Today …

- Basic SPJ Queries (cont)

Homework

- HW 6 out (due Thurs)
- Quiz on Tues (Basic ER, translation, SQL create table)
More Examples and Details

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>Downtown</td>
</tr>
<tr>
<td>102</td>
<td>Bob</td>
<td>Downtown</td>
</tr>
<tr>
<td>103</td>
<td>Alice</td>
<td>South Hill</td>
</tr>
<tr>
<td>104</td>
<td>Chuck</td>
<td>Downtown</td>
</tr>
</tbody>
</table>

```
SELECT owner
FROM account
WHERE main_branch = 'Downtown' AND main_branch = 'South Hill';
```

- AND is like `&&` — i.e., both conditions must be true
- OR is also supported — same as `||`
- can nest using parens, e.g., `(a1 = v1 OR a1 = v2) AND a2 = v3)`
- What does the query above return?

Query answer: 

<table>
<thead>
<tr>
<th>owner</th>
</tr>
</thead>
</table>

- The query answer is **empty**
**Note on Empty Query Answers**

An empty answer is not an error

- Can be very informative
- E.g., finding there are no accounts that have checked out any books

This example though is always empty! (unsatisfiable)

- branch can’t both be “Downtown” and “South Hill”

An optimization ...

- If a query will **always** produce an empty answer ...
- It would be nice if the query optimizer could recognize it
- And not run the query at all!
  - An example of "semantic" query optimization
- In MySQL:

  ```sql
  mysql> EXPLAIN SELECT *
  -> FROM account
  -> WHERE main_branch = 'Downtown' AND
  -> main_branch = 'South Hill';
  +-------------+-----+------------------+
  | id | select_type | Extra |
  +-------------+-----+------------------+
  | 1 | SIMPLE | Impossible WHERE |
  +-------------+-----+------------------+
  ```
Basic SQL Queries: Bank Account Example Tables

Account

<table>
<thead>
<tr>
<th>acct_num</th>
<th>owner</th>
<th>balance</th>
<th>acct_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Alice</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>102</td>
<td>Bob</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>103</td>
<td>Alice</td>
<td>5000.00</td>
<td>savings</td>
</tr>
<tr>
<td>104</td>
<td>Chuck</td>
<td>1000.00</td>
<td>checking</td>
</tr>
<tr>
<td>105</td>
<td>Debbie</td>
<td>10000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>

Deposit

<table>
<thead>
<tr>
<th>acct_num</th>
<th>transaction_id</th>
<th>deposit_date</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>1</td>
<td>10/22/18</td>
<td>500.00</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>10/29/18</td>
<td>200.00</td>
</tr>
<tr>
<td>104</td>
<td>3</td>
<td>10/29/18</td>
<td>1000.00</td>
</tr>
<tr>
<td>105</td>
<td>4</td>
<td>11/2/18</td>
<td>10000.00</td>
</tr>
</tbody>
</table>

Withdraw

<table>
<thead>
<tr>
<th>acct_num</th>
<th>check_num</th>
<th>check_date</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>924</td>
<td>10/23/18</td>
<td>125.00</td>
</tr>
<tr>
<td>101</td>
<td>925</td>
<td>10/24/18</td>
<td>23.98</td>
</tr>
<tr>
<td>102</td>
<td>128</td>
<td>10/29/18</td>
<td>200.00</td>
</tr>
</tbody>
</table>
Use the **DISTINCT** keyword to remove duplicates in query results

- E.g., the query

```
SELECT DISTINCT a.OWNER, a.BALANCE
FROM account a, deposit d
WHERE a.acct_num = d.acct_num AND
     a.balance > 1000;
```

- Returns

<table>
<thead>
<tr>
<th>owner</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>2000.00</td>
</tr>
<tr>
<td>Debbie</td>
<td>10000.00</td>
</tr>
</tbody>
</table>

Query answer:
A self join joins a table with itself

```
SELECT a1.owner, a1.acct_num, a1.balance, a2.acct_num, a2.balance
FROM account a1, account a2
WHERE a1.owner = a2.owner
```

- Here a1 and a2 refer to “copies” of the same instance
- What is this query trying to do?
- What is the answer?
Renaming Attributes and Materialized Views

We can rename attributes in query answers

```sql
SELECT owner, balance AS checking_balance
FROM account
WHERE acct_type = 'checking'
```

• What does this query return?

We can also save the result as a new table

```sql
CREATE TABLE checking_account
SELECT owner, balance AS checking_balance
FROM account
WHERE acct_type = 'checking'
```

• We can use the new table in subsequent queries
**SQL Null values**

**NULL is a special value in SQL**

- Indicates the value is **unknown**: either **missing** or **does not exist**
  - Car is new, but don’t know the retail price (missing)
  - Car is used, and so doesn’t have a retail price (does not exist)
  - In general, can stand in for many different cases

**Handling unknown (NULL) values in SQL can be tricky**

- WHERE only selects conditions that are True
  - E.g., What is the result of this query?
  ```sql
  SELECT *
  FROM account
  WHERE NULL
  ```
  - It is always empty! (since NULL != True)

- Comparisons involving NULL are always Unknown — evaluate to NULL
  - E.g., What is the result of this query?
  ```sql
  SELECT *
  FROM account
  WHERE type != NULL
  ```
  - Always returns the empty set (because "type != NULL" returns NULL)
• SQL provides IS NULL and IS NOT NULL comparators
  
  – E.g., What is the result of this query?
    ```sql
    SELECT *
    FROM account
    WHERE type IS NOT NULL
    ```
  
  – All accounts whose type attribute is something other than NULL

• Logical connectives based on “3-valued” logic (“Unknown” implies NULL):

<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>
• You can try these out in SQL (0=False, 1=True):

```
SELECT 1 AND NULL;
+-----------------+
| 1 AND NULL     |
+-----------------+
| NULL           |
+-----------------+
```

```
SELECT 0 AND NULL;
+-----------------+
| 0 and NULL      |
+-----------------+
| 0              |
+-----------------+
```

```
SELECT NULL OR 0;
+-----------------+
| NULL or 0       |
+-----------------+
| NULL           |
+-----------------+
```

```
SELECT NOT NULL;
+-----------------+
| not NULL        |
+-----------------+
| NULL           |
+-----------------+
```

```
SELECT (NOT NULL) IS NULL;
+------------------------+
| (not NULL) is NULL     |
+------------------------+
| 1                      |
+------------------------+
```
More SQL: Aggregation (SELECT)

SQL provides 5 aggregate operators

- COUNT, SUM, MIN, MAX, AVG

```sql
SELECT MIN(balance), MAX(balance), AVG(balance)
FROM account
WHERE acct_type = 'checking';
```

Query Answer:

<table>
<thead>
<tr>
<th>MIN(balance)</th>
<th>MAX(balance)</th>
<th>AVG(balance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>10000</td>
<td>3500</td>
</tr>
</tbody>
</table>

```sql
SELECT SUM(balance), COUNT(balance)
FROM account
WHERE acct_type = 'checking';
```

Query Answer:

<table>
<thead>
<tr>
<th>SUM(balance)</th>
<th>COUNT(balance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14000</td>
<td>4</td>
</tr>
</tbody>
</table>