MySQL Aggregate Functions

365 DataScience


COUNT()

COUNT()

So far:

COUNT()

So far:

- Theory of Relational Databases
- SQL Theory
- Coding Techniques and Best Practices
- SELECT, INSERT, UPDATE, DELETE

COUNT()

So far:

- Theory of Relational Databases
- SQL Theory
- Coding Techniques and Best Practices
- SELECT, INSERT, UPDATE, DELETE

Next:

COUNT()

So far:

- Theory of Relational Databases
- SQL Theory
- Coding Techniques and Best Practices
- SELECT, INSERT, UPDATE, DELETE

Next:

- Aggregate Functions

COUNT()

aggregate functions

they gather data from *many* rows of a table, then aggregate it into a *single* value

COUNT()

aggregate functions

they gather data from *many* rows of a table, then aggregate it into a *single* value

INPUT

COUNT()

aggregate functions

they gather data from *many* rows of a table, then aggregate it into a *single* value

INPUT

the information contained in *multiple* rows

COUNT()

aggregate functions

they gather data from *many* rows of a table, then aggregate it into a *single* value

INPUT

the information contained in *multiple* rows →

COUNT()

aggregate functions

they gather data from *many* rows of a table, then aggregate it into a *single* value

INPUT	OUTPUT
the information contained in *multiple* rows	→

COUNT()

aggregate functions

they gather data from *many* rows of a table, then aggregate it into a *single* value

INPUT		OUTPUT
the information contained in *multiple* rows	→	the *single* value they provide

COUNT()

COUNT()

COUNT()

- COUNT()
- SUM()

COUNT()

- COUNT()
- SUM()
- MIN()

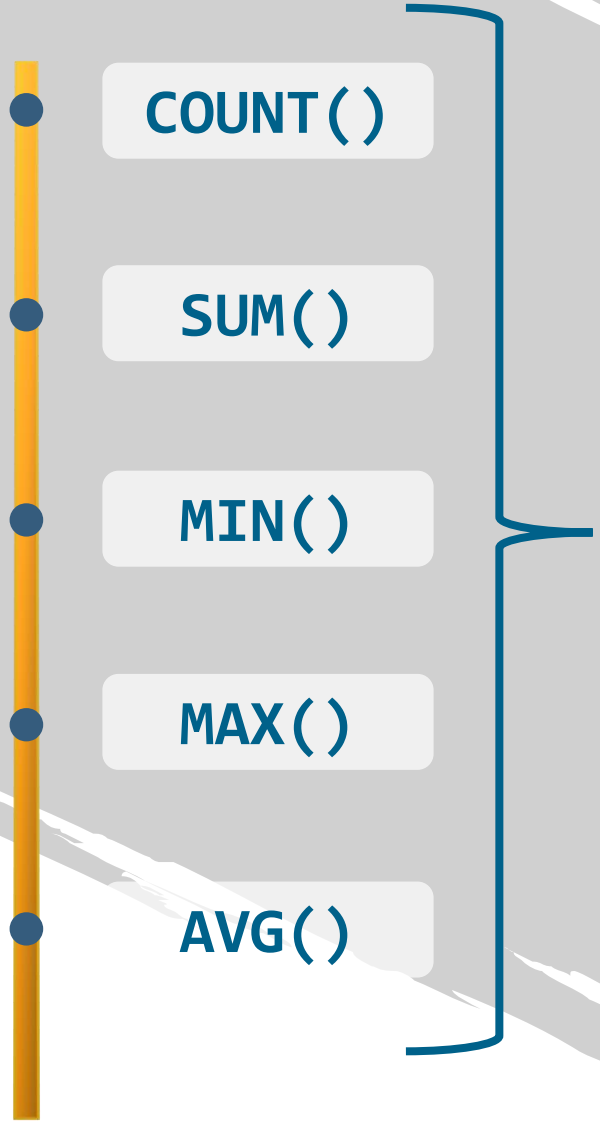
COUNT()

- COUNT()
- SUM()
- MIN()
- MAX()

