SQL Functions, Grouping, Aggregation, and Nested queries

DATA 604

Leanne Wu

lewu@ucalgary.ca

Department of Computer Science

SQL Functions

- As with built-in functions for Pythons, functions exist in SQL to reduce the amount of work required to perform common tasks. These include:
  - Mathematical (simple arithmetic and mathematical functions)
  - String (simple manipulations, creating substrings, pattern matching)
  - Date/Time (extracting values from something of DATETIME, arithmetic, conversions, constants)
  - Encryption (compute checksums, apply common encryption/decryption techniques, validate passwords)

Using functions in a query

- Can be used by themselves
  - `select CURDATE();` Current date
  - `select RAND();` Random numbe
- Can be used on literals
  - `select TRUNCATE(3.14159, 2);`
  - `select LENGTH("Hello world!");`
  - `select TRUNCATE(RAND(), 2);`
- Can be used on relations
  - `select length(library) from library_locations;`

Aggregate functions

- Some SQL functions do not work on an individual record; rather they summarize data for an entire table.
- For example:
  - COUNT() (AND COUNT() DISTINCT)
  - SUM()
  - AVG()
  - MAX()
  - MIN()
  - STD(), STDDEV(), STDDEV_POP()
  - VARIANCE()

Grouping results for further analysis

- It is useful to be able to summarize properties of subgroups of data
  - group by channels to see what is happening at each location
  - group by year or date to explore what is happening at different granularities of time

CHANNEL	YEAR	TIMESTAMP	RAINFALL	RG_ACTIVE	ID
42	2019	2019-09-14 5:05	0.2	Y	2019-09-14T05:05:00-42
42	2019	2019-09-14 1:45	0.4	Y	2019-09-14T01:45:00-42
42	2019	2019-09-14 1:40	0.4	Y	2019-09-14T01:40:00-42
48	2019	2019-09-13 10:45	0.2	Y	2019-09-13T10:45:00-48
21	2019	2019-09-13 10:40	0.2	Y	2019-09-13T10:40:00-21
2	2019	2019-09-12 17:40	0.2	Y	2019-09-12T17:40:00-02
23	2019	2019-09-12 6:20	0.2	Y	2019-09-12T06:20:00-23
34	2019	2019-09-11 15:50	0.2	Y	2019-09-11T15:50:00-34

Grouping and Aggregation

- We can use the GROUP BY keyword to summarize data by subgroups

```
SELECT SUM(rainfall_mm) from
rainchannel_measurements
where measurement_year = 2012;
```

```
SELECT SUM(rainfall_mm) from
rainchannel_measurements
where measurement_year = 2012
GROUP BY channel;
```

```
SELECT channel,
SUM(rainfall_mm) from
rainchannel_measurements
where measurement_year = 2012
GROUP BY channel;
```

```
SELECT channel,
SUM(rainfall_mm) as
channel_total from
rainchannel_measurements
where measurement_year = 2012
GROUP BY channel
ORDER BY channel_total;
```

Limits to aggregation

- Consider that now you wanted to filter your list of results
  - what would happen if you wanted a list of all channels that produced less than 250 mm of rainfall in 2012?

```
SELECT channel,
SUM(rainfall_mm) as
channel_total from
rainchannel_measurements
where measurement_year = 2012
GROUP BY channel
ORDER BY channel_total;
```

```
SELECT channel,
SUM(rainfall_mm) as
channel_total from
rainchannel_measurements
where measurement_year = 2012
and channel_total < 250
GROUP BY channel
ORDER BY channel_total;
```

Nested queries

- It is possible to use the results of an inner query to determine the results of an outer query
- For example:

```
SELECT * FROM
        (SELECT channel, SUM(rainfall_mm) AS
        channel_sum from rainchannel_measurements
        where measurement_year = 2012
        GROUP BY channel) AS channel_totals
        WHERE channel_sum < 250
ORDER BY channel_sum;
```

- Queries can be nested more than one level deep

Another example

- Select all channels which have never registered more than 8 mm of rain
  - first build a query to find all the channels which have registered more than 8mm of rain
    - SELECT DISTINCT might be useful
  - then build a query to find a list of all channels which do not satisfy the above query
  - then test it
    - try building a query to see if any of these channels have ever registered 8 mm of rain

Nested queries with joins

- Joins are frequently nested
- For example, for each year, return the channel which registered the most rain in June and the channel which registered the most rain in July, and the respective totals for each of these channels
  - Start with what you want your table to look like

Year	Channel_Max_June	Total_Max_June	Channel_Max_July	Total_Max_July

Nested queries with joins (2)

Next:

- see if you can select all measurements in July and June
- then see if you can sum these by channel and year (do each month separately)
  - (hint: you can GROUP BY multiple columns by specifying GROUP BY <attribute A>, <attribute B>)
- then, find the channel with the most rainfall for each year (do each month separately)

June

Year	Channel_max	Total_max

July

Year	Channel_max	Total_max

Nested queries with joins (2)

- Then join up your two smaller tables on year
- (And then you're done)

More practice

Try the same tasks as in the notebook:

- find the timestamp and channel where the highest rainfall values were recorded
- find the channel which returned the highest rainfall value in June 2013
- find the month in which channel 9 had record rainfall (i.e. higher than in all other months)
- find the average rainfall per channel in August 2017

What about NoSQL?

- What is supported varies by platform (and interface method)
  - MySQL supports quite a bit because collections have been implemented as tables (and thus can support SQL queries)
- Other NoSQL databases such as MongoDB natively supports grouping and aggregation
  - because documents vary by structure, more setup is required than in SQL