Advanced SQL

NULL

The special value `NULL` could mean:

- Unknown
- Unavailable
- Not Applicable

Three-Valued Logic - AND

AND | TRUE | FALSE | UNKNOWN | ———|———|———|———|
TRUE | TRUE | FALSE | UNKNOWN | FALSE | FALSE | FALSE |
FALSE | UNKNOWN | UNKNOWN | FALSE | UNKNOWN |

Three-Valued Logic - OR

OR | TRUE | FALSE | UNKNOWN | ——–|———–|———–|————-|
TRUE | TRUE | TRUE | TRUE | FALSE | TRUE | FALSE |
UNKKNOWN | UNKNOWN | TRUE | UNKNOWN | UNKNOWN |

Three-Valued Logic - NOT

NOT | TRUE | ——|———| TRUE | FALSE | FALSE | TRUE | UNKNOWN | UNKNOWN |

Comparisons with NULL Values

Each NULL is distinct, so comparisons with $<$, $>$, and $=$ don't make sense.
To compare with null, use SQL operator IS, e.g., "Which books don't have editors?":

```
SELECT * FROM book WHERE editor IS NULL;
```

Inner joins include only tuples for which the join condition evaluates to TRUE.

Queries in Depth

```
SELECT [DISTINCT] <select_header> FROM <source_tables>
WHERE <filter_expression>
GROUP BY <grouping_expressions>
HAVING <filter_expression>
ORDER BY <ordering_expressions>
LIMIT <count> OFFSET <count>
```

- The table is the fundamental data abstraction in a relational

database.

- The select command returns its result as a table
- Think of a select statement as creating a pipeline, each stage of which produces an intermediate working table

The SELECT Pipeline

The evaluation order of select clauses is approximately:

1. `FROM <source_tables>` - Designates one or more source tables and

combines them together into one large working table.

1. `WHERE <filter_expression>` - Filters specific rows of working table
2. `GROUP BY <grouping_expressions>` - Groups sets of rows in the working table based on similar values
3. `SELECT <select_heading>` - Defines the result set columns and (if applicable) grouping aggregates.
4. `HAVING <filter_expression>` - Filters specific rows of the grouped table. Requires a GROUP BY
5. `DISTINCT` - Eliminates duplicate rows.
6. ~ORDER BY <ordering$_{\text{expressions}}$> - Sorts the rows of the result set
7. `OFFSET <count>` - Skips over rows at the beginning of the result set. Requires a LIMIT.
8. `LIMIT <count>` - Limits the result set output to a specific number of rows.

SELECT Pipeline at a Glance

1. `FROM <source_tables>`
2. `WHERE <filter_expression>`
3. `GROUP BY <grouping_expressions>`
4. `SELECT <select_heading>`
5. `HAVING <filter_expression>`
6. `DISTINCT`
7. `ORDER BY <ordering_expressions>`
8. `OFFSET <count>`
9. `LIMIT <count>`

Evaluation order determines what can be cross referenced in clauses.

Aggregate Functions

Operate on groups of rows. Some common ones: COUNT, SUM, AVG

```
mysql> select count(*) from book;
+----------+
| count(*) |
+----------+
|        8 |
+----------+
```

There are 8 rows in the book table.

```
mysql> select count(editor) from book;
+---------------+
| count(editor) |
+---------------+
|             6 |
+---------------+
```

Notice that COUNT doesn't count NULL values.

GROUP BY

The `GROUP BY` clause groups rows in the working table by the values in the specified column(s) and collapses each group into a single row.

- We can apply an aggregate function to the resulting groups
- If we don't apply an aggregate function, only the last row of a group is returned.
    - Since rows within groups are in no particular order, failing to apply an aggregate function would essentially give us a random result.

Aggregate Functions on Groups

Aggregate functions apply some function the to the rows grouped together by a GROUP BY clause.
How many papers did each author write?

```
mysql> select author_id, last_name, count(author_id)
    -> from author join author_pub using (author_id)
    ->   join pub using (pub_id)
    -> group by author_id;
+-----------+-----------+------------------+
| author_id | last_name | count(author_id) |
+-----------+-----------+------------------+
|         1 | McCarthy |                1 |
|         2 | Ritchie  |                1 |
|         3 | Thompson |                1 |
|         4 | Shannon  |                1 |
|         5 | Turing   |                2 |
|         6 | Church   |                1 |
+-----------+-----------+------------------+
```

Aggregate function is applied to column in GROUP BY.

