Agenda

- What are MySQL's Aggregate Functions?
- Join and Union Operators
- How to deal with Timestamps and Dates in MySQL
- What are MySQL Triggers?
- Datatype Conversions in MySQL

In previous session we've learned how to perform basic CRUD operations on MySQL databases. Let's now dive deeper and learn about some mo advanced features of MySQL and how we can query data from multiple tables contained in the same database.

Aggregate Functions

Sometimes when working with the data stored in a MySQL database table, we are interested not in the data itself, but in statistics about that data. F example, whilst we may not be concerned about the specific content in each row, we may want to know how many rows are in a table. Alternatively, w may need to find the average of all the values in a particular table column. Information of this type can be obtained using a collection of built- MySQL *aggregate functions.*

Unless otherwise stated, aggregate functions ignore `NULL` values.

The aggregate functions are often used with the `GROUP BY` clause to calculate an aggregate value for each group e.g., the average value by the grou or the sum of values in each group. If you use an aggregate function in a statement containing no `GROUP BY` clause, it is equivalent to grouping on a rows.

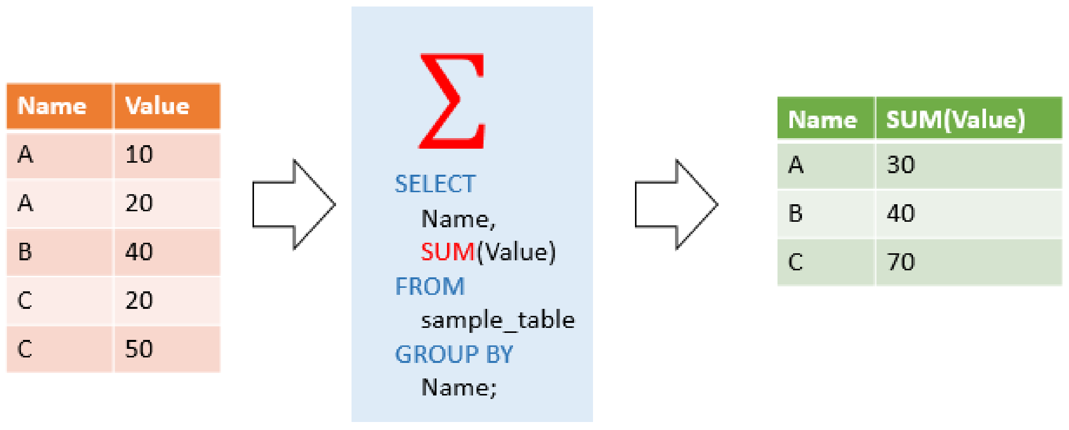
Syntax

```
function_name(DISTINCT | ALL expression)
```

In this syntax:

- First, specify the name of the aggregate function e.g., `AVG()` . See the list of aggregate functions in the following section.
- Second, use `DISTINCT` if you want to calculate based on distinct values or `ALL` in case you want to calculate all values including duplicates. The default is `ALL` .
- Third, specify an expression that can be a column or expression which involves column and arithmetic operators.

The following picture illustrates the `SUM()` aggregate function is used in conjunction with a `GROUP BY` clause:



Name	Value
A	10
A	20
B	40
C	20
C	50

```
SELECT
  Name,
  SUM(Value)
FROM
  sample_table
GROUP BY
  Name;
```

Name	SUM(Value)
A	30
B	40
C	70

`AVG(*expr*)`

Returns the average value of ***expr*** . The `DISTINCT` option can be used to return the average of the distinct values of ***expr*** .

Example :

```
SELECT student_name, AVG(test_score)
FROM student
GROUP BY student_name;
```

`COUNT(*expr*)`

Returns a count of the number of non- `NULL` values of ***expr*** in the rows retrieved by a `SELECT` statement. The result is a `BIGINT` value.

`COUNT(*)` is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain `NULL` values.