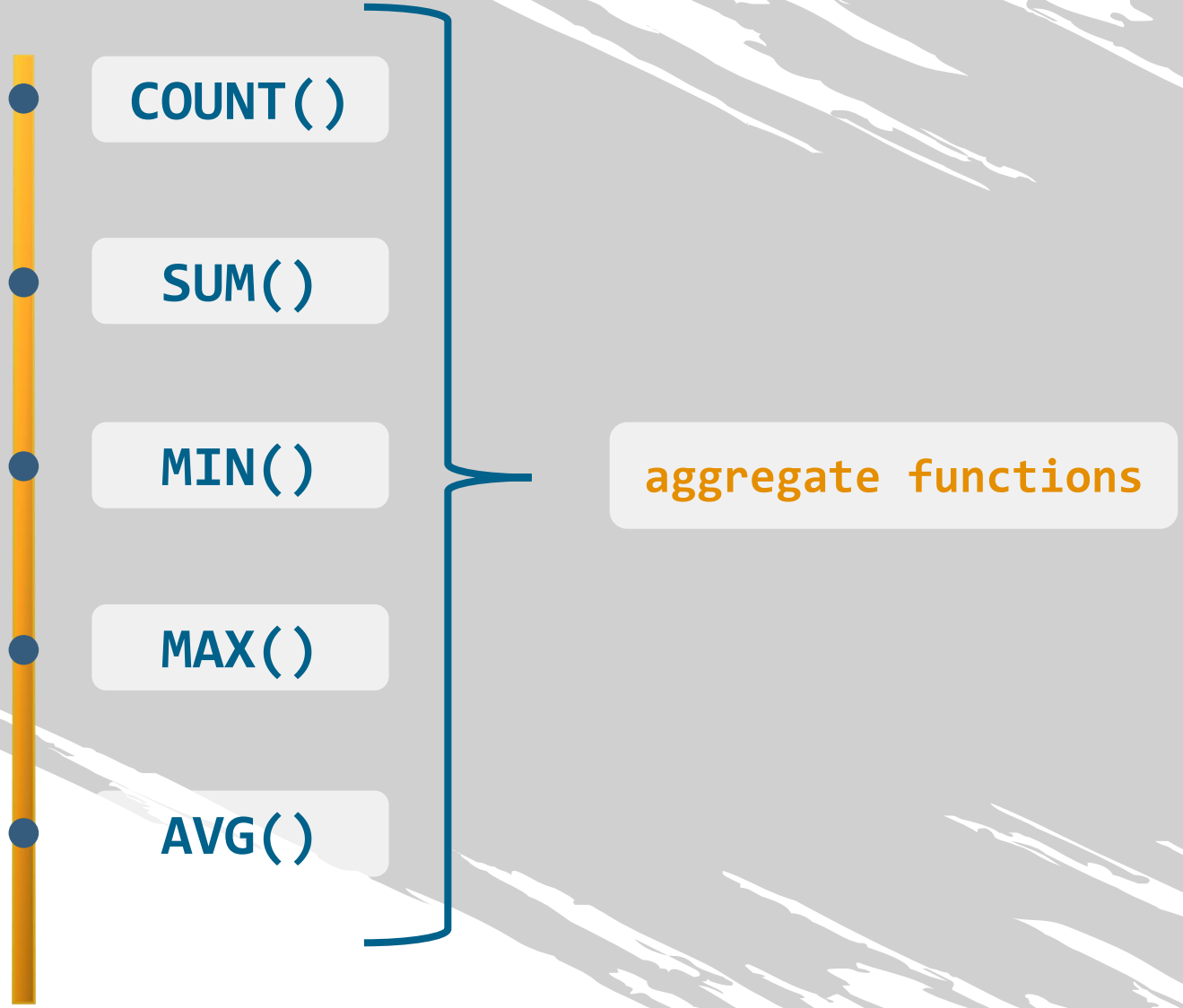
COUNT()

- COUNT()
- SUM()
- MIN()
- MAX()
- AVG()

COUNT()



COUNT()



COUNT()

- COUNT()
- SUM()
- MIN()
- MAX()
- AVG()

aggregate functions = summarizing functions

COUNT()

- Why do these functions exist?

COUNT()

Why do these functions exist?

- they are a response to the information requirements of a company's different organizational levels

COUNT()

Why do these functions exist?

