DATABASES

GS WEB APPLICATION DEVELOPMENT

Teacher: Vanesa Maldonado Guerrero
Curs 2025/26

DML Instructions

Data Modification Language (DML) is a fundamental part of the SQL language. It consists of instructions capable of modifying (adding, changing, or deleting) data in tables.

A set of **DML** instructions that are executed sequentially is called a transaction. The interesting thing about transactions is that we can cancel them, since they form a logical unit of work and their results are not final until they are accepted.

In all DML instructions, the only data returned by the system is the number of rows that have been modified when the instruction is executed.

Data Insertion

Adding data to a table is done using the **INSERT** statement. Its basic syntax is:

```
INSERT INTO table [ ( columnList ) ]
VALUES ( value1 [ , value2 ...] )
```

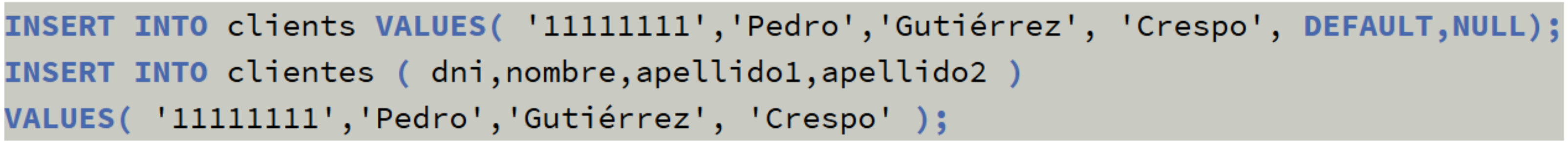
The table represents the table to which we want to add the record, and the values following the **VALUES** clause are the values we assign to the different fields of the record.
If the field list is not specified, the values list must follow the order of the columns as they were created.
The list of fields to fill in is indicated if we do not want to fill in all the columns.

Data Insertion

Columns not explicitly populated with the **INSERT** command are filled with their default value **(DEFAULT)** or with **NULL** if no default value was specified. If a column has a mandatory constraint **(NOT NULL)**, an error will occur if no value is specified for that column.

For example, suppose we have a customer table whose fields are: dni , name , surname1 , surname2 , location and address

Let's assume that's the order in which the fields in that table were created, and that the default value for the location is Palencia , while the address has no default value. In that case, these two instructions are equivalent:



```
INSERT INTO clients VALUES( '11111111','Pedro','Gutiérrez', 'Crespo', DEFAULT,NULL);
INSERT INTO clientes ( dni,nombre,apellido1,apellido2 )
VALUES( '11111111','Pedro','Gutiérrez', 'Crespo' );
```

They are equivalent since, in the second instruction, the unspecified fields are filled with their default value and the address has no default value.

Default values are specified, during the creation or modification of a table structure, through the keyword **DEFAULT**.

Data Modification

The modification of row data is done using the **UPDATE** instruction.
Its syntax is as follows:

```
UPDATE table
SET column1 = value1 [ , column2 = value2 ...]
[ WHERE condition ] ;
```

The columns specified in the SET section are modified with the indicated values. The WHERE clause allows you to specify which records will be modified.

Examples:

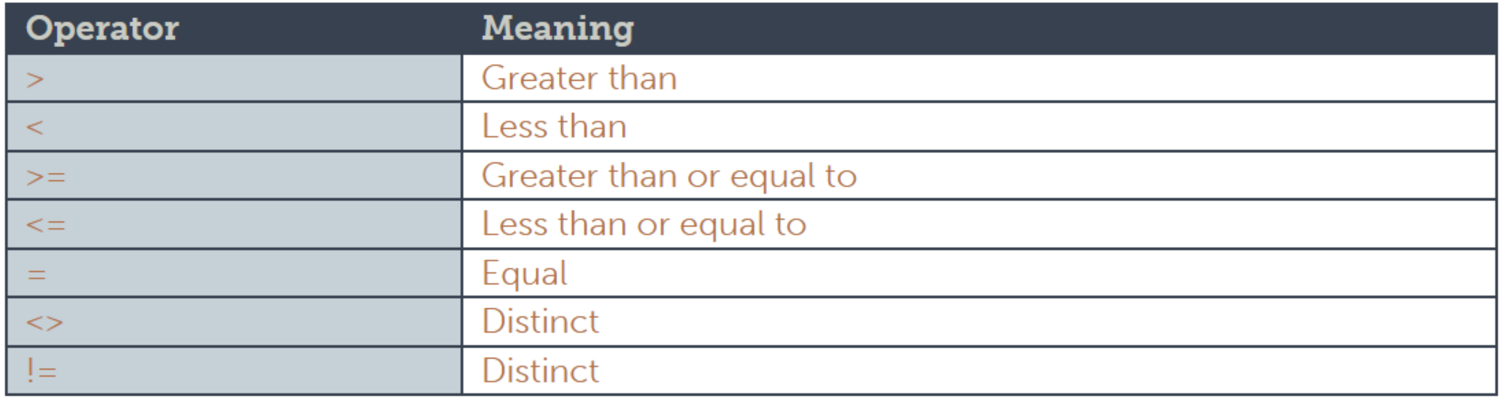
```
UPDATE clients SET province='Ourense'
WHERE provincia='Orense' ;
UPDATE products SET price=price*1.16 ;
```

