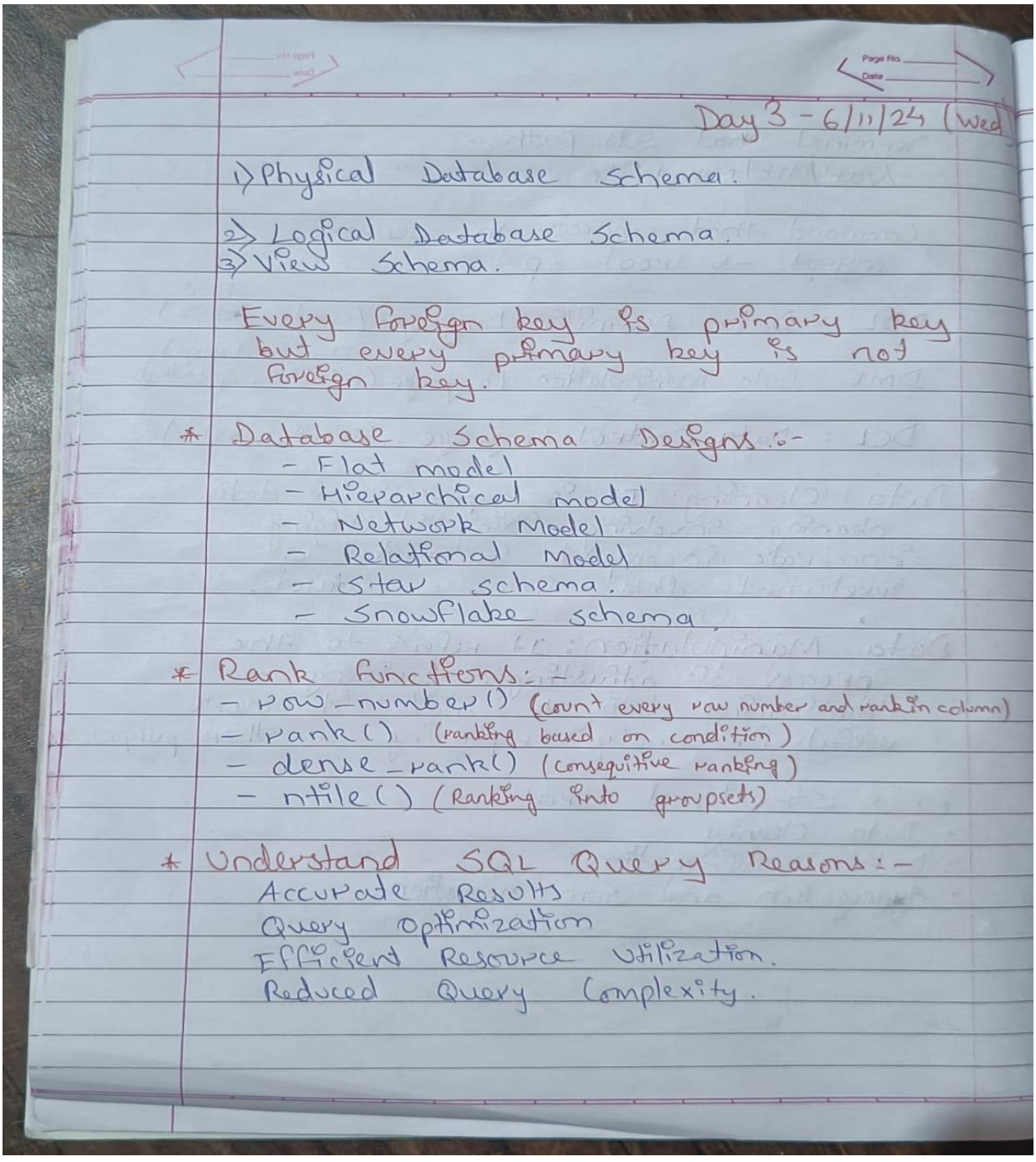
Summary of Day 3 – Data Engineering

-Sarthak Niranjan Kulkarni (Maverick)

- sarthakkul2311@gmail.com - (+91) 93256 02791

Day 3 – 6/11/2024 (Wednesday)

- **Handwritten Notes:-**



Day 3 - 6/11/24 (Wed)

1) Physical Database Schema.

2) Logical Database Schema.
3) View Schema.

Every foreign key is primary key but every primary key is not foreign key.

* Database Schema Designs :-
  - Flat model
  - Hierarchical model
  - Network Model
  - Relational Model
  - Star schema.
  - Snowflake schema.

* Rank functions :-
  - row_number() (count every row number and rank in column)
  - rank() (ranking based on condition)
  - dense_rank() (consequitive ranking)
  - ntile() (Ranking into groupsets)

* Understand SQL Query Reasons :-
  Accurate Results
  Query optimization
  Efficient Resource Utilization.
  Reduced Query Complexity.

Order (FWGHSOL) of execution.

The ROLLUP allows us to generate hierarchical subtotal rows according to its input columns and it also adds a grand total row to the result set.

* Aggregate Functions

- AVG()
- SUM()
- COUNT()
- MIN()
- MAX()
- VARIANCE()
- STDEV()

➢ Digital Notes: -

1. Types Of Database Schema: -

→ There are 3 types of Database Schema

- **Physical Database Schema**: Defines the actual storage of data on hardware, detailing files, indexes, and storage structures for performance optimization.
- **Logical Database Schema**: Outlines the logical structure of the database, including tables, relationships, and constraints, independent of physical storage.
- **View Schema**: Defines virtual tables or views that present specific data to users, often aggregating or filtering data without changing the underlying tables.

---

2. Types of Database Schema Designs: -

→ Types are mentioned below: -

- Flat Model
- Hierarchical Model
- Network Model
- Relational Model
- Star Schema
- Snowflake Schema

---

3. Rank Functions: -

→ **Rank functions** in SQL are used to assign a rank to each row within a partition of a dataset, based on the specified order:

- **ROW_NUMBER()**: Assigns a unique sequential number to each row within a partition, without gaps.
- **RANK()**: Assigns a rank with gaps in case of ties (rows with equal values share the same rank, and the next rank skips accordingly).

- **DENSE_RANK()**: Similar to RANK() but without gaps; ties share the same rank, and the next rank continues sequentially.
- **NTILE(n)**: Divides the rows into n equal buckets or groups, assigning a unique bucket number to each row, ideal for percentile-based ranking.

---

4. Why to understand Query Reasons: -

→ Understanding the reasons behind SQL queries is crucial for effective database management, as it helps to ensure that queries are well-optimized, accurate, and aligned with the goals of data retrieval. By analyzing the purpose of a query—whether it's for reporting, data transformation, or specific business logic—developers and analysts can write more efficient queries that reduce processing time and server load, ultimately enhancing performance. This understanding also aids in identifying areas where indexes, partitioning, or other optimizations could be applied to speed up query execution. Moreover, clear query reasoning allows for better troubleshooting, as it provides context when unexpected results or performance bottlenecks arise. Lastly, understanding query intent is key to maintaining data integrity and security, as it allows for controlled and relevant access to sensitive or critical information within a database system.

---

5. Order Of Execution: -

→ In SQL Server (and most relational database management systems), the order of execution of SQL queries follows a specific logical sequence, often abbreviated as **FWGHSOL**. Understanding this order is crucial for writing efficient queries, troubleshooting issues, and ensuring the intended results are achieved.

 FROM:

- The first step in query execution is to determine the source of the data, which can be a table, view, or join operation. SQL Server identifies which table(s) the query will operate on, and if there are joins, it resolves them at this stage.

 WHERE:

- Next, the filter conditions are applied. The WHERE clause eliminates rows from the data set that do not meet the specified conditions, reducing the number of rows that move forward in the query execution process.

 GROUP BY:

- If the query includes grouping, the GROUP BY clause is applied next. It groups rows based on the specified column(s) and prepares the data for aggregation (using functions like SUM(), COUNT(), etc.).

 HAVING:

- After grouping, the HAVING clause is applied to filter out groups that do not meet certain conditions. This is similar to the WHERE clause but operates on aggregated data.

 SELECT:

- Once filtering is done, the SELECT clause is evaluated, which determines which columns or expressions will appear in the final result set. This is the part where you specify the output of your query.

 ORDER BY:

- After the data has been selected and grouped, the ORDER BY clause is applied to sort the result set based on one or more columns, either in ascending or descending order.

 LIMIT/OFFSET (or **TOP** in SQL Server):

- Finally, if the query includes a limit on the number of rows returned (using LIMIT, OFFSET, or TOP in SQL Server), it's applied last, restricting the result set to the specified number of rows.

---

6. Aggregate Functions: -

→ **Aggregate functions** are essential tools used to perform calculations on multiple rows of data and return a single, summarized value. These functions are commonly used for data analysis, reporting, and to gain insights into large datasets by grouping and summarizing data.

SUM():

- The SUM() function calculates the total of a numeric column. It is widely used for financial calculations, such as totaling sales or expenses.

  " **SELECT SUM(salary) FROM employees;** "

COUNT():

- This function counts the number of rows or non-NULL values in a column. It's useful for finding the number of entries that meet certain criteria.

" **SELECT COUNT(employee_id) FROM employees WHERE department = 'Sales';** "

AVG():

- AVG() calculates the average of a numeric column, providing a mean value. This is often used for metrics like average salary, average sales, etc.

  " **SELECT AVG(salary) FROM employees;** "

MIN() and MAX():

- MIN() returns the smallest value, and MAX() returns the largest value in a column. These functions are helpful in finding range values, like the lowest or highest price, age, or date.

  " **SELECT MIN(salary), MAX(salary) FROM employees;** "

STDEV() and **VAR()**:

- These functions calculate statistical measures: STDEV() computes the standard deviation, and VAR() computes the variance of a numeric column. These are used to understand data dispersion or variation.

GROUPING Functions:

- SQL Server offers functions like GROUPING_ID() to identify if a row is part of a subtotal or grand total. This is particularly useful in complex aggregations with GROUP BY ROLLUP or GROUPING SETS for hierarchical grouping.
