SQL string, date, and miscellaneous functions

Miscellaneous functions

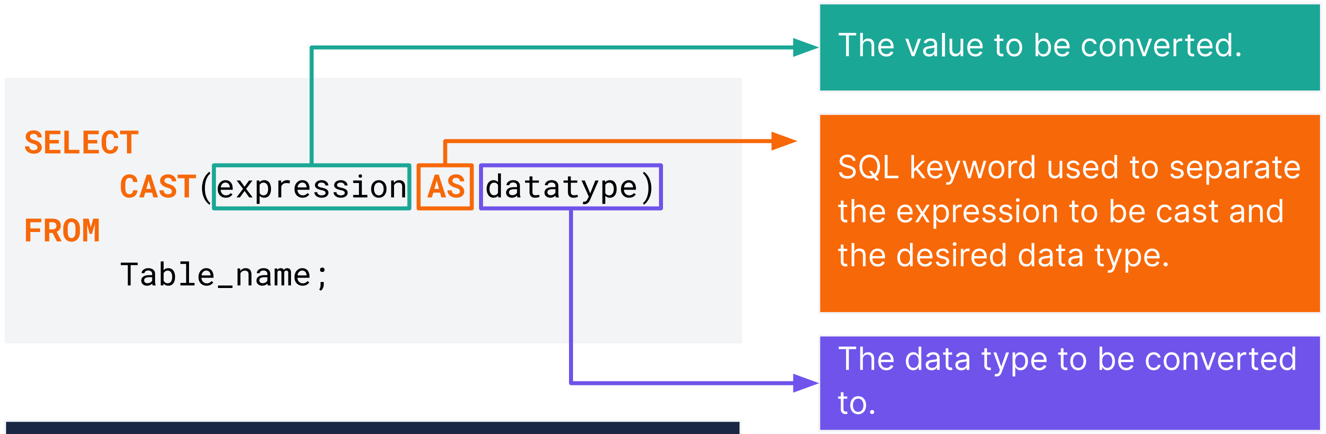
Data overview

We will use the following **Households_individuals** table which contains certain information about the individuals in all households in Kenya collected during a household survey in 2020.

ID	Sex	D_O_B	Age	Weight	Highest_ed	Ed_institution	Marital_status	Spouse	Spouse_ID
3901	Male	2020-06-30 00:00:00	0	7.24	NULL	NULL	Single	N/A	NULL
3821	Female	1998-05-21 00:00:00	22	67	Diploma	Public	Single	N/A	NULL
3961	Male	1970-11-15 00:00:00	35	59	Masters	NULL	Married	Yes	3331
3741	Female	2012-01-09 00:00:00	14	45.22	Primary	Private	Single	N/A	NULL
3661	Male	1989-10-04 00:00:00	69	77	PHD	NULL	Married	Yes	3891
63921	Female	2020-06-30 00:00:00	16	45.99	Secondary	Public	Single	N/A	NULL

CAST() function

The **CAST()** function is used to **convert** a value from its **current data type into a specified data type**. Its basic syntax is as follows:



```
SELECT
     CAST(expression AS datatype)
FROM
     Table_name;
```

The value to be converted.

SQL keyword used to separate the expression to be cast and the desired data type.

The data type to be converted to.

Ensure that the value you are attempting to cast is **compatible with** the **target data type**. Otherwise, the function will throw an error.

The following **target data types** are **supported:**

- DATE
- DATETIME
- TIME
- DECIMAL
- INTEGER
- FLOAT
- CHAR
- VARCHAR
- SIGNED
- UNSIGNED
- BINARY

CAST() function

The **D_O_B column** is set to the **DATETIME data type**. The values in the column do not have any time information, making the time part unnecessary. The **DATE data type** would be more suitable here.

Query

```
SELECT
      D_O_B,
      CAST(D_O_B AS DATE) AS New_D_O_B
FROM
      Household_individuals;
```

In this example, the DATETIME values in column D_O_B are cast to a DATE data type using the CAST() function. The results of the operation are then stored in a new column, New_D_O_B.

Output

D_O_B	New_D_O_B
2020-06-30 00:00:00	2020-06-30
1998-05-21 00:00:00	1998-05-21
1970-11-15 00:00:00	1970-11-15
2012-01-09 00:00:00	2012-01-09
1989-10-04 00:00:00	1989-10-04

CONVERT() function

`CONVERT()` is another function that can be used for **conversion from one data type to another**. Its basic syntax is as follows:

```
SELECT
     CONVERT(value, datatype)
FROM
     Table_name;
```

The value to be converted.

The data type to be converted to.

Ensure that the value you are attempting to convert is **compatible** with the **target data type**. Otherwise, the function will throw an error.

