

Comp 5311 Database Management Systems

4. SQL 2

Basic SQL Structure

- Typical SQL query:

```
select A1, A2, ..., An  
from R1, R2, ..., Rm  
where P
```

- A_i represent attributes
- R_i represent relations
- P is a predicate.

- Example Tables:

- Branch (branch-name, branch-city, assets)
- Customer (customer-name, customer-street, customer-city)
- Loan (loan-number, amount, *branch-name*)
- Account (account-number, balance, *branch-name*)
- Borrower (*customer-name*, loan-number)
- Depositor (*customer-name*, account-number)

SQL - Nested Subqueries

- The result of every SQL statement is considered a table even if it is a single value or null
- You can replace a value or set of values with a SQL statement (ie., a subquery)
- Illegal if the subquery returns the wrong type for the comparison

Example Query - IN

- Find all customers who have both an account and a loan in the bank.

```
select distinct customer-name
from borrower
where customer-name in (select customer-name
from depositor)
```

Check for each borrower
if he/she is *also* a depositor

Return the set of depositors

Example Query – NOT IN

- Find all customers who have a loan at the bank but do not have an account at the bank.

```
select distinct customer-name
from borrower
where customer-name not in (select customer-name
                             from depositor)
```

The **Some** Clause

- Find all branches that have greater assets than some branch located in Brooklyn
 - Equivalent to “find all branches that have greater assets than the **minimum** assets of any branch located in Brooklyn”

```
select branch-name
from branch
where assets > some
(select assets
from branch
where branch-city = "Brooklyn")
```

Assets of all branches in Brooklyn

Some Semantics

(5 < some

0
5
6

) returns true (5 < 6)

(5 < some

0
5

) returns false

(5 = some

0
5

) = true

(5 ≠ some

0
5

) = true (since 0 ≠ 5)

Note:

(= some) is equivalent to **in**
However, (≠ some) is not
equivalent to **not in**

The **All** Clause

- Find the names of all branches that have greater assets than *all* branches located in Brooklyn.
 - Equivalent to “find all branches that have greater assets than the **maximum** assets of any branch located in Brooklyn”

```
select branch-name  
from branch  
where assets > all
```

```
(select assets  
from branch  
where branch-city="Brooklyn")
```

Assets of all branches in Brooklyn

All Semantics

$$(5 < \text{all } \begin{array}{|c|} \hline 0 \\ \hline 5 \\ \hline 6 \\ \hline \end{array}) = \text{false}$$

$$(5 < \text{all } \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

$$(5 = \text{all } \begin{array}{|c|} \hline 4 \\ \hline 5 \\ \hline \end{array}) = \text{false}$$

$$(5 \neq \text{all } \begin{array}{|c|} \hline 6 \\ \hline 10 \\ \hline \end{array}) = \text{true}$$

Note:

(\neq all) is equivalent to **not in**
However, ($=$ all) is not equivalent to **in**

Test for Empty Relations

- `exists` returns `true` if the argument subquery is nonempty.
- Find all customer names who have both a loan and an account.

```
select customer-name from depositor as D where exists  
(select * from borrower as B where D.customer-name =  
B.customer-name)
```

- Find all customer names who have an account but no loan.

```
select customer-name from depositor as D where not exists  
(select * from borrower as B where D.customer-name =  
B.customer-name)
```

Test for Absence of Duplicate Tuples

- **unique** tests whether a subquery has any duplicate tuples in its result.
- Find all customers who have **only one account** at the Perryridge branch.

```
select T.customer-name  
from depositor as T
```

For each depositor T, check ...

```
where unique (
```

```
select R.customer-name  
from account, depositor as R  
where T.customer-name = R.customer-name and  
R.account-number = account.account-number and  
account.branch-name = "Perryridge")
```

Find depositors with
same name as T

Customers at Perryridge with same name as T

Example Query – NOT UNIQUE

- Find all customers with **at least 2 accounts** at the Perryridge branch.

```
select T.customer-name
from depositor as T
where not unique(
    select R.customer-name
    from account, depositor as R
    where T.customer-name = R.customer-name and
    R.account-number = account.account-number and
    account.branch-name = "Perryridge")
```

Division in SQL

- Find all customers with an account at *all* branches located in Brooklyn.

```
select distinct S.customer-name  
from depositor as S  
where not exist (
```

For each
customer S,
check ...