The first instruction updates the province of Orense customers to appear as Ourense .

The second UPDATE increases prices by 16%.

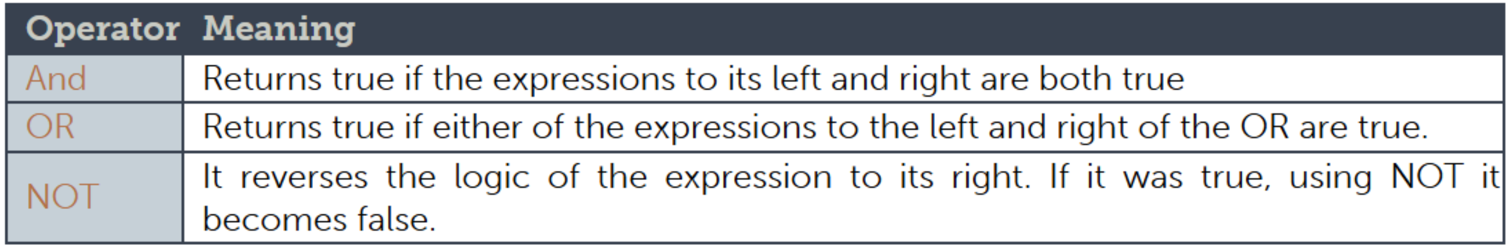
Data Modification

The condition can use any of the following comparison operators:



Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=	Equal
<>	Distinct
!=	Distinct

In addition, you can use:



Operator	Meaning
And	Returns true if the expressions to its left and right are both true
OR	Returns true if either of the expressions to the left and right of the OR are true.
NOT	It reverses the logic of the expression to its right. If it was true, using NOT it becomes false.

In short, the **WHERE** clause accepts any expression that returns true or false.

Data Deletion

It is done using the **DELETE** instruction:

```
DELETE [ FROM ]  table [ WHERE condition ] ;
```

This instruction deletes the rows from the table that meet the specified condition. Example:

```
DELETE FROM employees WHERE section=23 ;
```

The WHERE clause is the same as in the case of the **UPDATE** instruction .

It is important to note that deleting a record cannot cause integrity failures, and that the **ON DELETE CASCADE** integrity option ensures that not only the specified rows are deleted, but all related rows as well.

transactions

A **transaction** consists of a series of DML instructions. A transaction begins with the first DML instruction executed and ends when one of the following situations occurs:

- A **COMMIT** instruction (all instructions in the transaction are accepted)
- A **ROLLBACK** instruction (all instructions in the transaction are canceled)
- **A DDL** instruction , such as **CREATE TABLE**, **ALTER TABLE**, etc. In this case, the transaction is accepted (as if we had done a COMMIT).
- **A DCL** instruction (such as **GRANT**, for example). As in the previous case, the transaction will be accepted.
- The user must **log out**. In that case, the transaction is cancelled, unless the software the user is using automatically performs a COMMIT before logging out.
- In fact, many work environments ask, before closing the session, what we want to do with the ongoing transaction.
- Undoubtedly, if we close the session due to a failure or improperly, then the transaction will certainly be cancelled.
- **System crash**. Transaction cancelled.

Transactions are used to ensure that data modification instructions perform as intended. This ensures data consistency and allows for safe testing of the results of certain instructions before finalizing the transaction.

COMMIT Instruction

The **COMMIT** statement makes the changes made by the transaction permanent and irrevocable. It should only be used if you agree with the changes, and you should be absolutely certain before executing COMMIT, as the instructions can affect thousands of records.