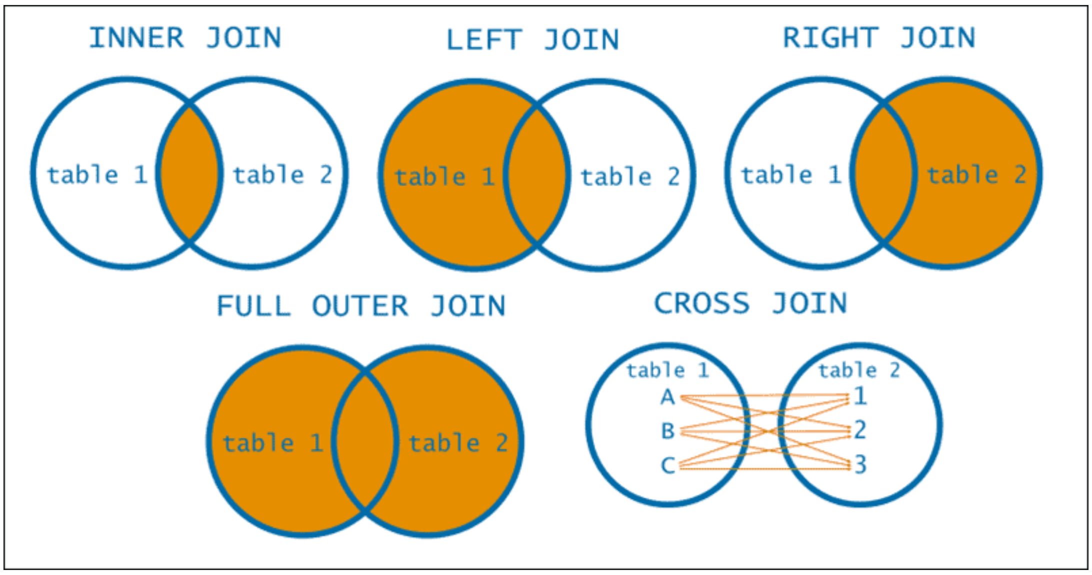
Example :

```
SELECT student.student_name,COUNT(*)
FROM student,course
WHERE student.student_id=course.student_id
GROUP BY student_name;
```

For more info on every Aggregate Function provided by MySQL, go through this link.

Join Operator

A `JOIN` clause is used to combine rows from two or more tables, based on a related column between them. The types of `JOIN` operations can b
classified as :



Inner Join

The simplest join type is `INNER JOIN` . The `INNER JOIN` results with a set of records that satisfy the given condition in joined tables. It matches eac
row in one table with every row in other tables and allows users to query rows containing columns from both tables.

The syntax for an `INNER JOIN` is:

```
SELECT table1.column1, table1.column2, table2.column1, ...
FROM table1
INNER JOIN table2
ON table1.matching_column = table2.matching_column;
```

The `matching_column` syntax represents the column common to both tables.

Since `INNER JOIN` is considered the default join type, using only the `JOIN` statement is accepted.

Left Join

The `LEFT JOIN` (or `LEFT OUTER JOIN`) returns all records from the table on the left side of the join and matching records from the table on the rig
side of the join. If there are rows for which there are no matching rows on the right-side table, the result value displayed is *NULL*.

The syntax for `LEFT JOIN` is:

```
SELECT table1.column1, table1.column2, table2.column1, ...
FROM table1
LEFT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Right Join

The `RIGHT JOIN` (`RIGHT OUTER JOIN`) is essentially the reverse of `LEFT OUTER JOIN` .

The `RIGHT JOIN` returns all records from the table on the right side of the join and matching records from the table on the left side of the join. If the are rows for which there are no matching rows on the left-side table, the result value displayed is *NULL*.

The syntax for `RIGHT JOIN` is:

```
SELECT table1.column1,table1.column2,table2.column1,....
FROM table1
RIGHT JOIN table2
ON table1.matching_column = table2.matching_column;
```

Cross Join

The `CROSS JOIN` (also called `CARTESIAN JOIN`) joins each row of one table to every row of another table. The `CROSS JOIN` happens when tl matching column or the `WHERE` condition are not specified. The result-set of a `CROSS JOIN` is the product of the number of rows of the joined tables

Use `CROSS JOIN` when you want a combination of every row from two tables. `CROSS JOIN` is useful when you want to make a **combination** of item for example, colors or sizes.

The syntax for `CROSS JOIN` is:

