Principles of Database Systems (CS307)

Lecture 4: Intermediate and Advanced SQL

Zhong-Qiu Wang

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.
- The slides are largely based on the slides provided by Dr. Yuxin Ma

Announcements

- First assignment is out, due date: 30th Sep., Tuesday, 10pm
  - Do not miss the deadline, or you will receive reduced scores
- We will check attendance today

Window Function

Scalar Functions and Aggregation Functions

- Scalar function
  - Functions that operate on values in the current row

```
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

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

Issues with Aggregate Functions

- A Problem: In aggregated 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.

```
select country,
       min(year_released) earliest_year
from movies
group by country
```

country	oldest_movie
fr	1896
ke	2008
si	2000
eg	1949
nz	1981
bg	1967
ru	1924
gh	2012
pe	2004
hr	1970
sg	2002
mx	1933
cn	1913
ee	2007
sp	1933
cl	1926
ec	1999
cz	1949
dk	1910
vn	1992
ro	1964
mn	2007
gb	1916
se	1913
tw	1971
ie	1970
ph	1975
ar	1945
th	1971

Issues with Aggregate Functions

- A Problem: In aggregated 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 get the title information
- The join conditions include (1) same country code; and (2) same release year

```
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 aggregated 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

- What if the title of the second oldest movie is what we want?

```
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 aggregated 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>`
    - Ranking window functions
    - 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;
```

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

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

`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`”

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

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
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?
    - “order by” for subgroups

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

You can also add “desc” here, similar to the “order by” we introduced before

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

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

- Aggregation functions can be used with window functions
  - `min/max(col)` , `sum(col)`, `count(col)`, `avg(col)`, `stddev(col)`, etc.
  - **If `order by` is specified**, return aggregation value from the first row to current row in each partition
  - **If not**, fill all rows with a single aggregation value computed on all rows in each partition

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

Need to specify a column in the parameter list

	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

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

- They are “treated like the same row” here

Aggregation Functions as Window Functions

- Aggregation functions can be used with window functions
  - `min/max(col)` , `sum(col)`, `count(col)`, `avg(col)`, `stddev(col)`, etc.
  - **If `order by` is specified,** return aggregation value from the first row to current row in each partition
  - **If not**, fill all rows with a single aggregation value computed on all rows in each partition

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

	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

Exercise

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

```sql
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
```

Update and Delete

So Far...

- We have learned:
  - How to access existing data in tables (select)
  - How to create new rows (insert)

- CRUD/CURD
  - create, read, update, delete
    - In SQL: insert, select, update, delete
    - In RESTful API: Post, Get, Put, Delete
  - Necessary operations for persistent storage

Update

- Make changes to the existing rows in a table
- `update` is the command that changes column values
  - You can even set a non-mandatory column to NULL
  - The change is applied to all rows selected by the `where`

```
update table_name
set column_name = new_value,
    other_col = other_new_val,
    ...
where ...
```

Update

- Remember
  - When you are doing any experiments with writing operations (update, delete), **backup the data first**
    - E.g., copy the tables

Update

- Example: In many Western cultures, a nobiliary particle is used in a surname or family name to signal the nobility of a family
  - We may want to modify some names in such a way as they sort as they should
    - `von Neumann -> Neumann (von)`

23 rows | Tx: Auto | DDL

	peopleid	first_name	surname	born	died	gender
1	16439	Axel	von Ambesser	1910	1988	M
2	16440	Daniel	von Bargen	1950	2015	M
3	16441	Eduard	von Borsody	1898	1970	M
4	16442	Suzanne	von Borsody	1957	<null>	F
5	16443	Tomas	von Brömssen	1943	<null>	M
6	16444	Erik	von Detten	1982	<null>	M
7	16445	Theodore	von Eltz	1893	1964	M
8	16446	Gunther	von Fritsch	1906	1988	M
9	16447	Katja	von Garnier	1966	<null>	F
10	16448	Harry	von Meter	1871	1956	M
11	16449	Jenna	von Oÿ	1977	<null>	F
12	16450	Alicia	von Rittberg	1993	<null>	F
13	16451	Daisy	von Scherler Mayer	1966	<null>	F
14	16452	Gustav	von Seyffertitz	1862	1943	M
15	16453	Josef	von Sternberg	1894	1969	M

John von Neumann

Update

- Example: In many Western cultures, a nobiliary particle is used in a surname or family name to signal the nobility of a family
  - We may want to modify some names in such a way as they sort as they should
    - `von Neumann -> Neumann (von)`

- First, how can we find these names?