Sorting, Aliasing, and Limiting

Who wrote the most publications?

```
mysql> select author_id, last_name, count(author_id) as
    pub_count
    -> from author join author_pub using (author_id) join
        pub using (pub_id)
    -> group by author_id
    -> order by pub_count desc;
+-----------+-----------+-----------+
| author_id | last_name | pub_count |
+-----------+-----------+-----------+
|         5 | Turing    |         2 |
|         1 | McCarthy  |         1 |
|         2 | Ritchie   |         1 |
|         6 | Church    |         1 |
|         3 | Thompson  |         1 |
|         4 | Shannon   |         1 |
+-----------+-----------+-----------+
6 rows in set (0.00 sec)
```

Notice that we also used an alias so we could reference the count in the `ORDER BY` clause

Limiting Results

If we want only the answer from the last query we can use `LIMIT`:
Who wrote the most publications?

```
mysql> select author_id, last_name, count(author_id) as
    pub_count
    -> from author join author_pub using (author_id) join
        pub using (pub_id)
    -> group by author_id
    -> order by pub_count desc
    -> limit 1;
+-----------+-----------+-----------+
| author_id | last_name | pub_count |
+-----------+-----------+-----------+
|         5 | Turing    |         2 |
+-----------+-----------+-----------+
1 row in set (0.00 sec)
```

HAVING

In the previous query we got the top author by pub count. If we want all authors having a particular pub count, we can use a HAVING clause.

```
mysql> select author_id, last_name, count(author_id) as
    pub_count
    -> from author join author_pub using (author_id)
    ->   join pub using (pub_id)
    -> group by author_id
    -> having pub_count = 1;
+-----------+-----------+-----------+
| Author_id | last_name | pub_count |
+-----------+-----------+-----------+
|         1 | McCarthy |         1 |
|         2 | Ritchie  |         1 |
|         3 | Thompson |         1 |
|         4 | Shannon  |         1 |
|         6 | Church   |         1 |
+-----------+-----------+-----------+
```

We can use comparisons like $<$, $>$. Notice that Turing is not in the result.

HAVING vs. WHERE Conditions

Functionally HAVING and WHERE do the same thing: they filter-in tuples. The difference is where they are evaluated in the SELECT pipeline.

- WHERE is evaluated only after the FROM clause that selects the source tables, so WHERE clauses can only reference expressions that do not contain aggregate functions
- HAVING is evaluated after GROUP BY, and SELECT, so HAVING clauses can reference any result column

Be aware that rows filtered out by a WHERE clause will not be included in a GROUP BY clause.

WHERE vs. HAVING Example

WHERE clause can't refer to column aliases and aggregates in the SELECT list or apply functions to groups greated by GROUP BY clauses.

```
mysql> select author_id, last_name, count(author_id) as
    pub_count
    -> from author natural join author_pub natural join
        pub
    -> where pub_count = 1
    -> group by author_id;
ERROR 1054 (42S22): Unknown column 'pub_count' in 'where
    clause'
```

HAVING can refer to select columns.

```
mysql> select author_id, last_name, count(author_id) as
    pub_count
    -> from author natural join author_pub natural join
        pub
    -> group by author_id
    -> having pub_count = 1;
+-----------+-----------+-----------+
| author_id | last_name | pub_count |
+-----------+-----------+-----------+
```

The IN Operator

```
mysql> select * from book where month in ('April',
    'July');
+---------+------------+-------+------+--------+
| book_id | book_title | month | year | editor |
+---------+------------+-------+------+--------+
|       1 | CACM       | April | 1960 |      8 |
|       2 | CACM       | July  | 1974 |      8 |
|       3 | BST        | July  | 1948 |      2 |
|       7 | AAAI       | July  | 2012 |      9 |
|       8 | NIPS       | July  | 2012 |      9 |
+---------+------------+-------+------+--------+
5 rows in set (0.00 sec)
```

