Introduction to SQL II

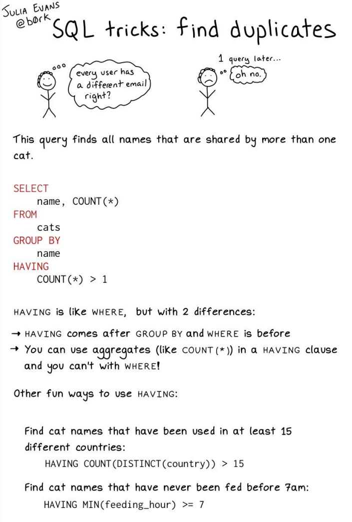
Lecture 7

Dirk Eddelbuettel

STAT 430: Data Science Programming Methods (Fall 2019)
Department of Statistics, University of Illinois

Summary of previous lecture

- SQL is a widely-used important data programming language
- Designed for relational structured data that is stored in tables
- While SQL is standardized different dialects exists
- We work with SQLite, a small and portable SQL engine
- We saw how to `CREATE` table and how to `INSERT` data.
- We used `SELECT` for a number of queries along with
    - several optional qualifiers
    - `WHERE` queries



Julia Evans just posted this:

Very nice illustration of 'how to think about / in SQL'

Source: https://twitter.com/b0rk/status/1172872684327383040)

Key elements we have seen

- `LIMIT`
- `ORDER BY`
- `DISTINCT`
- `WHERE`

Today:

- `JOIN` for inner, left, outer join of two tables
- `GROUP BY` for grouping

An important topic

- Key terms
    - Inner join aka "an intersection"
    - Left join aka "all from left, some matches from right"
    - Full outer join aka "left plus right join"
- Examples and diagrams can explain this well
- `http://www.sqltutorial.org/` is useful for these

Inner Join: Intersection

```
SELECT first_name, last_name,
       d.department_id, department_name
FROM employees e
INNER JOIN departments d
ON d.department_id = e.department_id
WHERE e.department_id IN (1 , 2, 3);
```

- The key is the link the tables department and employees.
- There are referenced as `d` and `e`.
- Join criteria is imposed on the department id.
- The final where clause just limits the overall result set.

Inner Join: Intersection

```
$ sqlite3 -column -header tutorial.sqlite < joinExample1.sql
first_name  last_name   department_id  department_name
----------  ----------  -------------  ---------------
Jennifer    Whalen      1              Administration
Michael     Hartstein   2              Marketing
Pat         Fay         2              Marketing
Den         Raphaely    3              Purchasing
Alexander   Khoo        3              Purchasing
Shelli      Baida       3              Purchasing
Sigal       Tobias      3              Purchasing
Guy         Himuro      3              Purchasing
Karen       Colmenares  3              Purchasing
$
```

Left Join: All rows from left whether or not right matches

```
SELECT c.country_name, c.country_id,
       l.street_address, l.city
FROM countries c
LEFT JOIN locations l ON l.country_id = c.country_id
WHERE c.country_id IN ('US', 'UK', 'CN')
```

- Combines on country id
- Result set has one row with empty address
    - that country and id had no entry in locations db
    - shows how in 'left join' the right table may not have match
- SQL has an equivalent `RIGHT JOIN` (but not in `sqlite`)

Left Join

```
$ sqlite3 -column -header tutorial.sqlite < joinExample2.sql
country_name  country_id  street_address  city
------------  ----------  --------------  ----------
China         CN
United Kingd  UK          8204 Arthur St  London
United Kingd  UK          Magdalen Centr  Oxford
United State  US          2014 Jabberwoc  Southlake
United State  US          2011 Interiors  South San
United State  US          2004 Charade R  Seattle
$
```

Full Outer Join: All rows from either left or right

- See the 'fruit basket example' in the tutorial

```
SELECT basket_name, fruit_name
FROM fruits f
FULL OUTER JOIN baskets b ON b.basket_id = f.basket_id;
```

Note that full outer joins are not currently supported in SQLite.

Full Outer Join: All rows from either left or right

The example yields

```
 basket_name | fruit_name
-------------+------------
 A           | Apple
 A           | Orange
 B           | Banana
 (null)      | Strawberry
 C           | (null)
```

as there are a basket without fuits, and a fruit without a basket.

