Lecture 11:

- Basic SQL Queries (cont)

Announcements:

- HW 4 out
- Project part 1 due
- Quiz 5 on Tues (basic queries)
- Note: Exam 1 next 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>
FROM tables

Can “access” multiple tables in a single query ...

```
SELECT *
FROM branch, account
WHERE branch_name = main_branch
```

This is similar to the relational algebra query: … except duplicates

```
σ_{branch\_name=main\_branch}(branch \times account)
```

Which can be rewritten (by the DBMS) into:

```
branch \bowtie_{branch\_name=main\_branch} account
```

Note: the above SQL query is sometimes called a “comma join”

- which is just a cartesian product
- also sometimes called a “cross join”
Check in: What does this query return?

```
SELECT *
FROM   branch, account
WHERE  branch_name = main_branch AND acct_name = 'Alice'
```

The table:

<table>
<thead>
<tr>
<th>branch_name</th>
<th>address</th>
<th>phone</th>
<th>acct_id</th>
<th>acct_name</th>
<th>main_branch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central</td>
<td>906 Main</td>
<td>444-5300</td>
<td>101</td>
<td>Alice</td>
<td>Central</td>
</tr>
<tr>
<td>South Hill</td>
<td>3324 Perry</td>
<td>444-5301</td>
<td>102</td>
<td>Bob</td>
<td>Central</td>
</tr>
<tr>
<td>Shadle</td>
<td>2111 Wellesley</td>
<td>444-5302</td>
<td>103</td>
<td>Alice</td>
<td>Shadle</td>
</tr>
<tr>
<td>Hillyard</td>
<td>4110 Cook</td>
<td>444-5303</td>
<td>104</td>
<td>Chuck</td>
<td>South Hill</td>
</tr>
</tbody>
</table>
Attribute Disambiguation

Often need to state what tables attributes are from ...

```sql
SELECT acct_name, barcode, due_date
FROM loan, account
WHERE acct_id = acct_id -- both tables have acct_id!
```

**Approach 1**: prefix attribute names with table names:

```sql
SELECT acct_name, barcode, due_date
FROM loan, account
WHERE loan.acct_id = account.acct_id
```

- can prefix in SELECT as well

**Approach 2**: use “correlation” names:

```sql
SELECT a.acct_name, a.barcode, a.due_date
FROM loan l, account a
WHERE l.acct_id = a.acct_id
```

- can leave off correlation names if clear from context as well
Check in: What is this query asking?

```sql
SELECT a1.acct_name, a1.acct_id, a2.acct_name, a2.acct_id
FROM account a1, account a2
WHERE a1.acct_name = a2.acct_name
```

In this example, correlation names are *required*

- that is, whenever the same table appears more than once in the FROM
- even if not required, correlation names generally make queries easier to read
JOIN Syntax

Q: What does this query do?

```
SELECT a.acct_id, a.main_branch, l.barcode
FROM loan l, account a
WHERE a.acct_id = 101
```

It returns the "cartesian product" of Loan and Account

- every combination of Loan with Account (acct_id 101) rows
- in this case, we technically aren't "joining" the tables (connecting rows)
- but we are computing a product of the tables

SQL provides special syntax (CROSS JOIN) to denote a cartesian product

```
SELECT a.acct_id, a.main_branch, l.barcode
FROM loan l CROSS JOIN account a
WHERE a.acct_id = 101
```

- helps make it clear a cartesian product is being done on purpose
- since otherwise, it could be just a mistake (missing join condition)
JOIN Syntax (cont)

For this query ...

```
SELECT a.acct_id, a.main_branch, l.barcode
FROM loan l, account a
WHERE l.acct_id = a.acct_id
```

We are “joining” loan and account using the acct_id attribute

- note we are joining the two tables on the same attribute name
- which is a common pattern when querying multiple tables

SQL provides the special JOIN- USING syntax for this case ...

```
SELECT acct_id, main_branch, barcode
FROM loan JOIN account USING (acct_id)
```

The general form:

```
R JOIN S USING (a_1, ..., a_n)
```

where:

- table joined using equality conditions: \( R.a_1 = S.a_1 \) AND \( ... \) AND \( R.a_n = S.a_n \)
- both tables must have all of the attribute names \( a_1, ..., a_n \)
- only one copy of the attributes \( a_1, ..., a_n \) are returned (similar to natural join)