Tutorial4 - Simple queries on a single table

The tutorial is based on the slides of Stephane Faroult

Designed by ZHU Yueming and Huang Yu'an.

Experimental-Objective

1. To learn how to query from a single table.
2. To understand key words - `like`, `in`, `or`, `range` etc.
3. To learn how to use some aggregate function along with `distinct, group by` and `having`.

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 bb website. 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 that, 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 movies where country='cn' or country='jp';
```

(2) and represent the logic and

```
select * from movies where country='cn' and year_released='2000';
```

(3) not represent the logic not

```
select * from movies where not (country='cn' and country='jp');
```

```
select * from movies where country not in ('cn','jp');
```

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 movies where year_released>=2000 and year_released<=2010;
```

```
select * from movies where year_released<>2000;
```

(2) between ... and ...

```
select * from movies where year_released between 2000 and 2010;
```

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

```
select * from movies where year_released>=2000 and year_released<=2010;
```

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 movies where RUNtime > 120 and (country == 'uk' or
country=='cn');
```

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

```
select * from movies where runtime > 120 and country in ('uk', 'cn');
```

(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 wildcard characters, % meaning "any number of characters, including none" and _ meaning "one and only one character"

```
select * from movies
where title not like '%A%' and title not like '%a%';
```

This expression for instance returns films the title of which doesn't contain any A. This A might be the first or last character as well. Note that if the DBMS is case- sensitive you need to cater both for upper and lower case.

Q1.
Query people whose first name starts with 'D' and surname ends with 'a'.

```
select * from people where first_name like 'D%' and surname like '%a';
```

Noted that *SQLite* isn't case sensitive with like_.

(3) Is Null , Is not Null

NULL in SQL is NOT a value, and if it's not a value, hard to say if a condition is true.

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

Q2.
Query people which burned between in 1970 and 1975 and has dead.

```
select * from people
where born between 1970 and 1975 and died is not NULL ;
```

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

Q3.
For each people that was born after 2000, Output a sentence in format "[who] was born in [year]."

```
select first_name || ' ' ||surname|| ' was born in ' || born
   from people
where born > 2000;
```

(2) as

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

Here we use **born_info** replace of **first_name || ' ' ||surname|| ' was born in ' || born**

```
select first_name || ' ' ||surname|| ' was born in ' || born as born_info
   from people
  where born > 2000;
```

(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.

```
select born, died,
case
  when died is null then
        'alive and kicking'
  else 'passed away'
end as status
from people limit 10;
```

(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()

Queries with aggregate functions

> As the name says, aggregate function will aggregate all rows that share a feature (such as being films from the same country) and return a characteristic of each group of aggregated rows. It will be clearer with an example.

(1) count(*)

aggregate functions ignore Nulls.

Q4.
How many people' first name is NULL which born after 1900?

```
select count(*)-count(first_name) as null_people_num from people where born >
1990;
```

All aggregate functions ignore NULLs, so result of count(*) may be different from count([column]).

round() and trunc()

Q5.
What is (rounded to the first decimal) the percentage of women in the database?

```
select round(100 * sum(
   case gender when 'F'
     then 1
     else 0
    end
) / count(*), 1)  as percentage_of_women from people;
```

Function round will save a few decimal places. Also you can use trunc().

- round(3.141592, 3) result is 3.142
- trunc(3.141592, 3) result is 3.141

(2) max(),min(), avg()

Q6.
Show age of the youngest person in the people table.

```
select current_date - max(born) young_age from people;
```

(3) group by

Q7. Show number of all genders in people table.

```
select count(gender) as num, gender from people group by gender;
```

The `group by` say that we want to group by gender, and for each gender the aggregate function count(*) says how many people it has. And pleased noted that we can group by multiple columns.

(4) having

`Having` is like a filter condition. But more than `where`, you can write something like `having count(gender) > 100`.

Q8. Show number of male and female in people table.

```
select count(gender) num, gender from people
 where gender in ('F', 'M')
group by gender;
```

But the efficiency may not be high as having (The reason why 'may' is that all DBMS have a highly important component called "query optimizer".).

You can also write by using **having** as follows:

```
select count(gender) as num, gender from people group by gender
having gender in ('F', 'M');
```

(5) distinct

If we only are interested in the different countries, there is a special keyword: `distinct`.

Q9.
There are films from how many countries/territories in the database?

```
select count(distinct country) as num_of_coun_movies from movies;
```

Query from the result set of another query

Q10.
The most common surname in the database can be found how many times?

```
select max(cnt) from (
    select surname, count(*) cnt from people group by surname
) name_count;
```

You can query from the result set of another query and also named the result set.

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.

What to submit

Please finish following query:

What is the number of Chinese films that we have per year, since 1960 (included)? The result set should contain the nunber and its corresponding year.

Submit a sql file into Sakai website.