```
SELECT table1.column1, table1.column2, table2.column1, ...
FROM table1
CROSS JOIN table2;
```

MySQL does not support `FULL OUTER JOIN` , but the same functionality can be implemented using `UNION` operator about which we'll learn next.

Union Operator

The `UNION` operator is used to combine the result-set of two or more `SELECT` statements.

- Every `SELECT` statement within `UNION` must have the same number of columns
- The columns must also have similar data types
- The columns in every `SELECT` statement must also be in the same order

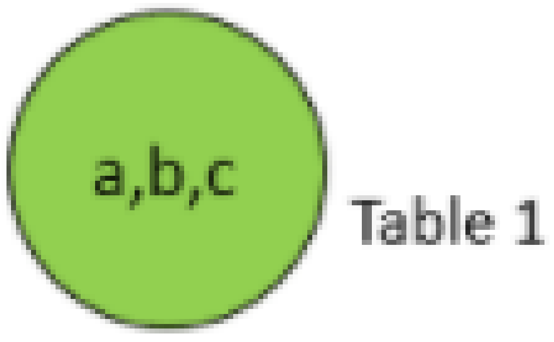
Syntax :

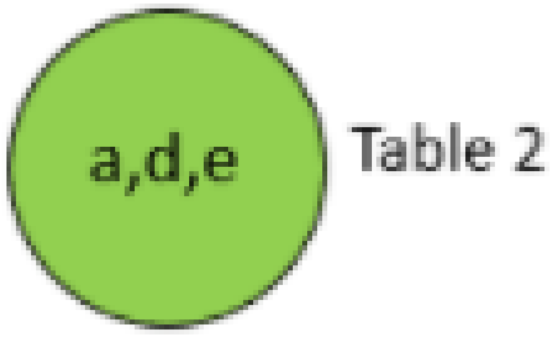
```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

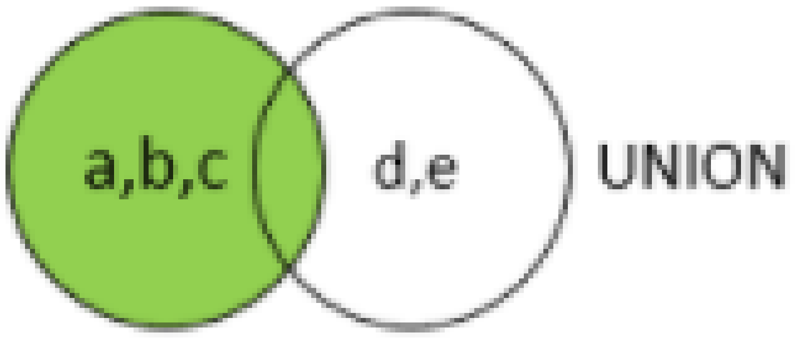
`UNION ALL`

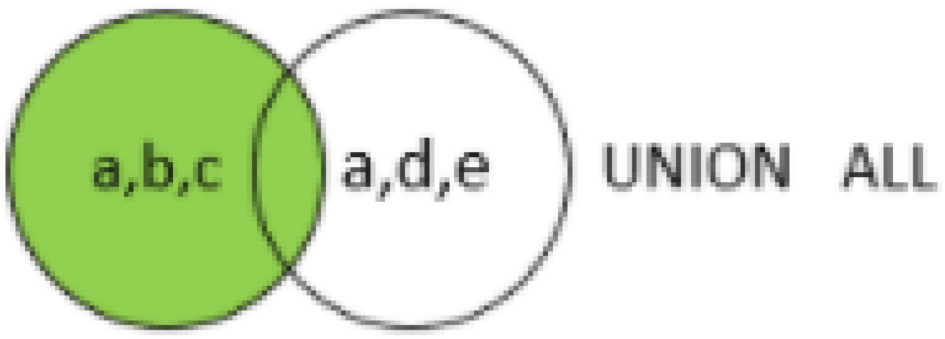
The `UNION` operator selects only distinct values by default. To allow duplicate values, use `UNION ALL` :

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```









Implementing `FULL OUTER JOIN` Using `UNION`

We can use a combination of `LEFT JOIN` , `UNION ALL` , and `RIGHT JOIN` , which outputs a union of table 1 and table 2, returning all records fro both tables. The columns existing only in one table will be displayed as *NULL* in the opposite table.