- they are a response to the information requirements of a company's different organizational levels

- top management executives are typically interested in *summarized* figures and rarely in detailed data

COUNT()

COUNT()

applicable to both *numeric* and *non-numeric* data

COUNT()

`COUNT(DISTINCT )`

helps us find the number of times unique values are encountered in a given column

COUNT()

- aggregate functions *typically* ignore null values throughout the field to which they are applied

COUNT()

- aggregate functions *typically* ignore null values throughout the field to which they are applied

COUNT()

- aggregate functions *typically* ignore null values throughout the field to which they are applied

COUNT()

- aggregate functions *typically* ignore null values throughout the field to which they are applied

  only if you have indicated a *specific* column name within the parentheses

COUNT()

- aggregate functions *typically* ignore null values throughout the field to which they are applied

only if you have indicated a *specific* column name within the parentheses

Alternatively:

COUNT()

- aggregate functions *typically* ignore null values throughout the field to which they are applied

  only if you have indicated a *specific* column name within the parentheses

Alternatively:

- `COUNT(*)`

COUNT()

- aggregate functions *typically* ignore null values throughout the field to which they are applied

  only if you have indicated a *specific* column name within the parentheses

Alternatively:

- `COUNT(*)`

  * returns the number of all rows of the table, `NULL` values *included*

COUNT()

COUNT()

COUNT()

COUNT()

the parentheses and the argument must be attached to the name of the aggregate function

COUNT()

COUNT()

the parentheses and the argument must be attached to the name of the aggregate function

- you shouldn’t leave white space before opening the parentheses

COUNT()

COUNT()

the parentheses and the argument must be attached to the name of the aggregate function

- you shouldn’t leave white space before opening the parentheses

COUNT()

COUNT()

COUNT()

the parentheses and the argument must be attached to the name of the aggregate function

- you shouldn’t leave white space before opening the parentheses

COUNT()

COUNT ()

COUNT()

COUNT()

the parentheses and the argument must be attached to the name of the aggregate function

- you shouldn’t leave white space before opening the parentheses

COUNT()

COUNT_()

COUNT()

COUNT()

the parentheses and the argument must be attached to the name of the aggregate function

- you shouldn’t leave white space before opening the parentheses

COUNT()


SUM()

SUM()

- COUNT(*)

SUM()

`COUNT(*)`

* returns all rows of the table, `NULL` values *included*

SUM()

- `COUNT(*)`

  * returns all rows of the table, `NULL` values *included*

- `SUM(*)`

SUM()

- `COUNT(*)`

  * returns all rows of the table, `NULL` values *included*

- ~~`SUM(*)`~~

SUM()

`COUNT(*)`

* returns all rows of the table, `NULL` values *included*

~~`SUM(*)`~~

* goes well with only the `COUNT()` function

SUM()

COUNT()

SUM()

- `COUNT()` - applicable to both *numeric* and *non-numeric* data

SUM()

- COUNT() - applicable to both *numeric* and *non-numeric* data
- SUM()
- MIN()
- MAX()
- AVG()

SUM()

- COUNT() - applicable to both *numeric* and *non-numeric* data
- SUM()
- MIN()
- MAX()
- AVG()

SUM()

- COUNT() - applicable to both *numeric and non-numeric* data
- SUM()
- MIN()
- MAX()
- AVG()

- work *only with numeric* data

MIN() and MAX()

MIN() and MAX()

MAX()

returns the maximum value of a column

MIN() and MAX()

MAX()

returns the maximum value of a column

MIN()

