COMP 3311
DATABASE MANAGEMENT SYSTEMS

LECTURE 8
STRUCTURED QUERY LANGUAGE (SQL)

EXAMPLE BANK RELATIONAL SCHEMA

Branch(branchName, branchCity, assets)

Client(clientName, clientStreet, clientCity)

Loan(loanNo, amount, *branchName*)

Account(accountNo, balance, *branchName*)

Borrower(*clientName, loanNo*)

Depositor(*clientName, accountNo*)

Attribute names in italics are foreign key attributes.

AGGREGATE FUNCTIONS

- An aggregate function operates on an attribute of a relation and returns a single value (i.e., a table with one row and one column).

avg	average value
count	number of tuples / values
max	maximum value
min	minimum value
sum	sum of values (total)

- For **sum**, **avg**, the input must be numbers.

- For other functions, the input can be non-numeric (e.g., strings).

AGGREGATE FUNCTIONS: COMPUTATION

Query: Find the average account balance at the Central branch.

```
select avg(balance) as avgBalance
from Account
where branchName='Central';
```

Select the tuples where branch name equals Central.

Project on the balance attribute retaining duplicates.

Calculate the average.

Account

accountNo	balance	branch Name
A-102	40000	Tsimshatsui
A-217	7500	Central
A-201	9000	Tsimshatsui
A-215	7500	Central
A-222	7500	Mongkok
A-224	10000	Central

accountNo	balance	branch Name
A-217	7500	Central
A-215	7500	Central
A-224	10000	Central

balance
7500
7500
10000

avg(balance)

avgBalance
8333.33

AGGREGATE FUNCTIONS: EXAMPLES

Query: Find the number of tuples in the account relation.

```
select count(*)
from Account;
```

☞ **Remember * stands for *all* attributes.**

Same as:

```
select count(branchName)
from Account;
```

Why?

Different from:

```
select count(distinct branchName)
from Account;
```

Why?

Cannot say:

```
select count(distinct *)
from Account;
```

SQL does not allow the use of distinct with count(*).

GROUP BY CLAUSE

Motivation: A group by clause permits aggregate results to be displayed (e.g., max, min, sum) for groups.
For example, group by x will get a result for every different value of x.

☞ **Aggregate queries without group by return a single number.**

Query: Find the number of accounts for *each* branch.

```
select branchName, count(*)
from Account
group by branchName;
```

GROUP BY CLAUSE (CONT'D)

Query: Find the number of accounts for *each* branch.

```
select branchName, count(*)
from Account
group by branchName;
```

Group the tuples by branch name.

For each group, count the number of tuples in the group.

Account

accountNo	balance	branch Name
A-102	40000	Tsimshatsui
A-217	7500	Central
A-201	9000	Tsimshatsui
A-215	7500	Central
A-222	7500	Mongkok
A-224	10000	Central

accountNo	balance	branch Name
A-102	40000	Tsimshatsui
A-201	9000	Tsimshatsui
A-217	7500	Central
A-215	7500	Central
A-224	10000	Central
A-222	7500	Mongkok

branch Name	count AccountNo
Tsimshatsui	2
Central	3
Mongkok	1

GROUP BY CLAUSE: ATTRIBUTES

accountNo	balance	branch Name
A-102	40000	Tsimshatsui
A-201	9000	Tsimshatsui
A-217	7500	Central
A-215	7500	Central
A-224	10000	Central
A-222	7500	Mongkok

Query: Find the balance and number of accounts for *each* branch.

```
select branchName, balance, count(*)
from Account
group by branchName;
```

accountNo	balance	branch Name
A-102	40000	Tsimshatsui
A-201	9000	Tsimshatsui
A-217	7500	Central
A-215	7500	Central
A-224	10000	Central
A-222	7500	Mongkok

Illegal! Why?

An attribute in the select clause must also appear in the group by clause.

The opposite is not true!

Attributes in the group by clause do not need to appear in the select clause.

