MySQL - RDBMS

Trainer: Mr. Rohan Paramane

Codd’s rules

- Proposed by Edgar F. Codd – pioneer of the RDBMS – in 1980.
- If any DBMS follow these rules, it can be considered as RDBMS.

- The 0th rule is the main rule known as “The foundation rule”.
  - For any system that is advertised as, or claimed to be, a relational data base management system, that system must be able to manage data bases entirely through its relational capabilities.

- The rest of rules can be considered as elaboration of this foundation rule.

Codd’s rules

- Rule 1: The information rule:
  - All information in a relational data base is represented explicitly at the logical level and in exactly one way – by values in tables.
- Rule 2: The guaranteed access rule:
  - Each and every datum (atomic value) in a relational data base is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.
- Rule 3: Systematic treatment of null values:
  - Null values (distinct from the empty character string or a string of blank characters and distinct from zero or any other number) are supported in fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of data type.

Codd’s rules

- Rule 4: Dynamic online catalog based on the relational model:
  - The data base description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

- Rule 5: The comprehensive data sublanguage rule:
  - A relational system may support several languages. However, there must be at least one language that supports all functionalities of a RDBMS i.e. data definition, data manipulation, integrity constraints, transaction management, authorization.

Codd’s rules

- Rule 6: The view updating rule:
  - All views that are theoretically updatable are also updatable by the system.
- Rule 7: Possible for high-level insert, update, and delete:
  - The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update and deletion of data.
- Rule 8: Physical data independence:
  - Application programs and terminal activities remain logically unbroken whenever any changes are made in either storage representations or access methods.
- Rule 9: Logical data independence:
  - Application programs & terminal activities remain logically unbroken when information-preserving changes of any kind that theoretically permit un-impairment are made to the base tables.

Codd’s rules

- Rule 10: Integrity independence:
  - Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

- Rule 11: Distribution independence:
  - The end-user must not be able to see that the data is distributed over various locations. Users should always get the impression that the data is located at one site only.

- Rule 12: The non-subversion rule:
  - If a relational system has a low-level (single-record-at-a-time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher level relational language (multiple-records-at-a-time).

Normalization

- Concept of table design: Table, Structure, Data Types, Width, Constraints, Relations.
- Goals:
  - Efficient table structure.
  - Avoid data redundancy i.e. unnecessary duplication of data (to save disk space).
  - Reduce problems of insert, update & delete.
- Done from input perspective.
- Based on user requirements.
- Part of software design phase.
- View entire appln on per transaction basis & then normalize each transaction separately.
- Transaction Examples:
  - Banking, Rail Reservation, Online Shopping.

Normalization

- For given transaction make list of all the fields.
- Strive for atomicity.
- Get general description of all field properties.
- For all practical purposes we can have a single table with all the columns. Give meaningful names to the table.
- Assign datatypes and widths to all columns on the basis of general desc of fields properties.
- Remove computed columns.
- Assign primary key to the table.
- At this stage data is in un-normalized form.
- UNF is starting point of normalization.
- UNF suffers from
  - Insert anomaly
  - Update anomaly
  - Delete anomaly

Normalization – 1 NF

- 1. Remove repeating group into a new table.
- 2. Key elements will be PK of new table.
- 3. (Optional) Add PK of original table to new table to give us Composite PK.
    - Repeat steps 1-3 infinitely -- to remove all repeating groups into new tables.
    - This is **1-NF**. No repeating groups present here. One to Many relationship between two tables.

Normalization – 2 NF

- 4. Only table with composite PK to be examined.
- 5. Those columns that are not dependent on the entire composite PK, they are to be removed into a new table.
- 6. The key elements on which the non-key elements were originally dependent, it is to be added to the new table, and it will be the PK of new table.
  - Repeat steps 4-6 infinitely -- to separate all non-key elements from all tables with composite primary key.
  - This is **2-NF**. Many-to-Many relationship.