returns the minimum value of a column

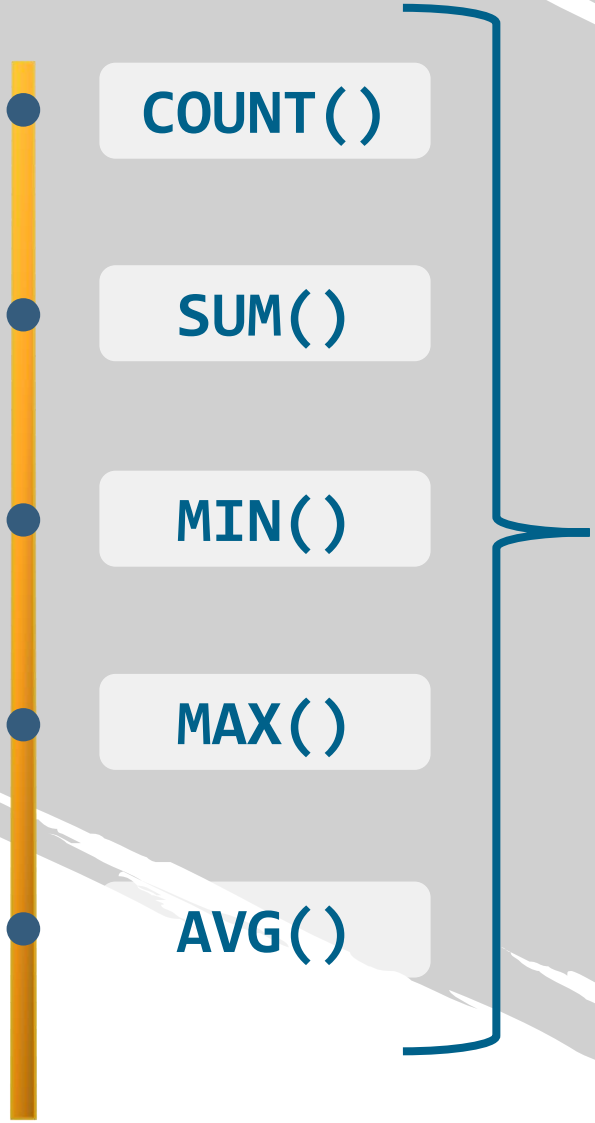

AVG()

AVG()

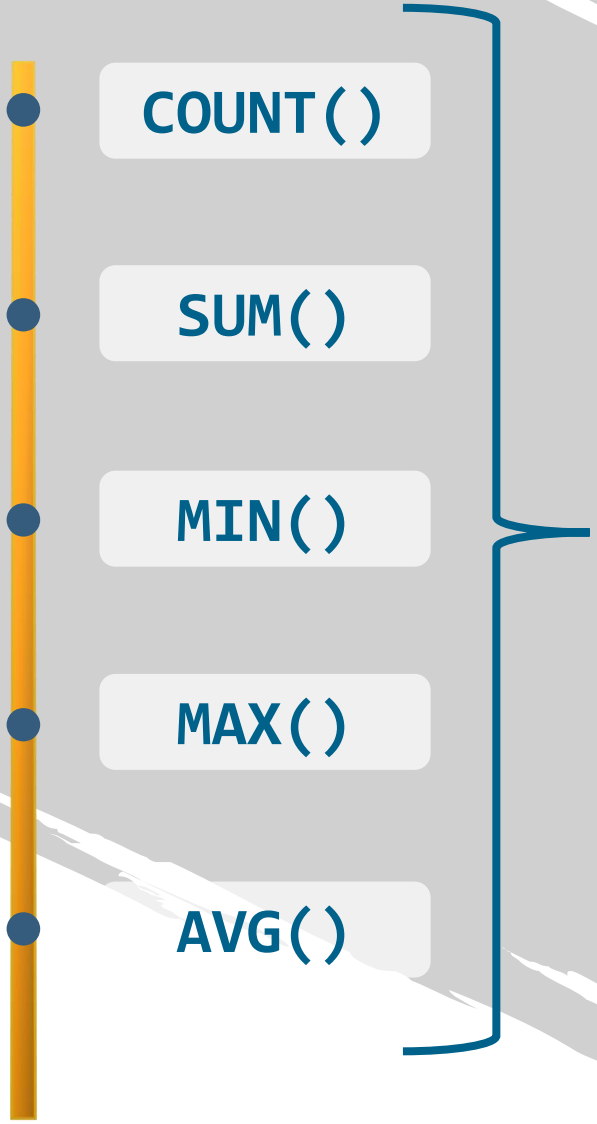
AVG()

extracts the average value of all non-null values in a field

AVG()

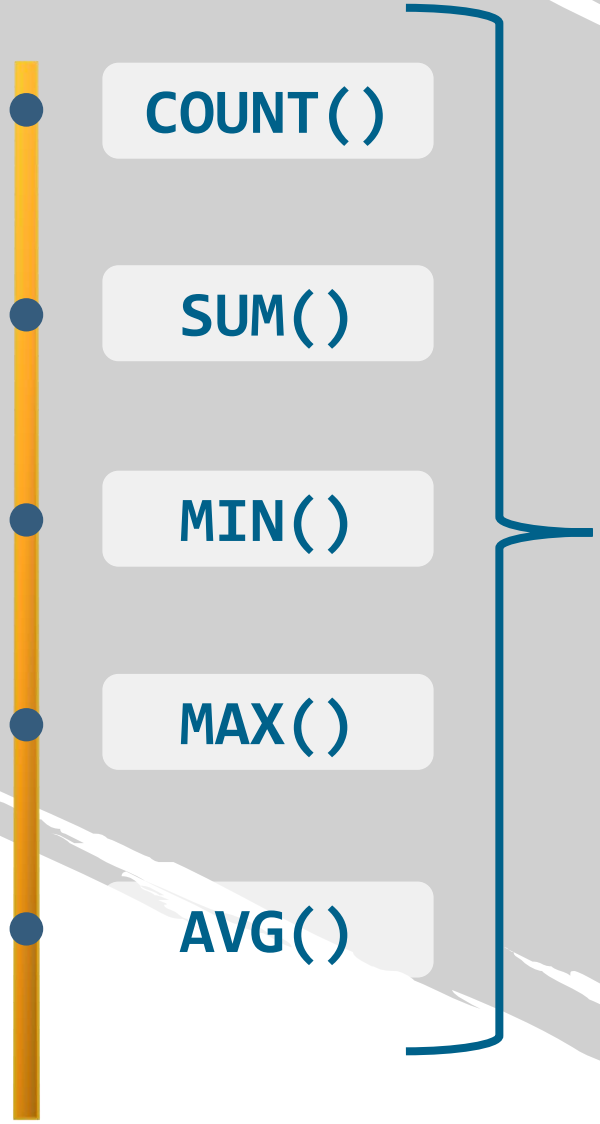


AVG()



- aggregate functions can be applied to any *group* of data values within a certain column

AVG()



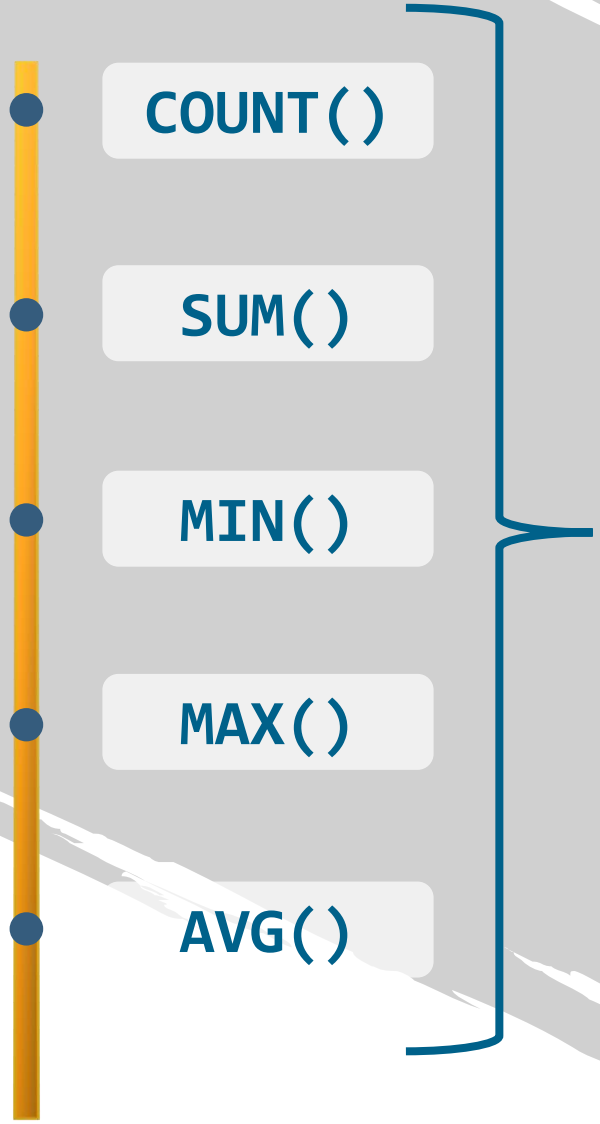
- aggregate functions can be applied to any *group* of data values within a certain column

AVG()

- COUNT()
- SUM()
- MIN()
- MAX()
- AVG()

- aggregate functions can be applied to any *group* of data values within a certain column

AVG()



- aggregate functions can be applied to any *group* of data values within a certain column

frequently used together with a GROUP BY clause

ROUND()

ROUND()

```
ROUND(#,decimal_places)
```

ROUND()

`ROUND(#,decimal_places)`

numeric, or math, function you can use

ROUND()

`ROUND(#,decimal_places)`

numeric, or math, function you can use

- usually applied to the single values that aggregate functions return

COALESCE() - Preamble

COALESCE() - Preamble

Here we will study something a bit more sophisticated.

