Database

Normalization

- Is a step-by-step operation to replace the relationships between data into two dimensional tabular form, in order to reduce the redundancy of data, increase the integrity and to free the relations from undesirable insertion, update, and deletion dependency.
- Normalization Objectives:

1. To make it possible to tabulate any relation in D.B.
2. To free relation from undesirable insertion, update, and deletion dependence.
3. To obtain a powerful retrieved capability by means of simple collection of relational operators to manipulate the relations.
4. To reduce the redundancy of data, achieve data independence, increase the integrity of data.
5. To make the relation more informative to the user.
6. To reduce the need for restructuring the relations as a new type of data are introduced when further applications or users view requires them.

FUNCTIONAL DEPENDENCY

Given a relation R, we say that attribute Y of R is functionally dependent on attribute X of R if each X value in R has associated with it precisely one Y-value in R.

A functional dependency is a relationship between two sets of attributes in a database table. It describes how the value of one attribute determines the value of another attribute.
Basically, a functional dependency is represented as X → Y, where X and Y are sets of attributes. This notation implies that any two rows in the table with the same value for X will also have the same value for Y. In other words, the value of Y is functionally dependent on the value of X.
Let's consider an example to illustrate functional dependencies. Suppose we have a table called "Employees" with attributes such as employee ID, name, department, and salary. If we define a functional dependency as "department → salary," it means that for any two employees who belong to the same department, their salaries will be the same. This allows us to infer the salary of an employee based on their department alone.

First normal form (FNF)

A relation R is in first normal form if all underlying domains contains atomic value only (no repeated group).

Second normal form (SNF)

A relation R is in second normal form iff:

- It is in 1NF.
- Every non-key attribute is depend on all parts off primary key.

Third Normal form (TNF)

A relation R is in third normal form iff:

- It is in 2NF.
- Every non-key attribute is depending on the primary key directly (Not transitively).

Structured Query Language (SQL)

- SQL known as SQL or simply “sequel,” is a programming language that communicates through relational databases. It is the easiest way to store, update, remove, search, or retrieve information on a database.
- The SQL language has several parts:-

1. The Data Manipulation Language (DML): This subset of SQL allows users to pose queries and to insert, delete, and modify rows.
2. The Data Definition Language (DDL): This subset of SQL supports the creation, deletion, and modification of definitions for tables and views.
3. Triggers and Advanced Integrity Constraints: The new SQL:1999 standard includes support for *triggers.*

4. Embedded SQL: Embedded SQL features allow SQL code to be called from a host language such as C or COBOL.

5. Dynamic SQL: Features allow SQL statement to constructed and executed at runtime based on input parameters.

6. Client-Server Execution and Remote Database Access: These commands control how a *client* application program can connect to an SQL database *server*, or access data from a database over a network.

7. Transaction Management (TCL): Various commands allow a user to explicitly control aspects of how a transaction is to be executed.

8. Security (DCL): SQL provides mechanisms to control users' access to data objects such as tables and views.

five main categories of SQL each with its own commands as shown in the table (1) below

Commands categories	Command	Command descriptions
(DML) Data Manipulation language	SELECT	Retrieve information from table
	INSERT	Insert information to table or object
	UPDATE	Edit information on table or object
	DELETE	delete information on table or object
(DML) Data Manipulation language	CREATE	Create table or object
	Alter	Edit table or object
	DROP	Delete table or object
	RENAME	Rename table or object
	TRUNCATE	Delete a part of table or object
Transaction Control	COMMIT	Commit the inf. On table
	ROLLBACK	Rollback on the new info.
	SAVEPOINT	Go back to the point where the system Is consistence
Transaction Control	GRANT	Give users access to the info.
	REVOKE	Blocking the user's access of info.