Branches in Brooklyn
where customer S
doesn't have an account

```
(select branch-name  
from branch  
where branch-city="Brooklyn")  
except
```

$$X - Y = \phi \Leftrightarrow X \subseteq Y$$

Branches where
customer S
has an account

```
(select R.branch-name  
from depositor as T, account as R  
where T.account-number = R.account-number and  
S.customer-name = T.customer-name) )
```

Aggregate Functions

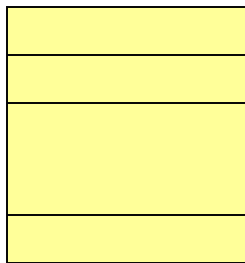
- Operate on a column of a relation, and return a value
 - `avg`: average value
 - `min`: minimum value
 - `max`: maximum value
 - `sum`: sum of values
 - `count`: number of values
- Note: for our examples we use the tables:
 - Branch (branch-name, branch-city, assets)
 - Account (account-number, balance, *branch-name*)
 - Depositor (*customer-name*, *account-number*)
 - Customer (customer-name, customer-street, customer-city)

Aggregate Function Computation

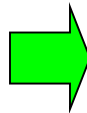
- Find the average account balance at the Perryridge branch.

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

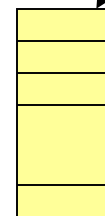
account



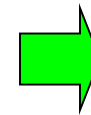
```
select balance
from account
where branch-name
="Perryridge"
```



Balances of Perryridge accounts



Avg()



120,000

Examples of Aggregate Functions

- Find the numbers of tuples in the account relation.

```
select count(*)  
from account
```

- remember * stands for all attributes
- Same as:
 select count(branch-name)
 from account
- Different from:
 select count(distinct branch-name)
 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

branch-name	account-number	balance
Perryridge	a-102	400
Brighton	a-217	750
Perryridge	a-201	900
Brighton	a-215	750
Redwood	a-222	700



branch-name	account-number	balance
Perryridge	a-102	400
Perryridge	a-201	900
Brighton	a-217	750
Brighton	a-215	750
Redwood	a-222	700

account table



branch-name	count-account-no
Perryridge	2
Brighton	2
Redwood	1



Group by Attributes

- 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
```

branch-name	account-number	balance
Perryridge	a-102	400
Perryridge	a-201	900
Brighton	a-217	750
Brighton	a-215	750
Redwood	a-222	700

Group by with Join

- Find the number of depositors for each branch.

```
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** ())
depositor (customer-name, account-number)
account (account-number, branch-name, balance)
Join \Rightarrow (customer-name, account-number, branch-name, balance)
- Group by and aggregate functions apply to the Join result

Group by Evaluation

```
select branch-name, customer-name
from depositor, account
where depositor.account-number
= account.account-number
```

branch-name	cust-name
Perryridge	John Wong
Perryridge	Jacky Chan
Uptown	John Wong
Uptown	Mary Kwan
Downtown	John Wong
Downtown	Pat Lee
Downtown	May Cheung

count

branch-name	count
Perryridge	2
Uptown	2
Downtown	3

↓ join

branch-name	cust-name
Perryridge	John Wong
Downtown	Pat Lee
Uptown	John Wong
Perryridge	Jacky Chan
Uptown	Mary Kwan
Downtown	John Wong
Perryridge	John Wong
Downtown	May Cheung

↑ distinct

branch-name	cust-name
Perryridge	John Wong
Perryridge	Jacky Chan
Perryridge	John Wong
Uptown	John Wong
Uptown	Mary Kwan
Downtown	John Wong
Downtown	Pat Lee
Downtown	May Cheung


group by



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

```
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



branch-name	account-number	balance
Perryridge	a-102	400
Perryridge	a-201	900
Brighton	a-217	750
Brighton	a-215	750
Redwood	a-222	700

Having Clause

- Display the names of all branches in Hong Kong where the average account balance is more than \$700

```
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.

```
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

