NESTED QUERIES AND AGGREGATION

CHAPTER 5 (6/E)

CHAPTER 8 (5/E)

LECTURE OUTLINE

- More Complex SQL Retrieval Queries
  - Self-Joins
  - Renaming Attributes and Results
  - Grouping, Aggregation, and Group Filtering
  - Ordering Results
  - Nested SPJ Queries

REVIEW OF SPJ QUERIES IN SQL

- SPJ (select-project-join) queries
  - SQL's basic `select-from-where` queries
  - Equivalent to using only σ, π, and $\bowtie$ (or $\times$) in Relational Algebra
    (and possibly ρ, if attributes need to be renamed before joining)

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:   SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     PROJECT, DEPARTMENT, EMPLOYEE
      WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
               Plocation='Stafford';
```

$$\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{Plocation='Stafford'}}(\text{PROJECT})$$
$$\text{CONTR\_DEPTS} \leftarrow (\text{STAFFORD\_PROJS} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT})$$
$$\text{PROJ\_DEPT\_MGRS} \leftarrow (\text{CONTR\_DEPTS} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$$
$$\text{RESULT} \leftarrow \pi_{\text{Pnumber, Dnum, Lname, Address, Bdate}}(\text{PROJ\_DEPT\_MGRS})$$

RENAMING IN SQL

- For convenience, include renaming (like ρ) as well
- **Aliases** or **tuple variables**
  - Provide alternative names for tables or columns

Customer

custid	name	address	phone

Sale

saleid	date	custid

LineItem

saleid	product	quantity	price

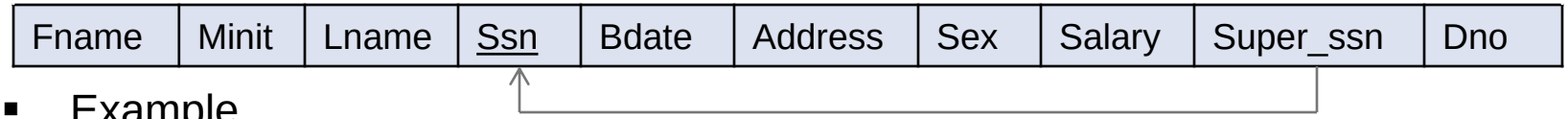
```
SELECT name, sale_date, product, quantity AS amount
FROM Customer C, Sale AS S(id,sale_date,custid), LineItem
WHERE C.custid = S.custid AND id = saleid;
```

  - Keyword `AS` is optional

SELF-JOINS

- Renaming is mandatory if table used more than once in a query

EMPLOYEE



- Example

Give the last names and salaries of employees and their managers whenever the employee earns more than the manager.

  - Think of the EMPLOYEE table as two tables, one for employees and one for managers.

E

M


```
SELECT E.Lname, E.Salary, M.Lname, M.Salary
FROM EMPLOYEE E, EMPLOYEE M
WHERE E.Super_ssn = M.Ssn and E.Salary > M.Salary;
```

AGGREGATE FUNCTIONS

- Used to accumulate information from multiple tuples, forming a single-tuple summary
- Built-in aggregate functions
  - `COUNT`, `SUM`, `MAX`, `MIN`, and `AVG`
- Used in the `SELECT` clause
- Examples:

  How many movies were directed by Steven Spielberg?

```
SELECT COUNT(*)
FROM Film
WHERE director='Steven Spielberg';
```

  - All tuples in result are counted, *with duplicates*!
    - `COUNT(title)` or `COUNT(director)` give same result!
  - `COUNT(DISTINCT year)` would include each year only once!

  What was the total movie profit since 2010, across how many directors?

```
SELECT SUM(gross - budget), COUNT(DISTINCT director)
FROM Film
WHERE year >= 2010;
```

GROUPING BEFORE AGGREGATION

- How can we answer a query such as

  "*How many films were directed by each director after 2001?*"

  - Need to produce a result with one tuple per director
    1. Partition relation into subsets of tuples based on **grouping column(s)**
    2. Apply function to each such group independently
    3. Produce one tuple per group
- **`GROUP BY`** clause to specify grouping attributes

```
SELECT director, COUNT(*)
FROM Film
WHERE year > 2001
GROUP BY director;
```

  - Every selector in `SELECT` clause must be a grouping column or an aggregation function
    - e.g., `SELECT director, year, COUNT(*)`

      would not be allowed unless *also* grouping by year

      i.e., `GROUP BY director, year`

HAVING CLAUSE

- After partitioning into groups, whole partitions can be discarded.
  - Provides a condition on the grouped tuples