IF NULL() and COALESCE() are among the advanced SQL functions in the toolkit of SQL professionals. They are used when null values are dispersed in your data table and you would like to substitute the null values with another value.

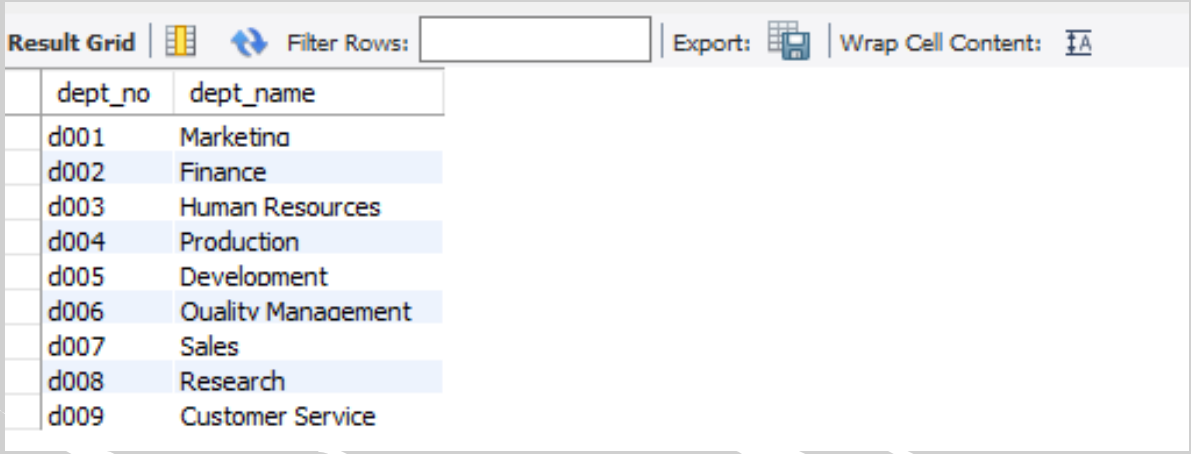
So, let's adjust the "Departments" duplicate in a way that suits the purposes of the next video, in which we will work with IF NULL() and COALESCE().

First, let's look at our table and see what we have there.

COALESCE() - Preamble

SQL

```
SELECT * FROM departments_dup;
```



Result Grid | Filter Rows: | Export: | Wrap Cell Content:

dept_no	dept_name
d001	Marketing
d002	Finance
d003	Human Resources
d004	Production
d005	Development
d006	Quality Management
d007	Sales
d008	Research
d009	Customer Service

Nine departments, with their department numbers and names provided. Ok!

COALESCE() - Preamble

Currently, as shown in the DDL statement of this table, the "Department name" field is with a NOT NULL constraint, which naturally means we must insert a value in each of its rows.

DDL for employees.departments_dup

```
1  CREATE TABLE `departments_dup` (
2      `dept_no` char(4) NOT NULL,
3      `dept_name` varchar(40) NOT NULL
4  ) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

COALESCE() - Preamble

Now, with the ALTER TABLE statement and the CHANGE COLUMN command, we will modify this constraint and allow null values to be registered in the “department name” column.


```sql
ALTER TABLE departments_dup

CHANGE COLUMN dept_name dept_name VARCHAR(40) NULL;
```

COALESCE() - Preamble

Right after that, we will insert into the department number column of this table a couple of data values – D-10 and D-11, the numbers of the next two potential departments in the “Departments Duplicate” table.


