Principles of Database Systems (CS307)

Lecture 7: More about NULL; Ordering; Window Function; Function

Ran Cheng

Department of Computer Science and Engineering
Southern University of Science and Technology

- Most contents are from slides made by Stéphane Faroult and the authors of Database System Concepts (7th Edition).
- Their original slides have been modified to adapt to the schedule of CS307 at SUSTech.

More about NULL

Expressions with NULL Values

- Most expressions with `NULL` will be evaluated into `NULL`
  - Arithmetic operations:

```
col+NULL -> NULL
col-NULL -> NULL
col*NULL -> NULL
col/NULL -> NULL

...
```

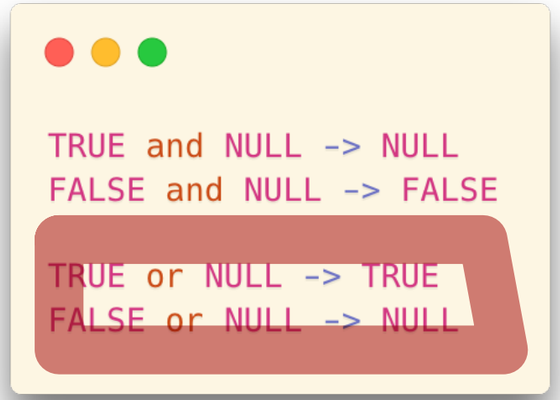
  - Comparison operations:

```
(col > NULL) -> NULL
(col = NULL) -> NULL

...
```

Expressions with NULL Values

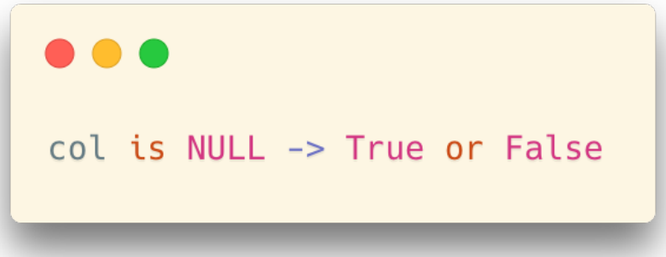
- Most expressions with `NULL` will be evaluated into `NULL`
  - But, there are some conditions where the values are not `NULL`



Logical operators (or, and):

- Three-valued logic (true, false, and unknown)

More on this: Three-valued logic and its application in SQL
`https://en.wikipedia.org/wiki/Three-valued_logic#SQL`



The way we use to check a `NULL` value: use `is`, not `=`

Recall: Subquery after Where

- Some important points for `in()`
  - `in()` means an implicit distinct in the subquery
    - `in('cn', 'us', 'cn', 'us', 'us')` is equal to `in('cn', 'us')`
  - null values in `in()`
    - Be extremely cautious if you are using `not in(...)` with a null value in it

```
value not in(2, 3, null)
⇒not (value=2 or value=3 or value=null)
⇒value<>2 and value<>3 and value<>null
⇒always false or null, never true
```

- If value is 2, the result is:
  `FALSE and TRUE and NULL -> FALSE`
- if value is 5, the result is:
  `TRUE and TRUE and NULL -> NULL`
- if value is NULL, the result is :
  `NULL and NULL and NULL -> NULL`

Ordering

Ordering in SQL

- `order by`
  - A simple expression in SQL to order a result set
  - It comes at the end of a query
    - ... and, you can have it in subqueries, definitely
  - Followed by the list of columns used as sort columns

```
select title, year_released
from movies
where country = 'us'
order by year_released;
```

	title	year_released
1	Ben Hur	1907
2	The Lonely Villa	1909
3	From the Manger to the Cross	1912
4	Falling Leaves	1912
5	Traffic in Souls	1913
6	At Midnight	1913
7	Lime Kiln Field Day	1913
8	The Sisters	1914
9	The Only Son	1914
10	Tess of the Storm Country	1914
11	Under the Gaslight	1914
12	Brute Force	1914
13	The Wishing Ring: An Idyll of Old England	1914

Ordering in SQL

- No matter how difficult the query is, you can apply order by to any result set

```
select m.title,
       m.year_released
from movies m
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.born >= 1970)
order by m.year_released
```

	title	year_released
1	Snehaseema	1954
2	Nairu Pidicha Pulivalu	1958
3	Mudiyanaya Puthran	1961
4	Puthiya Akasam Puthiya Bhoomi	1962
5	Doctor	1963
6	Aadyakiranangal	1964
7	Odayil Ninnu	1965
8	Adimakal	1969
9	Karakanakadal	1971
10	Ghatashraddha	1977
11	Kramer vs. Kramer	1979
12	The Champ	1979
13	The Shining	1980