Data Retrieval (SELECT):

What	Who	Example
All columns	SELECT * FROM table;	SELECT * FROM Students;
Some columns	SELECT column1,column2,... FROM table;	SELECT LastName, FirstName FROM Students;
Some rows/columns	SELECT column1,column2,... FROM table [WHERE condition(s)];	SELECT LastName,FirstName FROM Students WHERE StudentID LIKE '%123%
No Repeats	SELECT [DISTINCT] column(s) FROM table;	SELECT DISTINCT LastName FROM Students;
Ordering	SELECT column1,column2,... FROM table [ORDER BY column(s) [DESC]];	SELECT LastName,FirstName FROM Students ORDER BY LastName, FirstName DESC;
Column Aliases	SELECT column1 [AS alias1], column2 [AS alias2], ... FROM table1;	SELECT LastName,FirstName AS First FROM Students;

We can use the four Arithmetic Operators (+ , -, * , /) with select command and Some operator used with WHERE condition

BETWEEN AND	NOT (BETWEENAN)
IN( )	NOT IN( )
LIKE	NOT LIKE( )
IS NULL	Is Not NULL

Advanced Database Structures/ view

- A view is tables whose rows are not explicitly stored in the database but are computed as needed from a view definition. A database view is a searchable object in a database that is defined by a query. Though a view doesn't store data, some refer to a views as "virtual tables," you can query a view like you can a table. A view can combine data from two or more table, and also just contain a subset of information. We create view using SQL commands.

- **What is the difference between table and view?**

  A view is a virtual table. A view consists of rows and columns just like a table. The difference between a view and a table is that views are definitions built on top of other tables (or views), and do not hold data themselves. If data is changing in the underlying table, the same change is reflected in the view. A view can be built on top of a single table or multiple tables. It can also be built on top of another view.

- **Views offer the following advantages:**

1. Ease of use: A view hides the complexity of the database tables from end users. Essentially we can think of views as a layer of abstraction on top of the database tables.
2. Space savings: Views takes very little space to store, since they do not store actual data.
3. Additional data security: Views can include only certain columns in the table so that only the non-sensitive columns are included and exposed to the end user. In addition, some databases allow views to have different security settings, thus hiding sensitive data from prying eyes.

Trigger

- TRIGGERS AND ACTIVE DATABASES A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA.
- A database that has a set of associated triggers is called an active database.
- A trigger description contains three parts:
- - Event: A change to the database that activates the trigger.
- - Condition: A query or test that is run when the trigger is activated.
- - Action: A procedure that is executed when the trigger is activated and its condition is true.

Trigger Syntax and Examples in MySQL

- To create a trigger or drop a trigger, use the CREATE TRIGGER or DROP TRIGGER statement.
- Syntax :

CREATE TRIGGER trigger_name trigger_time trigger_event ON tbl_name

FOR EACH ROW trigger_body trigger_time: { BEFORE | AFTER }

- trigger_event: { INSERT | UPDATE | DELETE }
- Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.
- Trigger_time is the trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after each row to be modified.

Trigger_event indicates the kind of operation that activates the trigger.

These trigger_event values are permitted:

 INSERT: The trigger activates whenever a new row is inserted into the table; for example, through INSERT statements.

 UPDATE: The trigger activates whenever a row is modified; for example, through UPDATE statements.

 DELETE: The trigger activates whenever a row is deleted from the table; for example, through DELETE statements. DROP TABLE and TRUNCATE TABLE statements on the table do not activate this trigger, because they do not use DELETE.

Trigger_body is the statement to execute when the trigger activates. To execute multiple statements, use the BEGIN ... END compound statement construct.

Within the trigger body, you can refer to columns in the subject table (the table associated with the trigger) by using the aliases OLD and NEW. OLD.col_name refers to a column of an existing row before it is updated or deleted. NEW.col_name refers to the column of a new row to be inserted or an existing row after it is updated.

Here is a simple example that associates a trigger with a table, to activate for INSERT operations.

