CS3402: Chapter 6
SQL: Structured Query Language II

The SELECT-FROM-WHERE Structure of Basic SQL Queries

- SELECT statement
  - The basic statement for retrieving information from a database

SELECT <attribute list>
FROM <table list>
WHERE <condition>;

SELECT <attribute list>
FROM <table list>
[**WHERE** <condition>]
[**ORDER BY** <attribute list>];

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

Specified attributes

Satisfy the conditions →

Title	Year	Length	Type
Star War	1977	124	Color
Mighty Duck	1991	104	Color
Wayne's World	1992	95	Color

One possible database state for the COMPANY relational database schema

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

One possible database state for the COMPANY relational database schema

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Nested Queries

- Nested queries
  - Some queries require that existing values in the database be fetched and then used in a comparison condition.
  - Such queries can be conveniently formulated by using nested queries, which are complete select-from-where blocks within another SQL query.
  - The inner one is called nested query and the outer one is called the outer query.
  - These nested queries can also appear in the WHERE clause or the FROM clause or the SELECT clause or other SQL clauses as needed.

Nested Queries with IN

- **Comparison operator IN**
  - Compares a single value v with a set (or multiset) of values V
  - Evaluates "v IN V" to TRUE if v is one of the elements in V

e.g. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
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' );
```

The project no. of projects that have an **manager** with last name "Smith" →

The project no. of projects that have an **employee** with last name "Smith" →

Nested Queries with IN

- **Comparison operator IN**
  - If a nested query returns a single value, it is permissible to use = instead of IN for the comparison operator.
  - In general, the nested query will return a table (relation), which is a set or multiset of tuples. So to be safe IN is recommended.

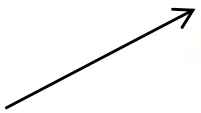
Nested Queries, IN

- **Comparison operator IN**
  - IN can be used to compare tuples of values by placing them within parentheses.
  - v IN V will return true if the tuple v exists in V

e.g. Select the Essn of all employees who work the same (project, hours) as the employee Essn =“123456789”

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

The (Pno, Hours) of the employee with Essn =“123456789”



Nested Queries, SOME/ANY, ALL

- In addition to the IN operator, a number of other comparison operators can be used to compare a single value v to a set or multiset V by combining the keywords ANY/SOME, ALL.

- ANY/SOME
  - = ANY (or = SOME) returns `TRUE` if the value *v* is equal to some value in the set *V* and is hence equivalent to `IN`
  - Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>.
- ALL
  - ALL can also be combined with each of these operators:=, >, >=, <, <=, and <>.
  - E.g. v > ALL V returns TRUE if the value v is greater than all the values in the set V.

Nested Queries, SOME/ANY, ALL

- **ANY/SOME，ALL**

e.g. Select the names of employees whose salary is greater than the salary of all the employees in department 5

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

The salary of Employee tuples with Dno = 5

- **What will be returned if we change the ALL to ANY?**

Correlated Nested Queries

- Correlated nested query
  - Whenever a condition in the WHERE clause of a nested query references some attributes of a relation declared in the outer query.
  - We can understand a correlated query better by considering that the nested query is evaluated once for each tuple in the outer query.

e.g. Retrieve the name of each employee who has a dependent with the same first name and the same sex as the employee.

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

Correlated Nested Queries , EXISTS

- **EXISTS/ NOT EXISTS function**
  - Boolean functions that return TRUE or FALSE, used in a WHERE clause condition.
  - Check whether the result of a nested query is empty or not
  - EXISTS returns TRUE if not empty; NOT EXISTS returns TRUE if empty.
  - Typically used in conjunction with a correlated nested query

e.g. Retrieve the name of each employee who has a dependent with the same first name and the same sex as the employee.

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

Correlated Nested Queries , EXISTS

- **EXISTS/ NOT EXISTS function**

e.g. retrieve the names of employees who have no dependents:

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE NOT EXISTS (SELECT *
                  FROM DEPENDENT
                  WHERE Ssn=Essn);
```

For each EMPLOYEE tuple, the correlated nested query selects all DEPENDENT tuples whose Essn value matches the EMPLOYEE Ssn; if the result is empty, no dependents are related to the employee, so we select that EMPLOYEE tuple and retrieve its Fname and Lname.

