SQL as Query Language, Part II

Some slides are based or modified from originals by
Elmasri and Navathe,
© 2004 Pearson Education, Inc.
and
© 2005 Silberschatz, Korth and Sudarshan
CORRELATED NESTED QUERIES

- If a condition in the WHERE-clause of a nested query references an attribute of a relation declared in the outer query, the two queries are said to be correlated.

- The result of a correlated nested query is different for each tuple (or combination of tuples) of the relation(s) the outer query.

- E.g. DB Company: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```sql
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN
(SELECT ESSN FROM DEPENDENT
WHERE ESSN=E.SSN
AND E.FNAME=DEPENDENT_NAME)
```
A query written with nested SELECT... FROM... WHERE... blocks and using the = or IN comparison operators can always be expressed as a single block query.

For example, the previous query could be

```sql
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE E, DEPENDENT D
WHERE E.SSN=D.ESSN
AND E.FNAME=D.DEPENDENT_NAME
```

The original SQL as specified for SYSTEM R also had a CONTAINS comparison operator, which is used in conjunction with nested correlated queries.

This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently.
EXPLICIT SETS

- It is also possible to use an explicit (enumerated) set of values in the WHERE-clause rather than a nested query.

- Ex. Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

  ```
  SELECT DISTINCT ESSN 
  FROM WORKS_ON 
  WHERE PNO IN (1, 2, 3)
  ```
Ordering the Display of Tuples

- List in alphabetic order the names of all customers having a loan in Perryridge branch

```sql
select distinct customer_name
from borrower, loan
where borrower loan_number = loan.loan_number
and branch_name = 'Perryridge'
ORDER BY customer_name
```

- We may specify `desc` for descending order or `asc` for ascending order, for each attribute;
  - ascending order is the default.
  - Example: `order by customer_name desc`
The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s).

**Ex2.** Retrieve a list of employees and the projects each works in, ordered by the employee's department, and within each department ordered alphabetically by employee last name.

```
SELECT DNAME, LNAME, FNAME, PNAME
FROM DEPARTMENT, EMPLOYEE,
    WORKS_ON, PROJECT
WHERE DNUMBER=DNO AND SSN=ESSN
    AND PNO=PNUMBER
ORDER BY DNAME, LNAME
```
Aggregate Functions

These functions operate on the multiset of values of 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
Aggregate Functions (Cont.)

- Find the average account balance at the Perryridge branch.
  
  ```sql
  select avg (balance) 
  from account 
  where branch_name = 'Perryridge'
  ```

- Find the number of tuples in the customer relation.
  
  ```sql
  select count (*) 
  from customer
  ```

- Find the number of depositors in the bank.
  
  ```sql
  select count (distinct customer_name) 
  from depositor
  ```
AGGREGATE FUNCTIONS

- Another Ex. Find the maximum salary, the minimum salary, and the average salary among all employees for the Company database

```sql
SELECT MAX(SALARY),
       MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE
```

Obs. Some SQL implementations may not allow more than one function in the SELECT-clause!
AGGREGATE FUNCTIONS (cont.)

Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```
AGGREGATE FUNCTIONS (cont.)

- Retrieve the total number of employees in the company

  ```sql
  SELECT COUNT (*)
  FROM EMPLOYEE
  ```

- and the number of employees in the 'Research' department.

  ```sql
  SELECT COUNT (*)
  FROM EMPLOYEE, DEPARTMENT
  WHERE DNO=DNUMBER AND DNAME='Research'
  ```
In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation.

Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s).

The function is applied to each subgroup independently.

SQL has a GROUP BY-clause for specifying the grouping attributes, which must also appear in the SELECT-clause.
Aggregate Functions – Group By

- 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
```

**Note:** Attributes in `select` clause outside of aggregate functions must appear in `group by` list.
For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT DNO, COUNT (*), AVG (SALARY)
FROM EMPLOYEE
GROUP BY DNO
```

- The EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO.
- The COUNT and AVG functions are applied to each such group of tuples separately.
- The SELECT-clause includes only the grouping attribute and the functions to be applied on each group of tuples.
- A join condition can be used in conjunction with grouping.
SQL Queries: Aggregation and Grouping

- Aggregate functions: AVG, COUNT, MIN, MAX, SUM, ... (user defined functions)
- Group-by

### Find average salary of all employees

SELECT Avg(Salary) AS AvgSal FROM Employee

<table>
<thead>
<tr>
<th>Name</th>
<th>Dept</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>Toys</td>
<td>45</td>
</tr>
<tr>
<td>Nick</td>
<td>PCs</td>
<td>50</td>
</tr>
<tr>
<td>Jim</td>
<td>Toys</td>
<td>35</td>
</tr>
<tr>
<td>Jack</td>
<td>PCs</td>
<td>40</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AvgSal</th>
</tr>
</thead>
<tbody>
<tr>
<td>42.5</td>
</tr>
</tbody>
</table>

### Find the average salary for each department

SELECT Dept, Avg(Salary) AS AvgSal FROM Employee GROUP-BY Dept

<table>
<thead>
<tr>
<th>Dept</th>
<th>AvgSal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toys</td>
<td>40</td>
</tr>
<tr>
<td>PCs</td>
<td>45</td>
</tr>
</tbody>
</table>
GROUPING Example

For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
```

- In this case, the grouping and functions are applied after the joining of the two relations
THE HAVING-CLAUSE

- Sometimes we want to retrieve the values of these functions for only those *groups that satisfy certain conditions*

- The HAVING-clause is used for specifying a selection condition on groups
  - rather than on individual tuples!
Aggregate Functions – Having Clause

Find the names of all branches where the average account balance is more than $1,200.

\[
\text{select} \quad \text{branch\_name, avg(balance)} \\
\text{from} \quad \text{account} \\
\text{group by} \quad \text{branch\_name} \\
\text{HAVING} \quad \text{avg(balance)} > 1200
\]

Note: predicates in the having clause are applied after the formation of groups whereas predicates in the where clause are applied before forming groups.
For each project *on which more than two employees work*, retrieve the project number, project name, and the number of employees who work on that project.

```sql
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT(*) > 2
```
Null Values and Aggregates

- Total all loan amounts

  ```sql
  select sum(amount) from loan
  ```

  - Above statement ignores null amounts
  - Result is `null` if there is no non-null amount

- All aggregate operations except `count(*)` ignore tuples with null values on the aggregated attributes.
Summary of SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory.
- The clauses are specified in the following order:

  SELECT <attribute list>
  FROM <table list>
  [WHERE <condition>]
  [GROUP BY <grouping attribute(s)>]
  [HAVING <group condition>]
  [ORDER BY <attribute list>]


Summary of SQL Queries (cont.)

- The SELECT-clause lists the attributes or functions to be retrieved.
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries.
- The WHERE-clause specifies the conditions for selection and join of tuples from the relations specified in the FROM-clause.
- GROUP BY specifies grouping attributes.
- HAVING specifies a condition for selection of groups.
- ORDER BY specifies an order for displaying the result of a query.
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause.