Database Management Systems:
Fundamentals and Introduction to SQL

MySQL Queries

JOINs

Self Join

- compare one table to itself
- must use table aliases (AS…)
- which pairs of rows in a table satisfy some conditions

Self Join

- Example: find all paintings in your collection by the artist who painted *The Potato Eaters*

1. Identify which `painting` *table row contains the title The Potato Eaters so that you can refer to its* `a_id` *value.*

2. Match other rows in the table that have the same `a_id` *value.*

3. Display the titles from those matching rows.

Multi-table Join

- One-to-many relationship
  - e.g., artist, paintings
- Many-to-many relationship
  - e.g., actors, movies
  - JOIN … ON … JOIN … ON… (>2 tables)
  - JOIN … ON … AND … : match two attributes
    - USING (col1, col2)

Aggregate Function

Summaries

- *How many?*
- *What is the total?*
- *What is the range of values?*
- …
- Summary types dependent on the nature of data
  - strings
  - numbers
  - dates

Summaries

- *aggregate functions*
- *returns a value*

- COUNT()
  - e.g., COUNT(DISTINCT COL1, COL2 …)
- **MIN(), MAX()**
- **SUM(), AVG()** – numeric functions

Summaries

- **MIN(), MAX()**
  - Return record with min or max value
  - min() and max() cannot be used in WHERE clause

  - possible solution:
  - **1. User-defined variable**
    - **SET @max = (SELECT max(…) FROM …);**
    - **SELECT …. WHERE col = @max;**
  - **2. JOIN**

    create temporary table containing min or max;

    Join with original table
  - 3. Using subquery

    WHERE col = (SELECT MAX() ……)

Summaries with Grouping

- **Vector aggregate**: multiple values returned from SQL query with aggregate function (via **GROUP BY**)
- Summary for each subgroup
- **GROUP BY**
  - SELECT **colname_group, summary_function** FROM …

    GROUP BY colname_group;
  - grouping can be fine-grained as needed
    - GROUP BY col1, col2….

Summaries with Grouping

- cannot include any nonaggregate column in SELECT

```
+--------+-------+------------+-------+
| rec_id | name  | trav_date  | miles |
+--------+-------+------------+-------+
|      1 | Ben   | 2014-07-30 |   152 |    ← Ben's longest trip
|      2 | Suzi  | 2014-07-29 |   391 |
|      3 | Henry | 2014-07-29 |   300 |    ← Henry's longest trip
|      4 | Henry | 2014-07-27 |    96 |
|      5 | Ben   | 2014-07-29 |   131 |
|      6 | Henry | 2014-07-26 |   115 |
|      7 | Suzi  | 2014-08-02 |   502 |    ← Suzi's longest trip
|      8 | Henry | 2014-08-01 |   197 |
|      9 | Ben   | 2014-08-02 |    79 |
|     10 | Henry | 2014-07-30 |   203 |
+--------+-------+------------+-------+
```

```
mysql> SELECT name, trav_date, MAX(miles) AS 'longest trip
    -> FROM driver_log GROUP BY name;
```

```
+-------+------------+--------------+
| name  | trav_date  | longest trip |
+-------+------------+--------------+
| Ben   | 2014-07-30 |          152 |
| Henry | 2014-07-29 |          300 |
| Suzi  | 2014-07-29 |          502 |
+-------+------------+--------------+
```

```
mysql> CREATE TEMPORARY TABLE t
    -> SELECT name, MAX(miles) AS miles FROM driver_log GROUP BY name;
mysql> SELECT d.name, d.trav_date, d.miles AS 'longest trip'
    -> FROM driver_log AS d INNER JOIN t USING (name, miles) ORDER BY name;
```

Grouping with Certain Characteristics

- calculate group summaries but display results for groups that match certain criteria
  - *e.g., drivers in the* `driver_log` *table who drove more than three days*
- issues with using WHERE – initial constraints to determine which rows to select
- HAVING – applies to group characteristics rather than single rows
  - Like a WHERE clause, but it operates on groups (categories), not on individual rows.
  - HAVING clause can use aliases

Grouping with Certain Characteristics

- DISTINCT eliminates duplicates but doesn't show which values are duplicated
- HAVING work together with COUNT() =1 OR >1
  - *days on which only one driver was active*
  - *days on which more than one driver was active*

Grouping with Certain Characteristics

- Smallest or largest summary value
  - Different levels of summary information required
  - *e.g., find the driver with the most total miles*

```
mysql> SELECT name, SUM(miles)
    -> FROM driver_log
    -> GROUP BY name
    -> HAVING SUM(miles) = MAX(SUM(miles));
ERROR 1111 (HY000): Invalid use of group function
```

The argument for summary values cannot be another aggregate function.