Normalization – 3 NF

- 7. Only non-key elements are examined for inter-dependencies.
- 8. Inter-dependent cols that are not directly related to PK, they are to be removed into a new table.
- 9. (a) Key ele will be PK of new table.
- 9. (b) The PK of new table is to be retained in original table for relationship purposes.
  - Repeat steps 7-9 infinitely to examine all non-key eles from all tables and separate them into new table if not dependent on PK.
  - This is **3-NF**.

Normalization – 4NF (BCNF)

- To ensure data consistency (no wrong data entered by end user).
- Separate table to be created of well-known data. So that min data will be entered by the end user.
- This is BCNF or 4-NF.

De-normalization

- Normalization will yield a structure that is non-redundant.
- Having too many inter-related tables will lead to complex and inefficient queries.
- To ensure better performance of analytical queries, few rules of normalization can be compromised.
- This process is de-normalization.

Window Functions

- Window functions are added in SQL-2003. Supported in MySQL 8.0+.
- Most powerful tool to solve typical analytical problems.
- Aggregate functions operate on group of rows and generates summary (fewer rows).
- Window functions also operate on group of rows, but not reduce number of rows.
- Unlike grouping, windowing retain the other columns.
- Windowing enable dividing data into multiple partitions, sorting each partition and
- perform window operations on each row.

Window Functions

- Window functions are of two types
  - Aggregate functions
    - Can be used with or without windowing.
    - SUM(), AVG(), MAX(), MIN(), COUNT(), STDEV(), ...
  - Non-aggregate functions
    - Can be used with windowing only.
    - ROW_NUMBER(), RANK(), DENSE_RANK(), FIRST_VALUE(), LAST_VALUE(), LEAD(), LAG(), ...
- SELECT window_function(…) _x0003_OVER (window_specification), col1, col2 FROM table;
- window_function(…) OVER (window_specification)
  - PARTITION BY columns
  - ORDER BY columns ASC | DESC ✓
  - ROWS | RANGE BETWEEN frame_start AND frame_end