GROUP BY CLAUSE: ATTRIBUTES (cont'd)

accountNo	balance	branch Name
A-102	40000	Tsimshatsui
A-201	9000	Tsimshatsui
A-217	7500	Central
A-215	7500	Central
A-224	10000	Central
A-222	7500	Mongkok

Query: Find the balance and number of accounts for *each* branch.

```
select branchName, balance, count(*)
from Account
group by branchName, balance;
```

branch Name	balance	count
Tsimshatsui	40000	1
Tsimshatsui	9000	1
Central	7500	2
Central	10000	1
Mongkok	7500	1

Either is legal SQL.

```
select branchName, sum(balance), count(*)
from Account
group by branchName;
```

branch Name	sum (balance)	count
Tsimshatsui	49000	2
Central	25000	3
Mongkok	7500	1

GROUP BY CLAUSE WITH JOIN: WITH JOIN

Query: Find the number of depositors for each branch.

```
select branchName, count(distinct clientName)
from Depositor natural join Account
group by branchName;
```

JOIN ⇒ (clientName, accountNo, balance, branchName)

client Name	branch Name
John Wong	Tsimshatsui
Pat Lee	Central
Mary Kwan	Mongkok
Jacky Chan	Central
John Wong	Tsimshatsui
May Cheung	Central

group by

client Name	branch Name
John Wong	Tsimshatsui
~~John Wong~~	~~Tsimshatsui~~
Mary Kwan	Mongkok
Jacky Chan	Central
Pat Lee	Central
May Cheung	Central

distinct

client Name	branch Name
John Wong	Tsimshatsui
Mary Kwan	Mongkok
Jacky Chan	Central
Pat Lee	Central
May Cheung	Central

count

branch Name	count
Tsimshatsui	1
Mongkok	1
Central	3

☞ **Group by and aggregate functions apply to the join result.**

HAVING CLAUSE

- Allows a condition to be applied to groups rather than to individual tuples.

Query: Find the names and average balances of all branches where the average account balance is more than $8000.

```
select branchName, avg(balance)
from Account
group by branchName
having avg(balance)>8000;
```

	accountNo	balance	branch Name	
✓	A-102	40000	Tsimshatsui	avg(24500.00)
	A-201	9000	Tsimshatsui	
	A-217	7500	Central	
✓	A-215	7500	Central	avg(8333.33)
	A-224	10000	Central	
X	A-222	7500	Mongkok	avg(7500.00)

Any condition that appears in the **having** clause refers to the groups and is applied ***after*** the formation of the groups.

The condition in the **having** clause must involve attributes or aggregate functions that appear in the **select** clause or **group by** clause.

HAVING CLAUSE: EVALUATION SEQUENCE

1. Evaluate the from clause to get a relation.
2. If a where clause is present, apply the predicate in the where clause to the from clause result relation before the formation of groups.

   ☞ **Records that do not pass the where predicate are eliminated *before* the formation of groups.**

3. Group tuples satisfying the where predicate into groups by the group by clause, if present. If the group by clause is absent, the entire set of tuples satisfying the where predicate is treated as a group.
4. Apply the having clause, if present, to each group retaining only those groups satisfying the having clause.
5. Apply the aggregate functions in the select clause to get a single result for each group.

☞ **Any attribute present in the having clause *that is not being aggregated* must appear in the group by clause.**

HAVING CLAUSE: EXAMPLE

Query: Find the names of all branches in Hong Kong where the average account balance is more than $8000.

```
select Branch.branchName
from Account, Branch
where Account.branchName=Branch.branchName
      and branchCity='Hong Kong'
group by Branch.branchName
having avg(balance)>8000
```

First, find the records that satisfy the where predicate.

Then, form the groups (include only those tuples that satisfy the where predicate).

Finally, apply the having clause to *each group*.

NESTED SUBQUERIES

- Every SQL statement returns a relation as the result.

  ☞ **A relation can be null or contain only a single, atomic value.**

- Consequently, a value or a set of values can be replaced with a SQL statement (i.e., with a subquery).

  ☞ **The query is illegal if the subquery returns the wrong number of tuples or the wrong type for the comparison.**

```
select *
from Loan
where amount>120000;
```

```
select *
from Loan
where amount>(select avg(amount)
              from Loan);
```

This subquery *must* return a single, numeric value else it is illegal.