Correlated Nested Queries , EXISTS

- **EXISTS/ NOT EXISTS function**

e.g. List the names of managers who have at least one dependent :

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE EXISTS (SELECT *
              FROM DEPENDENT
              WHERE Ssn=Essn)
      AND
      EXISTS (SELECT *
              FROM DEPARTMENT
              WHERE Ssn=Mgr_ssn);
```

The first nested query selects all DEPENDENT tuples related to an EMPLOYEE.
The second selects all DEPARTMENT tuples managed by the EMPLOYEE.
If at least one of the first and at least one of the second exists, we select the EMPLOYEE tuple.

Correlated Nested Queries , EXISTS

- **EXISTS/ NOT EXISTS function**

e.g. Retrieve the name of each employee who works on all the projects controlled by department number 5:

```
SELECT Fname, Lname
FROM EMPLOYEE
WHERE NOT EXISTS ((SELECT Pnumber
                   FROM PROJECT
                   WHERE Dnum = 5)
                   EXCEPT (SELECT Pno
                           FROM WORKS_ON
                           WHERE Ssn = Essn));
```

The first subquery selects all projects controlled by department 5.
The second subquery selects all projects that the particular employee being considered works on.
If the set difference of the first subquery result MINUS (EXCEPT) the second subquery result is empty, it means that the employee works on all the projects and is therefore selected.

Aggregate Functions

- Used to summarize information from multiple tuples into a single-tuple

- Built-in aggregate functions
  - `COUNT`: returns the number of values / tuples in the set or multiset .
  - `SUM`, `MAX`, `MIN`, and `AVG`: applied to a set or multiset of **numeric values** and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values.

- Functions can be used in the `SELECT` clause or in a `HAVING` clause

Aggregate Functions

- Used to summarize information from multiple tuples into a single-tuple

  e.g. Find the sum of the salaries of all employees, the maximum salary, the minimum salary, and the average salary

```
SELECT SUM(Salary), MAX(Salary), MIN(Salary), AVG(Salary)
FROM EMPLOYEE;
```

This query returns a single-row summary of all the rows in the EMPLOYEE table.

SUM(Salary)	MAX(Salary)	MIN(Salary)	AVG(Salary)
281000	55000	25000	35125

Aggregate Functions

- Aggregate Functions combining with where clause
  - Only the tuples satisfying condition will be aggregated.

e.g. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department

```
SELECT SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM EMPLOYEE, DEPARTMENT
WHERE Dno=Dnumber  AND Dname='Research';
```

e.g. Retrieve the number of employees in the 'Research' department .

```
SELECT COUNT (*)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO = DNUMBER AND DNAME = 'Research';
```

Here the asterisk (*) refers to the rows (tuples), so COUNT (*) returns the number of rows in the result of the query.

Aggregate Functions

- COUNT:
  - will not eliminate duplicated values/tuples
  - add keyword “DISTINCT” if we do not want to count duplicated values

    e.g. count the number of distinct salary values in the database

```
SELECT COUNT (DISTINCT Salary)
FROM EMPLOYEE;
```

  - can be used to count tuples with COUNT(*);

Aggregate Functions

- NULL in aggregate functions:
  - In general, NULL values are discarded when aggregate functions are applied to a particular column (attribute); the only exception is for COUNT(*) because tuples instead of values are counted.
  - When an aggregate function is applied to a collection of values, NULLs are removed from the collection before the calculation; if the collection becomes empty because all values are NULL, the aggregate function will return NULL (except in the case of COUNT, where it will return 0 for an empty collection of values).

Aggregate Functions

- Aggregate functions in nested query

e.g. retrieve the names of all employees who have two or more dependents:

```
SELECT Lname, Fname
FROM EMPLOYEE
WHERE (SELECT COUNT(*)
       FROM DEPENDENT
       WHERE Ssn = Essn) >= 2;
```

The correlated nested query counts the number of dependents that each employee has; if this is greater than or equal to two, the employee tuple is selected.

GROUP BY Clauses

- `GROUP BY`
  - Followed by attribute list called the grouping attribute(s).
  - partition the relation into nonoverlapping subsets (or groups) of tuples.
  - Each group (partition) will consist of the tuples that have the same value on the grouping attribute(s).
  - We can then apply aggregate functions to each such group independently to produce summary information about each group.

GROUP BY Clauses

- GROUP BY
  - The grouping attributes should also appear in the SELECT clause, so that the value resulting from applying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

e.g. 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;
```

Each group having the same value for the GROUP BY attribute Dno.

