UNIVERSITAS INDONESIA

FAKULTAS ILMU KOMPUTER

Slide 4

SQL: Advanced Query

CSF2600700 - BASIS DATA

SEMESTER GENAP 2017/2018

These slides are a modification to the supplementary slide of "Database System", 7th edition, Elmasri/Navathe, 2015: **Chapter 7 More SQL: Complex Queries, Triggers, Views, and Schema Modification**

Outline

Join SQL

More Complex SQL Retrieval Queries

Specifying Constraints as Assertions and Actions as Triggers

Views (Virtual Tables) in SQL

Schema Change Statements in SQL

Meanings of NULL values

Unknown value

- A person's date of birth is not known

Unavailable

- A person has a home phone but does not want it to be listed

Not applicable attribute

- Passport number

SQL does not distinguish between the different meanings of NULL

Operations on NULL value

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Operations on NULL value

SQL allows queries that check whether an attribute value is NULL

- **IS** or **IS NOT NULL**

SQL uses **IS** or **IS NOT** to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate.

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

Arithmetic Operations

The standard arithmetic operators '+', '-'. '*', and '/' (for addition, subtraction, multiplication, and division, respectively) can be applied to numeric values in an SQL query result

Query 13: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise.

```
Q13: SELECT    FNAME, LNAME, 1.1*SALARY
                AS INCREASED_SAL
     FROM      EMPLOYEE, WORKS_ON, PROJECT
     WHERE     SSN=ESSN AND PNO=PNUMBER AND
               PNAME='ProductX;
```

Arithmetic Operations

Query 14: Retrieve all employees in department 5 whose salary is between $30,000 and $40,000

```
Q14: SELECT   *
     FROM     EMPLOYEE
     WHERE    (SALARY BETWEEN 30000 AND 40000)
              AND DNO=5;

Q14A: SELECT  *
      FROM    EMPLOYEE
      WHERE   (SALARY >= 30000 AND SALARY <=40000)
              AND DNO=5;
```

The EXCEPT Function

Equal to **minus** operation

A except B means set of data in A **without** data that appears in B

```
(SELECT … FROM …. WHERE…. ) EXCEPT
(SELECT … FROM … WHERE …)
```

A

ID
12345
67890
54321
98760

EXCEPT

B

ID
12345
67890

C

ID
54321
98760

Joined Relations Feature in SQL

Can specify a “joined relation” in the FROM-clause

Looks like any other relation but is the result of a join

Allows the user to specify different types of joins (reguler “theta” JOIN, NATURAL JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, CROSS JOIN, etc)

Example CROSS-JOIN

Foods

Name	Cafe
Food 1	XYZ
Food 2	ABC
Food 3	ABC

Likes

Person	Food
Narpati	Food 1
Nizar	Food 1
Danu	Food 3

```
SELECT * FROM Foods
CROSS JOIN Likes
```

Name	Cafe	Person	Food
Food 1	XYZ	Narpati	Food 1
Food 1	XYZ	Nizar	Food 1
Food 1	XYZ	Danu	Food 3
Food 2	ABC	Narpati	Food 1
Food 2	ABC	Nizar	Food 1
Food 2	ABC	Danu	Food 3
Food 3	ABC	Narpati	Food 1
Food 3	ABC	Nizar	Food 1
Food 3	ABC	Danu	Food 3

EXAMPLE – THETA JOIN

Foods

Name	Cafe
Food 1	XYZ
Food 2	ABC
Food 3	ABC

Likes

Person	Food
Narpati	Food 1
Nizar	Food 1
Danu	Food 3

SELECT *
FROM Foods F
JOIN Likes L **ON**
F.name = L.food

Name	Cafe	Person	Food
Food 1	XYZ	Narpati	Food 1
Food 1	XYZ	Nizar	Food 1
Food 3	ABC	Danu	Food 3

EXAMPLE – OUTER JOIN

Foods

Name	Cafe
Food 1	XYZ
Food 2	ABC
Food 3	ABC

Likes

Person	Food
Narpati	Food 1
Nizar	Food 1
Danu	Food 3
Avi	Food 5

```
SELECT * FROM Foods B LEFT OUTER
JOIN Likes L ON B.name = L.Food
```