Ordering in SQL

- Ordering with joins
  - We can sort by any column of any table in the join (remember the super wide table with all the columns from all tables involved)

```
select c.country_name,
       m.title,
       m.year_released
from movies m
     inner join countries c
     on c.country_code = m.country
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.born >= 1970)
order by m.year_released
```

	country_name	title	year_released
1	India	Snehaseema	1954
2	India	Nairu Pidicha Pulivalu	1958
3	India	Mudiyanaya Puthran	1961
4	India	Puthiya Akasam Puthiya Bhoomi	1962
5	India	Doctor	1963
6	India	Aadyakiranangal	1964
7	India	Odayil Ninnu	1965
8	India	Adimakal	1969
9	India	Karakanakadal	1971
10	India	Ghatashraddha	1977
11	United States	Kramer vs. Kramer	1979
12	United States	The Champ	1979
13	United States	The Shining	1980

Ordering in SQL

- Ordering with joins
  - We can sort by any column of any table in the join (remember the super wide table with all the columns from all tables involved)

```sql
select c.country_name,
       m.title,
       m.year_released
from movies m
     inner join countries c
     on c.country_code = m.country
where m.movieid in
  (select distinct c.movieid
   from credits c
        inner join people p
        on p.peopleid = c.peopleid
   where c.credited_as = 'A'
     and p.born >= 1970)
order by m.year_released
```

	country_name	title	year_released
1	India	Snehaseema	1954
2	India	Nairu Pidicha Pulivalu	1958
3	India	Mudiyanaya Puthran	1961
4	India	Puthiya Akasam Puthiya Bhoomi	1962
5	India	Doctor	1963
6	India	Aadyakiranangal	1964
7	India	Odayil Ninnu	1965
8	India	Adimakal	1969
9	India	Karakanakadal	1971
10	India	Ghatashraddha	1977
11	United States	Kramer vs. Kramer	1979
12	United States	The Champ	1979
13	United States	The Shining	1980

Advanced Ordering

- Multiple columns
  - For example:
    - The result set will be order by `col1` first
    - If there are rows with the same value on `col1`, these rows will be ordered by `col2`.

```
order by col1, col2, ...
```

- Ascending or descending order
  - Add `desc` or `asc` after the column
    - However, `asc` is the default option and thus always omitted

```
-- Order col1 descendingly
order by col1 desc

-- Order based on col1 first, then col2.
-- col1 will be in the descending order, col2 ascending.
order by col1 desc, col2 asc, ...
```

Advanced Ordering

- Self-defined ordering
  - Use “`case` … `when`” in `order by` to define criteria on how to order the rows

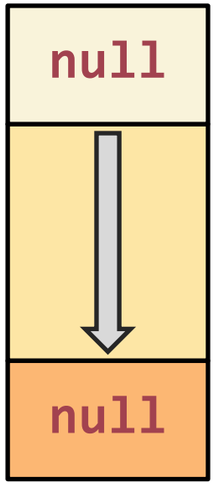
```
select * from credits
order by
    case credited_as
        when 'D' then 1
        when 'A' then 2
    end desc;
```

Data Types in Ordering

- Ordering depends on the data type
  - Strings: alphabetically,
  - Numbers: numerically
  - Dates and times: chronologically

Data Types in Ordering

- What about `NULL`?
  - It is implementation-dependent
  - SQL Server, MySQL and SQLite:
    - “nothing” is smaller than everything
  - Oracle and PostgreSQL:
    - “nothing” is greater than anything



Ordering in Text Data

- Remember, we have many different languages other than English
  - “Alphabetical order” in different languages means different things
    - Mandarin: Pinyin? Number of strokes?
    - Swedish and German
      - “ö” is considered the last letter in Swedish, while in German it is ordered after “o”.