```sql
INSERT INTO departments_dup(dept_no) VALUES ('d010'), ('d011');
```

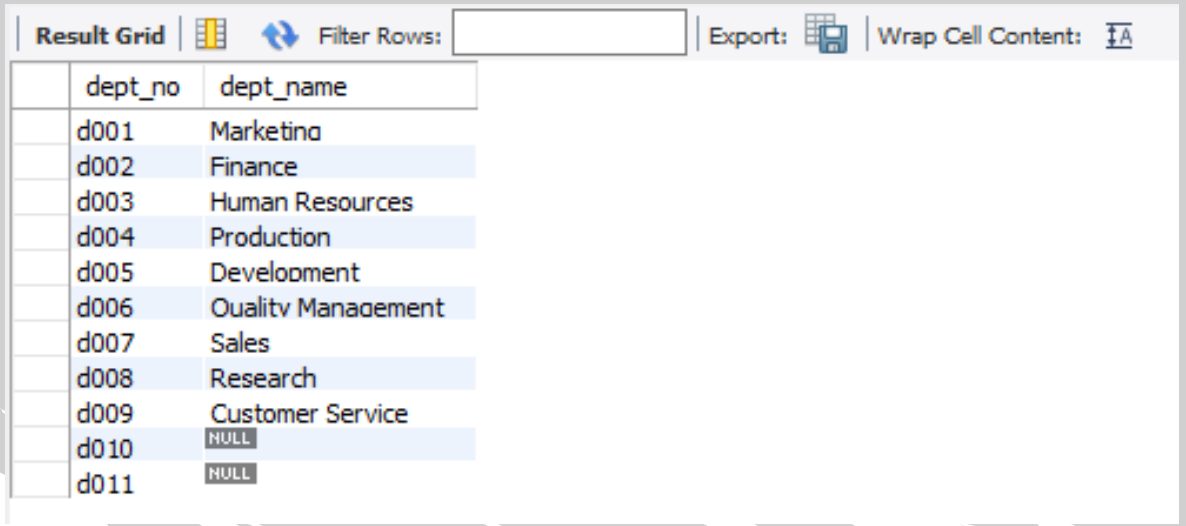
COALESCE() - Preamble

By running this SELECT query over here, you can see whether this operation was carried out successfully.


```sql
SELECT
    *
FROM
    departments_dup
ORDER BY dept_no ASC;
```

COALESCE() - Preamble

We have the two new department numbers listed below, and in the “Department name” column we can see two null values. The latter happened because we allowed for null values to exist in this field, “Department name”. Thus, Workbench will indicate that a value in a cell is missing by attaching a “null” label to it. Great!



dept_no	dept_name
d001	Marketing
d002	Finance
d003	Human Resources
d004	Production
d005	Development
d006	Quality Management
d007	Sales
d008	Research
d009	Customer Service
d010	NULL
d011	NULL

COALESCE() - Preamble

The next adjustment we'll have to make is adding a third column called "Department manager". It will indicate the manager of the respective department. For now, we will leave it empty, and will add the NULL constraint. Finally, we will place it next to the "Department name" column by typing "AFTER "Department name".


```sql
ALTER TABLE employees.departments_dup

ADD COLUMN dept_manager VARCHAR(255) NULL AFTER dept_name;
```

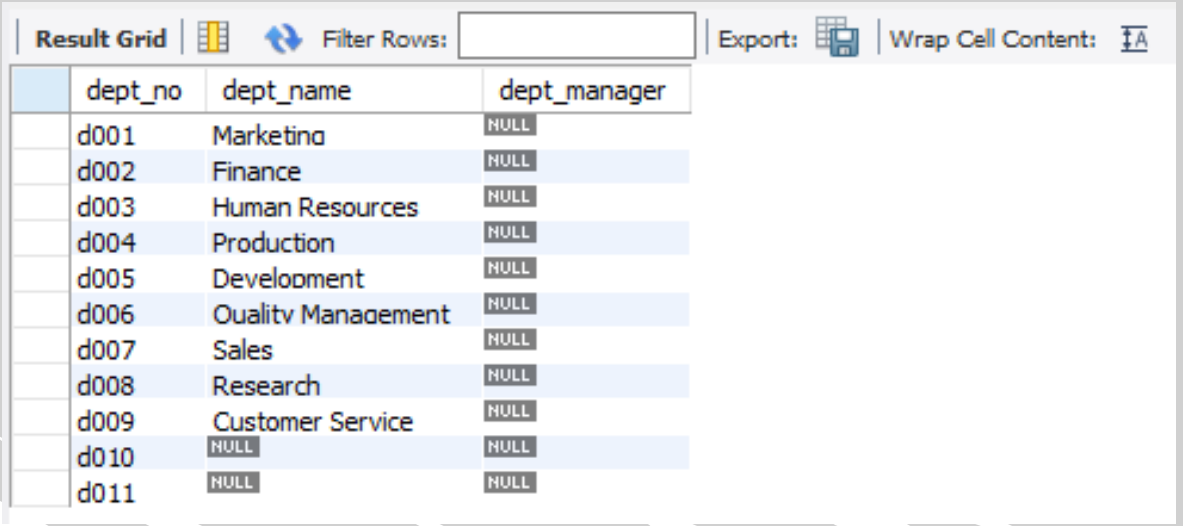
COALESCE() - Preamble

Let’s check the state of the “Departments duplicate” table now.