Update

- Example: In many Western cultures, a nobiliary particle is used in a surname or family name to signal the nobility of a family
  - We may want to modify some names in such a way as they sort as they should
    - **`von Neumann -> Neumann (von)`**
- First, how can we find these names?
  - Wildcards
  - Strings starting with “von ”

```
select * from people_1 where surname like 'von %';
```

Update

- Example: In many Western cultures, a nobiliary particle is used in a surname or family name to signal the nobility of a family
  - We may want to modify some names in such a way as they sort as they should.
    - `von Neumann -> Neumann (von)`
- Then, how should we update the names?

```
(first_name) John                    (first_name) John
(surname) von Neumann      ⇨         (surname) Neumann (von)
```

  - Try the transformation with select:

```
select replace('von Neumann', 'von ', '') || ' (von)';
```

	?column?
1	Neumann (von)

Update

- Example: In many Western cultures, a nobiliary particle is used in a surname or family name to signal the nobility of a family
  - We may want to modify some names in such a way as they sort as they should.
    - **`von Neumann -> Neumann (von)`**
- Finally, the `update` statement:

```
-- Specify the table
update people

-- Set the update rule
set surname = replace(surname, 'von ', '') || ' (von)'

-- Find the rows that need to be updated
where surname like 'von %';
```

Update

- The `where` clause specifies the affected rows
  - If `where` is not provided, the update will be applied to all rows in the table
    - Use with caution!
    - Sometimes, there will be a warning in IDEs such as DataGrip

Update

- The update operation may not be successful when constraints are violated
  - E.g., update the primary key but with duplicated values

```
update people set peopleid = 1 where peopleid < 10;
```

```
[23505] ERROR: duplicate key value violates unique constraint "people_pkey"
Detail: Key (peopleid)=(1) already exists.
```

  - This is why we need constraints when creating tables
    - Avoid unacceptable writing operations that break the integrity of the tables

Update

- Use Subqueries in update
  - Complex update operations where values are based on a query result
- Example: Add a column in people table to record the number of movies one has joined (either directed or played a role in)

```
update table_name
set column_name = new_value,
    other_col = other_new_val,
    ...
where ...
```

Update

- Example: Add a column in people table to record the number of movies one has joined (either directed or played a role in)
  - First, count the movies for a person
    - Used as the subquery part in the update statement

```
select count(*) from credits c where c.peopleid = [some peopleid];
```

Update

- Example: Add a column in people table to record the number of movies one has joined (either directed or played a role in)
  - First, count the movies for a person
    - Used as the subquery part in the update statement
  - Then, update

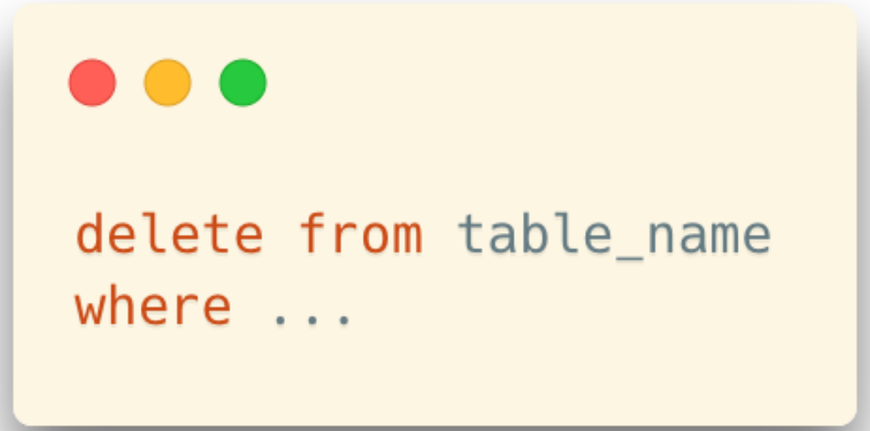
```
update people p

set num_movies = (
    select count(*) from credits c where c.peopleid = p.peopleid
)

where peopleid < 500;
-- This where is only for testing purpose;
-- You should change it (or remove it) when in actual use.
```

Delete

- As the name shows, `delete` removes rows from tables



  - If you omit the WHERE clause, then (as with UPDATE) the statement affects all rows and you **end up with an empty table!**
  - Well,
    - Many database products provide a roll-back mechanism when deleting rows
    - Transactions can also protect you (to some extent)

Constraints

- One important point with constraints (esp., foreign keys) is that they guarantee that data remains consistent
  - They not only work with `insert`, but with `update` and `delete`
  - Example: Try to delete some rows in the `country` table
    - Foreign-key constraints are especially useful in controlling `delete` operations

```
delete from countries where country_code = 'us';
```

```
[23503] ERROR: update or delete on table "countries" violates foreign key constraint "movies_country_fkey" on table "movies"
Detail: Key (country_code)=(us) is still referenced from table "movies".
```

movieid	title	country	year_released
1	Casab	us	1942
2	Goodfellas	us	1990
3	Bronenosets Potyomkin	ru	1925

country_code	country_name	continent
ru	Russia	Europe
us	United States	America

Constraints

- This is why constraints are so important:
  - They ensure that whatever happens, you'll always be able to make sense of ALL pieces of data in your database
  - For any changes made to the tables, all declared constraints need to be satisfied