The following **target data types** are **supported:**

- DATE
- DATETIME
- DECIMAL
- TIME
- CHAR
- NCHAR
- SIGNED
- UNSIGNED
- BINARY

CONVERT() function

The **Weight** column has been set to the FLOAT data type. This means that the **Weight** values have varying decimal precision depending on their declared values. We can convert to a DECIMAL data type with a precision of 4 and a scale of 2 to give all the values a fixed decimal precision to avoid rounding errors in calculations.

Query

```
SELECT
    Weight,
    CONVERT(Weight, DECIMAL(4,2)) AS New_weight
FROM
    Household_individuals;
```

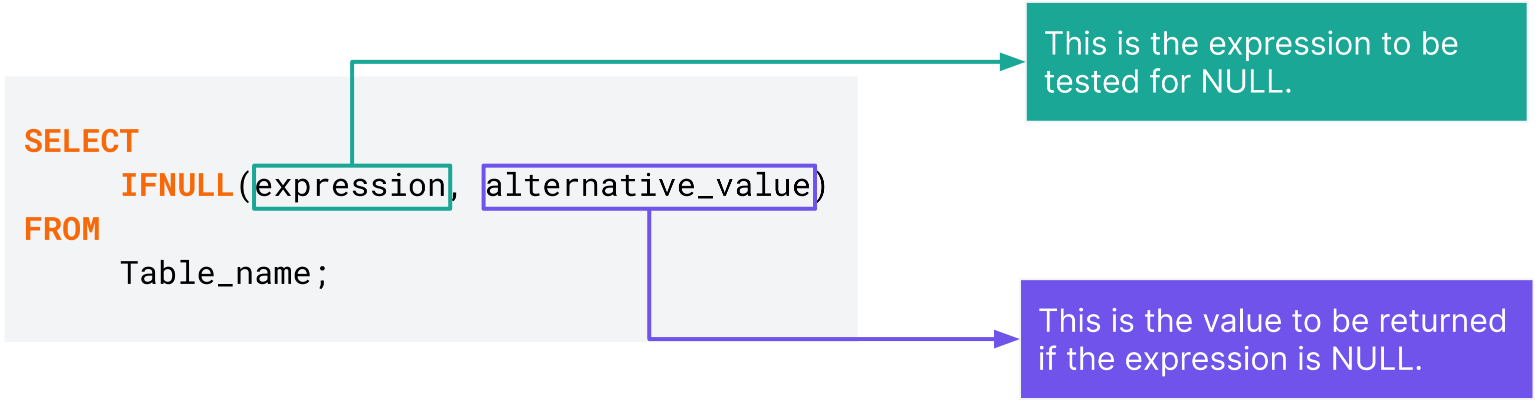
In this example, the floating-point values in the Weight column are converted to a DECIMAL(4,2) data type using the CONVERT() function. The results of the operation are then stored in a new column, New_weight.

Output

Weight	New_weight
7.24	7.24
67	67.00
56	56.00
45.22	45.22
9.1	9.10

IFNULL() function

The **`IFNULL()`** function **returns** a **specified value** if the given **expression is null**. Otherwise, it returns the value of the expression itself. Its basic syntax is as follows:



```
SELECT
     IFNULL(expression, alternative_value)
FROM
     Table_name;
```

This is the expression to be tested for NULL.

This is the value to be returned if the expression is NULL.

The **`IFNULL()`** function is usually used to **handle NULL values** in a column or expression by **replacing them** with an alternative value.

IFNULL() function

For records where **Highest_ed** readings are missing, they have been assigned a NULL value. We can **replace these NULL** values with a new category called **No schooling**.

Query

```
SELECT
      Highest_ed,
      IFNULL(Highest_ed,'No schooling') AS
      New_highest_ed
FROM
      Household_individuals;
```

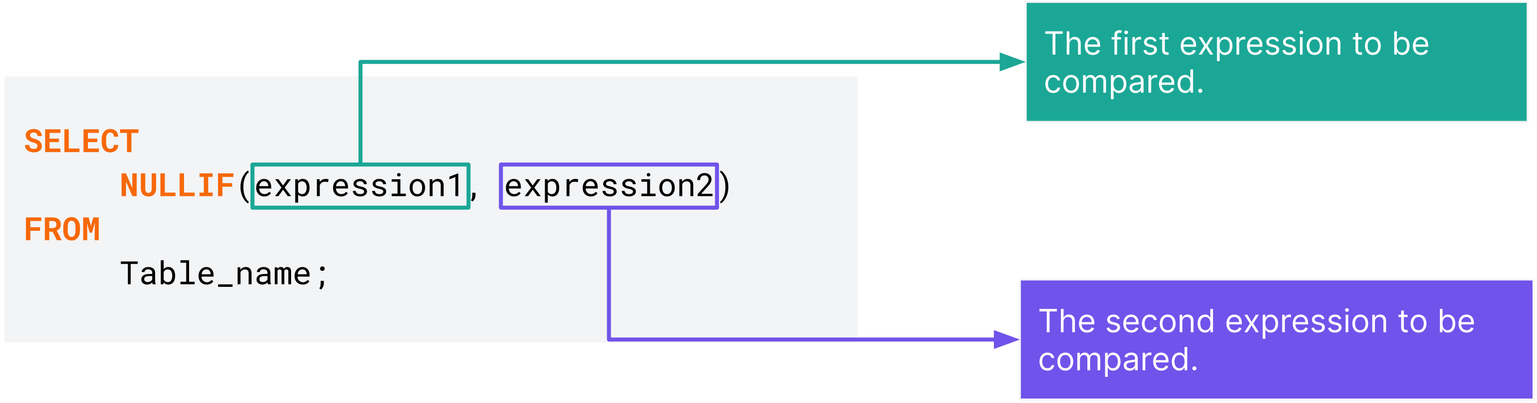
In this example, the IFNULL() function checks Highest_ed for NULL. If it encounters a NULL value, it replaces it with the alternative value, 'No schooling'.

