Lecture 12:

- Exam overview
- Quiz 5
- Basic SQL Queries (cont)

Announcements:

- HW 4 out
- Exam 1 on Thursday
### 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>

### Branch

<table>
<thead>
<tr>
<th>branch_name</th>
<th>address</th>
<th>phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td>906 Main</td>
<td>444-5300</td>
</tr>
<tr>
<td>South Hill</td>
<td>3324 Perry</td>
<td>444-5301</td>
</tr>
<tr>
<td>Shadle</td>
<td>2111 Wellesley</td>
<td>444-5302</td>
</tr>
<tr>
<td>Hillyard</td>
<td>4110 Cook</td>
<td>444-5303</td>
</tr>
</tbody>
</table>

### Account

<table>
<thead>
<tr>
<th>acct_id</th>
<th>acct_name</th>
<th>main_branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Alice</td>
<td>Central</td>
</tr>
<tr>
<td>102</td>
<td>Bob</td>
<td>Central</td>
</tr>
<tr>
<td>103</td>
<td>Alice</td>
<td>Shadle</td>
</tr>
<tr>
<td>104</td>
<td>Chuck</td>
<td>South Hill</td>
</tr>
</tbody>
</table>
Exam 1 Overview:

Basics

- Closed notes, etc.
- Worth 100 total points
- Approx. 5 multipart questions

Possible Topics: ... everything we've covered is fair game!

- DBMSs basics, transaction vs analytical processing, ACID
- Relational model (incl. domains, keys, foreign keys, etc.)
- Relational algebra (select, project, join, set ops, etc.)
- SQL table creation (with PKs, FKs, UNIQUE, NOT-NULL, CHECK)
- Basic SQL queries (SELECT-FROM-WHERE, joins, ordering, duplicates, etc.)

To Study:

- Go back over quizzes (ensure can answer all questions without viewing answer)
- Go back over lecture notes (quiz yourself!)
- Review homework assignments
JOIN Syntax (cont)

For this query ...

```
SELECT a.acct_id, a.acct_name, b.phone
FROM   branch b, account a
WHERE  b.branch_name = a.main_branch
```

We are joining branch and account using different attributes

- here the join involves different named attributes
- note this is also a common pattern when querying multiple tables
- in this case, the "join condition" is `b.branch_name = a.main_branch`

SQL provides a special `JOIN-ON` syntax for this (and other) cases ...

```
SELECT a.acct_id, a.acct_name, b.phone
FROM   branch b JOIN account a ON (b.branch_name = a.main_branch)
```

The general form:

```
R JOIN S ON ( condition )
```

where:

- any valid boolean expression (i.e., \( \theta \)) can be used for the join condition
- does not remove duplicate columns (like with `USING`)
JOIN Syntax (cont)

An “equi-join” if join conditions only involve equality

\[ R_1 \text{ JOIN } R_2 \text{ ON } (x < u) \] is not an equi-join

Can chain together JOINs ...

\[
\text{loan JOIN account USING (acct_id)} \\
\text{JOIN branch ON (main_branch = branch_name)}
\]

- JOIN is a left associative binary operator
- can add parens if desired: \( R \text{ JOIN } (S \text{ JOIN } T \text{ USING } (a_1)) \text{ USING } (a_2) \)

If no JOIN syntax used, often called a “comma join”
- better to use JOIN syntax for readability
JOIN Syntax (cont)

A NATURAL JOIN is a special equi-join ...

```
SELECT *
FROM branch NATURAL JOIN account;
```

- automatically joins on attributes with the same name
- only includes one of the attributes in the output table

If $R_1(x, y, z)$ and $R_2(x, u, z)$ then:

```
SELECT * FROM $R_1$ NATURAL JOIN $R_2$
```

is the same as (in MariaDB):

```
SELECT $x$, $z$, $y$, $u$
FROM $R_1$ JOIN $R_2$ USING ($x$, $z$)
```

which is the same as (in MariaDB):

```
SELECT *
FROM $R_1$ JOIN $R_2$ USING ($x$, $z$)
```

Q: What does the following query return?

```
SELECT *
FROM branch JOIN account;
```

- the cartesian product!
More on NULL values

Recall: NULL indicates the value is unknown

(1) WHERE only selects conditions that are true

Q: What does this query return?

SELECT * FROM Account WHERE NULL

empty set because the WHERE condition isn’t true – it is unknown

(2) Comparisons involving NULL are always unknown — evaluate to NULL

Q: What does this query return?

SELECT * FROM Account WHERE main_branch = NULL

empty set because “main_branch = NULL” returns unknown (NULL) even if some account’s have a NULL main branch

note: this is why we use IS NULL and IS NOT NULL
Logical connectives in SQL are based on “3-valued” logic

<table>
<thead>
<tr>
<th>X AND Y</th>
<th>True</th>
<th>False</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>False</td>
<td>Unknown</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>False</td>
<td>False</td>
</tr>
<tr>
<td>Unknown</td>
<td>Unknown</td>
<td>False</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X OR Y</th>
<th>True</th>
<th>False</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>False</td>
<td>Unknown</td>
</tr>
<tr>
<td>Unknown</td>
<td>True</td>
<td>Unknown</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT Y</th>
<th>True</th>
<th>False</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>False</td>
<td>True</td>
<td>True</td>
<td>Unknown</td>
</tr>
</tbody>
</table>

Can verify in MySQL, e.g., explain the results for (where 0=\text{False}, 1=\text{True}):

- `SELECT true AND NULL;` -- returns NULL
- `SELECT false AND NULL;` -- returns 0
- `SELECT false OR NULL;` -- returns NULL
- `SELECT NOT NULL;` -- returns NULL
- `SELECT NULL = NULL;` -- returns NULL
- `SELECT NULL != NULL;` -- returns NULL
- `SELECT NULL IS NULL;` -- returns 1
- `SELECT NULL IS NOT NULL;` -- returns 0

Can get into trouble comparing potentially NULL-valued attributes:

- `WHERE main_branch = branch_name` ... vs ...
- `WHERE main_branch IS NOT NULL AND main_branch = branch_name`