The syntax is:

```
SELECT * FROM table1
LEFT JOIN table2 ON table1.matching_column = table2.matching_column
UNION ALL
SELECT * FROM table1
RIGHT JOIN table2 ON table1.matching_column = table2.matching_column
```

Dealing with Dates / Timestamps

MySQL stores date and time values using 5 different data types i.e. – `YEAR` , `DATE` , `TIME` , `DATETIME` , and `TIMESTAMP` . Let's try to understar these briefly in the below table, before diving deeper into `DATE FORMATS` and `DATETIME FUNCTIONS` provided by MySQL.

Type	Syntax	Range	Format
YEAR	Used to store just the year part	1901 - 2155	'YYYY'
DATE	Used to store just the DATE part	'1000-01-01' to '9999-12-31'	'YYYY-MM-DD'
TIME	TIME → Used to store just the TIME part		
TIME(n) n -> accuracy or fractional component	'00:00:00' to '23:59:59'	'hh:mm:ss.[fraction]'	
DATETIME	DATETIME → Used to store both DATE and TIME		
DATETIME(n) n -> accuracy or fractional component	'1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'	'YYYY-MM-DD hh:mm:ss.[fraction]'	
TIMESTAMP	TIMESTAMP → Used to store both DATE and TIME		
TIMESTAMP(n) n -> accuracy or fractional component	'1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC.	'YYYY-MM-DD hh:mm:ss.[fraction]'	

Getting Current Date and Time

MySQL has the following functions to get the current date and time:

```
SELECT now();  -- date and time
SELECT curdate(); --date
SELECT curtime(); --time in 24-hour format
```

Adding and Subtracting Dates

MySQL provides functions to add and subtract date and time values to an existing value stored in the column or to be used during querying data.

`ADDDATE()`

This function is used to add a date to an existing date value which could be of type – `DATE` , `DATETIME` , or `TIMESTAMP` .

Example :

```
SELECT ADDDATE('2020-08-15', 31);

// output
'2020-09-15'
```

When an integer value is supplied as the second argument, it is treated as the number of days to be added to the given date.

A particular Interval of time can be specified by using the `INTERVAL` keyword. Examples:

```
SELECT ADDDATE('2020-01-02', INTERVAL 1 MONTH);
//output
2020-02-02

SELECT ADDDATE('2020-01-02 12:11:11', INTERVAL 5 HOUR);
//output
2020-01-02 17:11:11

SELECT ADDDATE('2020-01-02 12:11:11', INTERVAL 1 WEEK);
//output
2020-01-09 12:11:11
```

Other examples of intervals could be – `MINUTE` , `SECOND` , `QUARTER` , etc.

`ADDTIME()`

This function is used to add time component to an existing date value which could be of type – `DATE` , `DATETIME` , or `TIMESTAMP` . The time to b
added should be passed in as second argument in `hh:mm:ss` format. Example :

```
SELECT ADDTIME('2020-01-02 12:11:11', '01:10:00');
// output
2020-01-02 13:21:11
```

`SUBDATE()`

`SUBDATE` is exactly similar to `ADDDATE` , the only difference being, `SUBDATE` subtracts the given Interval/No. of days from the column/field value.

Examples:

```
SELECT SUBDATE('2020-08-15', 31);
// output
2020-07-15

//The below examples show subtracting an Interval mentioned in hours/days week, etc.

SELECT SUBDATE('2020-01-02 12:11:11', INTERVAL 1 WEEK);
// output
2019-12-26 12:11:11

SELECT SUBDATE('2020-01-02 12:11:11', INTERVAL 12 HOUR);
// output
2020-01-02 00:11:11
```

`SUBTIME()`

`SUBTIME` subtracts the time value specified in `hh:mm:ss` from a given datetime or time field value.

