**Case Study: Airport Parking Management Toolkit**

**Use Case Summary**

|  |  |  |
| --- | --- | --- |
| **#** | **Use Case** | **Description** |
| 1 | Vehicle Entry/Exit Logging | Capture when a vehicle enters/exits a zone, with timestamps and amount paid. |
| 2 | Validate Parking Records | Ensure exit time is not before entry time, and paid amount is not negative. |
| 3 | Analyze Zone Utilization | Identify top-performing zones by revenue or usage. |
| 4 | Track Frequent Parkers | List vehicles with the highest number of visits or longest durations. |
| 5 | Detect Data Quality Issues | Nulls in vehicle IDs, overlaps in time ranges, invalid timestamps. |
| 6 | Compare Parking Zone Performance | Compare zones (e.g., short-term vs valet) by occupancy and revenue. |
| 7 | Package, Test, and Reuse ETL | Build a reusable Python package for ingestion, transformation, and validation. |
| 8 | Enable Scalable Historical Analysis | Use Spark to analyze large volumes of archived parking data. |

**Functional Requirements**

|  |  |
| --- | --- |
| Component | Requirement Description |
| CLI Tool | Load CSV data to clean, validate, write Parquet, and log events |
| Data Validation | Ensure proper timestamps, null handling, vehicle ID checks |
| Unit Testing | Parametrized tests, edge-case coverage > 80% |
| Packaging | Poetry-based package with pre-commit hooks |
| SQL Schema | Normalized schema with PK/FK relationships |
| SQL Queries | Joins, groupings, filters, rankings, triggers |
| Analytics SQL | CTEs, window functions, MERGE/UPSERT, stored procedures |
| Spark Optimization | .cache(), .explain(), broadcast join, partition tuning |

**Technical Topics Covered**

|  |  |
| --- | --- |
| Track | Concepts |
| Python I | Object model, mutability, truthy/falsy, dict/set ops, logging, argparse CLI, CSV to Parquet |
| Python II | Decorators, comprehensions, testing, poetry, fixtures, Git hooks, wheel publishing |
| SQL I | PK/FK, SELECT, JOIN, GROUP BY, NULL handling, CRUD, EXPLAIN, indexing, ACID |
| SQL II | Window functions, CTEs, MERGE, triggers, stored procedures, isolation levels |
| Spark Fundamentals | DAG, DataFrame API, cache, checkpoint, Spark UI, Parquet read, broadcast |

**Tables and Schema**

|  |  |
| --- | --- |
| Table | Key Columns |
| vehicles | vehicle\_id, plate\_number, type, owner\_name |
| parking\_zones | zone\_id, zone\_name, rate\_per\_hour, is\_valet |
| parking\_events | event\_id, vehicle\_id, zone\_id, entry\_time, exit\_time, paid\_amount |