Self Study: Text Encoding

- Key Question: How does characters represented in a computer?
  - Wikipedia – Character encoding: `https://en.wikipedia.org/wiki/Character_encoding`
  - A video on Bilibili: `https://www.bilibili.com/video/BV1xP4y1J7CS`

Self Study: Text Encoding

手持两把锟斤拷，
口中疾呼烫烫烫。
脚踏千朵屯屯屯，
笑看万物锘锘锘。

- Try to answer the following questions:
  - What are ASCII, Unicode, UTF-8, and UTF-16? What are the relationships between them?
  - What are GB2312, GB18030, and GBK? What are “锟斤拷” and “烫烫烫”? How can you make it (not) happen?
  - Given a string with several characters, can you print the bitmap of this string?
  - Are emojis characters? How can you insert an emoji in a text editor?
  - What are the default character encodings in different platforms?
    - OS: Windows, MacOS, Linux
    - DBMS: PostgreSQL, etc.
    - Programming Languages: Java, C, C++, Python, etc.
  - How can we translate strings from one encoding to another?
    - E.g., with text editors (Windows Notepad, VSCode, Sublime Text, etc.); in programming languages; in DBMS

Limit and Offset

- Get a slice of the long query result
  - `limit k offset p`
    - Return the top-k rows in the result set and skip the first p rows
    - `offset` is optional (which means "`offset 0`")
  - Always used together with order by
    - E.g., get the top-k query results under a certain ordering criteria
  - * In some DBMS, the syntax can be different
    - Always refer to the software manual for specific features

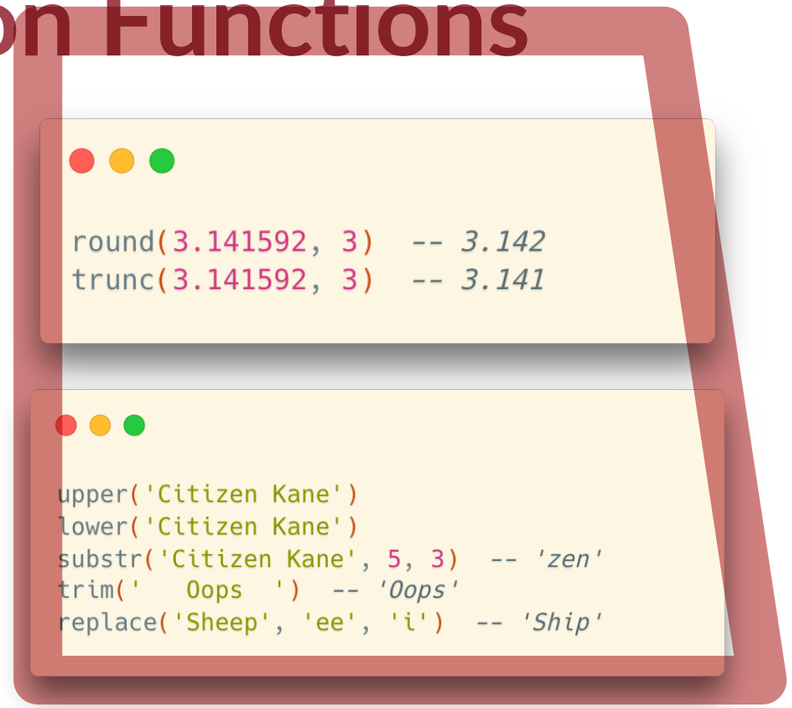
```
select * from movies
where country = 'us'
order by year_released
limit 10 offset 5
```

```
select * from movies
where country = 'us'
order by year_released
limit 10
```

Window Function

Scalar Functions and Aggregation Functions

- Scalar function
  - Functions that operate on values in the current row
    - Recall: “Some Functions”, Lecture 3



```
round(3.141592, 3)  -- 3.142
trunc(3.141592, 3)  -- 3.141
```

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

- Aggregation function
  - Functions that operate on sets of rows and return an aggregated value
    - Recall: “Aggregate Functions”, Lecture 4

```
count(*)/count(col), min(col), max(col), stddev(col), avg(col)
```

Issues with Aggregate Functions

- A Problem: In aggregate functions, the details of the rows are vanished
  - For example: If we ask for the year of the oldest movie per country,
    - … we get a country, a year, and nothing else.

Issues with Aggregate Functions

- A Problem: In aggregate functions, the details of the rows are vanished
  - For example: If we ask for the year of the oldest movie per country,
    - … we get a country, a year, and nothing else.

If we want some more details, like the title of the oldest movies for each country, we can only use self-join to keep the columns

```
select m1.country,
       m1.title,
       m1.year_released
from movies m1
     inner join
     (select country,
        min(year_released) minyear
        from movies
        group by country) m2
on m2.country = m1.country and m2.minyear = m1.year_released
order by m1.country
```

Issues with Aggregate Functions

- A Problem: In aggregate functions, the details of the rows are vanished
  - Another example: How can we rank the movies in each country separately based on the released year?
    - “order by” for subgroups
  - One more example: Get the top-3 oldest movies for each country.
    - How can we implement it?