Name	Cafe	Person	Food
Food 1	XYZ	Narpati	Food 1
Food 1	XYZ	Nizar	Food 1
Food 2	ABC		
Food 3	ABC	Danu	Food 3

```
SELECT * FROM Foods B RIGHT OUTER
JOIN Likes L ON B.name = L.Food
```

Name	Cafe	Person	Food
Food 1	XYZ	Narpati	Food 1
Food 1	XYZ	Nizar	Food 1
Food 3	ABC	Danu	Food 3
		Avi	Food 5

Example – FULL OUTER JOIN

Foods

Name	Cafe
Food 1	XYZ
Food 2	ABC
Food 3	ABC

Likes

Person	Food
Narpati	Food 1
Nizar	Food 1
Danu	Food 3
Avi	Food 5

Name	Cafe	Person	Food
Food 1	XYZ	Narpati	Food 1
Food 1	XYZ	Nizar	Food 1
Food 2	ABC		
Food 3	ABC	Danu	Food 3
		Avi	Food 5

```
SELECT *
FROM Foods B
FULL OUTER JOIN Likes L ON
B.name = L.Food
```

EXAMPLE- NATURAL JOIN

Likes

Person	Food
Narpati	Food 1
Nizar	Food 1
Danu	Food 3
Harith	Food 2

Frequents

Person	Cafe
Avi	ABC
Danu	XYZ
Nizar	ABC
Jack	SB

```
SELECT * FROM Likes
NATURAL JOIN Frequents
```

Person	Food	Cafe
Nizar	Food 1	ABC
Danu	Food 3	XYZ

Nested Queries

Some queries require that existing values in the database be fetched and then used in a comparison condition -> using **nested query**

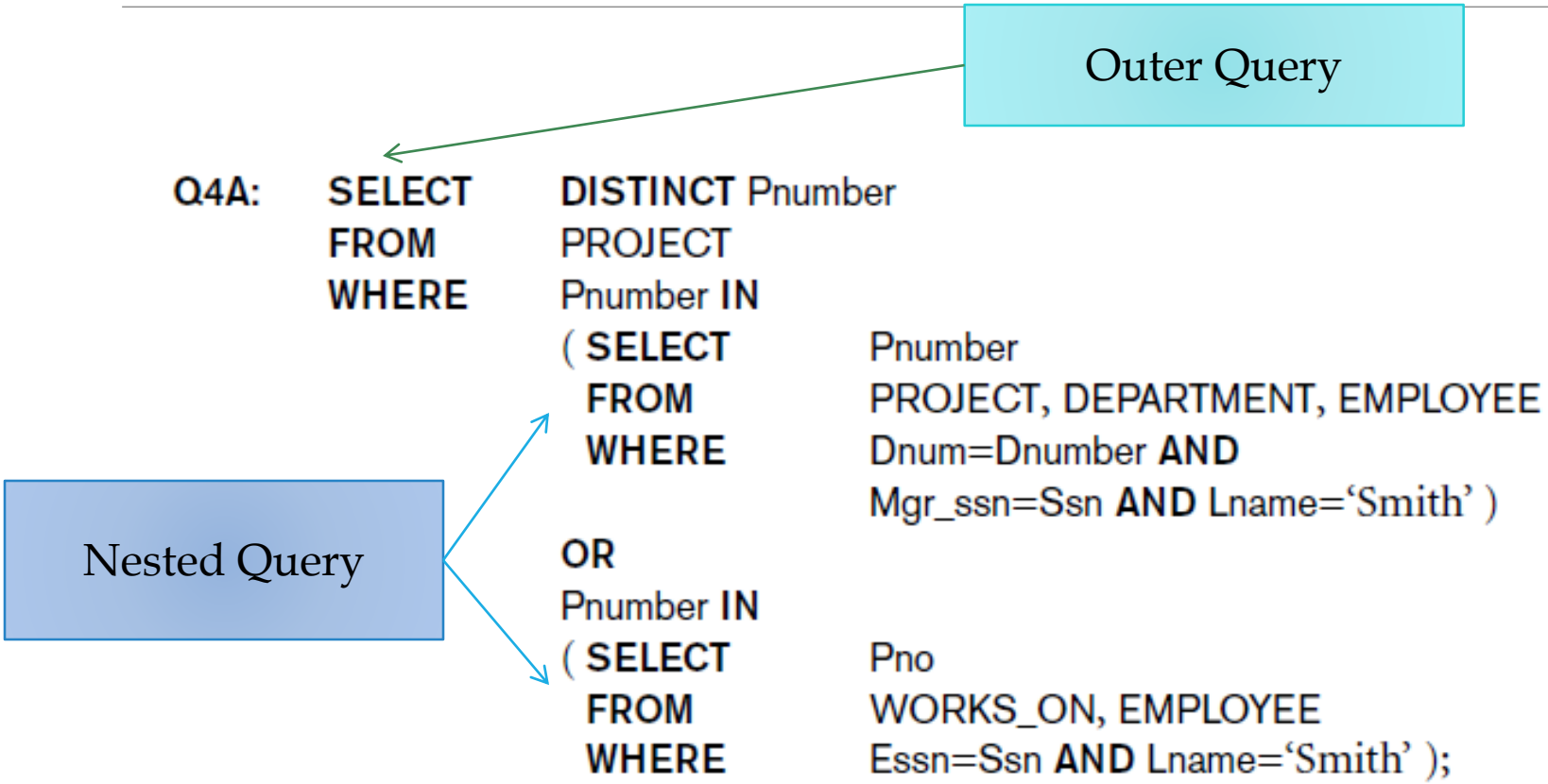
A nested query is a complete SELECT-FROM-WHERE block, within in the WHERE-clause of another query

