Introduction to SQL

Tara Murphy and James Curran

15th April, 2009
What happens when you run an SQL query?

- To run an SQL query the following steps occur
  1. You connect to the server which verifies your username and password
  2. You send a query to the server
  3. The server checks that
     - You have permission to execute the statement
     - You have permission to access the data
     - The statement syntax is correct
  4. Then your query is passed to the *query optimizer* which determines the most efficient way to execute the query
  5. The optimizer chooses an *execution plan* which is used by the server to execute your query
  6. Finally the *result set* is returned to the original application client
There are 6 possible clauses in a select statement

In MySQL only one clause is mandatory (select) however most queries will consist of a few clauses

- **select** determines which columns to include in the result set
- **from** identifies the tables to query and how to join them
- **where** restricts the number of rows in the result set by applying a condition
- **group by** groups rows together using common column values
- **having** restricts the number of rows in the result set using grouped data
- **order by** sorts the rows of the result set
Case study: AT20G Bright Source Sample

- For the rest of this lecture we will be doing queries on a database I have set up using AT20G Survey Data.
- The database contains the following tables:
  - **bss**: Position and flux data for ~800 sources observed at 20 GHz with the ATCA. These are the “Bright Source Sample” (BSS) at 20 GHz with the ATCA.
  - **duplicates**: Information about sources that were observed multiple times.
**describe** shows the structure of a table.

We can inspect any of the tables in the database using **DESC**.

```sql
1  mysql> DESC bss;

+---------------------+-----------------------+-------+-----+---------+
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>varchar(30)</td>
<td>NO</td>
<td>PRI</td>
<td></td>
</tr>
<tr>
<td>obs</td>
<td>varchar(15)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>ra</td>
<td>double</td>
<td>YES</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>decl</td>
<td>double</td>
<td>YES</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>tinyint(3) unsigned</td>
<td>YES</td>
<td></td>
<td>NULL</td>
</tr>
<tr>
<td>run</td>
<td>varchar(10)</td>
<td>YES</td>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>

17 rows in set (0.01 sec)
**select chooses columns from a table**

- **Show me all columns from the duplicate table**

```sql
mysql> SELECT * -> FROM duplicate;
```

```
+------------+---------+------------+---------+----------+---------+
| name1      | run1    | name2      | run2    | offset   | type    |
| AT20G J004426-842239 | apr06   | AT20G J004426-842242 | oct05   | 2.8      | Duplicate |
| AT20G J010915-604948 | oct05   | AT20G J010915-604948 | oct04   | 1.51     | Duplicate |
| AT20G J010915-604948 | oct05   | AT20G J010915-604948 | apr06   | 0.44     | Duplicate |
...```

172 rows in set (0.01 sec)
select chooses columns from a table

- Show me the name1 column from the duplicate table

```
1  mysql> SELECT name1
2       -> FROM duplicate;

+---------------------------------------------+
| name1                                       |
|---------------------------------------------+
| AT20G J004426.05-842239.3                  |
| AT20G J010915.40-604948.6                  |
| AT20G J010915.40-604948.6                  |
| AT20G J010915.46-604948.5                  |
| ...                                         |
+---------------------------------------------+
172 rows in set (0.00 sec)
```
distinct removes duplicates from your results

- You want to retrieve a list of run names

```sql
mysql> SELECT run1
-> FROM duplicate;

+---------+
| run1    |
+---------+
| apr06   |
| oct05   |
| oct05   |
| apr06   |
| apr06   |
| apr06   |
| ...     |
+---------+
172 rows in set (0.00 sec)
```
distinct removes duplicates from your results

```
mysql> SELECT DISTINCT run1
   -> FROM duplicate;

+---------+
| run1    |
+---------+
| apr06   |
| oct05   |
| oct04   |
| oct06   |
| may07GT |
| may07K  |
+---------+
6 rows in set (0.00 sec)

Warning: distinct requires sorting and can be slow on large queries.
```
The where clause filters out unwanted rows

- You want to retrieve information about all sources observed in oct04

```sql
mysql> SELECT name, ra, dec, run
    -> FROM bss
    -> WHERE run = 'oct04';
```

<table>
<thead>
<tr>
<th>name</th>
<th>ra</th>
<th>decl</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT20G J000435.65-473619.1</td>
<td>1.1485</td>
<td>-47.6053</td>
<td>oct04</td>
</tr>
<tr>
<td>AT20G J001035.92-302748.3</td>
<td>2.6497</td>
<td>-30.4634</td>
<td>oct04</td>
</tr>
<tr>
<td>AT20G J001259.89-395426.4</td>
<td>3.2495</td>
<td>-39.9073</td>
<td>oct04</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

148 rows in set (0.01 sec)
You can combine multiple filtering conditions

- Conditional expressions: AND, OR or NOT
- Comparison operators (\(=\), \(!=\), \(<\), \(>\), \(<\rangle\))
- You want to retrieve information about all sources observed in the oct04 run, South of \(-70^\circ\) declination.

```
mysql> SELECT name, ra, decl, run
    -> FROM bss
    -> WHERE (run = 'oct04') AND (decl < -65);
```