Example:

```
SELECT SUBTIME('2020-01-02 12:11:11', '02:20:05');
// output
2020-01-02 09:51:06
```

Converting Dates

MySQL provides a variety of in-built functions from converting dates from one format to another. Let's see the most widely used functions for convertir
dates.

`CONVERT_TZ()`

This function is used to convert the date from one timezone to another. These come handy in situations where suppose your data was stored using UT
timezone and while displaying you want to convert into timezone of your choice.

For example**,** convert UTC to MET (Middle European Time).

```
SELECT CONVERT_TZ('2004-01-01 12:00:00', 'UTC', 'MET');
```

`FROM_UNIXTIME()`

This function is used to convert a given `UNIX TIMESTAMP` to MySQL `datetime` format.

```
SELECT FROM_UNIXTIME(1547430881);
//output
2019-01-14 07:24:41
```

`UNIX_TIMESTAMP()`

This function is used to convert a given MySQL `datetime` to `UNIX timestamp` .

```
SELECT UNIX_TIMESTAMP('2020-03-15 07:10:56.123')
//output
```

```
1584236456.123
```

`UNIX timestamp` is the representation of a given date in the form of seconds elapsed since January 1, 1970, UTC.

Fetching Specific Parts Of DateTime Columns

At times, it is desired to fetch a specific part of the date value for a `datetime` field value in the table. For example, suppose you want to count the n of orders and group them by day, week, month, etc.

Generally speaking, these values are not stored as separate columns in the table – for example**,** you would have just one column like `created_on` which would be a `datetime` field and not having separate columns like – date, month, year, etc.

MySQL provides a lot of useful methods where you can extract the desired part from the `Datetime` field. Let's have a look at some of the widely use functions with examples

- **Extract DAY** `DATE()` **–** Extract the DATE() part of `datetime` field. `DAYOFMONTH()` **–** Extract the DAY part. Its essentially the day of the month. Shorthand for using this function is `DAY()` `DAYOFWEEK()` **–** Extracts the index corresponding to the day of week – values between 1 to 7. `DAYOFYEAR()` **–** Extracts the day in terms of no. of days passed in a year. The values range between 1 and 365 (or 366 if it is a leap year). `DAYNAME()` **–** Extracts the name of the day with values ranging from Monday to Sunday. Examples :

```
SELECT DATE('2020-03-15 07:10:56.123');
//output
2020-03-15

SELECT DAYOFMONTH('2020-03-15 07:10:56.123');
//output
15

SELECT DAYOFWEEK('2020-03-15 07:10:56.123');
//output
1

SELECT DAYOFYEAR('2020-03-15 07:10:56.123');
//output
75

SELECT DAYNAME('2020-03-15 07:10:56.123');
//output
Sunday
```

- **Extract TIME** `TIME()` **–** Extracts the time portion in format `hh:mm:ss` (with fractional seconds component if available). `HOUR()` **–** Extracts the hour part of the field/value – values ranging between 1-24. `MINUTE()` **–** Extracts the minute part of the given `datetime` – values ranging between 1-60. `SECOND()` **–** Extracts the seconds part of the given `datetime` – values ranging between 1-60. `MICROSECOND()` **–** Extracts the microseconds part of the given DateTime. Examples :

```
SELECT TIME('2020-03-15 07:10:56.123');
//output
'07:10:56.123'

SELECT HOUR('2020-03-15 07:10:56.123');
//output
7

SELECT MINUTE('2020-03-15 07:10:56.123');
//output
10

SELECT SECOND('2020-03-15 07:10:56.123');
//output
56

SELECT MICROSECOND('2020-03-15 07:10:56.123');
//output
123000
```

- **Extract MONTH** `MONTH()` **–** Extracts the month index. Values ranging between 1 – 12. `MONTHNAME()` ****– Extracts the month name. Values ranging between January to December. Examples :

```
SELECT MONTH('2020-03-15 07:10:56.123');
//output
3

SELECT MONTHNAME('2020-03-15 07:10:56.123');
//output
March
```