```
SELECT      Dname, COUNT (*)
FROM        DEPARTMENT, EMPLOYEE
WHERE       Dnumber=Dno AND Salary>40000
GROUP BY    Dname
HAVING      COUNT (*) > 5;
```

- Having clause cannot reference individual tuples within group
  - Can reference grouping column(s) and aggregates only

- Contrast `WHERE` clause to `HAVING` clause

Note: As for aggregation, no `GROUP BY` clause means relation treated as one group

ORDERING OF QUERY RESULTS

- Final output of a query can be sorted by one or more column values
- Use **`ORDER BY`** clause
  - Keyword **`DESC`** for descending order of values
  - Optionally use keyword **`ASC`** for ascending order (default)
- Example

Course

dept	cnum	instructor	term

```
SELECT dept, term,
    COUNT(DISTINCT instructor) AS num_instructors
FROM Course
GROUP BY dept, term;
ORDER BY dept, term DESC;
```

  - Note that this is sorted ascending by department.
  - Within each department, terms sorted in descending order.
  - What if `DISTINCT` omitted? What if `term` omitted from `SELECT` clause? What if `dept` omitted from `GROUP BY` clause? What if `dept` omitted from `ORDER BY` clause?

SUMMARY OF SQL QUERIES

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

1. Assemble all tables according to **From** clause (“,” means to use $\times$).
2. Keep only tuples matching **Where** clause.
3. Group into blocks based on **Group By** clause.
4. Keep only blocks matching **Having** clause.
5. Create one tuple for each block using **Select** clause.
6. Order resulting tuples according to **Order By** clause.

NESTED QUERIES

- Any table can be used in FROM clause.
- `select-from-where` produces a table.
- Thus can nest one query within another.
- Example:

Give the biographical information for directors of profitable movies.

Film

title	genre	year	director	minutes	budget	gross

Person

name	birth	city

```
SELECT name, birth, city
FROM  (SELECT director
       FROM Film
       WHERE gross > budget) AS Profitable,
      Person
WHERE director = name
```

NESTED QUERIES (CONT'D.)

- Any column can be used in `SELECT` and `WHERE` clauses.
  - *But* refers to only one tuple value at a time
- `select-from-where` can produce a one-column table that contains only one tuple.
- Thus queries can also be nested in `SELECT` and `WHERE` clauses
- Example:

  Which film(s) had the highest budget?

```
SELECT *
FROM Film
WHERE budget = (SELECT MAX(budget)
                FROM Film);
```

USING IN FOR MEMBERSHIP TEST

- Comparison operator `IN`
  - Compares value *v* with a set (or bag) of values *V*
  - Evaluates to `TRUE` if *v* is one of the elements in *V*
  - Allows any relation in `WHERE` clause

```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT     Pnumber
                 FROM       PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE      Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT     Pno
                 FROM       WORKS_ON, EMPLOYEE
                 WHERE      Essn=Ssn AND Lname='Smith' );
```

  - Can omit `DISTINCT` from this solution. Why?

USING IN (CONT'D.)

- Use tuples of values in comparisons
  - Requires parentheses

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

NESTED 1-COLUMN QUERIES

- Use other comparison operators to compare a single value *v*
  - `= ANY` (or `= SOME`) operator
    - Returns `TRUE` if the value *v* is equal to *some* value in the set *V*
    - Equivalent to `IN`
  - Also available for >, >=, <, <=, and <>
  - `>= ALL` operator
    - Returns `TRUE` if the value *v* is greater than or equal to *every* value in the set *V*
    - Equivalent to `= (SELECT MAX (...) ...)`
    - Also available for =, >, <, <=, and <>

```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    Salary > ALL  ( SELECT   Salary
                         FROM     EMPLOYEE
                         WHERE    Dno=5 );
```

CORRELATED NESTED QUERIES

- **Correlated** nested query
  - Evaluated once for each tuple in the outer query

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

```
Q16:  SELECT   E.Fname, E.Lname
      FROM     EMPLOYEE AS E
      WHERE    E.Ssn IN ( SELECT   Essn
                          FROM     DEPENDENT AS D
                          WHERE    E.Fname=D.Dependent_name
                          AND E.Sex=D.Sex );
```

- Such queries are easiest to understand (and write correctly) if all column names are qualified by their relation names.
- *Note that the inner query can refer to E, but the outer query* cannot *refer to D.*

EXISTS AND UNIQUE FUNCTIONS

- `[NOT] EXISTS` function
  - Check whether result of correlated nested query is empty or not
  - `EXISTS` equivalent to `(SELECT COUNT(*) … ) <> 0`

Customer

custid	name	address	phone

Sale

saleid	date	custid

```
SELECT name, phone
FROM Customer C
WHERE NOT EXISTS ( SELECT *
                   FROM Sale S
                   WHERE C.custid = S.custid);
```

  - Note that columns selected in inner query are irrelevant.

- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if no duplicate tuples in result of query Q

LECTURE SUMMARY

- Complex SQL:
  - Self joins
  - Aggregate functions
  - Grouping
  - Sorting
  - Nested queries
- Relational algebra expressions can handle self joins and nested queries with no additional operators
  - Grouping, aggregations, and sorting require additional operators