That other query is called the *outer query*

Comparison operator IN

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

Nested Queries



Nested Queries

Use tuples of values in comparisons

- Place them within parentheses

Query: retrieve the SSN from all employees who work the same (project,hours) combination on same project that employee 'John Smith' (ESSN = '123456789') works on.

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

Nested 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 and is hence **equivalent** to IN
- Other operators that can be combined with ANY (or SOME): >, >=, <, <=, and <>

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

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*

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 );
```

Refer to sex attribute in outer query (EMPLOYEE)

Correlated Nested Queries

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, Q16 may be written as in Q12A

```
Q12A:   SELECT   E.FNAME, E.LNAME
        FROM     EMPLOYEE E, DEPENDENT D
        WHERE    E.SSN=D.ESSN AND
                 E.FNAME=D.DEPENDENT_NAME
                 AND
                 E.SEX = D.SEX
```

The EXISTS Functions

Check whether the result of a correlated nested query is empty (contains no tuples) or not

EXISTS and NOT EXISTS are **usually used in conjunction** with a correlated nested query

The EXISTS Function

Query 12: Retrieve the name of each employee who has a dependent with the same first name and same sex as the employee.

Q12B:

```
SELECT    Fname, Lname
FROM      EMPLOYEE E
WHERE     EXISTS (SELECT * FROM DEPENDENT
          WHERE SSN = ESSN AND
          Fname = DEPENDENT_NAME AND
          E.Sex = Sex);
```

The EXISTS Function

Query 6: Retrieve the names of employees who have no dependents

Q6:

```
SELECT   Fname, Lname
FROM     EMPLOYEE
WHERE    NOT EXISTS (
         SELECT * FROM DEPENDENT
         WHERE SSN = ESSN)
```

The correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist, the EMPLOYEE tuple is selected

The EXISTS Function

Query 7: 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 = MGRSSN);
```

- The first nested query select all DEPENDENT tuples related to an EMPLOYEE
- The second nested query select 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.

Can you rewrite that query using only on a nested query or no nested query?