Grouping with Certain Characteristics

- Working with per-group and overall summary values simultaneously
- Different levels of summary information required
  - overall summary
  - per-group summary

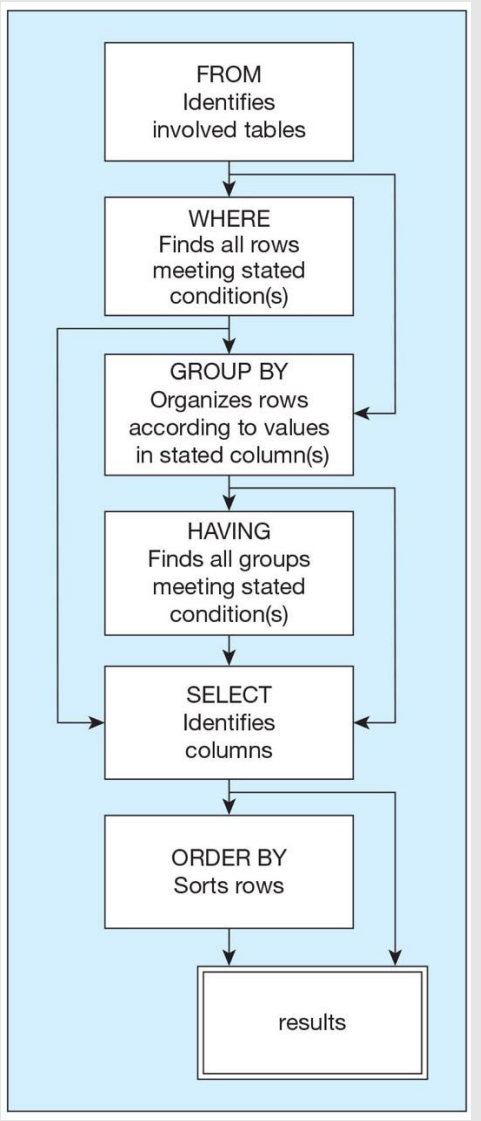
  e.g.,
  - *total miles per driver*
  - *percentage of each driver's mileage*

SELECT Statement Summary

- Used for queries on single or multiple tables
- Clauses of the SELECT statement:
  - SELECT: List the columns (and expressions) to be returned from the query
  - FROM: Indicate the table(s) or view(s) from which data will be obtained
  - WHERE: Indicate the conditions under which a row will be included in the result
  - GROUP BY: Indicate categorization of results
  - HAVING: Indicate the conditions under which a category (group) will be included
  - ORDER BY: Sorts the result according to specified criteria

SELECT Statement Processing Order

SELECT [ALL/DISTINCT] column_list
FROM table list
[WHERE conditional expression]
[GROUP BY group_by_column_list]
[HAVING conditional expression]
[ORDER BY order_by_column_list]



In-class Exercise (Camino)

- Find the solution to the following query:

- *find the driver with the most total miles*

```
mysql> SELECT name, SUM(miles)
    -> FROM driver_log
    -> GROUP BY name
    -> HAVING SUM(miles) = MAX(SUM(miles));
ERROR 1111 (HY000): Invalid use of group function
```

In-class Exercise (Camino)

- Assignment 2 rating.sql
- *For all cases where the same reviewer rated the same movie twice and gave it a higher rating the second time, return the reviewer's name and the title of the movie.*
- *For each movie that has at least one rating, find the highest number of stars that movie received. Return the movie title and number of stars. Sort by movie title.*
- *For each movie, return the title and the 'rating spread', that is, the difference between the highest and lowest ratings given to that movie. Sort by rating spread from highest to lowest, then by movie title.*