- **Extract WEEK, YEAR, and QUARTER** `WEEK()` – Extracts the WEEK of year for the given DateTime value. `YEAR()` **–** Extracts the YEAR part of the date. `QUARTER()` **–** Extracts the quarter with respect to the year. Values ranging between 1 to 4. Examples :

```
SELECT WEEK('2020-03-15 07:10:56.123');
//output
11

SELECT YEAR('2020-03-15 07:10:56.123');
//output
2020

SELECT QUARTER('2020-03-15 07:10:56.123');
//output
1
```

All these above methods are mostly used for data analysis and making predictions – ****for example, which days of the month were the no. of orde highest, etc.

Finding Difference between two dates

In order to find the difference between 2 given `datetime` field values, MySQL provides `DATEDIFF()` & `TIMEDIFF()` functions.

`DATEDIFF()`

Returns the difference between 2 Datetime (or date) values in no of days.

Examples :

```
SELECT DATEDIFF('2020-01-05 12:11:11.11', '2020-01-03 14:11:11')
//output
2

//DATEDIFF() using Date inputs.
SELECT DATEDIFF('2020-04-02', '2020-01-04')
//output
89

//DATEDIFF() using Date inputs with negative output.
SELECT DATEDIFF('2020-04-02', '2020-04-05')
//output
-3
```

`TIMEDIFF()`

This function returns the difference between 2 datetime (or time) values expressed in `hh:mm:ss` with an optional fractional part depending on the inp the `TIMEDIFF()` function was called with.

Examples :

```
SELECT TIMEDIFF('2020-01-4 12:11:11.11', '2020-01-03 12:11:11')
//output
24:00:00.11

//TIMEDIFF() using Time inputs.
SELECT TIMEDIFF('12:11:11.11', '02:12:11')
```

```
//output
09:59:00.11

//TIMEDIFF() returning negative output.
SELECT TIMEDIFF('2020-01-01 12:11:11.11', '2020-01-03 12:11:11')
//output
-47:59:59.89
```

Formatting Dates

`DATE_FORMAT()` function converts the date in the given format.

For example: Converting the required date in format mm/yyyy.

```
SELECT DATE_FORMAT('2020-03-15 07:10:56.123', '%m/%Y');
//output
03/2020
```

Let's see the common conventions for using format specifiers :

Format	Description
%Y	Year value - 4 digit
%y	Year value - 2 digit
%M	Month name like- January, February
%m	Month name - numeric
%d	Day of month
%h	Hour - 00 - 12
%H	Hour - 00 - 23
%i	Minutes
%S	Seconds

Depending on the requirement these format specifiers can be used and MySQL DateTime can be converted into required format.

Example: Converting to dd/mm/yyyy format can be specified as:

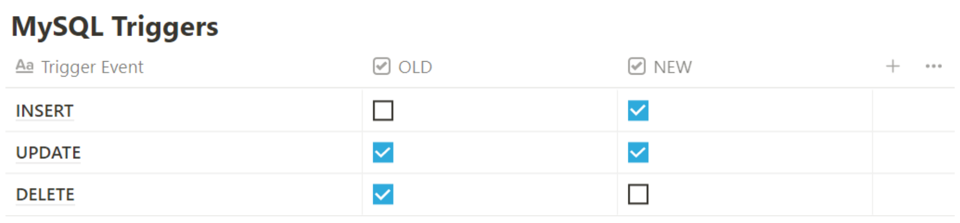
```
SELECT DATE_FORMAT('2020-03-15 07:10:56.123', '%d/%m/%Y');
//output
15/03/2020
```

MySQL Triggers

What is a Trigger in MySQL?

A trigger is a named MySQL object that activates when an event occurs in a table. Triggers are a particular type of stored procedure associated with specific table.

Triggers allow access to values from the table for comparison purposes using **`NEW`** and **`OLD`**. The availability of the modifiers depends on the trigg event you use:


MySQL Triggers

Trigger Event	OLD	NEW
INSERT	☐	☑
UPDATE	☑	☑
DELETE	☑	☐

Checking or modifying a value when trying to insert data makes the `NEW.<column name>` modifier available. This is because a table is updated wi new content. In contrast, an `OLD.<columnname>` value does not exist for an insert statement because there is no information exists in its plac beforehand.

When updating a table row, both modifiers are available. There is `OLD.<colum name>` data which we want to update to `NEW.<column name>` data.

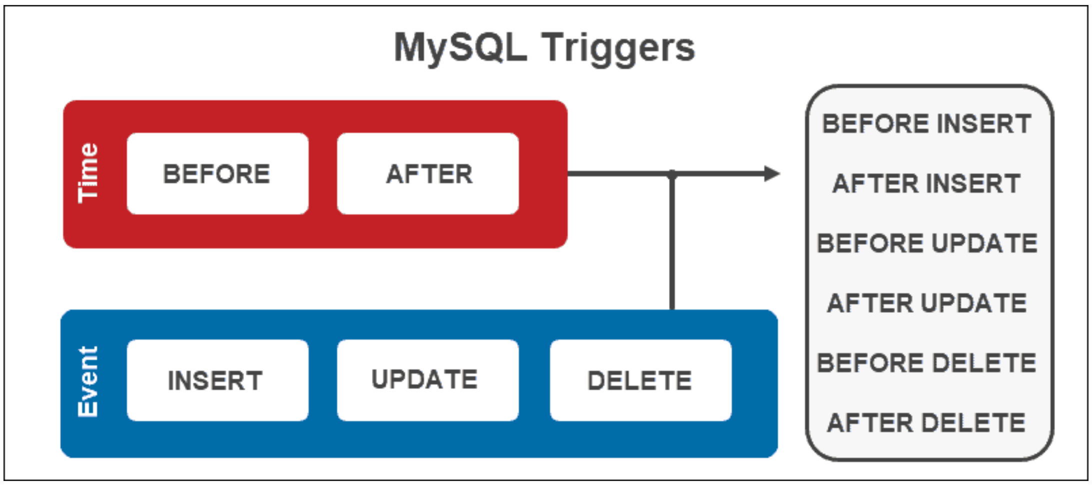
Finally, when removing a row of data, the `OLD.<column name>` modifier accesses the removed value. The `NEW.<column name>` does not exi because nothing is replacing the old value upon removal.

Using MySQL Triggers

Every trigger associated with a table has a unique name and function based on two factors:

1. Time. `BEFORE` or `AFTER` a specific row event.

2. Event. `INSERT` , `UPDATE` or `DELETE` .



MySQL triggers fire depending on the activation time and the event for a total of six unique trigger combinations. The before statements help to chec data and make changes before making commitments, whereas the after statements commit the data first and then execute statements.

The execution of a set of actions happens automatically, affecting all inserted, deleted, or updated rows in the statement.

Create Triggers

Use the `CREATE TRIGGER` statement syntax to create a new trigger:

```
CREATE TRIGGER <trigger name> <trigger time > <trigger event>
ON <table name>
FOR EACH ROW
<trigger body>;
```

The best practice is to name the trigger with the following information:

```
<trigger time>_<table name>_<trigger event>
```

For example, if a trigger fires **before insert** on a table named **employee**, the best convention is to call the trigger:

```
before_employee_insert
```

Delete Triggers

To delete a trigger, use the `DROP TRIGGER` ****statement:

```
DROP TRIGGER <trigger name>;

// Alternatively, you can use :
DROP TRIGGER IF EXISTS <trigger name>;
```

An example trigger to check if an employee's age is above 18, before inserting the values into DB table would look like the following :

```
delimiter //
CREATE TRIGGER person_bi BEFORE INSERT
ON person
FOR EACH ROW
IF NEW.age < 18 THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Person must be older than 18.';
END IF; //
delimiter ;
```

NOTE: A MySQL client uses the delimiter (`;`) to separate statements and executes each statement separately. However, a stored procedure consis of multiple statements separated by a semicolon (`;`).