Output

Highest_ed	New_highest_ed
NULL	No schooling
Undergraduate	Undergraduate
Primary	Primary
Diploma	Diploma
Secondary	Secondary

NULLIF() function

The **`NULLIF()`** function is used to **compare two expressions** and **return NULL if** they are **equal**. Otherwise, the first expression is returned. Its basic syntax is as follows:



```
SELECT
     NULLIF(expression1, expression2)
FROM
     Table_name;
```

The first expression to be compared.
The second expression to be compared.

The **`NULLIF()`** function provides a way of **marking certain values as NULL** in an effort to treat them as missing or unknown or to avoid particular errors.

NULLIF() function

On the **Age** column, the age of children below 1 year has been assigned 0. If we wish to exclude these records from the aggregations performed on the Age column, we can **convert the 0 values to NULL**. This ensures that the aggregation functions disregard these values.

Query

Output

```
SELECT
    Age,
    NULLIF(Age, 0) AS New_age
FROM
    Household_individuals;
```

In this example, if a value in the column Age is equal to 0, the NULLIF( ) function returns NULL. Otherwise, the original value is retained.

Age	New_age
0	NULL
22	22
35	35
14	14
2	2

ISNULL() function

The **`ISNULL()`** function helps to **determine whether** an **expression is NULL or not**. If the expression is NULL, this function returns 1. Otherwise, it returns 0. Its basic syntax is as follows:

```
SELECT
     ISNULL(expression)
FROM
     Table_name;
```

$\xrightarrow{}$ The value to test for NULL.

The **`ISNULL()`** function helps when we want to **filter our data** or **perform conditional logic** based on the presence of NULLS.

ISNULL() function

Suppose we want to investigate the cause behind the NULL values in **`Ed_institution`**. We can filter our data to **only remain with the NULL values** in that particular column. Can you identify some potential causes?

Query

```
SELECT
      Sex,
      Age,
      New_highest_ed
FROM
      Household_individuals
WHERE
      ISNULL(Ed_institution) = 1;
```

The `ISNULL()` function in the `WHERE` clause checks whether the values in `Ed_institution` are NULL. If NULL, it returns 1, and 0 otherwise. The `WHERE` clause then filters out the rows where the `ISNULL()` function returns 1, i.e. `Ed_institution` is NULL.

Output

Sex	Age	New_highest_ed
Male	0	No schooling
Male	35	Masters
Female	17	No schooling
Female	55	PHD

COALESCE() function

The **`COALESCE()`** function **evaluates a list of expressions** from left to right, searching for the **first non-NULL value** and **returning it**. If all the expressions are NULL, the function returns NULL. Its basic syntax is as follows:

```
SELECT
    COALESCE(expression1, expression2, expression3, ...)
FROM
    Table_name;
```

The list of values we want to check for NULL.

The **`COALESCE()`** function allows us to handle NULL values by providing an alternative or fallback value.

COALESCE() function

The **Spouse** column seems **redundant** since an individual will automatically have a spouse if married, or no spouse if single. We can **combine Spouse and Spouse_ID** to form a new column that reads the string 'N/A' if one is single or the spouse's ID if married.

Query

```
SELECT
      Marital_status,
      Spouse_ID,
      Spouse,
      COALESCE(Spouse_ID, Spouse) AS New_spouse_ID
FROM
      Household_individuals;
```

The COALESCE function starts by checking the Spouse_ID column and if its value is not NULL, it will be assigned as the value for the new column, New_spouse_ID. However, if the Spouse_ID value is NULL, the function will move on to evaluate the Spouse column for a non-null value. It's value, which in this case is the string 'N/A', is then returned in the new column.

Output

Marital_status	Spouse_ID	Spouse	New_spouse_ID
Single	NULL	N/A	N/A
Single	NULL	N/A	N/A
Married	3331	Yes	3331
Single	NULL	N/A	N/A
Married	3891	Yes	3891