MySQL - RDBMS

Trainer: Mr. Nilesh Ghule

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.

insert anomaly
update anomaly
delete anomaly

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.

SQL Keys

- An SQL key is either a single column (or attribute) or a group of columns that can uniquely identify rows (or tuples) in a table.
- Super key is a single key or a group of multiple keys that can uniquely identify tuples in a table.
- Candidate key is a single key or a group of multiple keys that uniquely identify rows in a table.
- Primary key is the Candidate key selected by the database administrator to uniquely identify tuples in a table.
- Alternate keys are those candidate keys which are not the Primary key.
- Foreign key is an attribute which is a Primary key in its parent table, but is included as an attribute in another related table.

SQL Keys

empno	ename	address	dept	job	ssn/aadhar	passport	email	phone
1	A	PUNE	10	ANALYST	111100001111	11111111	a@sun.com	9822012345
2	B	DELHI	10	CLERK	222200002222	22222222	b@sun.com	9422012345
3	C	MUMBAI	30	PRESIDENT	333300003333	33333333	c@sun.com	9822123456
4	D	CHENNAI	20	MANAGER	444400004444	44444444	d@sun.com	9422123456
5	E	GOA	20	CLERK	555500005555	55555555	e@sun.com	9822123321
S	S	S			S	S	S	S
C					C	C	C	C
PK					AK	AK	AK	AK

Normalization

- 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

- 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

- 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

- 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.

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 0^{th} 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).

MySQL Exceptions

- Exceptions are runtime problems, which may arise during execution of stored procedure, function or trigger.
- Required actions should be taken against these errors.
- SP execution may be continued or stopped after handling exception.
- MySQL error handlers are declared as:
  - DECLARE *action* HANDLER FOR *condition handler_impl*;
- The *action* can be: CONTINUE or EXIT.
- The *condition* can be:
  - MySQL error code: e.g. 1062 for duplicate entry.
  - SQLSTATE value: e.g. 23000 for duplicate entry, NOTFOUND for end-of-cursor.
  - Named condition: e.g. DECLARE duplicate_entry CONDITION FOR 1062;
- The *handler_impl* can be: Single liner or PSM block i.e. BEGIN … END;

MySQL Stored Functions

- Stored Functions are MySQL programs like stored procedures.
- Functions can be having zero or more parameters. MySQL allows only IN params.
- Functions must return some value using RETURN statement.
- Function entire code is stored in system table. information_schema.routines
- Like procedures, functions allows statements like local variable declarations, if-else, case, loops, etc. One function can invoke another function/procedure and vice-versa. The functions can also be recursive.
- There are two types of functions: DETERMINISTIC and NOT DETERMINISTIC.

CREATE FUNCTION

```
CREATE FUNCTION fn_name(p1 TYPE)
RETURNS TYPE
[NOT] DETERMINISTIC
BEGIN
   body;
   RETURN value;
END;
```

SHOW FUNCTION

```
SHOW FUNCTION STATUS LIKE 'fn_name';

SHOW CREATE FUNCTION fn_name;
```

DROP FUNCTION

```
DROP FUNCTION IF EXISTS fn_name;
```

SELECT fn_name(...);

MySQL Triggers

- Triggers are supported by all standard RDBMS like Oracle, MySQL, etc.
- Triggers are not supported by WEAK RDBMS like MS--Access.
- Triggers are not called by client's directly, so they don't have args & return value.
- Trigger execution is caused by DML operations on database.
  - BEFORE/AFTER INSERT, BEFORE/AFTER UPDATE, BEFORE/AFTER DELETE.
- Like SP/FN, Triggers may contain SQL statements with programming constructs. They may also call other SP or FN.
- However COMMIT/ROLLBACK is not allowed in triggers. They are executed in same transaction in which DML query is executed.

CREATE TRIGGER

```
CREATE TRIGGER trig_name
AFTER|BEFORE dml_op ON table
FOR EACH ROW
BEGIN
   body;
   -- use OLD & NEW keywords
   -- to access old/new rows.
   -- INSERT triggers - NEW rows.
   -- DELETE triggers - OLD rows.
END;
```

SHOW TRIGGERS

```
SHOW TRIGGERS FROM db_name;
```

DROP TRIGGER

```
DROP TRIGGER trig_name;
```

MySQL Triggers

- Applications of triggers:
  - Maintain logs of DML operations (Audit Trails).
  - Data cleansing before insert or update data into table. (Modify NEW value).
  - Copying each record AFTER INSERT into another table to maintain "Shadow table".
  - Copying each record AFTER DELETE into another table to maintain "History table".
  - Auto operations of related tables using cascading triggers.

- Cascading triggers
  - One trigger causes execution of 2nd trigger, 2nd trigger causes execution of 3rd trigger and so on.
  - In MySQL, there is no upper limit on number of levels of cascading.
  - This is helpful in complicated business processes.
- Mutating table error
  - If cascading trigger causes one of the earlier trigger to re-execute, "mutating table" error is raised.
  - This prevents infinite loop and also rollback the current transaction.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>