It is important to remember that if we have performed an **INSERT**, a **DELETE** and two **UPDATEs** (for example), COMMIT accepts all of those instructions at once, since the transaction will be the set of all of them (if we have not written a DDL or DCL instruction in between).

ROLLBACK

It cancels all instructions in the transaction. This is a dangerous instruction that should only be used if we have realized that executing the transaction instructions results in an undesirable effect on the data.

An incorrect session exit, a communication problem, or a system crash results in an implicit **ROLLBACK**.

SAVEPOINT, save points

It's not a standard action, but it is supported by Oracle. Since **ROLLBACK** cancels the entire transaction, savepoints are a utility that allows us to cancel part of the transaction.

It works as follows:

- During a transaction, we use the **SAVEPOINT** instruction followed by the name we want to give to that save point. For example:

```sql
SAVEPOINT pointA;
```

- We can repeat the previous instruction to save more points.
- We can return to the save point using the instruction:

```sql
ROLLBACK TO SAVEPOINT pointA;
```

Save points only work during the current transaction. They are deleted when the transaction is complete.

The names we give to each save point must be different; otherwise, if we use the same name twice, the last use of that name cancels the previous one.

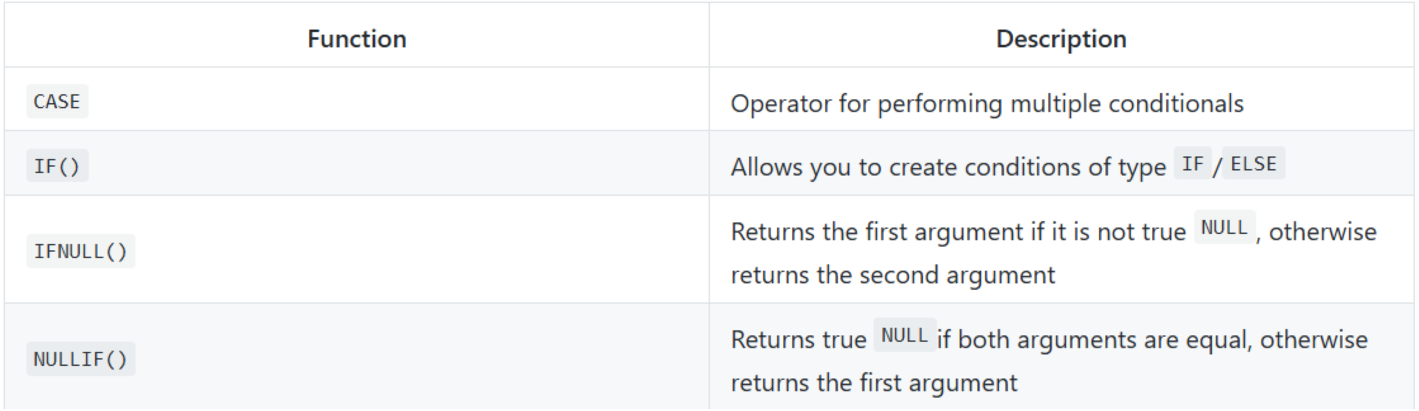
Naturally, if we have gone back to a distant save point, we will not be able to go to a more recent save point afterwards.

Flow control functions

The **control flow function** evaluates the condition specified in it.

The output generated by them can be a true, false, static value or column expression. We can use the control flow functions in the SELECT, WHERE, ORDER BY, and GROUP BY clause.

Following are the most common functions:



Function	Description
`CASE`	Operator for performing multiple conditionals
`IF()`	Allows you to create conditions of type `IF` / `ELSE`
`IFNULL()`	Returns the first argument if it is not true `NULL`, otherwise returns the second argument
`NULLIF()`	Returns true `NULL` if both arguments are equal, otherwise returns the first argument

Case

The **CASE** statement is one of the most common control flow expressions of MySQL.

The case expression is used to implement the IF ELSE logic in the query. The CASE expression can be used in the SELECT, WHERE, and ORDER BY clause.

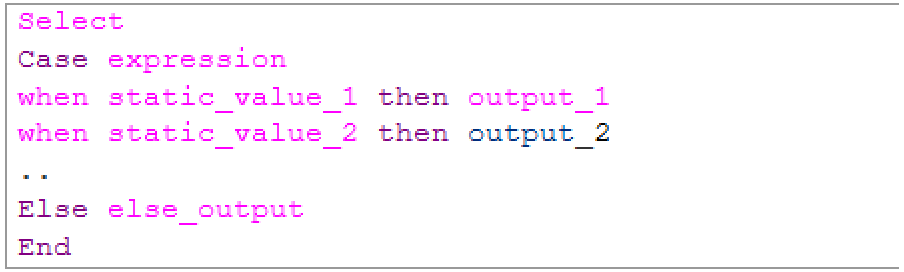
There are two variants of the CASE expression.