The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account FOR EACH ROW
SET @sum = @sum + NEW.amount;
```

In the example, the trigger body is a simple SET that accumulates into a user variable the values inserted into the amount column. The statement refers to the column as NEW.amount which means “the value of the amount column to be inserted into the new row.”

The CREATE TRIGGER statement creates a trigger named ins_sum that is associated with the account table. It also includes clauses that specify the trigger action time, the triggering event, and what to do when the trigger activates:

- The keyword BEFORE indicates the trigger action time. In this case, the trigger activates before each row inserted into the table. The other permitted keyword here is AFTER.
- The keyword INSERT indicates the trigger event; that is, the type of operation that activates the trigger. In the example, INSERT operations cause trigger activation. You can also create triggers for DELETE and UPDATE operations.
- The statement following FOR EACH ROW defines the trigger body; that is, the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering event.

To use the trigger, set the accumulator variable to zero, execute an INSERTstatement, and then see what value the variable has afterward:

```
 mysql> SET @sum = 0;
 mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
 +----------------------+
 | Total amount inserted |
+----------------------+
| 1852.48 |
 +----------------------+
```

In this case, the value of @sum after the INSERT statement has executed is 14.98 + 1937.50 - 100, or 1852.48.

To destroy the trigger, use a DROP TRIGGER statement.

You must specify the schema name if the trigger is not in the default schema:

mysql> DROP TRIGGER test.ins_sum;

If you drop a table, any triggers for the table are also dropped.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema.

Triggers in different schemas can have the same name.

In an **INSERT trigger,** only NEW.col_name can be used; there is no old row.

In a **DELETE trigger**, only OLD.col_name can be used; there is no new row.

In an **UPDATE trigger**, you can use OLD.col_name to refer to the columns of a row before it is updated and NEW.col_name to refer to the columns of the row after it is updated.

By using the BEGIN ... END construct, you can define a trigger that executes multiple statements. Within the BEGIN block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the mysql program to define a trigger that executes multiple

statements, it is necessary to redefine the mysql statement delimiter so that you can use the ; statement delimiter within the trigger definition.

The **DELIMITER statement** changes the standard delimiter which is semicolon (;) to another. The delimiter is changed from the semicolon (;) to double-slashes //. Why do we have to change the delimiter? To pass the trigger, stored procedure to the server as a whole rather than letting mysql tool to interpret each statement at a time.

The following example illustrates these points. It defines an UPDATE trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a BEFORE trigger because the value must be checked before it is used to update the row:

```
mysql> delimiter //

 mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account

 -> FOR EACH ROW

-> BEGIN

-> IF NEW.amount < 0 THEN

-> SET NEW.amount = 0;

-> ELSEIF NEW.amount > 100 THEN

->SET NEW.amount = 100;

-> END IF;

 -> END;//

 mysql> delimiter ;
```

SQL :Join

- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
- Let's look at a selection from the "person" table:
- Then, look at a selection from the "color" table:
- Notice that the "fk" column in the "person" table refers to the "id" in the "color" table.

```
mysql> select *from color;
+------+--------+
| id   | color  |
+------+--------+
|    1 | read   |
|    2 | green  |
|    3 | blue   |
+------+--------+
```