GROUP BY Clauses

- GROUP BY

```
SELECT Dno, COUNT(*), AVG(Salary)
FROM EMPLOYEE
GROUP BY Dno;
```

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

GROUP BY Clauses

- GROUP BY
  - If NULLs exist in the grouping attribute, then a separate group is created for all tuples with a NULL value in the grouping attribute.
  - GROUP BY can be applied after joining two or more relations.
  - WHERE clause will be performed before GROUP BY, that means only tuples satisfying condition will be grouped.

e.g. for each project, retrieve the project number, the 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;
```

HAVING Clauses

- HAVING clause
  - Provides a condition to select or reject an entire group. Only the groups that satisfy the condition are retrieved in the result of the query.
  - appear in conjunction with a GROUP BY clause.

e.g. For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pnumber, Pname
HAVING COUNT(*)>2;
```

HAVING Clauses

- HAVING clause

```
SELECT Pnumber, Pname, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE Pnumber=Pno
GROUP BY Pnumber, Pname
HAVING COUNT(*)>2;
```

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING

Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

HAVING Clauses

- HAVING clause
  - the WHERE clause is executed first, to select individual tuples or joined tuples; the HAVING clause is applied later, to select individual groups of tuples.

e.g. count the total number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.

```
SELECT Dno, COUNT(*)
FROM EMPLOYEE
WHERE Salary>40000
GROUP BY Dno
HAVING COUNT(*)>5;
```

Incorrect!

It will select only departments that have more than five employees who each earn more than $40,000.

HAVING Clauses

- HAVING clause
  - the WHERE clause is executed first, to select individual tuples or joined tuples; the HAVING clause is applied later, to select individual groups of tuples.

e.g. count the total number of employees whose salaries exceed $40,000 in each department, but only for departments where more than five employees work.

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

Correct!

EXPANDED Block Structure 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> ];
```

Summary of SQL Syntax

Table 7.2 Summary of SQL Syntax

```
CREATE TABLE <table name> ( <column name> <column type> [ <attribute constraint> ]
                           { , <column name> <column type> [ <attribute constraint> ] }
                           [ <table constraint> { , <table constraint> } ] )

DROP TABLE <table name>
ALTER TABLE <table name> ADD <column name> <column type>

SELECT [ DISTINCT ] <attribute list>
FROM ( <table name> { <alias> } | <joined table> ) { , ( <table name> { <alias> } | <joined table> ) }
[ WHERE <condition> ]
[ GROUP BY <grouping attributes> [ HAVING <group selection condition> ] ]
[ ORDER BY <column name> [ <order> ] { , <column name> [ <order> ] } ]

<attribute list> ::= ( * | ( <column name> | <function> ( ( [ DISTINCT ] <column name> | * ) ) )
                       { , ( <column name> | <function> ( ( [ DISTINCT] <column name> | * ) ) } ) )

<grouping attributes> ::= <column name> { , <column name> }

<order> ::= ( ASC | DESC )

INSERT INTO <table name> [ ( <column name> { , <column name> } ) ]
( VALUES ( <constant value> , { <constant value> } ) { , ( <constant value> { , <constant value> } ) }
| <select statement> )
```

Summary of SQL Syntax

Table 7.2 Summary of SQL Syntax

DELETE FROM <table name> [WHERE <selection condition>]
UPDATE <table name> SET <column name> = <value expression> { , <column name> = <value expression> } [WHERE <selection condition>]
CREATE [UNIQUE] INDEX <index name> ON <table name> (<column name> [<order>] { , <column name> [<order>] }) [CLUSTER]
DROP INDEX <index name>
CREATE VIEW <view name> [(<column name> { , <column name> })] AS <select statement>
DROP VIEW <view name>

NOTE: The commands for creating and dropping indexes are not part of standard SQL.

References

- 6e
  - Ch. 4. p. 83 – 107
  - Ch. 5, p. 111 – 126

Mid-Term examination

- Date: Oct 19th (Thu)

  Time: 12:30 - 2:30 pm (2 hours)

  Please arrive at least 15 min before the exam starts.

  Venue: YEUNG LT-17 OR YEUNG P4703 OR LI 1106

  Please find your seat and venue from this link:

  http://8.210.38.3/midseat/

  Format: Open-book exam

  Printed version of lecture/tutorial notes, personal notes, and textbook. No electronical devices.

  Exam mode: face-to-face, paper writing

  Coverage: Lecture 1-6