Lecture 10:

- Quiz 4
- Basic SQL Queries

Announcements:

- HW 3 due
- HW 4 out
- Project part 1 due Thursday
- Note: Exam 1 next Thursday
Surrogate Keys in MySQL

Define surrogate keys with AUTO_INCREMENT

CREATE TABLE pet (
    pet_id INT UNSIGNED NOT NULL AUTO_INCREMENT, -- surrogate key
    pet_name TINYTEXT NOT NULL,
    pet_type TINYTEXT NOT NULL DEFAULT 'dog', -- default pet type
    PRIMARY KEY (pet_id)
);

Then use named-attribute version of INSERT:

    INSERT INTO pet (pet_name) VALUES ('fido');
    INSERT INTO pet (pet_name, pet_type) VALUES ('bill', 'cat');
    INSERT INTO pet VALUES (4, 'tom', 'cat');

Note that AUTO_INCREMENT values start at 1 and increment from highest value
SQL Query Basics

SQL queries use the SELECT command ... or WITH (more later)

Basic form of an SQL Query:

```
SELECT attributes ... π attributes
FROM tables ... R, R₁ × R₂ · · ·, R₁ ▷◁ R₂ · · ·, etc.
WHERE conditions ... σ conditions
```

Like with relational algebra ..

- each query is defined over one or more tables
- the result of a query is an “output” table (duplicate rows allowed)

Our Plan:

- cover basic "Select-From-Where" queries this week (plus ordering)
- come back to more advanced query constructs later (e.g., for analytics)
**Basic Select-From-Where Queries**

```
loan

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
<th>checkout_date</th>
<th>due_date</th>
<th>return_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
<td>8/12</td>
<td>8/26</td>
<td>8/24</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>8/12</td>
<td>8/19</td>
<td>NULL</td>
</tr>
<tr>
<td>102</td>
<td>4242</td>
<td>8/25</td>
<td>9/7</td>
<td>8/29</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>7/10</td>
<td>7/17</td>
<td>7/18</td>
</tr>
</tbody>
</table>
```

**Example**: Find barcodes of books loaned by account 101

```
SELECT  barcode
FROM     loan
WHERE    acct_id = 101
```

**Result**: three barcodes returned in a 1-column table!

```
<table>
<thead>
<tr>
<th>barcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>4242</td>
</tr>
<tr>
<td>4243</td>
</tr>
<tr>
<td>4243</td>
</tr>
</tbody>
</table>
```

*Note*: queries return duplicates by default in SQL
WHERE conditions

Example: Find due dates of book 4243 checked out by account 101

```sql
SELECT due_date
FROM loan
WHERE acct_id = 101 AND
     barcode = 4243
```

Returns:

<table>
<thead>
<tr>
<th>due_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>8/19</td>
</tr>
<tr>
<td>7/17</td>
</tr>
</tbody>
</table>

Basic WHERE clause conditions:

- expressions involving **AND, OR, NOT** ... can be parenthesized
- relational comparators: =, <, >, <=, >=, != ... or <> for !=
- also BETWEEN $x$ AND $y$ ... e.g., barcode BETWEEN 4200 AND 4300
- **IS NULL** instead of = NULL ... more on NULLs later
- **IS NOT NULL** insted of != NULL
Check in: write a query for barcodes loaned to account 101 that are either still checked out or were returned after the due date

```
SELECT barcode
FROM loan
WHERE acct_id = 101 AND
    (return_date IS NULL OR return_date > due_date)
```

- like with the CHECK constraint, must use IS NULL here
**SELECT attributes**

SELECT can have more than one attribute ...

```
SELECT barcode, due_date
FROM   loan
WHERE  acct_id = 101
```

The query returns a result table with two attributes:

<table>
<thead>
<tr>
<th>barcode</th>
<th>due_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>4242</td>
<td>8/26</td>
</tr>
<tr>
<td>4243</td>
<td>8/19</td>
</tr>
<tr>
<td>4243</td>
<td>7/17</td>
</tr>
</tbody>
</table>