Window Function

- Syntax:

```
<function> over (partition by <col_p> order by <col_o1, col_o2, …>)
```

  - `<function>`: we can apply (1) ranking window functions, or (2) aggregation functions
  - `partition by`: specify the column for grouping
  - `order by`: specify the column(s) for ordering in each group

Ranking Window Function

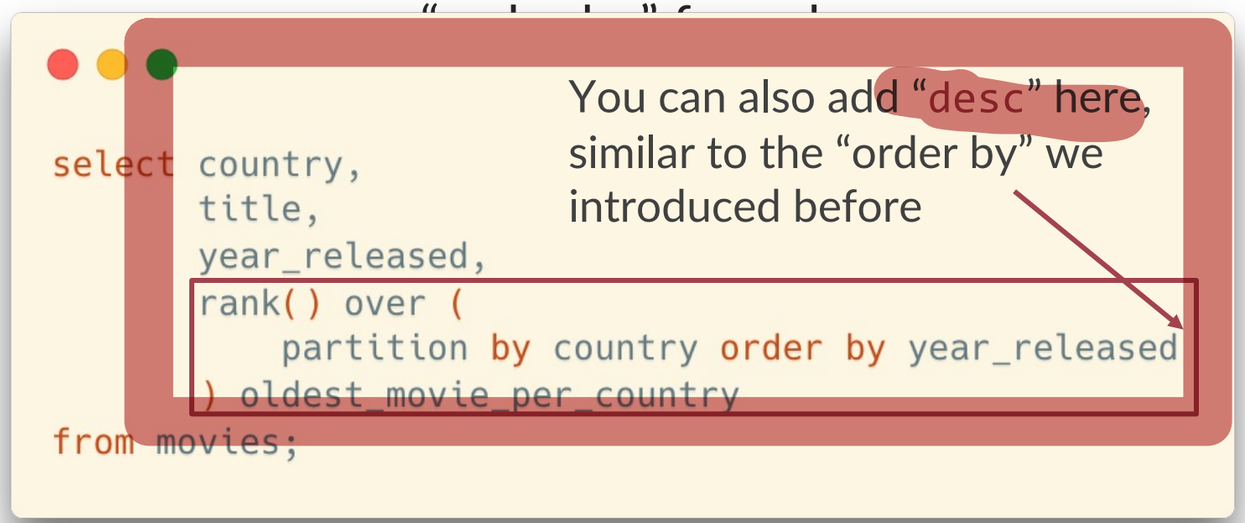
- Example
  - How can we rank the movies in each country separately based on the released year?
    - “order by” for subgroups

```
select country,
       title,
       year_released,
       rank() over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

	country	title	year_released	oldest_movie_per_country
1	am	Sayat Nova	1969	1
2	ar	Pampa bárbara	1945	1
3	ar	Albéniz	1947	2
4	ar	Madame Bovary	1947	2
5	ar	La bestia debe morir	1952	4
6	ar	Las aguas bajan turbias	1952	4
7	ar	Intermezzo criminal	1953	6
8	ar	La casa del ángel	1957	7
9	ar	Bajo un mismo rostro	1962	8
10	ar	Las aventuras del Capitán Piluso	1963	9
11	ar	Savage Pampas	1966	10
12	ar	La hora de los hornos	1968	11
13	ar	Waiting for the Hearse	1985	12
14	ar	La historia oficial	1985	12
15	ar	Hombre mirando al sudeste	1986	14

Ranking Window Function

- Example
  - How can we rank the movies in each country separately based on the released year?



country	title	year_released	oldest_movie_per_country
ar	some title	1948	1
ar	some title	1959	2
ar	some title	1980	3
cn	some title	1987	1
cn	some title	2002	2
uk	some title	1985	1
uk	some title	1992	2
uk	some title	2010	3

`partition by country`
- the selected rows will be grouped (partitioned) according to the values in the column `country`

`rank()`
- A function to say that "I want to order the rows in each partition"
- No parameters in the parentheses

`order by year_released`
- In each group (partition), the rows will be ordered by the column "`year_released`"

Ranking Window Function

- Example
  - How can we rank the movies in each country separately based on the released year?

