SQL Table Transform Date, Number and String functions

1
Oftentimes, data in columns of tables is not in the exact format we need to complete our desired analysis. We may need to extract a date from a full timestamp, manipulate a number, or combine first and last name columns to create a full name.
In this lesson, we'll be learning about some of SQL's built-in functions for transforming dates, numbers and strings. We'll be using database of bakeries in this lesson.
It is important to note that **date, number, and string functions are highly database dependent**. Here, we focus on built-in functions in the SQLite database management system.

2
Dates are often written in the following format
1. Date: **YYYY-MM-DD**
2. **Datetime or Timestamp**: **YYYY-MM-DD hh:mm:ss**
We can use SQL's date functions to transform data into a desired format. Since date functions can be database specific, verify the functions that exist on your relational database management system.
For example, this statement
SELECT DATETIME(manufacture_time)
FROM baked_goods;
Would return the date and time for the manufacture_time column.

3.
Now let's assume that we have a column in our baked_goods table named manufacture_time in the format YYYY-MM-DD hh:mm:ss.
We'd like to know **the number of baked_goods manufactured by day**, and not by second. We can use the DATE() function to easily convert timestamps to dates and complete the following query:

SELECT **DATE(manufacture_time), count(*)** as count_baked_goods
FROM baked_goods
GROUP BY DATE(manufacture_time);

Similarly, we can query the time with
SELECT TIME(manufacture_time), count(*) as count_baked_goods
FROM baked_goods
GROUP BY TIME(manufacture_time);

4.
Given a datepart and a column of date or timestamp data type, we can

increment date or timestamp values by a specified interval.
For example, in SQLite, the statement
DATETIME(time1, '+3 hours', '40 minutes', '2 days');
Would return a time 3 hours, 20 minutes, and 2 days after time1.

Imagine that **each dessert in our baked_goods table is inspected 2 hours, 30 minutes, and 1 day after the manufacture time**. To derive the inspection date for each baked good, we can use the following query

SELECT DATETIME(manufacture_time, '+2 hours', '30 minutes', '1 day') as inspection_time
FROM baked_goods;

5.
Great work! **Numeric functions can be used to transform numbers**. Some common **SQLite** mathematical functions are included below that take numeric data types as inputs:

- **SELECT (number1 + number2)**;: Returns the sum of two numbers. Similar, SQL can be used for **subtraction, multiplication, and division**.
- **SELECT CAST(number1 AS REAL) / number3;**: Returns the result as a real number by **casting one of the values as a real number, rather than an integer.**
- SELECT ROUND(number, precision);: Returns the numeric value rounded off to the next value specified.

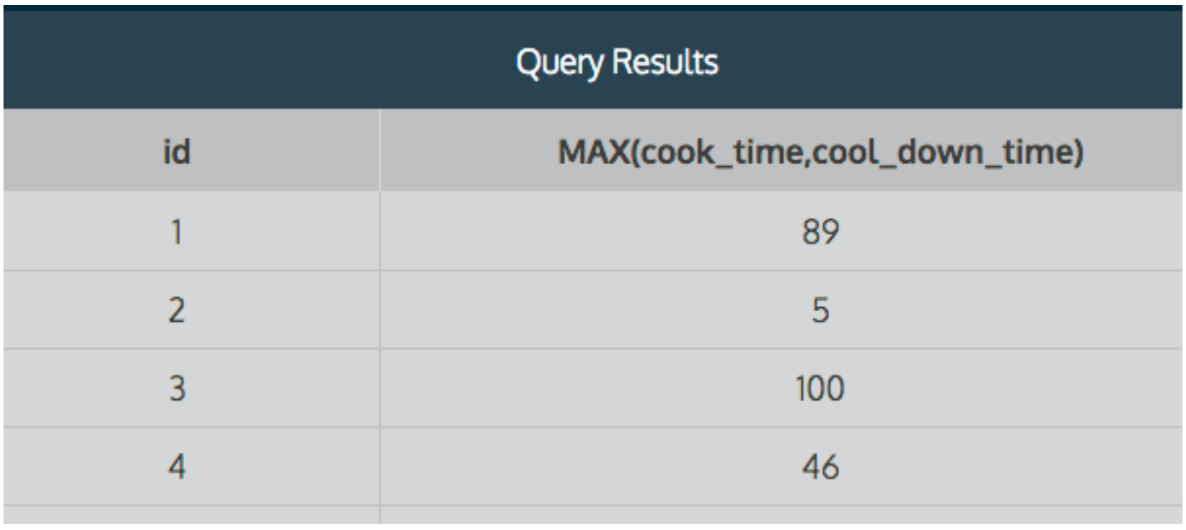
In our baked_goods table, we have information about cost designated by ingredients_cost. For accounting purposes, we'd like to **make sure that each ingredient cost is rounded to four decimal places rather than two,** to account for currency fluctuations.

SELECT **ROUND(ingredients_cost, 4)** as rounded_cost
FROM baked_goods;

6.
A couple more useful numeric SQL functions are included below: MAX and MIN.
MAX(n1,n2,n3,...): returns the **greatest value in the set of the input numeric expressions** MIN(n1,n2,n3,...): returns the least value in the set of the input numeric expressions
In our baked_goods table, in addition to the numeric ingredients_cost we have information about the packaging cost located in the packaging_cost column.
We can use the MAX function to determine the overall greatest value of cost for each item using the following query:
SELECT id, MAX(ingredients_cost, packaging_cost)
FROM baked_goods;

We also have information about cook time designated as cook_time and cool down time designated as cool_down_time in the baked_goods table. Find the greatest time value for each item in the table.

SELECT id,MAX(cook_time,cool_down_time)



Query Results

id	MAX(cook_time,cool_down_time)
1	89
2	5
3	100
4	46

from baked_goods;

7.
String manipulation can be useful to derive information from columns. We'll cover a couple of the common string functions here.
A common use case for string manipulation in **SQL is concatenation of strings**. In **SQLite**, this is written as
SELECT string1 || ' ' || string2;
For example, the bakeries table contains both city and state columns. In order to create a route for these columns, we use the || function to concatenate them as in the following query:
SELECT city || ' ' || state as location
FROM bakeries;
String functions are again, very database specific, and it is best practice to consult documentation before proceeding.

8.
Another useful string function in SQL is REPLACE():
REPLACE(string,from_string,to_string)
The function returns the string string with all occurrences of the string from_string replaced by the string to_string.
For example in baked_goods, there is a column named ingredients. The ingredients strings are formatted with underscores, such as baking_soda and vanilla_extract. To make these values more readable, we might like to replace the underscores with spaces. We can do so by using the following query:
SELECT id, REPLACE(**ingredients,'_',' ')** as item_ingredients
from baked_goods;

REPLACE(din_field, ‘string_de_inlocuit’, ‘string_inlocuitor’)

Date Functions:

- **DATETIME;** Returns the date and time of the column specified. This can be modified to return only the date or only the time.
- **DATETIME(time1, +X hours, Y minutes, Z days):** Increments the specificed column by a given number of hours, minutes, or days.

Numeric Functions:

- (number1 + number2);: Returns the sum of two numbers, or other mathematical operations, accordingly.
- **CAST(number1 AS REAL) / number2;:** Returns the result as **a real number by casting one of numeric inputs** as a real number
- **ROUND(number, precision);**: Returns the numeric value rounded off to the next value specified.

String Functions:

- **'string1' || ' ' || 'string2'**;: Concatenates string1 and string 2, with a space between.
- **REPLACE(string,from_string,to_string):** Returns the string with all occurrences of the string from_string replaced by the string to_string.