strictly than usual: it is a reliable marker of unedited LLM output and creates a natural opening for a conversation about the appropriate role of LLMs in course assignments. Telling students in advance that the data covers only one year makes clear that the filter is not required.