Lakehouse and the Query Layer :-

Bronze layer optimises the storage of raw data.
Silver layer provides access to full history of data, with guaranteed validation, & data quality.

Then comes the Query Layer / Serving Layer / Gold Layer

The Query Layer is primarily the end goal of the lakehouse, enabling analytics and ML on optimised and highly refined data.

What is the Query Layer?

- Stores refined datasets for use by data scientists
- Serves results for pre-computed ML models
- Contains enriched, aggregated views for use by analysts
- Star-schemas and data marts for BI queries
- Powers data-driven applications, dashboards, and reports

Also called the serving layer; gold tables exist at this level.

⇒ Contains a mix of Tables & Views

⇒ In this layer, ACLs are imposed heavily to limit access to data.

⇒ Where I how data is stored in this layer (i.e Tables/ Views, saved queries) depends on a no. of factors.

⇒ Various logics can be implemented, to update the Gold Tables — in near-realtime, or in batches hourly, daily, etc.

⇒ Major compute job to create these tables are executed On Write.

⇒ Reading gold table requires deserialization.

⇒ With Views, we store the physical plan/query that computes results each time when queried. It does not store the results.

Views should be avoided when the results generation needs joins & aggregations of large no. of records or tables. It increases latency, and is computationally expensive, and users needing access to the results, will also need access to powerful clusters.

⇒ For these reasons, Databricks SQL Saved queries are better:-

⇒ Databricks SQL Saved Queries:-

⇒ Similar to saved views, as generate results by executing the query, when they are queried. But efficient because:- They also cache the data & end results, but, checks whether or not the source delta tables are changed, after last execution.

If source tables donot changed, upon being queried, they simply return the last cached results. Thus, saves computational costs on frequent returns by multiple users.

Computes the results only if source tables changes.

⇒ The results are cached in Customer's Object Storage, and thus, data access during dashboarding do not need access to clusters. and can also download the results as CSV, without an active Cluster or SQL warehouse

The queries keeps the results in the cache updated as soon as source data changes.

≥ Databricks SQL Saved Query results can be scheduled to get refreshed automatically, thus, dashboards data will also get periodically updated.

≥ Databricks SQL Endpoints :-

≥ Databricks SQL endpoints are Clusters optimised for SQL queries.

≥ Has serverless options for Quick Startup and Auto Scaling.

≥ Photon-enabled for vectorized execution.

≥ Enhanced throughput for data exchange with external systems.

≥ Data, where to be stored for optimum throughput & transfer latencies, should be decided.
If data needs to be regularly processed using Databricks, using external storage system could be costly, and time-taking.

Sample approach of an optimisation case:-

⇒ Suppose, data for sales are recorded for an entire nation, in a bronze table.
A regional office needs sales data for a particular state.
We can provide them this data, by creating a view, that will only fetch that state data, & give the office access to that, instead of giving access to entire data.
This will reduce user's need of computational cost.

Also, in this scenario, data will be provided optimally if we partition the data in files based on region. This will make greater efficiency, as for providing the data, only specific files needs to be accessed, thus computationally beneficial.

(2)

⇒ Column renaming needs can be solved by creating views with column name aliases.
This will not change the schema of the original production table.

⇒ By Default, ACLs are not enforced for standard Data Engg Clusters So :-

```
%sql
spark.sql(" SHOW GRANT ON <view or table name> ")
```

⇓

these queries will generate exceptions.

⇒ ACLs are primarily intended for managing data access in Databricks workspace for BI & Data Science use cases.

⇒ Results obtained by querying Materialised/Saved views, are stored in Delta Lake for efficient deserialization.