ILLINOIS

GROUP BY

- A very powerful mechanism for relational data involves grouping
- Given a grouping column, data can be subset to distinct values
- Works particularly well with *aggregating* functions
    - `MIN`, `MAX`, `SUM`, `COUNT`, `AVERAGE` are examples
    - many aggregating functions common, some only in dialects
- Simple example:

```
SELECT department_id, COUNT(employee_id) headcount
FROM employees
GROUP BY department_id;
```

```
$ sqlite3 -column -header tutorial.sqlite < groupBy1.sql
department_id  headcount
-------------  ----------
1              1
2              2
3              6
4              1
5              7
6              5
7              1
8              6
9              3
10             6
11             2
$
```

GROUP BY

- A more complicated example with a `join`
- We added final `ORDER BY`

```
SELECT e.department_id, department_name,
       COUNT(employee_id) headcount
FROM employees e
INNER JOIN departments d ON d.department_id = e.department_id
GROUP BY e.department_id
ORDER BY headcount DESC;
```

```
$ sqlite3 -column -header tutorial.sqlite < groupBy2.sql
department_id  department_name  headcount
-------------  ---------------  ----------
5              Shipping         7
3              Purchasing       6
8              Sales            6
10             Finance          6
6              IT               5
9              Executive        3
2              Marketing        2
11             Accounting       2
1              Administration   1
4              Human Resources  1
7              Public Relation  1
$
```

GROUP BY and HAVING

- Another example with HAVING
- This imposes a constraint

```
SELECT e.department_id, department_name,
       COUNT(employee_id) headcount
FROM employees e
INNER JOIN departments d
ON d.department_id = e.department_id
GROUP BY e.department_id
HAVING headcount > 5
ORDER BY headcount DESC;
```

```
$ sqlite3 -column -header tutorial.sqlite < groupBy3.sql
department_id  department_name  headcount
-------------  ---------------  ----------
5              Shipping         7
3              Purchasing       6
8              Sales            6
10             Finance          6
$
```

GROUP BY and MIN/MAX/AVERAGE

- The next example shows min, max and average
- We added an ORDER BY

```
SELECT e.department_id, department_name,
       MIN(salary) min_salary,
       MAX(salary) max_salary,
       ROUND(AVG(salary), 2) average_salary
FROM employees e
INNER JOIN departments d
ON d.department_id = e.department_id
GROUP BY e.department_id
ORDER BY average_salary DESC;
```

```
$ sqlite3 -column -header tutorial.sqlite < groupBy4.sql
department_id  department_name  min_salary  max_salary  average_salary
-------------  ---------------  ----------  ----------  --------------
9              Executive        17000.0     24000.0     19333.33
11             Accounting       8300.0      12000.0     10150.0
7              Public Relation  10000.0     10000.0     10000.0
8              Sales            6200.0      14000.0     9616.67
2              Marketing        6000.0      13000.0     9500.0
10             Finance          6900.0      12000.0     8600.0
4              Human Resources  6500.0      6500.0      6500.0
5              Shipping         2700.0      8200.0      5885.71
6              IT               4200.0      9000.0      5760.0
1              Administration   4400.0      4400.0      4400.0
3              Purchasing       2500.0      11000.0     4150.0
$
```

GROUP BY and multiple groups

- We can also use `GROUP BY` over more than one column
- Produces analysis with distinct values over *both* columns
- Example 5 provides headcount by department *and* job id

```
SELECT e.department_id, department_name,
       e.job_id, job_title, COUNT(employee_id)
FROM employees e
INNER JOIN departments d
ON d.department_id = e.department_id
INNER JOIN jobs j
ON j.job_id = e.job_id
GROUP BY e.department_id, e.job_id;
```

```
$ sqlite3 -column -header tutorial.sqlite < groupBy5.sql
department_id  department_name  job_id      job_title                 COUNT(employee_id)
-------------  ---------------  ----------  ------------------------  ------------------
1              Administration   3           Administration Assistant  1
2              Marketing        10          Marketing Manager         1
2              Marketing        11          Marketing Representative  1
3              Purchasing       13          Purchasing Clerk          5
3              Purchasing       14          Purchasing Manager        1
4              Human Resources  8           Human Resources Represen  1
5              Shipping         17          Shipping Clerk            2
5              Shipping         18          Stock Clerk               1
5              Shipping         19          Stock Manager             4
6              IT               9           Programmer                5
7              Public Relation  12          Public Relations Represe  1
8              Sales            15          Sales Manager             2
8              Sales            16          Sales Representative      4
9              Executive        4           President                 1
9              Executive        5           Administration Vice Pres  2
10             Finance          6           Accountant                5
10             Finance          7           Finance Manager           1
11             Accounting       1           Public Accountant         1
11             Accounting       2           Accounting Manager        1
$
```

DBI

- We have focused on SQL via SQLite
- This allowed us to concentrate on the SQL *language*
- R deployment will often involve sending SQL from R
- Result are then returned in particular R objects
- We are not ready for this as we have not really started with R
- A package you will likely encounter is `DBI`: Database Interface
- It "normalizes" access to a particular database backend and makes access more interchangeable

Summary

We have seen

- JOIN allows to combine several tables
- We look at the difference between inner, left, right and full joins
- GROUP BY: powerful + automatic grouping, one or more columns
- Many other SQL commands exist and can be studied individually
- But this should provide you with a solid base