4. Structured Query Language 2
Aggregate Functions

• Operate on a column of a relation, and return a value
  
  \textbf{avg}: average value
  \textbf{min}: minimum value
  \textbf{max}: maximum value
  \textbf{sum}: sum of values
  \textbf{count}: number of values

• Note: for our examples we use the tables:
  – Branch (*branch-name*, *branch-city*, assets)
  – Account (*account-number*, balance, *branch-name*)
  – Depositor (*customer-name*, *account-number*)
  – Customer (*customer-name*, *customer-street*, *customer-city*)
Find the average account balance at the Perryridge branch.

```sql
select avg(balance)
from account
where branch-name="Perryridge"
```

Balances of Perryridge accounts

```
Avg() 120,000
```
Examples of Aggregate Functions

• Find the numbers of tuples in the account relation.
  
  \[
  \text{select count(*)} \\
  \text{from account}
  \]
  
  – remember * stands for all attributes
  
  – Same as:
    
    \[
    \text{select count(branch-name)} \\
    \text{from account}
    \]
  
  – Different from:
    
    \[
    \text{select count(\textbf{distinct} branch-name)} \\
    \text{from account}
    \]
    
  – Because branch-name is not a key in account
Group by

- Find the number of accounts for each branch.
  select branch-name, count(account-number)
  from account
  group by branch-name

- For each group of tuples with the same branch-name, count the account-numbers for this group

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>a-102</td>
<td>400</td>
</tr>
<tr>
<td>Brighton</td>
<td>a-217</td>
<td>750</td>
</tr>
<tr>
<td>Perryridge</td>
<td>a-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>a-215</td>
<td>750</td>
</tr>
<tr>
<td>Redwood</td>
<td>a-222</td>
<td>700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>a-102</td>
<td>400</td>
</tr>
<tr>
<td>Perryridge</td>
<td>a-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>a-217</td>
<td>750</td>
</tr>
<tr>
<td>Brighton</td>
<td>a-215</td>
<td>750</td>
</tr>
<tr>
<td>Redwood</td>
<td>a-222</td>
<td>700</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>branch-name</th>
<th>count-account-no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>2</td>
</tr>
<tr>
<td>Brighton</td>
<td>2</td>
</tr>
<tr>
<td>Redwood</td>
<td>1</td>
</tr>
</tbody>
</table>
Attributes in `select` clause outside of aggregate functions must appear in `group by` list, why?

```
select branch-name, balance, count( distinct account-number)
from account
group by branch-name
```

**Correct**

```
select ... from account
group by branch-name, balance
OR
select branch-name, sum(balance), count(...)
from account group by branch-name
```
Group by with Join

- Find the number of depositors for each branch.

```sql
select branch-name, count( distinct customer-name) 
from depositor, account 
where depositor.account-number = account.account-number 
group by branch-name
```

- Perform Join  then group by then count ( distinct () )

```sql
depositor (customer-name, account-number) 
account (account-number, branch-name, balance) 
Join ⇒ (customer-name, account-number, branch-name, balance)
```

- Group by and aggregate functions apply to the Join result
select branch-name, customer-name
from depositor, account
where depositor.account-number = account.account-number

group by branch-name, customer-name

<table>
<thead>
<tr>
<th>branch-name</th>
<th>cust-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>John Wong</td>
</tr>
<tr>
<td>Perryridge</td>
<td>Jacky Chan</td>
</tr>
<tr>
<td>Uptown</td>
<td>John Wong</td>
</tr>
<tr>
<td>Uptown</td>
<td>Mary Kwan</td>
</tr>
<tr>
<td>Downtown</td>
<td>John Wong</td>
</tr>
<tr>
<td>Downtown</td>
<td>Pat Lee</td>
</tr>
<tr>
<td>Downtown</td>
<td>May Cheung</td>
</tr>
</tbody>
</table>

distinct count

<table>
<thead>
<tr>
<th>branch-name</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>2</td>
</tr>
<tr>
<td>Uptown</td>
<td>2</td>
</tr>
<tr>
<td>Downtown</td>
<td>3</td>
</tr>
</tbody>
</table>
Having Clause (condition on the groups)

- Find the names and average of balances of all branches where the average account balance is more than $700
  
  ```sql
  select branch-name, avg(balance)
  from account
  group by branch-name
  having avg (balance) > 700
  ```

- Predicates in the **having** clause are applied to **each group** after the formation of groups

<table>
<thead>
<tr>
<th>branch-name</th>
<th>account-number</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perryridge</td>
<td>a-102</td>
<td>400</td>
</tr>
<tr>
<td>Perryridge</td>
<td>a-201</td>
<td>900</td>
</tr>
<tr>
<td>Brighton</td>
<td>a-217</td>
<td>750</td>
</tr>
<tr>
<td>Brighton</td>
<td>a-215</td>
<td>750</td>
</tr>
<tr>
<td>Redwood</td>
<td>a-222</td>
<td>700</td>
</tr>
</tbody>
</table>
Having Clause

• Display the names of all branches in Hong Kong where the average account balance is more than $700
  
  ```sql
  select branch-name
  from account, branch
  where account.branch-name=branch.branch-name
  and branch-city="Hong Kong"
  group by branch-name
  having avg (balance) >700
  ```

• first you find the records that satisfy the `where` condition, then you form the groups (including only those records), and finally you apply the `having` clause to `each group`
Derived Relations

- Find the name(s) of branches with the **maximum average** account balance.

```sql
select branch-name
from (select branch-name, avg(balance)
      from account
      group by branch-name)
as result (branch-name, avg-balance)
where avg-balance =
    (select max(avg-balance)
     from result))
```

Return avg balance of each branch