Simple queries on a single table

> The tutorial is based on the slides of Stephane Faroult
>
> Designed by ZHU Yueming and Huang Yu'an in 2019.
>
> Improved by ZHU Yueming in Feb. 21th, 2022.

Experimental-Objective

1. To learn how to query from a single table.
2. To understand key words - `like`, `in`, `or`, `range` etc.

SQL: two main components

- Data Definition Language: The data definition language(DDL) deals with tables. (`create, alter, drop`)
- Data Manipulation Language: The data manipulation language (DML) deals with data. (`insert, update, delete, select`)

The basic syntax of SQL is very simple. SELECT is followed by the names of the columns you want to return, FROM by the name of the tables that you query, and WHERE by filtering conditions.
`select ... from ... where ...`

Let's query!

Before you start, please download the **filmdb.sqlite** which has been uploaded in Sakai. You can use datagrip to connect it (need to download relative driver, datagrip can do it automatically) or use other tools (such as SQLiteStudio).

After you doing this, try thinking those questions and write your queries to validate them.

Queries without any aggregate

Restriction

> When tables contains thousands or millions or billions of rows, you are usually interested in only a small subset, and only want to return some of the rows.
>
> Filtering is performed in the "where" clause, with conditions that are usually expressed by a column name followed by a comparison operator and the value to which the content of the column is compared.

1. Basic Filtering

(1) or represent the logic or

```
select * from stations where station_id='10' or station_id='11';
```

(2) and represent the logic and

```
select * from stations where district='Futian' and latitude>22.57;
```

(3) not represent the logic not

```
select * from stations where not (district='Futian' and latitude>22.57);
```

```
select * from stations where district not in ('Futian', 'Nanshan');
```

One thing to remember is that, like numerical operators, all logical operators haven't the same precedence, like `*` is stronger than `+` *1+2*3*, `and` is stronger than `or` and `not`.

2. number 'constraint' column

(1) >, <, >=, <=, != or <>

```
select * from stations where latitude>22.6 and latitude<23;
```

```
select * from stations where district<>'Nanshan';
```

(2) between ... and ...

```
select * from stations where latitude between  22.6 and 23;
```

Using `between ... and ...` can be shorter than the query below.

```
select * from stations where latitude>22.6 and latitude<23;
```

3. other 'constraint' column

(1) in

It can be used as the equivalent for a series of equalities with OR (it has also other interesting uses). It may make a comparison clearer than a parenthesized expression.

```
select * from stations where longitude>114.14 and (district like 'Longgang' or
district like 'Luohu');
```

Keywords and identifiers are not case-sensitive in sql. More, for this problem, you can also write:

```
select * from stations where longitude>114.14 and district in ('Longgang'
,'Luohu');
```

(2) like '%X%', like '%X', like 'X%'

For strings, you also have LIKE which is a kind of regex (regular expression) for dummies. LIKE compares a string to a pattern that can contain two percent characters, % meaning "any number of characters, including none and _ meaning one and only one character

```
select * from lines where line_color like '%Green';
```

(3) Is Null , Is not Null

NULL in SQL is NOT a value, so that the sign `=` can not be used with `NULL`.

The only thing you can test is whether a column contains a value or not, which is done with the special operator IS (NOT) NULL

Example: Find all lines which is not being opened.

```
select * from lines where opening is null;
```

In this case, NULL is not a value, so you should not use `opening == NULL` which is always false (`opening != NULL` return false either).

4. Transform data in Select part

> One important feature of SQL is that you needn't return data exactly as it was stored. Operators, and a large number of (mostly DBMS specific) functions allow to return transformed data.

(1) ||

Symbol `||` is to concatenate two strings together.

Return a sentence in the format "The code of [name] is [hex]"

```
select 'The code of '||c.name||' is '|| c.hex
from color_names c;
```

?column?
The code of AliceBlue is #F0F8FF
The code of AntiqueWhite is #FAEBD7
The code of Aqua is #00FFFF
The code of Aquamarine is #7FFFD4
.....

(2) as

Symbol `as` is to rename this column (can ignore). There are other useful functions to operate strings.

Here we use `combine` replace of `The code of '||c.name||' is '|| c.hex`

```
select 'The code of ' || c.name || ' is ' || c.hex as combine
from color_names c;
```

combine
The code of AliceBlue is #F0F8FF
The code of AntiqueWhite is #FAEBD7
The code of Aqua is #00FFFF
The code of Aquamarine is #7FFFD4
.....

(3) case... when... when ... else... end

A very useful construct is the case... end construct that is similar to IF or SWITCH statements in a program.

The Grammar 1：

```
case when [condition1] then [result1]
     when [condition2] then [result2]
     ......
     else [resultn]
     end as [user_defined_name]
from [table]
where ......
```

The Grammar 2:

```
case [column]
      when [value1] then [result1]
      when [value2] then [result2]
      ......
      else [resultn]
      end as [user_defined_name]
from [table]
where ......
```

Example for Grammar 1:

```
select s.english_name,
       case
           when s.latitude is null then 'missing'
           when s.district in ('Luohu','Futian','Nanshan') then 'Inside Area'
           else 'Outside Area' end as area
from stations s;
```

Example for Grammar 2:

```
select round(100.0 * sum(case s.district
                         when 'Luohu' then 1
                         when 'Futian' then 1
                         when 'Nanshan' then 1
                         else 0 end) / count(*),3)||'%' as Inside_rate
from stations s;
```

(4) Some useful functions

- upper(), lower()
- trim(' Oops ') return 'Oops'
- substr('Citizen Kane', 5, 3) return 'zen'
- replace('Sheep', 'ee', 'i') return 'Ship'
- length()
- round() trunc()

Some important things to remember about SQL

- Keywords and identifiers are **not** case-sensitive (Data: can be CASE-SENSITIVE).
- **NULL** is not a value so you could not use `where column_name == NULL`.
- Avoid applying function to columns that are used for comparison (`where upper(column_name)=='AAXX'`).
- Aggregate functions ignore NULLs.