Subqueries are commonly used to test for set membership, do set comparisons or determine set cardinality.

SET MEMBERSHIP: IN

Query: Find all clients who have both an account and a loan.

```
select distinct clientName
from Borrower
where clientName in (select clientName
                     from Depositor);
```

The in operator tests for the presence of set membership (i.e., selects clients in the Borrower set *only if* they are in the Depositor set).
Duplicates are retained.

The *set* of clients who have an account.

SET MEMBERSHIP: NOT IN

Query: Find all clients who have a loan, but do not have an account.

```
select distinct clientName
from Borrower
where clientName not in (select clientName
                         from Depositor);
```

The not in operator tests for the absence of set membership (i.e., selects clients in the Borrower set *only if* they are not in the Depositor set).
Duplicates are retained.

The *set* of clients who have an account.

SET COMPARISON: SOME

Query: Find the names of all branches that have greater assets than *some* (i.e., at least one) branch located in Hong Kong.

☞ Equivalent to: Find the names of all branches that have greater assets than the minimum assets of any branch located in Hong Kong.

```
select branchName
from Branch
where assets >some (select assets
                    from Branch
                    where branchCity='Hong Kong');
```

The where clause is true if the assets value of a **Branch** tuple is greater than at least one member of the set of all assets values of branches in Hong Kong (i.e., greater than the minimum assets value in the set). ***Duplicates are retained.***

The *set* of assets values of all branches in Hong Kong.

SET COMPARISON: SEMANTICS OF SOME

(5 < **some** | 0 | 5 | 6 |) returns true (since 5 is less than the maximum value 6 in the set)

(5 < **some** | 0 | 5 |) returns false (since 5 is not less than the maximum value 5 in the set)

(5 = **some** | 0 | 5 |) returns true (since 5 = 5)

(5 ≠ **some** | 0 | 5 |) returns true (since 5 ≠ 0)

Note

(= **some**) is equivalent to **in**

(≠ **some**) is not equivalent to **not in**

SET COMPARISON: ALL

Query: Find the names of those branches that have greater assets than *all* branches located in Hong Kong.

☞ Equivalent to: Find the names of all branches that have greater assets than the maximum assets of any branch located in Hong Kong.

```
select branchName
from Branch
where assets >all (select assets
                   from Branch
                   where branchCity='Hong Kong');
```

The where clause is true if the assets value of a **Branch** tuple is greater than each of the members of the set of all assets values of branches in Hong Kong (i.e., greater than the maximum assets value in the set). ***Duplicates are retained.***

The *set* of assets values of all branches in Hong Kong.

SET COMPARISON: SEMANTICS OF ALL

(5 < all $\boxed{0 \mid 5 \mid 6}$) returns false (since 5 is not less than all of the members in the set)

(5 < all $\boxed{6 \mid 9}$) returns true (since 5 is less than all of the members in the set)

(5 = all $\boxed{4 \mid 5}$) returns false (since 5 ≠ 4)

(5 ≠ all $\boxed{6 \mid 9}$) returns true (since 5 ≠ 6 or 9)

Note

(≠ all) is equivalent to not in

(= all) is not equivalent to in

EMPTY RELATION TEST

- The exists operator returns true if the subquery is *not empty* (i.e., the subquery returns at least one tuple).

Query: Find all client names who have both a loan and an account.

If the join result is not empty, return the client name.

```
select clientName
from Depositor D
where exists (select *
              from Borrower B
              where D.clientName=B.clientName);
```

Clients who have both a loan and an account are in the join result.

Scoping rules for correlation names (aliases) in subqueries.

- A correlation name defined in a subquery can be used only in the subquery itself or in any query contained in the subquery (e.g., D can be used in the nested select; B cannot be used in the outer select).
- Locally defined correlation names override globally defined names.

EMPTY RELATION TEST (CONT'D)

- The not exists operator returns true if the subquery is *empty* (i.e., the subquery returns no rows).

Query: Find all client names who have an account, but no loan.

