kokchun giang

working with **strings** in sql and duckdb to format and clean text

using array-like **slicing and indexing** to get substrings

```
SELECT
    sql_word,
    sql_word[:2],
    sql_word[2:5]
FROM
    staging.sql_glossary;
```

slices from index 1 to 2 (SQL counts from 1)

slices from index 2 to 5, so gets the substring composing of characters on index 2 to 5

```
SELECT
    sql_word,
    sql_word[0],
    sql_word[1],
    sql_word[-1],
FROM
    staging.sql_glossary;
```

returns empty string as there is no strings in 0th position

gets the last character in the string

using **string functions** to transform text

```sql
SELECT
    upper(trim(sql_word, ' ')) as trimmed_word,
    trimmed_word[1],
    trimmed_word[-1]
FROM
    staging.sql_glossary;
```

composition of functions or
nested functions

many useful **text functions** in the documentation

Text Functions and Operators

This section describes functions and operators for examining and manipulating `STRING` values.

Name	Description
`string ^@ search_string`	Return true if `string` begins with `search_string`.
`string || string`	Concatenate two strings. Any NULL input results in NULL. See also `concat(string, ...)`.
`string[index]`	Extract a single character using a (1-based) index.
`string[begin:end]`	Extract a string using slice conventions, see slicing.
`string LIKE target`	Returns true if the `string` matches the like specifier (see Pattern Matching).
`string SIMILAR TO regex`	Returns `true` if the `string` matches the `regex`; identical to `regexp_full_match` (see Patter Matching).
`array_extract(list, index)`	Extract a single character using a (1-based) index.
`array_slice(list, begin, end)`	Extract a string using slice conventions. Negative values are accepted.
`ascii(string)`	Returns an integer that represents the Unicode code point of the first character of the `string`.
`bar(x, min, max[, width])`	Draw a band whose width is proportional to (`x - min`) and equal to `width` characters when `x`

using **regex** for more complex pattern matching

```sql
SELECT
    regexp_replace (trim(description), ' +', ' ', 'g') AS cleaned_description,
    regexp_replace (trim(lower(example)), ' +', ' ', 'g') AS example,
FROM
    staging.sql_glossary;
```

replaces every matching of the pattern ' +' (one or more spaces) and replaces it with ' ' (one space)

using schemas to divide into **layers** for structure

```sql
CREATE SCHEMA IF NOT EXISTS staging;

CREATE TABLE IF NOT EXISTS staging.sql_glossary AS (
    SELECT * FROM read_csv_auto('data/sql_terms.csv')
);
```

staging schema is used to represent a staging layer, kind of landing zone for data

```sql
CREATE SCHEMA IF NOT EXISTS refined;

CREATE TABLE
    IF NOT EXISTS refined.sql_glossary AS (
        SELECT
            UPPER(TRIM(sql_word)) AS sql_word,
            description,
            example
        FROM
            staging.sql_glossary
    );
```

refined layer or warehouse layer is where data is transformed