<table>
<thead>
<tr>
<th>name</th>
<th>ra</th>
<th>decl</th>
<th>run</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT20G J040820.33-654508.6</td>
<td>62.0847</td>
<td>-65.7524</td>
<td>oct04</td>
</tr>
<tr>
<td>AT20G J064428.14-671257.6</td>
<td>101.1173</td>
<td>-67.216</td>
<td>oct04</td>
</tr>
<tr>
<td>AT20G J074331.60-672625.8</td>
<td>115.8817</td>
<td>-67.4405</td>
<td>oct04</td>
</tr>
</tbody>
</table>
```
...
You can sort and limit results

Find the 5 pairs of duplicates with the greatest offsets

```
mysql> SELECT name1, name2, offset
    -> FROM duplicate
    -> ORDER BY offset desc
    -> LIMIT 5;
```

<table>
<thead>
<tr>
<th>name1</th>
<th>name2</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>AT20G J132649-525623</td>
<td>AT20G J132649-525535</td>
<td>47.99</td>
</tr>
<tr>
<td>AT20G J142431-491301</td>
<td>AT20G J142432-491349</td>
<td>47.75</td>
</tr>
<tr>
<td>AT20G J142741-330531</td>
<td>AT20G J142743-330527</td>
<td>26.5</td>
</tr>
<tr>
<td>AT20G J145427-374733</td>
<td>AT20G J145429-374728</td>
<td>24.86</td>
</tr>
<tr>
<td>AT20G J142756-420618</td>
<td>AT20G J142758-420616</td>
<td>24.57</td>
</tr>
</tbody>
</table>

5 rows in set (0.00 sec)
Information from multiple tables requires a join

- You often want to find information that is **split across several tables**.
- For example, we want to find the RA and Dec of all sources classified as sidelobes.
- This requires information from two tables `bss` and `duplicate`.

```sql
mysql> SELECT bss.ra, bss.decl, duplicate.type
-> FROM bss, duplicate
-> WHERE duplicate.type = 'Sidelobe';
```

- This results in a **cross join**, showing every permutation of `bss` and `duplicate` (824 × 2 = 1648 rows)
An *inner join* is the most common type of join

- To correct the previous query we need to make use of the foreign key to link the tables
- These tables are linked by the foreign key *name*.

```sql
mysql> SELECT bss.ra, bss.decl, duplicate.type
    -> FROM bss JOIN duplicate
    -> ON bss.name = duplicate.name1
    -> WHERE duplicate.type = 'Sidelobe';
```

- As there are other types of joins, you should specify INNER JOIN rather than just JOIN
Results from the inner join

```
mysql> SELECT bss.ra, bss.decl, duplicate.type
    -> FROM bss JOIN duplicate
    -> ON bss.name = duplicate.name1
    -> WHERE duplicate.type = 'Sidelobe';
```

```
+----------+----------+----------+
| ra       | decl     | type     |
+----------+----------+----------+
| 216.1343 | -49.2304 | Sidelobe |
+----------+----------+----------+
1 row in set (0.00 sec)
```
Using *table aliases* makes the query shorter

- When referring to more that one table it is necessary to prefix the column name with the table name to remove ambiguity.
- Hence we refer to them as `employee.dept_id`, `department.dept_id`
- It is more convenient to assign a table alias in the `FROM` clause:

```sql
mysql> SELECT b.ra, b.decl, d.type
    -> FROM bss b
    -> JOIN duplicate d ON b.name = d.name1;
```
You should use SQL92 standard syntax

- This is a standard that all the major DBMS follow
- You should use it for portability across systems
- However you should also be aware of pre-SQL92 syntax for joins as it is still commonly used

```sql
mysql> SELECT b.ra, b.decl, d.type
    -> FROM bss b, duplicate d
    -> WHERE b.name = d.name1;
```
Grouping is good for finding trends in your data

- We want to find the number of sources observed in each run

```sql
mysql> SELECT run, COUNT(*) n_sources
       -> FROM bss
       -> GROUP BY run;

+---------+-----------+
| run    | n_sources |
+---------+-----------+
| apr06  |    14     |
| may07GT|    24     |
| may07K |   449     |
| oct04  |   148     |
| oct05  |    59     |
| oct06  |   130     |
+---------+-----------+
6 rows in set (0.00 sec)
```
Group filter conditions are specified using `HAVING`

- The **HAVING** clause lets you apply filters to groups as follows

```sql
mysql> SELECT run, COUNT(*) n_sources
    -> FROM bss
    -> GROUP BY run
    -> HAVING n_sources 100;
```

```
+--------+-----------+
| run    | n_sources |
+--------+-----------+
| may07K | 449       |
| oct04  | 148       |
| oct06  | 130       |
+--------+-----------+
3 rows in set (0.00 sec)
```
Order is important when combining filters

Image Ref: Kifer, Bernstein and Lewis, *Database Systems*
Aggregation functions work on groups

- Aggregation functions perform a specific operation over all rows in a group
- MySQL supports the following aggregation functions:
  - `max()` returns the maximum value within a set
  - `min()` returns the minimum value within a set
  - `avg()` returns the average value within a set
  - `sum()` returns the sum of values within a set
  - `count()` returns the number of values within a set
- We have already used `count(*)` in previous examples
Tutorial options

- **Option 1**: Astronomy SQL
  - Try SQL questions on SDSS and OpenSkyQuery sites

- **Option 2**: SQL tutorial
  - If you have MySQL installed, try walkthrough tutorial

- **Option 3**: MySQL tutorial
  - If you have MySQL installed, try MySQL walkthrough
References

- Beaulieu, 2005, *Learning SQL*, O’Reilly