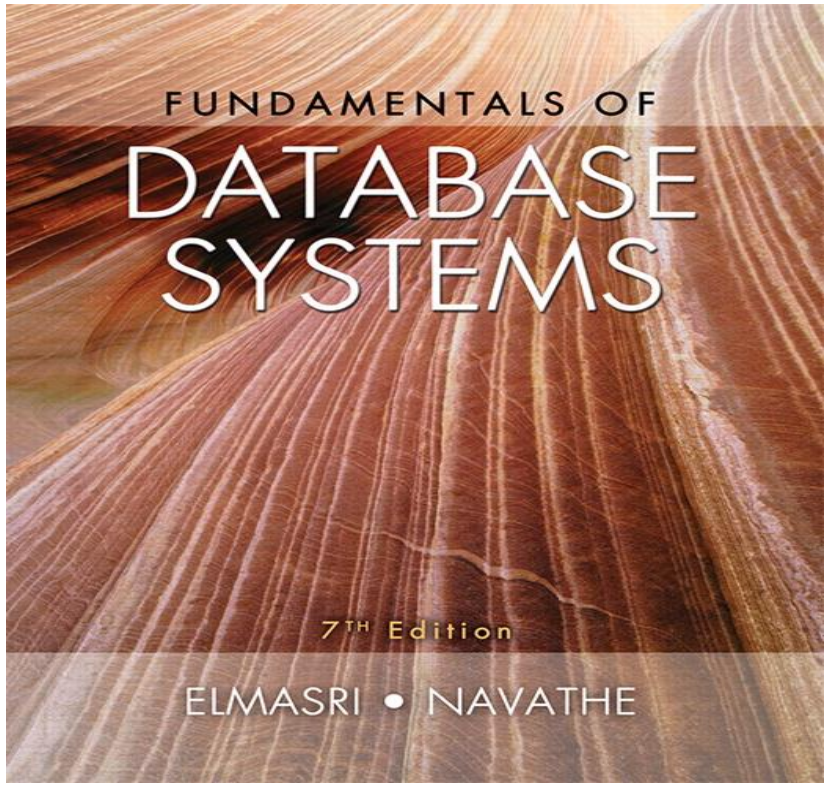
Comp-3150: Database Management Systems

- Ramez Elmasri , Shamkant B. Navathe(2016) Fundamentals of Database Systems (7th Edition), Pearson, isbn 10: 0-13-397077-9; isbn-13:978-0-13-397077-7.

- Chapter 7:
  More SQL: Complex Queries, Triggers, Views, and Schema Modification



Chapter 7: More SQL: Complex Queries, Triggers, Views, and Schema Modification: Outline

- 1. More Complex SQL Retrieval Queries
- 2. Views (Virtual Tables) in SQL
- 3. Schema Modification in SQL

1. More Complex SQL Retrieval Queries

- Additional features allow users to specify more complex retrievals from database:
  - Nested queries, joined tables, and outer query (e.g., in the FROM clause), aggregate functions, and grouping

1.1 Comparisons involving NULL

- A NULL value can stand for unknown, not available, or not applicable
- When a record with NULL in one of its attributes is involved in a comparison operation, result is UNKNOWN (TRUE or FALSE)
- Thus, with this three- valued logic expressions used by SQL, the results of AND, OR, NOT on expressions involving NULL are summarized below and in table 7.1.
  - TRUE AND UNKNOWN= UNKNOWN
    - FALSE AND UNKNOWN= FALSE
    - UNKNOWN AND UNKNOWN= UNKNOWN
    -
    - TRUE OR UNKNOWN= TRUE
    - FALSE OR UNKNOWN= UNKNOWN
    - UNKNOWN OR UNKNOWN= UNKNOWN
    - NOT (UNKNOWN) = UNKNOWN

1.1 Comparisons involving NULL

- Note that the position of the operands in the AND, OR operations can be switched.
- In select- project-join queries, the general rule is that only those combinations of tuples that evaluate the logical expression in the WHERE clause of the query to TRUE are selected.
- however, for aggregate operations such as counting the number of tuples and some other operations, all values may be selected.
- SQL uses the comparison operators IS or IS NOT to check whether an attribute value is NULL. Example, Query 18: Get the names of all employees who do not have supervisors.

1.1 Comparisons Involving NULL and Three-Valued Logic

- SQL allows queries that check whether an attribute value is `NULL`
  - `IS` or `IS NOT NULL`

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

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

1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- **Nested queries are:**
  - Complete select-from-where blocks within WHERE clause of another query called outer query.
  - The nested queries can appear in the WHERE clause or the FROM clause or the SELECT clause or other SQL clause as needed.
- Comparison operator `IN is used with nested queries.`
  - It compares value *v* in the outer where clause with a set (or multiset) of values *V retrieved from the inner query.*
  - And it evaluates to `TRUE` if *v* is one of the elements in *V.*

1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- For example, Query 4 in Q4 can be rephrased to use nested queries as in Q4A

- Query 4: 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.

1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

```
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' );
```

1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- In query Q4A,
- we can replace the IN with = ONLY if the nested query returns a single attribute or tuple. In general, it returns a table.
- We can use tuples (eg., more than one attribute) in the comparison operator WHERE clause of outer query before IN as in the following query that selects the Essns of employees who work the same (project, hours) as employee with Essn '123456789'.

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

1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- Other comparision operators that can be used to compare a single value v (eg: an attribute value) to a set or multiset V (typically a nested query) include = ANY (or = SOME) operator.
- = ANY (or SOME) is equivalent to IN as it returns TRUE if the value v is equal to some value in the set V.
- The operators >,>=, <, <= and < > can be combined with ANY (or SOME).
- The ALL can also be combined with each of these operators.

