Standard SQL Functions Cheat Sheet

Text Functions

CONCATENATION

Use the || operator to concatenate two strings:

```
SELECT 'Hi' || 'there!';
-- result: Hi there!
```

Remember that you can concatenate only character strings using ||. Use this trick for numbers:

```
SELECT '' || 4 || 2;
-- result: 42
```

Some databases implement non-standard solutions for concatenating strings like CONCAT() or CONCAT_WS(). Check the documentation for your specific database.

LIKE OPERATOR - PATTERN MATCHING

Use the_character to replace any single character. Use the % character to replace any number of characters (including 0 characters).

Fetch all names that start with any letter followed by 'atherine':

```
SELECT names
FROM names
WHERE name LIKE '_atherine';
```

Fetch all names that end with 'a':

```
SELECT names
FROM names
WHERE name LIKE '%a';
```

USEFUL FUNCTIONS

Get the count of characters in a string:

```
SELECT LENGTH('CoderSQL.com');
-- result: 12
```

Convert all letters to lowercase:

```
SELECT LOWER('CODERSQL.com');
-- result: codersql.com
```

Convert all letters to uppercase:

```
SELECT UPPER('CODERSQL.com');
-- result: CODERSQL.COM
```

Convert all letters to lowercase and all first letters to uppercase (not implemented in MySQL and SQL Server):

```
SELECT INITCAP('edgar frank ted cODD');
-- result: Edgar Frank Ted Codd
```

Get just a part of a string:

```
SELECT SUBSTRING('CODERSQL.com', 9);
-- result: .com
```

Get just a part of a string:

```
SELECT SUBSTRING('CODERSQL.com', 0, 6);
-- result: Coder
```

Replace part of string:

```
SELECT REPLACE('CODERSQL.com', 'SQL', 'Python');
-- result: CoderPython.com
```

Numeric Functions

BASIC OPERATIONS

Use +, -, *, / to do some basic math. To get the number of seconds in a week:

```
SELECT 60 * 60 *60 * 7;
-- result: 604800
```

CASTING

From time to time, you need to change the type of a number. The CAST( ) function is there to help you out. It lets you change the type of value to almost anything (integer, numeric, double precision, varchar, and many more).
Get the number as an integer (without rounding):

```
SELECT CAST (1234.567 AS integer);
-- result: 1234
```

Change a column type to double precision

```
SELECT CAST (column AS double
precision);
```

USEFUL FUNCTIONS

Get the remainder of a division:

```
SELECT MOD (13, 2);
-- result: 1
```

Round a number to its nearest integer:

```
SELECT ROUND (1234.56789);
-- result: 1235
```

Round a number to three decimal places:

```
SELECT ROUND (1234.56789, 3);
-- result: 1234.568
```

PostgreSQL requires the first argument to be of the type numeric - cast the number when needed.

To round the number **up**:

```
SELECT CEIL(13.1); -- result: 14
SELECT CEIL(-13.9); -- result: -13
```

The CEIL(x) function returns the **smallest** integer **not less** than x. In SQL Server, the function is called CEILING().

To round the number **down**:

```
SELECT FLOOR(13.8);-- result: 13
SELECT FLOOR(-13.2);-- result: -14
```

The FLOOR(x) function returns the **greatest** integer **not greater** than x.

To round towards 0 irrespective of the sign of a number:

```
SELECT TURNC(13.5);-- result: 13
SELECT TURNC(-13.5);-- result: -13
```

TRUNC(x) works the same way as CAST (x AS integer). In MySQL, the function is called TRUNCATE().

To get the absolute value of a number:

```
SELECT ABS(-12);-- result: 12
```

To get the square root of a number:

```
SELECT SQRT(9);-- result: 3
```

Standard SQL Functions Cheat Sheet

NULLs

To retrieve all rows with a missing value in the price column:

```
WHERE price IS NULL
```

To retrieve all rows with the weight column populated:

```
WHERE weight IS NOT NULL
```

Why shouldn't you use price = NULL or weight != NULL?
Because databases don't know if those expressions are true or false - they are evaluated as NULLs.
Moreover, if you use a function or concatenation on a column that is NULL in some rows, then it will get propagated. Take a look:

domain	LENGTH(domain)
CODERSQL.com	12
CoderPython.com	15
NULL	NULL
vertabelo.com	13

USEFUL FUNCTIONS

COALESCE(x, y, ...)

To replace NULL in a query with something meaningful:

```
SELECT
    domain,
    COALESCE (domain, 'domain missing');
SELECT contacts;
```

domain	coalesce
CODERSQL.com	CODERSQL.com
NULL	domain missing

The COALESCE() function takes any number of arguments and returns the value of the first argument that isn't NULL.

NULLIF(x, y)

To save yourself from division by 0 errors:

```
SELECT
    last_month,
    this_month,
    this_month * 100.0
    /  NULLIF (last_month, 0);
    AS better_by_percent
FROM video_views;
```

last_month	this_month	better_by_percent
723786	1085679	1085679
0	178123	NULL

The NULLIF (x, y) function will return NULL if x is the same as y, else it will return the x value.

Standard SQL Functions Cheat Sheet

Case When

The basic version of CASE WHEN checks if the values are equal (e.g., if fee is equal to 50, then ‘normal’ is returned). If there isn't a matching value in the CASE WHEN, then the ELSE value will be returned (e.g., if fee is equal to 49, then ‘not available’ will show up.

```
SELECT
    CASE fee
      WHEN 50 THEN ‘normal’
      WHEN 10 THEN ‘reduced’
      WHEN  0 THEN ‘free’
      ELSE  ‘not available’
    END  AS tariff
FROM ticket_types;
```

The most popular type is the **searched CASE WHEN** — it lets you pass conditions (as you'd write them in the WHERE clause), evaluates them in order, then returns the value for the first condition met.

```
SELECT
    CASE
      WHEN score >= 90 THEN ‘A’
      WHEN score > 60 THEN ‘B’
      ELSE ‘F’
    END  AS grade
FROM test_results;
```

Here, all students who scored at least 90 will get an A, those with the score above 60 (and below 90) will get a B, and the rest will receive an F.

Troubleshooting

Integer division

When you don't see the decimal places you expect, it means that you are dividing between two integers. Cast one to decimal:

```
CAST(123 AS decimal) / 2
```

Division by 0

To avoid this error, make sure that the denominator is not equal to 0. You can use the NULLIF( ) function to replace 0 with a NULL, which will result in a NULL for the whole expression:

```
count / NULLIF(count_all, 0)
```

Inexact calculations

If you do calculations using real (floating point) numbers, you'll end up with some inaccuracies. This is because this type is meant for scientific calculations such as calculating the velocity.
Whenever you need accuracy (such as dealing with monetary values), use the decimal / numeric type (or money if available).

Errors when rounding with a specified precision

Most databases won't complain, but do check the documentation if they do. For example, if you want to specify the rounding precision in PostgreSQL, the value must be of the numeric type.

Source: Internet