Today …

- Quiz 5
- Aggregate Operators

Homework

- HW 6 out (due Thurs)
- Quiz 6 Thurs
- Midterm next Thurs
## 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>
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';
```

<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';
```

<table>
<thead>
<tr>
<th>SUM(balance)</th>
<th>COUNT(balance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>14000</td>
<td>4</td>
</tr>
</tbody>
</table>
If an aggregate operator appears in a SELECT clause ... 

- all entries in the select clause **must** be aggregate operators
- unless the query includes a GROUP BY clause (more later)

- For example:
  
  ```sql
  SELECT owner, AVG(balance)
  FROM account;
  ```

- this query is **not well formed**

**Q: What does MySQL do?**

- calculate the average for each owner? **NO!!!**
- pick an owner and then average balance across all owners **YES!!!**

Note: This is confusing. So beware!
The COUNT aggregate can take multiple attributes

Q: What does this query return?

```
SELECT COUNT(*)
FROM account;
```

- The number of rows in account

Q: What do you think this query returns?

```
SELECT COUNT(DISTINCT balance, acct_type)
FROM account;
```

- The number of unique pairs of (balance, acct_type) values

However, the following query is not well-formed ...

```
SELECT COUNT(balance, acct_type)
FROM account;
```

This query is well-formed, what does it return?

```
SELECT COUNT(acct_type)
FROM account;
```

- The number of NON NULL acct_type values

Generally better to use COUNT(*) unless using DISTINCT ... why?

- since usually we really are just counting rows not actual values
- unless we really want to count the number of NON NULL values (likely rare)
Using **DISTINCT with aggregates**

Q: Can you guess what the difference is between these two queries?

```
SELECT SUM(balance)
FROM account;
```

```
SELECT SUM(DISTINCT balance)
FROM account;
```

- sum computed over all balances vs. only distinct balances
- first query returns 19000
- second returns 18000

Q: Can you guess what the difference is between these two queries?

```
SELECT MIN(balance)
FROM account;
```

```
SELECT MIN(DISTINCT balance)
FROM account;
```

- Nothing!
**More SQL: SELECT expressions and comparators**

Arithmetic expressions can be used in the SELECT clause

```
SELECT a.owner, d.deposit_date, (a.balance + d.amount) AS new_bal
FROM account a, deposit d
WHERE a.acct_num = d.acct_num
```

MySQL also allows comparison operators

```
SELECT owner, balance > 5000 as large_balance
FROM account
WHERE acct_type = 'checking'
```

<table>
<thead>
<tr>
<th>owner</th>
<th>large_balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>0</td>
</tr>
<tr>
<td>Bob</td>
<td>0</td>
</tr>
<tr>
<td>Chuck</td>
<td>1</td>
</tr>
<tr>
<td>Debbie</td>
<td>0</td>
</tr>
</tbody>
</table>

Query Answer:
SQL largely ignores NULL values when computing aggregates

Consider the following table

<table>
<thead>
<tr>
<th>id</th>
<th>retail_price</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5000</td>
<td>economy</td>
</tr>
<tr>
<td>2</td>
<td>NULL</td>
<td>sports</td>
</tr>
<tr>
<td>3</td>
<td>10000</td>
<td>luxury</td>
</tr>
</tbody>
</table>

The query:

```sql
SELECT AVG(retail_price)
FROM Car;
```

Returns:

| AVG(retail_price) | 7500 |

And the query:

```sql
SELECT COUNT(retail_price)
FROM Car;
```

Returns:

| COUNT(retail_price) | 2     |
Whereas the query:

```
SELECT COUNT(*)
FROM Car;
```

Returns:

| COUNT(*) | 3 |

- again with COUNT better to use COUNT(*)
- unless you are counting non-null values
The UNION keyword in SQL

(SELECT owner
 FROM account
 WHERE balance >= 1/zero.alt1/zero.alt1/zero.alt1/zero.alt1
) UNION
(SELECT owner
 FROM account
 WHERE balance <= 5/zero.alt1/zero.alt1);

Note we can add literals to help distinguish results

(SELECT owner, 'high balance' as at_risk_type
 FROM account
 WHERE balance >= 10000
) UNION
(SELECT owner, 'low balance' as at_risk_type
 FROM account
 WHERE balance <= 100);

<table>
<thead>
<tr>
<th>owner</th>
<th>at_risk_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debbie</td>
<td>high balance</td>
</tr>
<tr>
<td>Alice</td>
<td>low balance</td>
</tr>
</tbody>
</table>
The INTERSECT keyword in SQL

\[
\text{(SELECT owner} \\
\text{FROM account} \\
\text{WHERE acct_type = 'checking'}) \\
\text{INTERSECT} \\
\text{(SELECT owner} \\
\text{FROM account} \\
\text{WHERE acct_type = 'savings');}
\]

Q: What does this query do? What does it return?

Result

<table>
<thead>
<tr>
<th>owner</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
</tr>
</tbody>
</table>

Can we perform this intersection without INTERSECT?

\[
\text{SELECT DISTINCT a1.owner} \\
\text{FROM account a1 JOIN account a2 USING (owner)} \\
\text{WHERE a1.acct_type = 'checking' AND a2.acct_type = 'savings'};
\]

- We use DISTINCT since INTERSECT is a set-based operation (more later)

There are other ways as well ... e.g., using EXISTS (which we'll see later)