1) The simple CASE expression
2) The searched CASE expression

The simple CASE expression

The **simple CASE expression** matches the input with the values for equality and returns the corresponding result.

Following is the syntax of the simple CASE expression:



```
Select
Case expression
when static_value_1 then output_1
when static_value_2 then output_2
..
Else else_output
End
```

The simple CASE expression

Example 1

Suppose you want to populate the list of the students, their grades, and grade description. The query output must show the grade description based on the following condition.

1) If the student's grade is A+, then he should be considered an Intelligent
2) If the student's grade is B+, then he should be considered as an Average Student
3) If the student's grade is D+ or D- then he should be considered a Weak Student

In the case statement, we can specify an operator as well. I have specified an OR operator to ensure that the student whose grade is D+ or D should be considered a Weak student.

The query is the following:

```
select studentname, grade,
case grade
when 'A+' then 'Intelligent'
when 'B+' then 'Average Student'
when 'D' or 'D-' then 'Weak student' end as 'grade description'
from tblstudent;
```

The Searched CASE expression

The **searched case expression** evaluates the expression specified in the WHEN clause and returns the corresponding value.

The syntax is the following:

```
Select
Case
when expression_1 then output_1
when expression_2 then output_2
..
Else else_expression_output
End
```

The Searched CASE expression

Example 1

Suppose we want to populate the name of the parents based on the name of the surname of the student. The query must populate the values of studentname, grade, and classroom columns.

The condition for the CASE expression is following.

1) If the student's surname is Chauhan, then the father's name must be Hasmukh Chauhan
2) If the student's surname is Bhatt, then the father's name must be Jivan Bhatt
3) If the student's surname is Upadhyay, then the father's name must be Lalshankar Upadhyay

Query

```
select studentname,
case
when studentname like '%Chauhan%' Then 'Hasmukh Chauhan'
when studentname like '%Bhatt%' then 'Jivan Bhatt'
when studentname like '%Upadhyay%' then 'Lalshankar Upadhyay' end as ParentName,
classroom,grade
from tblstudent
where classroom is not null;
```

IF() function

The **IF function** returns the value based on the condition specified within the function.

The syntax of the IF function is the following:

```
SELECT IF(condition, if_condition_true, if_condition_false)
```

In the syntax:

- **Condition** is a condition specified in the IF function
- **If_condition_true** is the value returned by the function
- **If_condition_false** is the value returned by the function

The IF function evaluates the values returned by the SELECT statement with the specified condition. If the function evaluates as TRUE, then it returns the value specified in if_condition_true.

If the function evaluates as FALSE, then it returns the value specified in the if_condition_false.

IF() function

When the input string matches with each other, then print TRUE else, print FALSE.

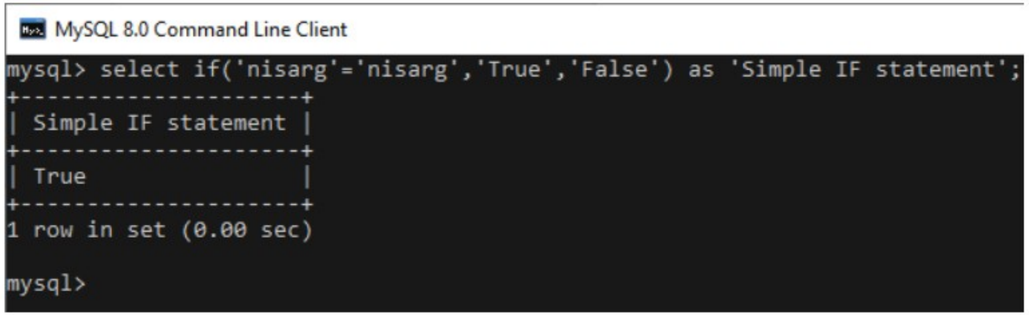
We can use the IF function in the SELECT statement.

The query should be written as follows:

Query 1

```
mysql> select if('nisarg'='nirali','True','False') as 'Simple IF statement';
```

Output 1



MySQL 8.0 Command Line Client

```
mysql> select if('nisarg'='nisarg','True','False') as 'Simple IF statement';
+---------------------+
| Simple IF statement |
+---------------------+
| True                |
+---------------------+
1 row in set (0.00 sec)

mysql>
```

IFNULL function

The **IFNULL** function is another common control flow function of MySQL.