```
select country,
       title,
       year_released,
       rank() over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

country	title	year_released	oldest_movie_per_country
ar	some title	1948	1
ar	some title	1959	2
ar	some title	1980	3
cn	some title	1987	1
cn	some title	2002	2
uk	some title	1985	1
uk	some title	1992	2
uk	some title	2010	3

Note: partition functions can only be used in the `select` clause

- … since it is designed to work on the query result

Ranking Window Function

- Example
  - How can we rank the movies in each country separately based on the released year?

```
select country,
       title,
       year_released,
       rank() over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

country	title	year_released	oldest_movie_per_country
ar	some title	1948	1
ar	some title	1959	2
ar	some title	1980	3
cn	some title	1987	1
cn	some title	2002	2
uk	some title	1985	1
uk	some title	1992	2
uk	some title	2010	3

Partitioned by country
- i.e., a country in a group

An order value is computed for each row in a partition.
- Only inside the partition, not across the entire result set

Ranking Window Function

- Why window function, not group by?
  - “Group by” reduces the rows in a group (partition) into one result, which is the meaning of “aggregation”
    - Then, the values in non-aggregating columns are vanished
  - Window functions do not reduce the rows
    - Instead, they attach computed values next to the rows in a group (partition) and keep the details
    - Actually, the partition here means “window”: an affective range for statistics

Ranking Window Function

- Some more ranking window functions
  - Besides `rank()`, we also have `dense_rank()` and `row_number()`
  - The difference is about how they treat rows with the same rank

```
select country,
       title,
       year_released,

       rank() over (
           partition by country order by year_released
       ) rank_result,

       dense_rank() over (
           partition by country order by year_released
       ) dense_rank_result,

       row_number() over (
           partition by country order by year_released
       ) row_number_result
from movies;
```

country	title	year_released	rank_result	dense_rank_result	row_number_result
cn	some title	1948	1	1	1
cn	some title	1959	2	2	2
cn	some title	1959	2	2	3
cn	some title	1987	4	3	4
cn	some title	2002	5	4	5
uk	some title	1985	1	1	1
uk	some title	1992	2	2	2
uk	some title	2010	3	3	3

Aggregation Functions as Window Functions

- `max(col)` and `min(col)`

```
select country,
       title,
       year_released,
       min(year_released) over (
           partition by country order by year_released
       ) oldest_movie_per_country
from movies;
```

Need to specify a column in the parameter list

	cou...	title	year_released	oldest_movie_per_country
1	am	Sayat Nova	1969	1969
2	ar	Pampa bárbara	1945	1945
3	ar	Albéniz	1947	1945
4	ar	Madame Bovary	1947	1945
5	ar	La bestia debe morir	1952	1945
6	ar	Las aguas bajan turbias	1952	1945
7	ar	Intermezzo criminal	1953	1945
8	ar	La casa del ángel	1957	1945
9	ar	Bajo un mismo rostro	1962	1945
10	ar	Las aventuras del Capitán Piluso	1963	1945
11	ar	Savage Pampas	1966	1945
12	ar	La hora de los hornos	1968	1945
13	ar	Waiting for the Hearse	1985	1945
14	ar	La historia oficial	1985	1945
15	ar	Hombre mirando al sudeste	1986	1945

The min/max value for each partition is assigned for all the rows inside this partition

Aggregation Functions as Window Functions

- `sum(col)`, `count(col)`, `avg(col)`, `stddev(col)`, etc.
  - Different from min/max: for these aggregation functions, it means the aggregation value from the first row to the current row in its partition when `order by` is specified

```
select country,
       title,
       year_released,
       sum(runtime) over (
           partition by country order by year_released
       ) total_runtime_till_this_row
from movies;
```

	country	title	year_released	total_runtime_till_this_row
1	am	Sayat Nova	1969	78
2	ar	Pampa bárbara	1945	98
3	ar	Albéniz	1947	308
4	ar	Madame Bovary	1947	308
5	ar	La bestia debe morir	1952	494
6	ar	Las aguas bajan turbias	1952	494
7	ar	Intermezzo criminal	1953	494
8	ar	La casa del ángel	1957	570
9	ar	Bajo un mismo rostro	1962	695
10	ar	Las aventuras del Capitán Piluso	1963	785
11	ar	Savage Pampas	1966	897
12	ar	La hora de los hornos	1968	1157
13	ar	Waiting for the Hearse	1985	1354
14	ar	La historia oficial	1985	1354

However, if there is no `order by`, the behavior will be similar to `min()` and `max()`

- One result for all rows

Pay attention to the behavior on rows with the same rank:

- They are “treated like the same row” here

Exercise

- Question: How can we get the top-5 most recent movies for each country?
  - Hint: Use a subquery in the “`from`” clause

Exercise

- Question: How can we get the top-5 most recent movies for each country?
  - Hint: Use a subquery in the “`from`” clause