```
mysql> select * from person;
+----+-------+------+
| id | name  | fk   |
+----+-------+------+
|  1 | steve |    1 |
|  2 | aron  |    3 |
|  3 | mary  | NULL |
+----+-------+------+
```

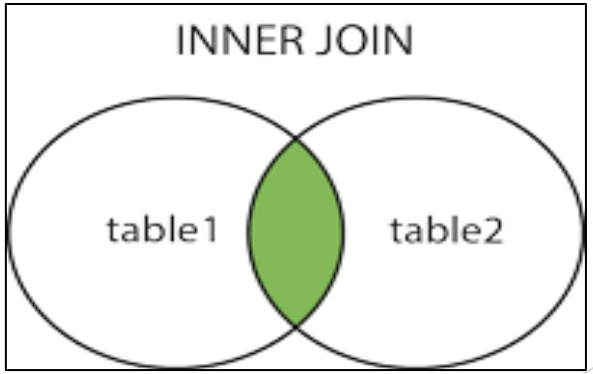
Then, we can create the following SQL statement (that contains an INNER JOIN), that selects records that have matching values in both tables:

The MySQL INNER JOIN clause matches rows in one table with rows in other tables and allows you to query rows that contain columns from both tables.

Here are the different types of the JOINs in SQL:

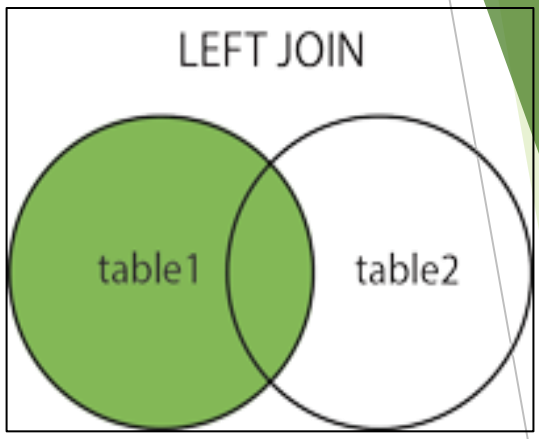
- **(INNER) JOIN**: Returns records that have matching values in both tables

```
mysql> select * from person inner join color on person.fk=color.id;
+----+-------+------+------+-------+
| id | name  | fk   | id   | color |
+----+-------+------+------+-------+
|  1 | steve |    1 |    1 | read  |
|  2 | aron  |    3 |    3 | blue  |
+----+-------+------+------+-------+
```



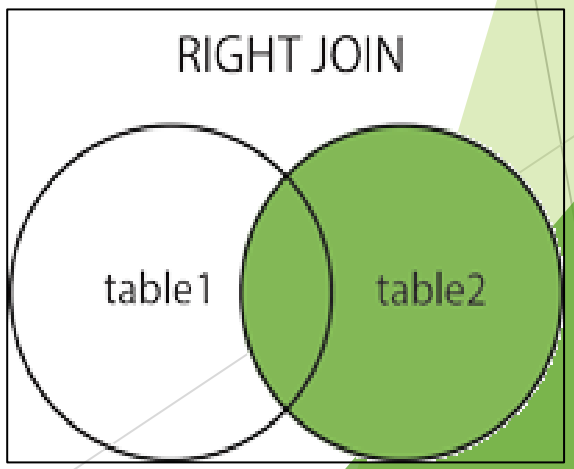
- **LEFT (OUTER) JOIN**: Return all records from the left table, and the matched records from the right table

```
mysql> select * from person left join color on person.fk=color.id;
+----+-------+------+------+-------+
| id | name  | fk   | id   | color |
+----+-------+------+------+-------+
|  1 | steve |    1 |    1 | read  |
|  2 | aron  |    3 |    3 | blue  |
|  3 | mary  | NULL | NULL | NULL  |
+----+-------+------+------+-------+
```



- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table

```
mysql> select * from person right join color on person.fk=color.id;
+------+-------+------+------+-------+
| id   | name  | fk   | id   | color |
+------+-------+------+------+-------+
|    1 | steve |    1 |    1 | read  |
|    2 | aron  |    3 |    3 | blue  |
| NULL | NULL  | NULL |    2 | green |
+------+-------+------+------+-------+
```



- **Cross join** : Return all records when there is a match in either left or right table

```
mysql> select * from person cross join color ;
+----+-------+------+------+-------+
| id | name  | fk   | id   | color |
+----+-------+------+------+-------+
|  1 | steve |    1 |    1 | read  |
|  2 | aron  |    3 |    1 | read  |
|  3 | mary  | NULL |    1 | read  |
|  1 | steve |    1 |    2 | green |
|  2 | aron  |    3 |    2 | green |
|  3 | mary  | NULL |    2 | green |
|  1 | steve |    1 |    3 | blue  |
|  2 | aron  |    3 |    3 | blue  |
|  3 | mary  | NULL |    3 | blue  |
+----+-------+------+------+-------+
9 rows in set (0.00 sec)
```