A “*” means return all attributes ...

```
SELECT *
FROM   loan
WHERE  acct_id = 101
```

Returns:

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
<th>checkout_date</th>
<th>due_date</th>
<th>return_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
<td>8/12</td>
<td>8/26</td>
<td>8/24</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>8/12</td>
<td>8/19</td>
<td>NULL</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>7/10</td>
<td>7/17</td>
<td>7/18</td>
</tr>
</tbody>
</table>
Use `SELECT DISTINCT` to remove duplicates:

```sql
SELECT DISTINCT barcode
FROM loan
WHERE acct_id = 101
```

Returns:

<table>
<thead>
<tr>
<th>barcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>4242</td>
</tr>
<tr>
<td>4243</td>
</tr>
</tbody>
</table>

Can also use `DISTINCT` with multiple attributes ...

```sql
SELECT DISTINCT acct_id, barcode
FROM loan
```

Returns:

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
</tr>
<tr>
<td>102</td>
<td>4242</td>
</tr>
</tbody>
</table>

**However, use *judiciously*** ... why?

- often is expensive depending on the query
- only use in your queries if absolutely needed!
Result attributes can be renamed

```
SELECT acct_id AS account, barcode AS returned_book
FROM   loan
WHERE  return_date IS NOT NULL
```

Returns:

<table>
<thead>
<tr>
<th>account</th>
<th>returned_book</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
</tr>
<tr>
<td>102</td>
<td>4242</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
</tr>
</tbody>
</table>

Result can be ordered using ORDER BY

```
SELECT acct_id, barcode, due_date
FROM   loan
WHERE  acct_id = 101
ORDER BY due_date ASC
```

- note: ASC means “ascending” ... the default (can leave ASC off)

Returns:

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
<th>due_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4243</td>
<td>2022-07-17</td>
</tr>
<tr>
<td>102</td>
<td>4243</td>
<td>2022-08-19</td>
</tr>
<tr>
<td>101</td>
<td>4242</td>
<td>2022-08-26</td>
</tr>
</tbody>
</table>
Instead of ascending (smallest-to-largest) can also use DESC (descending):

```
SELECT acct_id, barcode, due_date
FROM loan
WHERE acct_id = 101
ORDER BY due_date DESC
```

Returns:

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
<th>due_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
<td>2022-08-26</td>
</tr>
<tr>
<td>102</td>
<td>4243</td>
<td>2022-08-19</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>2022-07-17</td>
</tr>
</tbody>
</table>

Can sort on *multiple* attributes to “break ties” ...

```
SELECT acct_id, barcode, due_date
FROM loan
ORDER BY acct_id ASC, due_date DESC
```

- sorting on account id s.t. rows with same account id are sorted on due date

Returns:

<table>
<thead>
<tr>
<th>acct_id</th>
<th>barcode</th>
<th>due_date</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>4242</td>
<td>2022-08-26</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>2022-08-19</td>
</tr>
<tr>
<td>101</td>
<td>4243</td>
<td>2022-07-17</td>
</tr>
<tr>
<td>102</td>
<td>4242</td>
<td>2022-09-07</td>
</tr>
</tbody>
</table>

*Note:* In MySQL, only SELECT is required ...

- e.g: SELECT 'hello world'
Q: What does this query return?

```sql
SELECT *
FROM account
WHERE main_branch = 'Downtown AND main_branch = 'South Hill'
```

This query is always empty (unsatisfiable) ... but the optimizer catches it!

```sql
mysql> EXPLAIN SELECT * FROM account WHERE main_branch = 'Downtown' -> AND main_branch = 'South Hill';
+----+-------------+-----+------------------+
<table>
<thead>
<tr>
<th>id</th>
<th>select_type</th>
<th>...</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SIMPLE</td>
<td>...</td>
<td>Impossible WHERE</td>
</tr>
</tbody>
</table>
+----+-------------+-----+------------------+
```