```
select x.country,
       x.title,
       x.year_released
from (
 select country,
        title,
        year_released,
        row_number()
        over (partition by country
              order by year_released desc) rn
 from movies) x
where x.rn <= 5
```

Function

Built-in Functions

- Most DBMS provides a series of built-in functions
  - E.g., Scalar function, aggregation function, window function

```
round(3.141592, 3)  -- 3.142
trunc(3.141592, 3)  -- 3.141
```

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

```
count(*)/count(col), min(col), max(col), stddev(col), avg(col)
```

```
<function> over (partition by <col_p> order by <col_o1, col_o2, …>)
```

- <function>: we can apply (1) ranking window functions, or (2) aggregation functions
- partition by: specify the column for grouping
- order by: specify the column(s) for ordering in each group

Self-defined Function

- Sometimes the built-in functions cannot fulfill our requirements
  - And the power of declarative language (SQL) is not enough

- Most DBMS implement a built-in, SQL-based programming language
  - A procedural extension to SQL

Procedural vs. Declarative

- Two different programming paradigms
  - Imperative: Describe the algorithms step-by-step (**命令式编程**)
    - Procedural: C (and many other legacy languages)
    - Object-oriented: Java
  - Declarative: Describe the result without specifying the detailed steps (**声明式编程**)
    - (Pure) declarative: SQL, Regular Expressions, Markup (HTML, XML), CSS
    - Functional: Scheme, Haskell, Scala, Erlang
    - Logic programming: Prolog

Procedural vs. Declarative

- E.g., How can we get a cup of tea?
  - In a procedural way:

  - In a declarative way:

Procedural vs. Declarative

- E.g., How can we get a cup of tea?
  - In a procedural way:

1. Get a cup
2. Get some tea
3. Get some hot water
4. Put tea into the cup
5. Pour hot water into the cup
6. `return tea;`

  - In a declarative way:

Procedural vs. Declarative

- E.g., How can we get a cup of tea?
  - In a procedural way:

1. Get a cup
2. Get some tea
3. Get some hot water
4. Put tea into the cup
5. Pour hot water into the cup
6. `return tea;`

  - In a declarative way:

`<a cup of tea/>`

    - You don't really need to know how to make a cup of tea
    - The system can do it in a black-box manner

大佬喝茶

Procedural vs. Declarative

- E.g., Find all Chinese movies before 1990 in the `movies` table?
  - In a procedural way:

1. Read the `movies` table into the memory
2. For each row `i` in the table, repeat:
2.1 In row `i`, read the value of the column "country"
2.2 if …

  - In a declarative way:

```
select * from movies where country = 'cn' and year_released < 1990
```

    - You don't really need to know how to filter the table
    - The DBMS system can do it in a black-box manner

Procedural vs. Declarative

- Benefits in declarative languages
  - No need to understand the details
    - The systems take in charge of all the details
  - Easier to use than imperative programming
    - More user-friendly

- Problem in declarative languages
  - Cannot specify the control flow of a program
    - “If there is no such command as <a cup of tea/>, you need to create it by yourself”

Procedural Extension to SQL

- Many DBMS products provide a proprietary procedural extension to the standard SQL
  - Transact-SQL (T-SQL) 

  - PL/SQL 

  - PL/PGSQL 

  - (No specific name) 

  - (Not supported) 

… well, sometimes SQLite is even not considered a DBMS

Function in (Postgre)SQL

- Example: Display the full name for people with “von”
  - When introducing `update`, we have modified the names starting with “von” into “… (von)” for ordering

	peopleid	first_name	surname	born	died	gender
1	16439	Axel	Ambesser (von)	1910	1988	M
2	16440	Daniel	Bargen (von)	1950	2015	M
3	16441	Eduard	Borsody (von)	1898	1970	M
4	16442	Suzanne	Borsody (von)	1957	<null>	F
5	16443	Tomas	Brömssen (von)	1943	<null>	M
6	16444	Erik	Detten (von)	1982	<null>	M
7	16445	Theodore	Eltz (von)	1893	1964	M
8	16446	Gunther	Fritsch (von)	1906	1988	M
9	16447	Katja	Garnier (von)	1966	<null>	F
10	16448	Harry	Meter (von)	1871	1956	M
11	16449	Jenna	Oÿ (von)	1977	<null>	F
12	16450	Alicia	Rittberg (von)	1993	<null>	F
13	16451	Daisy	Scherler Mayer (von)	1966	<null>	F
14	16452	Gustav	Seyffertitz (von)	1862	1943	M

Function in (Postgre)SQL

- If we simply concatenate the first name and the last name, it looks like this:
  - A little bit weird format (a trailing “von”)