If the join result is empty, return the client name.

```
select clientName
from Depositor D
where not exists (select *
                  from Borrower B
                  where D.clientName=B.clientName);
```

Clients who have both a loan and an account are in the join result.

☞ The not exists operator can be used to simulate set containment.

relation A contains relation B ⇔ not exists (B minus A)

Not implemented in Oracle.

DUPLICATE TUPLES TEST: UNIQUE

- The unique operator tests for the *non existence* (i.e., absence) of duplicate tuples in a subquery.

☞ **Returns true if the subquery contains no duplicate tuples.**

Query: Find all clients who have *only one account* at the Tsimshatsui branch.

See later slide for an alternate way to answer this query.

```
select D1.clientName
from Depositor D1
where unique (select D2.clientName
              from Account, Depositor D2
              where D1.clientName=D2.clientName
                    and D2.accountNo=account.accountNo
                    and account.branchName='Tsimshatsui');
```

For each depositor D1, check …

Clients at Tsimshatsui with the same name as D1.

Find depositors with the same name as D1.

Not implemented in Oracle.

DUPLICATE TUPLES TEST: NOT UNIQUE

- The not unique operator tests for the *existence* (i.e., presence) of duplicate tuples in a subquery.

☞ **Returns true if the subquery contains two or more duplicate tuples.**

Query: Find all clients who have *at least two accounts* at the Tsimshatsui branch.

See later slides for an alternate way to answer this query.

```
select D1.clientName
from Depositor D1
where not unique (select D2.clientName
                  from Account, Depositor D2
                  where D1.clientName=D2.clientName
                        and D2.accountNo=account.accountNo
                        and account.branchName='Tsimshatsui');
```

☞ **Fails if tuples contain null values.**

DUPLICATE TUPLES TEST: REVISITED

- The group by and having clauses can test for the *non existence* (absence) and *existence* (presence) of duplicate tuples.

Query: Find all clients who have *only one account* at the Tsimshatsui branch.

```
select clientName
from Depositor D, Account A
where D.account#=A.account#
        and branchName='Tsimshatsui'
group by clientName
having count(*)=1;
```

Query: Find all clients who have *at least two accounts* at the Tsimshatsui branch.

How would you answer this query?

SUBQUERIES IN THE FROM CLAUSE

- The from clause can contain a subquery expression.

Why?

Query: Find the name(s) of branches whose average balance is greater than the average account balance.

```
select branchName
from (select branchName, avg(balance) as avgBalance
      from Account
      group by branchName) result
where avgBalance>(select avg(balance)
                  from Account);
```

The average balance of all accounts.

result

branch Name	avg Balance
Tsimshatsui	24500.00
Mongkok	7500.00
Central	8333.33

The result relation contains the branch name and average balance of each branch.

☞ **The relation "result" is called a *derived relation*.**

SUBQUERIES IN THE FROM CLAUSE (CONT'D)

Query: Find the name and average balance of branches with the *maximum average* account balance.

```
select branchName, avgBalance
from (select branchName, avg(balance) as avgBalance
      from Account
      group by branchName) result
where avgBalance=(select max(avgBalance)
                  from result);
```

result

branch Name	avg Balance
Tsimshatsui	24500.00
Mongkok	7500.00
Central	8333.33

The result relation contains the branch name and average balance of each branch.

The maximum average balance in the result relation.

Oracle Note

This query is *not allowed in Oracle* due to Oracle's scoping rules.
(The scope of the result relation is restricted to the outer select clause.)
See the next slide.

WITH CLAUSE

- Allows a temporary (derived) relation to be defined that is available only to the query in which the with clause occurs.

Query: Find the name and average balance of branches with the *maximum average* account balance.

```
with result (branchName, avgBalance) as
    (select branchName, avg(balance)
     from Account
     group by branchName)
select branchName, avgBalance
from result
where avgBalance=(select max(avgBalance)
                  from result);
```

result

branch Name	avg Balance
Tsimshatsui	24500.00
Mongkok	7500.00
Central	8333.33

The result relation contains the branch name and average balance of each branch.

The maximum average balance in the result relation.

Oracle Note
This query *is allowed* in Oracle.