**Assignment Tasks**

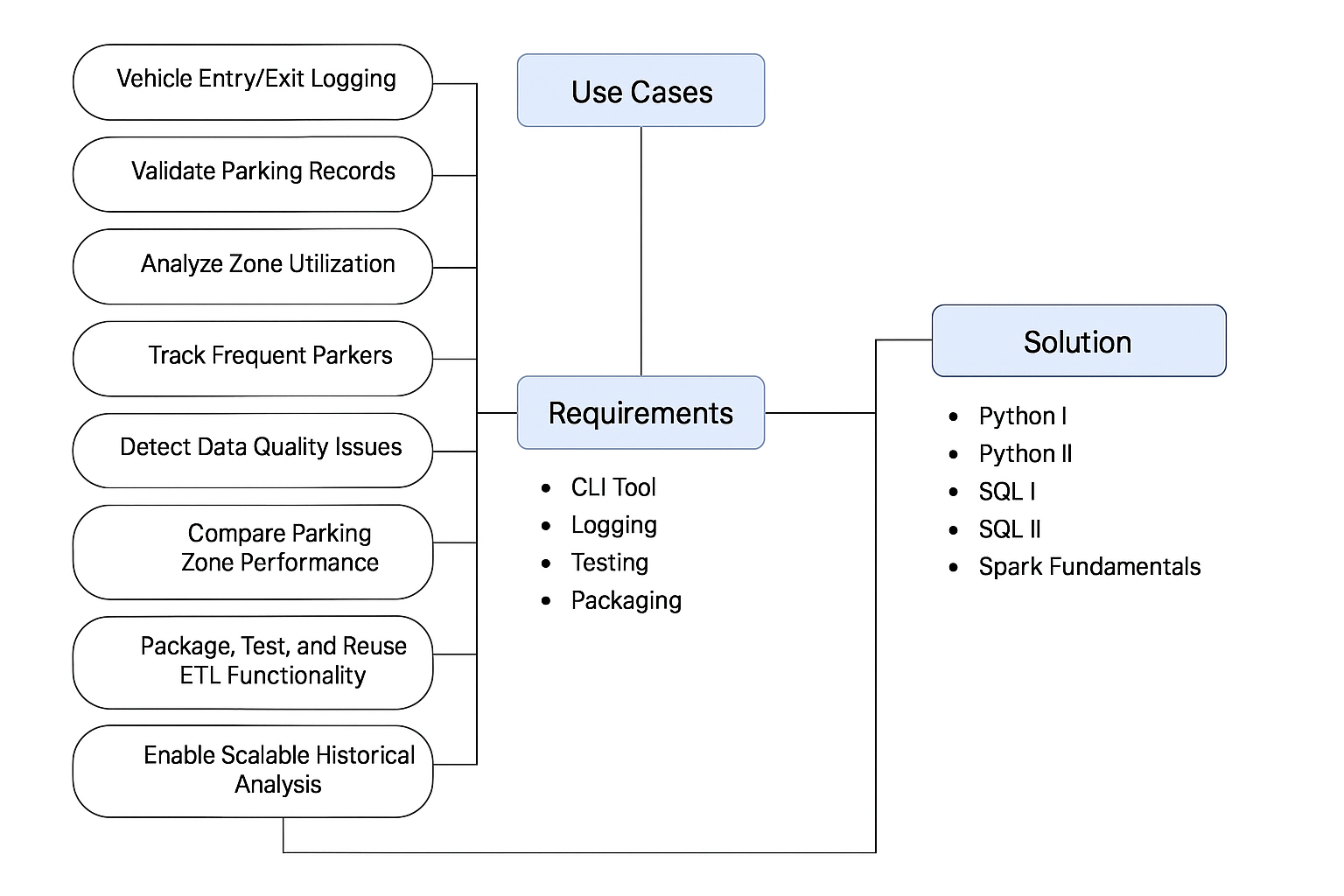
Participants are required to design and implement a unified solution that addresses all of the above use cases and functional requirements using the following components:

1. **Python CLI Script** to ingest and validate data, with logging and structured outputs.
2. **Python Packaging & Testing** with >80% test coverage, use of fixtures, decorators, and Git pre-commit hooks.
3. **SQL Schema Design & Queries** for relational joins, indexing, and windowed analytics.
4. **Advanced SQL** for stored procedures, triggers, and transaction integrity.
5. **Spark Job** to process large Parquet datasets efficiently with partition tuning and broadcast optimization.

**Expected Deliverables**

|  |  |
| --- | --- |
| Component | Description |
| cli\_tool.py | Python script for ETL (ingest-clean-validate-write) |
| airport\_parking\_toolkit/ | Python module with CLI, validation, and logging logic |
| tests/ | PyTest-based unit tests for all modules |
| parking\_etl.log | Structured log file for all ETL runs |
| coverage/ | Report showing test coverage ≥ 80% |
| schema.sql | DDLs for all tables with constraints |
| queries.sql | Join, rank, aggregation, trigger queries |
| spark\_job.py | PySpark script for Parquet processing and performance tuning |

**Flow Diagram**



**Evaluation Rubric**

|  |  |
| --- | --- |
| Criteria | Weight |
| Python CLI functionality | 20% |
| Unit test coverage + PyTest usage | 15% |
| SQL schema normalization + relational queries | 15% |
| Analytics SQL logic (rank, lag, trigger) | 15% |
| Spark performance & DAG optimization | 20% |
| Packaging, logging, and CLI usability | 15% |