```
select first_name || ' ' || surname
from people
where surname like '%(von)';
```

	?column?
1	Axel Ambesser (von)
2	Daniel Bargen (von)
3	Eduard Borsody (von)
4	Suzanne Borsody (von)
5	Tomas Brömssen (von)
6	Erik Detten (von)
7	Theodore Eltz (von)
8	Gunther Fritsch (von)
9	Katja Garnier (von)
10	Harry Meter (von)
11	Jenna Oÿ (von)
12	Alicia Rittberg (von)
13	Daisy Scherler Mayer (von)
14	Gustav Seyffertitz (von)

Function in (Postgre)SQL

- Question: How can we restore the format into “first_name von surname”?
  - String operations

Function in (Postgre)SQL

- Question: How can we restore the format into “first_name von surname”?
  - String operations

```
select case
    when first_name is null then ''
    else first_name || ' '
end || case position('(' in surname)
    when 0 then surname
    else trim(')' from substr(surname, position('(' in surname) + 1))
        || ' '
        || trim(substr(surname, 1, position('(' in surname) - 1))
end
from people
where surname like '%(von)';
```

Function in (Postgre)SQL

- Question: How can we restore the format into “first_name von surname”?
  - String operations

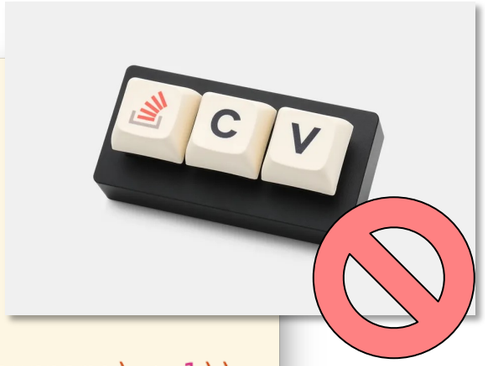
Then, how can we store this part to reuse it in the future?

```
    case
    when first_name is null then ''
    else first_name || ' '
end || case position('(' in surname)
    when 0 then surname
    else trim(')' from substr(surname, position('(' in surname) + 1))
        || ' '
        || trim(substr(surname, 1, position('(' in surname) - 1))
end
from people
where surname like '%(von)';
```

Function in (Postgre)SQL

- “Copy and paste” is not a good habit
  - Whenever you have painfully written something as complicated, which is pretty generic, you'd rather not copy and paste the code every time you need it

```
case
    when first_name is null then ''
    else first_name || ' '
end || case position('(' in surname)
    when 0 then surname
    else trim(')' from substr(surname, position('(' in surname) + 1))
        || ' '
        || trim(substr(surname, 1, position('(' in surname) - 1))
end
```



Function in (Postgre)SQL

- Store for Reuse
  - In PostgreSQL, we can store the expression and reuse it in another context
- Self-defined Function
  - `create function`

```
CREATE [OR REPLACE] FUNCTION function_name (arguments)
RETURNS return_datatype AS $variable_name$
   DECLARE
      declaration;
      [...]
   BEGIN
      < function_body >
      [...]
      RETURN { variable_name | value }
   END; LANGUAGE plpgsql;
```

...or, a simpler version

```
CREATE [ OR REPLACE ] FUNCTION
    name ( [ [ argmode ] [ argname ] argtype [ { DEFAULT | = } default_expr ] [, ...] ] )
    [ RETURNS rettype
      | RETURNS TABLE ( column_name column_type [, ...] ) ]
  { LANGUAGE lang_name
    | TRANSFORM { FOR TYPE type_name } [, ... ]
    | WINDOW
    | { IMMUTABLE | STABLE | VOLATILE }
    | [ NOT ] LEAKPROOF
    | { CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT }
    | { [ EXTERNAL ] SECURITY INVOKER | [ EXTERNAL ] SECURITY DEFINER }
    | PARALLEL { UNSAFE | RESTRICTED | SAFE }
    | COST execution_cost
    | ROWS result_rows
    | SUPPORT support_function
    | SET configuration_parameter { TO value | = value | FROM CURRENT }
    | AS 'definition'
    | AS 'obj_file', 'link_symbol'
    | sql_body
  } ...
```

https://www.postgresql.org/docs/current/sql-createfunction.html

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Function name and the parameter list

- Format for variables and parameters: [name] [type]

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Return type

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

Body

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

A very simple body: return the value of an expression

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position
        || ' '
        || trim(substr(p_sname, 1, position('(' in
    end;
end;
$$ language plpgsql;
```