```sql
SELECT
    *
FROM
    departments_dup
ORDER BY dept_no ASC;
```

COALESCE() - Preamble

Exactly as we wanted, right? The third column is completely empty and we have null values in the last two records. These are the “department name” and “manager” fields.



dept_no	dept_name	dept_manager
d001	Marketing	NULL
d002	Finance	NULL
d003	Human Resources	NULL
d004	Production	NULL
d005	Development	NULL
d006	Quality Management	NULL
d007	Sales	NULL
d008	Research	NULL
d009	Customer Service	NULL
d010	NULL	NULL
d011	NULL	NULL

COALESCE() - Preamble

To save the “Departments duplicate” table in its current state, execute a COMMIT statement.


```
COMMIT;
```

Here we’ll end the setup for the video about IF NULL() and COALESCE().

Good luck!

IFNULL() and COALESCE()

IFNULL() and COALESCE()

```
IFNULL(expression_1, expression_2)
```

IFNULL() and COALESCE()

```
IFNULL(expression_1, expression_2)
```

returns the first of the two indicated values if the data value found in the table is *not null*, and returns the second value if there is a *null* value

IFNULL() and COALESCE()

```
IFNULL(expression_1, expression_2)
```

returns the first of the two indicated values if the data value found in the table is *not null*, and returns the second value if there is a *null* value

- prints the returned value in the column of the output

IFNULL() and COALESCE()

```
COALESCE(expression_1, expression_2 …, expression_N)
```

IFNULL() and COALESCE()

```
COALESCE(expression_1, expression_2 …, expression_N)
```

allows you to insert N arguments in the parentheses

IFNULL() and COALESCE()

```
COALESCE(expression_1, expression_2 …, expression_N)
```

allows you to insert N arguments in the parentheses

- think of COALESCE() as IFNULL() with more than two parameters

IFNULL() and COALESCE()

```
COALESCE(expression_1, expression_2 …, expression_N)
```

allows you to insert N arguments in the parentheses

- think of COALESCE() as IFNULL() with more than two parameters

- COALESCE() will always return a *single* value of the ones we have within parentheses, and this value will be *the first non-null value* of this list, reading the values from left to right

IFNULL() and COALESCE()

```
COALESCE(expression_1, expression_2 …, expression_N)
```

- if COALESCE() has two arguments, it will work precisely like IFNULL()

IFNULL() and COALESCE()

`IFNULL()` and `COALESCE()` do not make any changes to the data set. They merely create an output where certain data values appear in place of NULL values.

IFNULL() and COALESCE()

```
COALESCE(expression_1, expression_2 …, expression_N)
```

IFNULL() and COALESCE()

```
COALESCE(expression_1, expression_2 …, expression_N)
```

- we can have a single argument in a given function

IFNULL() and COALESCE()

COALESCE(expression_1, expression_2 …, expression_N)

- we can have a single argument in a given function

- practitioners find this trick useful if some *hypothetical result* must be provided in a supplementary column

IFNULL() and COALESCE()

COALESCE(expression_1, expression_2 …, expression_N)

- we can have a single argument in a given function

- practitioners find this trick useful if some *hypothetical result* must be provided in a supplementary column

- COALESCE() can help you visualize *a prototype of the table's final version*

IFNULL() and COALESCE()

`IFNULL()` works with precisely *two* arguments

IFNULL() and COALESCE()

`IFNULL()` works with precisely *two* arguments

`COALESCE()` can have *one, two, or more* arguments