Nested Queries, a.k.a., Sub-Selects

List all the books published in the same month in which an issue of CACM was published.

```
mysql> select book_title, month
    -> from book
    -> where month in (select month
    ->                 from book
                       where book_title = 'CACM');
+------------+-------+
| book_title | month |
+------------+-------+
| CACM       | April |
| CACM       | July  |
| BST        | July  |
| AAAI       | July  |
| NIPS       | July  |
+------------+-------+
5 rows in set (0.00 sec)
```

Simple Summation

Here are the data in the dorm table:

```
mysql> select * from dorm;
+---------+-----------+--------+
| dorm_id | name      | spaces |
+---------+-----------+--------+
|       1 | Armstrong |   124 |
|       2 | Brown     |   158 |
|       3 | Caldwell |   158 |
+---------+-----------+--------+
3 rows in set (0.00 sec)
```

What is the total capacity (number of spaces) for all dorms?

SUM

To find the total capacity for all dorms, sum the spaces column:

```
mysql> select sum(spaces) from dorm;
+-------------+
| sum(spaces) |
+-------------+
|         440 |
+-------------+
1 row in set (0.00 sec)
```

Or use a column alias in the select list to make output clearer:

```
mysql> select sum(spaces) as total_capacity from dorm;
+----------------+
| total_capacity |
+----------------+
|            440 |
+----------------+
1 row in set (0.00 sec)
```

Grouping and Counting

What is the occupancy of each dorm?
First, get a feel for the data:

```
mysql> select * from dorm join student using (dorm_id)
    order by dorm.name;
+---------+-----------+--------+------------+--------+------+
| dorm_id | name      | spaces | student_id | name   | gpa  |
+---------+-----------+--------+------------+--------+------+
|       1 | Armstrong |   124  |          1 | Alice  | 3.60 |
|       1 | Armstrong |   124  |          2 | Bob    | 2.70 |
|       1 | Armstrong |   124  |          3 | Cheng  | 3.90 |
|       2 | Brown     |   158  |          4 | Dhruv  | 3.40 |
|       2 | Brown     |   158  |          5 | Ellie  | 4.00 |
|       2 | Brown     |   158  |          6 | Fong   | 2.30 |
|       3 | Caldwell  |   158  |          7 | Gerd   | 4.00 |
|       3 | Caldwell  |   158  |          8 | Hal    | 2.20 |
|       3 | Caldwell  |   158  |          9 | Isaac  | 2.00 |
|       3 | Caldwell  |   158  |         10 | Jacque | 4.00 |
+---------+-----------+--------+------------+--------+------+
```

We can see that there are three groups of dorms in the result, which we could group by `dorm_id` or `dorm.name`.

Dorm Occupancy

So we group by `dorm.name` and count the rows in each group.

```
mysql> select dorm.name as dorm_name, count(*) as
    occupancy
    -> from dorm join student using (dorm_id)
    -> group by dorm.name;
+-----------+-----------+
| dorm_name | occupancy |
+-----------+-----------+
| Armstrong |         3 |
| Brown     |         3 |
| Caldwell  |         4 |
+-----------+-----------+
3 rows in set (0.00 sec)
```

Ordering

```
mysql> select dorm.name as dorm_name, count(*) as
    occupancy
    -> from dorm join student using (dorm_id)
    -> group by dorm.name
    -> order by occupancy desc;
+-----------+-----------+
| dorm_name | occupancy |
+-----------+-----------+
| Caldwell  |         4 |
| Armstrong |         3 |
| Brown     |         3 |
+-----------+-----------+
3 rows in set (0.00 sec)
```

Nested Queries

Which dorms have fewer occupants than Caldwell?
Step 1: how many occupants in Caldwell?

```
mysql> select count(*) as caldwell_occupancy
    -> from dorm join student using(dorm_id)
    -> where dorm.name = 'caldwell';
+--------------------+
| caldwell_occupancy |
+--------------------+
|                  4 |
+--------------------+
1 row in set (0.00 sec)
```

Occupancy Less than Caldwell

Now we use the previous "caldwell$_{occupancy}$" query as a subquery.

```
mysql> select dorm.name as dorm_name, count(*) as
    occupancy
    -> from dorm join student using (dorm_id)
    -> group by dorm_name
    -> having occupancy < (select count(*) as
       caldwell_occupancy
    ->                    from dorm join student
       using(dorm_id)
    ->                    where dorm.name = 'caldwell');
+-----------+-----------+
| dorm_name | occupancy |
+-----------+-----------+
| Armstrong |         3 |
| Brown     |         3 |
+-----------+-----------+
2 rows in set (0.00 sec)
```

Notice that we couldn't use a where clause here because occupancy is computed from a group, which isn't available at the `WHERE` stage of the SQL `SELECT` pipeline.