If you use a MySQL client program to define a stored procedure that contains semicolon characters, the MySQL client program will not treat the who stored procedure as a single statement, but many statements.

Therefore, we must redefine the delimiter temporarily (as done in the first line) so that you can pass the whole stored procedure to the server as a sing statement. Once the entire procedure is passed as single statement, we change the delimiter back to `;` .

MySQL Datatypes

MySQL supports SQL data types in several categories: numeric types, date and time types, string (character and byte) types, spatial types, ar the `JSON` data type. Apart from the Date and Datetime datatype we already discussed, the most frequently used datatype are :

Types	Description
INT	A standard integer
DECIMAL	A fixed-point number
FLOAT	A single-precision floating point number
DOUBLE	A double-precision floating point number
BIT	A bit field
CHAR	A fixed-length nonbinary (character) string
VARCHAR	A variable-length non-binary string
BLOB	A small BLOB
LARGEBLOB	A large BLOB
TEXT	A small non-binary string
LARGETEXT	A large non-binary string

NOTE : MySQL does not have the built-in `BOOLEAN` or `BOOL` data type. To represent boolean values, MySQL uses the smallest integer type whic is `TINYINT(1)` . In other words, `BOOLEAN` and `BOOL` are synonyms for `TINYINT(1)` .

For more information on every datatype in MySQL, go through official MySQL Docs Chapter 11 - Datatypes.

Datatype Conversions

The `CONVERT()` function in MySQL is used to convert a value from one data type to the other data type specified in the expression. MySQL also allov it to convert a specified value from one character set to the different **character set**.

The following are the data types on which this function works perfectly:

Datatype	Descriptions
DATE	It converts the value into DATE datatype that responsible for the date portion only. It always results in the "YYYY-MM-DD" format. It supports the range of DATE in '1000-01-01' to '9999-12-31'.
DATETIME	It converts the value into the DATETIME data type that responsible for the date and time portion both. It always results in the "YYYY-MM-DD HH:MM:SS" format. It support the range in '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

Datatype	Descriptions
TIME	It converts the value into a TIME data type that responsible for the time portion only. It always results in the "HH:MM:SS" format. It supports the range of time in '-838:59:59' to '838:59:59'.
CHAR	It converts a value to the CHAR data type, which has a fixed-length string.
SIGNED	It converts a value to SIGNED datatype, which has signed 64-bit integer.
UNSIGNED	It converts a value to the UNSIGNED datatype, which has unsigned 64-bit integer.
DECIMAL	It converts a value to the DECIMAL data type, which has a decimal string.
BINARY	It converts a value to the BINARY data type, which has a binary string.

Syntax :

The following are the syntax of `CONVERT()` ****function :

```
CONVERT(expression, datatype);
// OR
CONVERT(expression USING character_set);
```

where,

Parameter	Requirement	Descriptions
expression	Required	It is a specified value going to be converted into another specific datatype.
datatype	Required	It specifies the desired data type in which we want to be converted.
character_set	Required	It specifies the desired character set in which we want to be converted.

Examples :

```
SELECT CONVERT("2018-11-30", DATETIME);
//output
2018-11-30 00:00:00

SELECT CONVERT(4-7, UNSIGNED);
// output
18446744073709551613

SELECT CONVERT(CONVERT(4-7, UNSIGNED), SIGNED);
// output
-3

SELECT CONVERT('AlmaBetter' USING utf8mb4);
// output
AlmaBetter
```

Conclusion

In this session we've learned about :

- Using Aggregate Functions in MySQL
- Using Join and Union operators in MySQL
- Dealing with Datetime in MySQL
- Managing triggers in a MySQL DB
- Frequently used datatype in MySQL and their Conversions

Interview Questions

- What are the types of `JOIN` operators supported by MySQL?

- Inner Join
- Left Join
- Right Join
- Cross Join

- How many Triggers are possible in MySQL? There are six Triggers allowed to use in the MySQL database:

- Before Insert
- After Insert
- Before Update
- After Update
- Before Delete
- After Delete

Thank You !