The EXISTS Function

Query 3: Retrieve the name of each employee who works on **all** the projects controlled by department number 5

Can be used: (S1 CONTAINS S2) that logically equivalent to (S2 EXCEPT S1) is empty.

```
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 select all projects controlled by dept 5
- The second subquery select all projects that particular employee being considered works on.
- If the set difference of the first subquery MINUS (EXCEPT) the second subquery is empty, it means that the employee works on all the projects and is hence selected

EXPLICIT SETS

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

Query 17: Retrieve the social security numbers of all employees who work on project number 1, 2, **or** 3.

Q17:

```
SELECT    DISTINCT ESSN
FROM WORKS_ON
WHERE     PNO IN (1, 2, 3);
```

Q17A:

```
SELECT    DISTINCT ESSN
FROM WORKS_ON
WHERE     PNO = ANY (array[1, 2, 3]);
```

Renaming Attribute

In SQL, its possible to rename attribute that appears in the result of a query by adding the qualifier AS followed by the desired new name

Q8A:

```
SELECT E.Lname AS EMPLOYEE_NAME,
    S.Lname AS SUPERVISOR_NAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE E.SUPERSSN = S.SSN;
```

Aggregate Function

Include **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG**

Query : Find the maximum salary, the minimum salary, and the average salary among all employees.

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

- Some SQL implementations *may not allow more than one function* in the SELECT-clause

AGGREGATE FUNCTION

Query : 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 FUNCTION

Queries : Retrieve the total number of employees in the company (QA), and the number of employees in the 'Research' department (QB).

QA:

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

QB:

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

Grouping

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*

GROUPING

Query 24: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:SELECT      DNO, COUNT (*), AVG (SALARY)
    FROM        EMPLOYEE
    GROUP BY    DNO
```

- In Q24, 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

GROUPING

Query 25: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
Q25: 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)

THE HAVING CLAUSE

Query 26: 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.

```
Q26:    SELECT   PNUMBER, PNAME, COUNT (*)
        FROM     PROJECT, WORKS_ON
        WHERE    PNUMBER=PNO
        GROUP BY     PNUMBER, PNAME
        HAVING       COUNT (*) > 2
```

VIEWS in SQL

A view is a “virtual” table that is derived from other tables

Allows for limited update operations (since the table may not physically be stored)

Allows full query operations

A convenience for expressing certain operations

Specification of Views

SQL command: `CREATE VIEW`

- a table (view) name
- a possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
- a query to specify the table contents

SQL Views: An Example

Specify a different WORKS_ON table

```
CREATE VIEW WORKS_ON_NEW AS
SELECT FNAME, LNAME, PNAME, HOURS
    FROM EMPLOYEE, PROJECT, WORKS_ON
    WHERE SSN=ESSN AND PNO=PNUMBER
```

Using a Virtual Table

We can specify SQL queries on a newly create table (view):

```
SELECT FNAME, LNAME FROM WORKS_ON_NEW
WHERE PNAME='Seena';
```

When no longer needed, a view can be dropped:

```
DROP VIEW WORKS_ON_NEW;
```

Efficient View Implementation

Query modification: present the view query in terms of a query on the underlying base tables

- disadvantage: inefficient for views defined via complex queries (especially if additional queries are to be applied to the view within a short time period)

Efficient View Implementation

View materialization: involves physically creating and keeping a temporary table

- assumption: other queries on the view will follow
- concerns: maintaining correspondence between the base table and the view when the base table is updated
- strategy: incremental update

View Update

Update on a single view without aggregate operations: update may map to an update on the underlying base table

Views involving joins: an update *may* map to an update on the underlying base relations

- not always possible

Un-updatable Views

Views defined using groups and aggregate functions are not updateable

Views defined on multiple tables using joins are generally not updateable

`WITH CHECK OPTION`: must be added to the definition of a view if the view is to be updated

- to allow check for updatability and to plan for an execution strategy

```
CREATE VIEW authors_CA AS ( SELECT * FROM
  Authors WHERE state='CA' ) WITH CHECK
  OPTION
```

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

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

Exercise

1, For each project, list the project name and the total hours per week (by all employees) spent on that project.

2, Retrieve the names of employees who work on every project.

3, Retrieve the names of employees who do not work on any project.

4, For each department, retrieve the department name, and the average salary of employees working in that department.

5, Retrieve the average salary of all female employees.

6, Find the names and addresses of employees who work on at least one project located in Houston but whose department has no location in Houston.

7, List the last names of department managers who have no dependents.