SQL Queries

The SELECT-FROM-WHERE Structure

```
SELECT <attributes>
FROM <tables>
WHERE <conditions>
```

From relational algebra:

- `SELECT <attributes>` corresponds to projection
- `FROM <tables>` specifies the table in parentheses in a relational algebra expression and joins
- `WHERE <conditions>` corresponds to selection

Projection

$\pi_{first\_name, last\_name}(author)$

```
mysql> select first_name, last_name from author;
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| John       | McCarthy  |
| Dennis     | Ritchie   |
| Ken        | Thompson  |
| Claude     | Shannon   |
| Alan       | Turing    |
| Alonzo     | Church    |
| Perry      | White     |
| Moshe      | Vardi     |
| Roy        | Batty     |
+------------+-----------+
9 rows in set (0.00 sec)
```

Asterisk

```
mysql> select * from author;
+-----------+------------+-----------+
| author_id | first_name | last_name |
+-----------+------------+-----------+
|         1 | John       | McCarthy  |
|         2 | Dennis     | Ritchie   |
|         3 | Ken        | Thompson  |
|         4 | Claude     | Shannon   |
|         5 | Alan       | Turing    |
|         6 | Alonzo     | Church    |
|         7 | Perry      | White     |
|         8 | Moshe      | Vardi     |
|         9 | Roy        | Batty     |
+-----------+------------+-----------+
9 rows in set (0.00 sec)
```

Notice that with no condition on select, all rows returned.

Select

$\sigma_{year=2012}(book)$

```
mysql> select * from book where year = 2012;
+---------+------------+-------+------+--------+
| book_id | book_title | month | year | editor |
+---------+------------+-------+------+--------+
|       7 | AAAI       | July  | 2012 |      9 |
|       8 | NIPS       | July  | 2012 |      9 |
+---------+------------+-------+------+--------+
2 rows in set (0.00 sec)
```

String Matching with LIKE

Our where condition can match a pattern with like. Use a % for wildcard, i.e., matching any character sequence.
Which publications have "Turing" in their titles?

```
select * from pub where title like 'Turing%';
+--------+-----------------+---------+
| pub_id | title           | book_id |
+--------+-----------------+---------+
|      4 | Turing Machines |       4 |
|      5 | Turing Test     |       5 |
+--------+-----------------+---------+
2 rows in set (0.00 sec)
```

Note that strings are not case-sensitive.

Joins

The `FROM` clause takes one or more source tables from the database and combines them into one (large) table using the `JOIN` operator. Three kinds of joins:

- CROSS JOIN
- INNER JOIN
- OUTER JOIN

Since DB designs are typically factored into many tables, the join is the most important part of a query.

CROSS JOIN

A CROSS JOIN matches every row of the first table with every row of the second table. Think of a cross join as a cartesian product.
The general syntax for a cross join is:

```
SELECT <select_header> FROM <table1> CROSS JOIN <table2>
```

or

```
SELECT <select_header> FROM <table1>, <table2>
```

CROSS JOIN EXAMPLE

```
mysql> select * from pub cross join book;
+--------+-----------------+---------+---------+------------+----------+---
| Pub_id | title           | book_id | book_id | book_title | month
       | year | editor |
+--------+-----------------+---------+---------+------------+----------+---
|      1 | LISP            |       1 |       1 | CACM       | April
       | 1960 |      8 |
|      2 | Unix            |       2 |       1 | CACM       | April
       | 1960 |      8 |
|      3 | Info Theory     |       3 |       1 | CACM       | April
       | 1960 |      8 |
|      4 | Turing Machines |       4 |       1 | CACM       | April
       | 1960 |      8 |
|      5 | Turing Test     |       5 |       1 | CACM       | April
       | 1960 |      8 |
|      6 | Lambda Calculus |       6 |       1 | CACM       | April
       | 1960 |      8 |
|      1 | LISP            |       1 |       2 | CACM       | July
        | 1974 |      8 |
|      2 | Unix            |       2 |       2 | CACM       | July
        | 1974 |      8 |
|      3 | Info Theory     |       3 |       2 | CACM       | July
        | 1974 |      8 |
|      4 | Turing Machines |       4 |       2 | CACM       | July
        | 1974 |      8 |
|      5 | Turing Test     |       5 |       2 | CACM       | July
        | 1974 |      8 |
|      6 | Lambda Calculus |       6 |       2 | CACM       | July
        | 1974 |      8 |
|      1 | LISP            |       1 |       3 | BST        | July
        | 1948 |      2 |
```

LIMITing Results

If we don't want many results to scroll past the bottom of the screen we can limit the number of results using a LIMIT clause.

```
mysql> select * from pub, book limit 3;
+--------+-------------+---------+---------+------------+-------+-----+---
| pub_id | title       | book_id | book_id | book_title | month |
    year | editor |
+--------+-------------+---------+---------+------------+-------+-----+---
|      1 | LISP        |       1 |       1 | CACM       | April |
    1960 |      8 |
|      2 | Unix        |       2 |       1 | CACM       | April |
    1960 |      8 |
|      3 | Info Theory |       3 |       1 | CACM       | April |
    1960 |      8 |
+--------+-------------+---------+---------+------------+-------+-----+---
3 rows in set (0.00 sec)
```

The general form of the LIMIT clause is `LIMIT` start, count, where start is the first row returned and count is the number of rows returned. If a single value is given, start assumes the value 0.