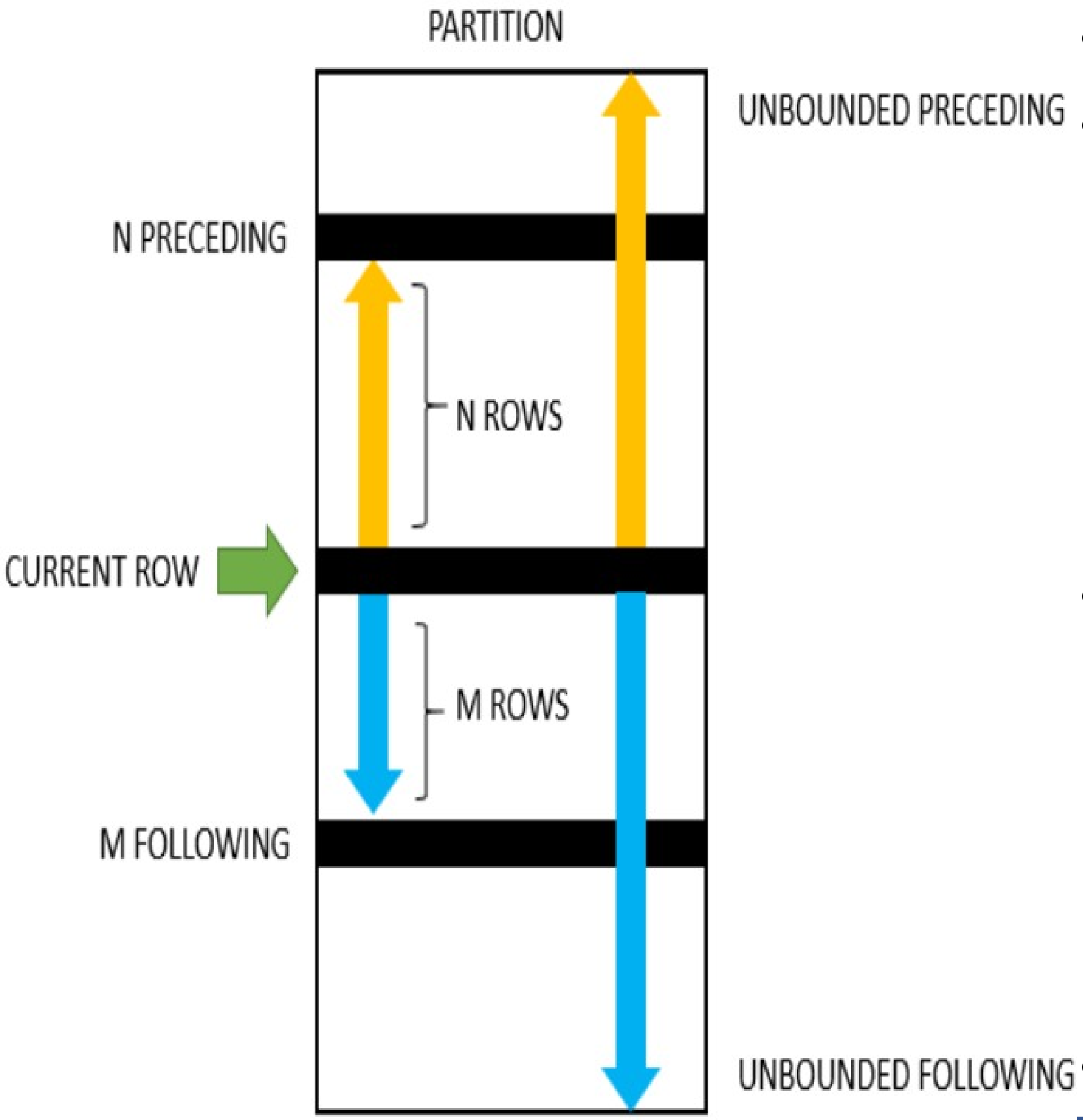
Window Functions

- Partition
  - Breaks up rows into partitions/groups.
  - Rows can be partitioned by one or more columns/expressions.
  - Window function is performed within partitions and re-initialized when crossing partition boundary.
- Order
  - Sorts rows within a partition in ASC or DESC order.
  - Rows can be sorted on one or more columns/expressions.
  - Partition and Order is supported by all window functions.
  - Order by is useful only for order-sensitive functions e.g. ROW_NUMBER(), RANK(), etc.
- ❯ Frame
  - Frame is a subset of current partition.
  - Frame is defined w.r.t. current row and is moving within a partition depending on position of thecurrent row.
  - The default frame is RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW.

Window Functions

Name	Description
ROW_NUMBER	Assigns a sequential integer to every row within its partition
RANK	Assigns a rank to every row within its partition based on the ORDER BY clause. It assigns the same rank to the rows with equal values.
DENSE_RANK	Same as RANK(), but if two or more rows have the same rank, then there will be no gaps in the sequence of ranked values.
PERCENT_RANK	Calculates the percentile rank of a row in a partition. (RANK() – 1) / (NumberOfRows – 1)
FIRST_VALUE	Returns the value of the specified expression with respect to the first row in the window frame.
LAST_VALUE	Returns the value of the specified expression with respect to the last row in the window frame.
LAG	Returns the value of the Nth row before the current row in a partition. It returns NULL if no preceding row exists.
LEAD	Returns the value of the Nth row after the current row in a partition. It returns NULL if no subsequent row exists.
NTH_VALUE	Returns value of argument from Nth row of the window frame
NTILE	Distributes the rows for each window partition into a specified number of ranked groups.
CUME_DIST	Calculates the cumulative distribution of a value in a set of values. Same as COUNT(*) OVER (ORDER BY Col1) / COUNT(*) OVER ().

Window Frame



- ROWS/RANGE BETWEEN start AND end.
- start is one of:
  - UNBOUNDED PRECEDING: The window starts in the first row of the partition
  - CURRENT ROW: The window starts in the current row
  - N PRECEDING or M FOLLOWING
- end is one of:
  - UNBOUNDED FOLLOWING: The window ends in the last row of the partition
  - CURRENT ROW: The window ends in the current row
  - N PRECEDING or M FOLLOWING
- ROWS count number of rows, while RANGE follows number of unique values in given column.

Thank you!

Rohan Paramane<rohan.paramane@sunbeaminfo.com>