Extended Example: Which dorm has the highest average GPA?

- Step 1: Group students and their GPAs by dorm.
- Step 2: Get the average GPAs of each dorm.
- Step 3: Get the max avg GPA from step 2.

Step 1: Group students and their GPAs by dorm

```
mysql> select dorm.name as dorm_name, student.name as
    student_name, gpa
    -> from dorm join student using (dorm_id)
    -> group by dorm_name, student_name, gpa;
+-----------+--------------+------+
| dorm_name | student_name | gpa  |
+-----------+--------------+------+
| Armstrong | Alice        |  3.6 |
| Armstrong | Bob          |  2.7 |
| Armstrong | Cheng        |  3.9 |
| Brown     | Dhruv        |  3.4 |
| Brown     | Ellie        |    4 |
| Brown     | Fong         |  2.3 |
| Caldwell  | Gerd         |    4 |
| Caldwell  | Hal          |  2.2 |
| Caldwell  | Isaac        |    2 |
| Caldwell  | Jacque       |    5 |
+-----------+--------------+------+
10 rows in set (0.00 sec)
```

Step 2: Get the average GPAs of each dorm.

```
mysql> select dorm.name as dorm_name, avg(gpa) as
    average_gpa
    -> from dorm join student using (dorm_id)
    -> group by dorm_name;
+-----------+--------------------+
| dorm_name | average_gpa        |
+-----------+--------------------+
| Armstrong | 3.400000015894572 |
| Brown     | 3.2333333492279053 |
| Caldwell |  3.300000011920929 |
+-----------+--------------------+
3 rows in set (0.00 sec)
```

Step 2.1 Formatting Numeric Values

```
mysql> select dorm.name as dorm_name, format(avg(gpa), 2)
    as average_gpa
    -> from dorm join student using (dorm_id)
    -> group by dorm_name;
+-----------+-------------+
| dorm_name | average_gpa |
+-----------+-------------+
| Armstrong | 3.40        |
| Brown     | 3.23        |
| Caldwell  | 3.30        |
+-----------+-------------+
3 rows in set (0.01 sec)
```

FORMAT(x,d[,locale])

- Formats the number x to d decimals using a format like 'nn,nnn.nnn' and returns the result as a string. If d is 0, the result has no decimal point or fractional part.
- locale defaults to the value of the lc_time_names system variable.

```
mysql> select @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)
```

Step 3: Get max average gpa from average gpa results.

Using a nested query:

```
mysql> select dorm_name, max(average_gpa) as
    max_average_gpa
    -> from (select dorm.name as dorm_name,
        format(avg(gpa), 2) as average_gpa
    ->      from dorm join student using (dorm_id)
    ->      group by dorm_name) as avg_gpas;
+-----------+-----------------+
| dorm_name | max_average_gpa |
+-----------+-----------------+
| Armstrong | 3.40            |
+-----------+-----------------+
1 row in set (0.00 sec)
```

Views

```
mysql> create view cacm_issues as
    ->   select * from book
    ->   where book_title = 'CACM';
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+----------------+
| Tables_in_pubs |
+----------------+
| author         |
| author_pub     |
| book           |
| cacm_issues    |
| pub            |
+----------------+
5 rows in set (0.00 sec)
```

A View is Like a Table

```
mysql> select * from cacm_issues;
+---------+------------+-------+------+--------+
| book_id | book_title | month | year | editor |
+---------+------------+-------+------+--------+
|       1 | CACM       | April | 1960 |      8 |
|       2 | CACM       | July  | 1974 |      8 |
+---------+------------+-------+------+--------+
2 rows in set (0.00 sec)
```