Inner Joins

A simple inner join uses an `ON` condition.

```
mysql> select * from pub join book on pub.book_id = book.book_id;
+--------+-----------------+---------+---------+------------+----------+---
| pub_id | title           | book_id | book_id | book_title | month
       | year | editor |
+--------+-----------------+---------+---------+------------+----------+---
|      1 | LISP            |       1 |       1 | CACM       | April
       | 1960 |      8 |
|      2 | Unix            |       2 |       2 | CACM       | July
        | 1974 |      8 |
|      3 | Info Theory     |       3 |       3 | BST        | July
        | 1948 |      2 |
|      4 | Turing Machines |       4 |       4 | LMS        |
    November | 1936 |      7 |
|      5 | Turing Test     |       5 |       5 | Mind       |
    October  | 1950 |   NULL |
|      6 | Lambda Calculus |       6 |       6 | AMS        | Month
       | 1941 |   NULL |
+--------+-----------------+---------+---------+------------+----------+---
6 rows in set (0.00 sec)
```

Notice that `book_id` appears twice, becuase we get one from each source table. We can fix that . . .

Natural Joins

The USING clause, also called a natural join, equijoins on a like-named column from each table and includes the join column only once.

```
mysql> select * from pub join book using (book_id);
+---------+--------+-----------------+------------+----------+------+------
| book_id | pub_id | title           | book_title | month    | year
    | editor |
+---------+--------+-----------------+------------+----------+------+------
|       1 |      1 | LISP            | CACM       | April    | 1960
    |      8 |
|       2 |      2 | Unix            | CACM       | July     | 1974
    |      8 |
|       3 |      3 | Info Theory     | BST        | July     | 1948
    |      2 |
|       4 |      4 | Turing Machines | LMS        | November | 1936
    |      7 |
|       5 |      5 | Turing Test     | Mind       | October  | 1950
    |   NULL |
|       6 |      6 | Lambda Calculus | AMS        | Month    | 1941
    |   NULL |
+---------+--------+-----------------+------------+----------+------+------
6 rows in set (0.00 sec)
```

Many to Many Relationships

A single author can write many publications, and a single publication can have many authors. This is a many-to-many relationship, which is modeled in relational databases with a relationship (or link or bridge) table.

```
CREATE TABLE IF NOT EXISTS author_pub (
  author_id INTEGER NOT NULL REFERENCES author(author_id),
  pub_id INTEGER NOT NULL REFERENCES publication(pub_id),
  author_position INTEGER NOT NULL, -- first author, second, etc?
  PRIMARY KEY (author_id, pub_id)
);
```

`author_pub` tables links the `author` and `pub` tables

- `author_id` and `pub_id` are foreign keys to `author` and `pub` tables
- ~(author$_{id}$, pub$_{id}$) is composite key for the table

Joining Multiple Tables

We can join all three tables by chaining join clauses:

```
mysql> select *
    -> from author join author_pub using (author_id)
    ->   join pub using (pub_id);
+--------+------+------------+-----------+------------+-----------------+--
| pub_id | a_id | first_name | last_name | a_pos      | title
               | book_id |
+--------+------+------------+-----------+------------+-----------------+--
|      1 |    1 | John       | McCarthy  |     1      | LISP
                |       1 |
|      2 |    2 | Dennis     | Ritchie   |     1      | Unix
                |       2 |
|      2 |    3 | Ken        | Thompson  |     2      | Unix
                |       2 |
|      3 |    4 | Claude     | Shannon   |     1      | Info Theory
         |       3 |
|      4 |    5 | Alan       | Turing    |     1      | Turing
    Machines  |       4 |
|      5 |    5 | Alan       | Turing    |     1      | Turing Test
         |       5 |
|      6 |    6 | Alonzo     | Church    |     1      | Lambda
    Calculus  |       6 |
+--------+------+------------+-----------+------------+-----------------+--
7 rows in set (0.01 sec)
```

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 source tables and

combining into one 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 column 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.

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.

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)
```

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 |
+-----------+-----------+-----------+
|         1 | McCarthy  |         1 |
|         2 | Ritchie   |         1 |
|         3 | Thompson  |         1 |
|         4 | Shannon   |         1 |
|         6 | Church    |         1 |
+-----------+-----------+-----------+
5 rows in set (0.00 sec)
```