A very simple body: return the value of an expression

Procedural extensions provide all the bells and whistles in a true (procedural) programming languages, such as:

- Variables
- Conditions
- Loops
- Arrays
- Error management
- ...

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_fname varchar, p_sname varchar)
returns varchar
as $$
begin
    return case
        when p_fname is null then ''
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

Language Type

PostgreSQL supports 4 procedural languages:
PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python

- Tcl, Perl, and Python are famous scripting languages in case you don't know

Function in (Postgre)SQL

- How do we rewrite the name conversion expression into a function?

```
create function full_name(p_
returns varchar
as $$
begin
    return case
        when p_fname is null
        else p_fname || ' '
    end || case position('(' in p_sname)
        when 0 then p_sname
        else trim(')' from substr(p_sname, position('(' in p_sname) + 1))
        || ' '
        || trim(substr(p_sname, 1, position('(' in p_sname) - 1))
    end;
end;
$$ language plpgsql;
```

```
create function append_test(p_code varchar)
returns varchar
as $$
    if p_code == 'cn':
        return 'China'
    else:
        return 'not China'
$$ language plpython3u;
```

Yes, we can even use Python to write functions

Language Type

PostgreSQL supports 4 procedural languages:
PL/pgSQL, PL/Tcl, PL/Perl, and PL/Python

- Tcl, Perl, and Python are famous scripting languages in case you don't know

Function in (Postgre)SQL

- Once your function is created, you can use it as if it were any built-in function.

```
select full_name(first_name, surname)
from people
where surname like '%(von)';
```

Function in (Postgre)SQL

- We can run `select` queries in functions
  - Example: design a function “get_country_name” to transform the country codes into country names based on the `countries` table

Function in (Postgre)SQL

- We can run `select` queries in functions
  - Example: design a function “get_country_name” to transform the country codes into country names based on the `countries` table

```
create function get_country_name(p_code varchar)
returns countries.country_name%type
as $$
declare
    v_name countries.country_name%type;
begin
    select country_name
    into v_name
    from countries
    where country_code = p_code;
    return v_name;
end;
$$ language plpgsql;
```

```
select get_country_name(country) from movies;
```

Function in (Postgre)SQL

- We can run `select` queries in functions
  - Example: design a function “get_country_name” to transform the country codes into country names based on the `countries` table

```
create function get_country_name(p_code varchar)
returns countries.country_name%type
as $$
declare
    v_name countries.country_name%type;
begin
    select country_name
    into v_name
    from countries
    where country_code = p_code;
    return v_name;
end;
$$ language plpgsql;
```

```
select get_country_name(country) from movies;
```

… seems to be an easy way to get rid of `join` operations?

```
select c.country_name
from countries c join movies m
on c.country_code = m.country;
```

Function in (Postgre)SQL

- “Cultural Mismatch”
  - Here we have a problem, because there is a big cultural gap between the relational mindset and procedural processing.

  - A “look-up function” forces a “one row at a time” join which in most cases will be dreadful

```
select get_country_name(country) from movies;
```

For each row in movies, the `select` query in `get_country_name()` is executed once

Comment on Procedural SQL (PL/SQL)

- **Tom Kyte**, who is a Senior Technology Architect at Oracle, says that his mantra is:
  - You should do it in a single SQL statement if at all possible.
  - If you cannot do it in a single SQL statement, then do it in PL/SQL (as little PL/SQL as possible!)

- And some other suggestions (from your lecturer):
  - You should ask for help from someone more experienced than you
    - Stackoverflow, forums, Google, etc.

More to Read

- We may not cover all the details in functions in the theoretical session, so here are some more materials on procedural programming in PostgreSQL:
  - Lab tutorial on Functions
    - Please read it before your next lab sessions
  - Chapter 5.2 “Functions and Procedures,” Database System Concepts (7th Edition)
  - Chapter 43 “PL/pgSQL,” PostgreSQL Documentation
    - https://www.postgresql.org/docs/current/plpgsql.html