If the specified expression is NULL then the NULLIF function returns the specific value, else it returns the expression.

Syntax

```
SELECT IFNULL(NULL, expr2);
```

In the syntax, if the value of the expr1 is NULL then it returns expr2 else it returns a value of expr1.

IFNULL function

Example

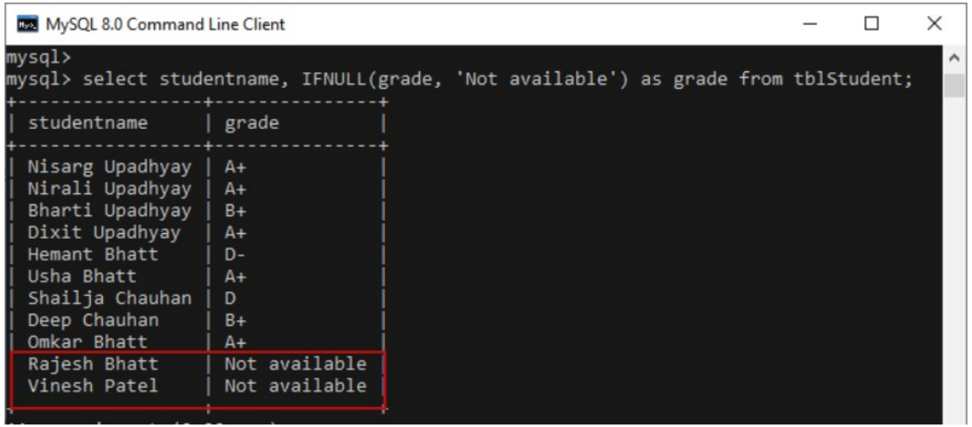
Suppose when the query finds the NULL value for the grade column, we want to print details not available else we want to print the value of the column.

This can be achieved by using the CASE expression, but we are using the IFNULL function.

Query

```
select studentname, IFNULL(grade, 'Not available') as grade from tblStudent
```

Output



MySQL 8.0 Command Line Client

```
mysql>
mysql> select studentname, IFNULL(grade, 'Not available') as grade from tblStudent;
+-----------------+---------------+
| studentname     | grade         |
+-----------------+---------------+
| Nisarg Upadhyay | A+            |
| Nirali Upadhyay | A+            |
| Bharti Upadhyay | B+            |
| Dixit Upadhyay  | A+            |
| Hemant Bhatt    | D-            |
| Usha Bhatt      | A+            |
| Shailja Chauhan | D             |
| Deep Chauhan    | B+            |
| Omkar Bhatt     | A+            |
| Rajesh Bhatt    | Not available |
| Vinesh Patel    | Not available |
+-----------------+---------------+
```

NULLIF function

The **NULLIF** function compares the expressions specified in the function.

If the values of the expressions are equal then it returns the NULL else it returns the first expression.

```
SELECT NULLIF(expr1, expr2);
```

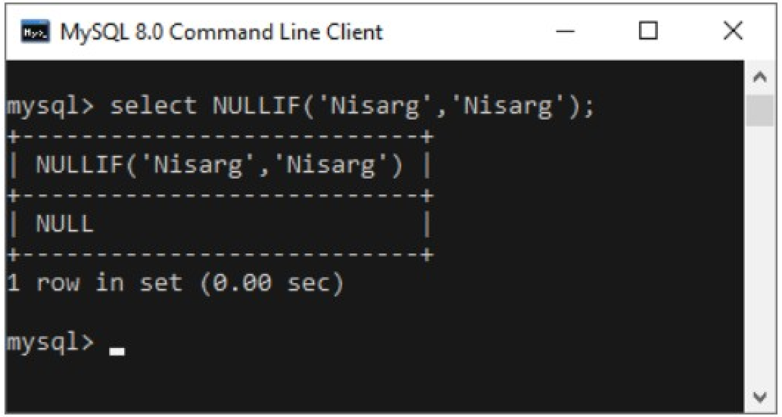
NULLIF function

Example

Query

```
Select NULLIF('Nisarg','Nisarg')
```

Output



```
mysql> select NULLIF('Nisarg','Nisarg');
+---------------------------+
| NULLIF('Nisarg','Nisarg') |
+---------------------------+
| NULL                      |
+---------------------------+
1 row in set (0.00 sec)

mysql> _
```