Lecture 19:
• Quiz 4
• Intro to Aggregates

Announcements:
• Project Step 2 due
• PS-3 due
• PS-4 out, due ...
• HW-5 due Thurs, Nov 10

Bank Account Example Tables

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

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

<table>
<thead>
<tr>
<th>Withdraw</th>
<th>acct_num</th>
<th>check_num</th>
<th>check_date</th>
<th>amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>101</td>
<td>924</td>
<td>10/23/18</td>
<td>125.00</td>
</tr>
<tr>
<td></td>
<td>101</td>
<td>925</td>
<td>10/24/18</td>
<td>23.98</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>128</td>
<td>10/29/18</td>
<td>200.00</td>
</tr>
</tbody>
</table>
SQL Aggregate Functions

SQL provides 5 aggregate operators: COUNT, SUM, MIN, MAX, AVG

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

Returns:
<table>
<thead>
<tr>
<th>COUNT(balance)</th>
<th>SUM(balance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>3</td>
</tr>
</tbody>
</table>

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

Returns:
<table>
<thead>
<tr>
<th>MIN(balance)</th>
<th>MAX(balance)</th>
<th>AVG(balance)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>2000</td>
<td>1333.333333</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:

SELECT owner, AVG(balance)
FROM account

this query is not well formed ... but MySQL allows it

Q: How does MySQL answer the query?
- calculate the average for each owner? ... NO!
- pick an owner and then average balance across all owners ... YES!

Note: This is confusing. So beware!
Aggregate Functions (cont)

DBMSs have additional aggregates beyond standard 5 ... e.g., in MySQL:

- `STDEV()` – returns standard deviation
- `VARIANCE()` – returns the standard variance
- `GROUP_CONCAT()` – concate selected strings
- `JSON_ARRAYAGG()` – return list of values (as JSON list)
- `JSON_OBJECTAGG(key, val)` – return dictionary of key-val pairs
- `BIT_AND(), BIT_OR(), BIT_XOR()` – apply op to list of values

Notes:

- aggregates are special because they are “column-based” ... lists
- aggregates only allowed in SELECT and HAVING clauses (more later)
- there are also non-aggregate functions supported in most DBMSs

Aggregate Functions: Variations of COUNT

`COUNT(attribute)` returns the number of NON-NULL attribute values

Q: What does this query return?

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

Returns: 4 ... since the last row has a NULL value

`COUNT(*)` is a special case that returns number of rows (regardless of NULLs)

Q: What does this query return?

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

Returns: 5 ... even though last row has a NULL attribute value
Aggregate Functions: Variations of COUNT

COUNT(DISTINCT attribute) returns number of unique NON-NULL values

Q: What does this query return?

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

*Returns: 2 ... since only checking and savings values*

COUNT(DISTINCT attribute₁, ..., attributeₖ) is similar but unique k-tuples

Q: What does this query return?

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

*Returns: 3 ... two duplicate rows, one contains a NULL-valued attribute*

Generally better to use COUNT(*) unless using DISTINCT

- usually just counting rows (especially if using a NON-NULL attribute)
- counting attribute values can be misleading (if we just want rows)
- unless we really want to count number of NON-NULL values (likely rare)
Aggregate Functions: Using DISTINCT

Note: The other 4 standard aggregates only take a single attribute
• where the aggregate is applied only to the NON-NULL attribute values
• e.g., \( \text{SUM}(x) \) only sums the NON-NULL \( x \) values

Also possible to use DISTINCT in other 4 standard aggregates

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

```sql
SELECT \text{SUM}(\text{balance})
FROM account;

SELECT \text{SUM(DISTINCT balance)}
FROM account;
```

*1st Returns: 19000.00, 2nd Returns: 18000.00*

Limits

Can use LIMIT to restrict number of rows returned by a query

```sql
SELECT *
FROM account
LIMIT 1;
```

*Returns:*

<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>
</tbody>
</table>

```sql
SELECT *
FROM account
LIMIT 2;
```

*Returns:*

<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>
</tbody>
</table>
Limits

Can use `LIMIT x, y` to return `y` rows starting at row offset `x`

```sql
SELECT *
FROM account
LIMIT 1, 2;
```

Returns:

<table>
<thead>
<tr>
<th>acct_num</th>
<th>owner</th>
<th>balance</th>
<th>acct_type</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

```sql
SELECT *
FROM account
LIMIT 3, 2;
```

Returns:

<table>
<thead>
<tr>
<th>acct_num</th>
<th>owner</th>
<th>balance</th>
<th>acct_type</th>
</tr>
</thead>
<tbody>
<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>NULL</td>
</tr>
</tbody>
</table>

Limits and Ordering

Can approximate “top-k” queries using `LIMIT` with `ORDER BY`

E.g., the “top 2” highest checking balances (not really the top 2, but close)

```sql
SELECT *
FROM account
WHERE acct_type = 'checking'
ORDER BY balance DESC
LIMIT 2;
```

Returns:

<table>
<thead>
<tr>
<th>acct_num</th>
<th>owner</th>
<th>balance</th>
<th>acct_type</th>
</tr>
</thead>
<tbody>
<tr>
<td>102</td>
<td>Bob</td>
<td>2000.00</td>
<td>checking</td>
</tr>
<tr>
<td>101</td>
<td>Alice</td>
<td>1000.00</td>
<td>checking</td>
</tr>
</tbody>
</table>