1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- For example, (v > ALL) returns TRUE if the value v is greater than all the values in the set (or multiset) V.
- The following query returns the names of employees whose salary is greater than the salary of all employees in department 5.

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

1.2 Nested Queries, Tuples, and Set/Multiset Comparisons

- Ambiguities in variable names are best resolved using tuple variable (aliases) to qualify attribute names.

- However, unqualified reference to common attribute names would be taken by SQL as a reference to the innermost nested query attributes.

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 );
```

1.3 Correlated Nested Queries:

- The Two queries are correlated if a condition in the WHERE clause of a nested query references some attributes of a relation declared in the outer query.
- Query 16 is a correlated query.
- In general, 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 re-written as Q16A.

- **Q16A:**

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

1.4 The EXISTS and UNIQUE functions in SQL:

- EXISTS and UNIQUE are Boolean functions that return TRUE or FALSE and can be used as a WHERE clause condition.
- EXISTS checks whether the result of a nested query is empty (has no types) or not.
- Example: is another form of query 16 called Q16B shown below.

- SELECT E.Fname, E.Lname
- FROM Employee E
- WHERE **EXISTS** (SELECT *
- FROM DEPENDENT D
- WHERE E.Ssn= D.Essn
  AND E.sex= D.sex
  AND E.fname= D.Dependent_name);

1.4 The EXISTS and UNIQUE functions in SQL

- `EXISTS` and `NOT EXISTS`
  - Typically used in conjunction with a correlated nested query
- SQL function `UNIQUE(Q)`
  - Returns `TRUE` if there are no duplicate tuples in the result of query Q

- Query 6 : Get the names of employees who have no dependents
- Q6: select Fname, Lname from EMPLOYEE
  where NOT EXISTS
  (SELECT * from DEPENDENT
  where Ssn=Essn);

1.4 The EXISTS and UNIQUE functions in SQL

- Query 7: List the names of managers who have at least one dependent.

- **Q7:**
- SELECT Fname, Lname
- FROM Employee
- WHERE **EXISTS** (SELECT *
- FROM DEPENDENT
- WHERE Ssn= Essn)

- AND **EXISTS** (SELECT *
- FROM Department
- WHERE Ssn= Mgr_Ssn);
-

1.5. Explicit sets and renaming in SQL

- We can also use explicit set of values in the WHERE clause in place of a nested query as in Q17:
- Query 17 : get the social security numbers of all employees who work on project numbers 1,2, or 3.

```
Q17:  SELECT    DISTINCT Essn
      FROM      WORKS_ON
      WHERE     Pno IN (1, 2, 3);
```

1.5. Explicit sets and renaming in SQL

- Use qualifier AS followed by desired new name
  - To Rename any attribute that appears in the result of a query, eg. Q8A

```
Q8A:  SELECT  E.Lname AS Employee_name, S.Lname AS Supervisor_name
      FROM    EMPLOYEE AS E, EMPLOYEE AS S
      WHERE   E.Super_ssn=S.Ssn;
```

- NOTE: 1.6 Explicitly Specifying Joined tables and Outer joins is Omitted & can be done in advanced class or read up if needed by student.

1.7 Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- These 5 Built-in aggregate functions exist.
  - **`COUNT`**, **`SUM`**, **`MAX`**, **`MIN`**, and **`AVG`**
- **Grouping**
  - Is used to create subgroups of tuples before summarizing
- To select entire groups, `HAVING` clause is used
- Aggregate functions can be used in the `SELECT` clause or in a `HAVING` clause

1.7 Aggregate Functions in SQL

- Query 19: find the sum of the salaries of all employees , the maximum salary, the minimum salary and the average salary.
- Following query returns a single row of computed values from EMPLOYEE table:

Q19:

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

- The result can be presented with new names:

Q19A:

```
SELECT SUM (Salary) AS Total_Sal, MAX (Salary) AS
       Highest_Sal, MIN (Salary) AS Lowest_Sal, AVG
       (Salary) AS Average_Sal
FROM  EMPLOYEE;
```

1.7 Aggregate Functions in SQL

- NULL values are discarded when aggregate functions are applied to a particular column

Query 20. 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.

```
Q20:    SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
        FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
        WHERE     Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:    SELECT    COUNT (*)
        FROM      EMPLOYEE;

Q22:    SELECT    COUNT (*)
        FROM      EMPLOYEE, DEPARTMENT
        WHERE     DNO=DNUMBER AND DNAME='Research';
```

1.8 Grouping : The GROUP BY and HAVING CLAUSES

- Application of aggregation functions are mostly to subgroups of tuples in a relation such as:
  - total number of students in each department;
  - average salary of employees in each department.
- We can apply the functions to each group independently to get summary information about each group.
- SQL uses the GROUP BY clause to accomplish this.
- The GROUP BY clause specifies the grouping attributes,
  - which should also appear in the SELECT clause with the aggregate functions
  - Examples follow.

1.8 Grouping : The GROUP BY and HAVING CLAUSES

```
Q24:    SELECT  Dno, COUNT (*), AVG (Salary)
        FROM    EMPLOYEE
        GROUP BY  Dno;
```

- If the grouping attribute has NULL as a possible value, then a separate group is created for the null value (e.g., null Dno in the above query)
- GROUP BY may be applied to the result of a JOIN:

```
Q25:SELECT      Pnumber, Pname, COUNT (*)
    FROM        PROJECT, WORKS_ON
    WHERE       Pnumber=Pno
    GROUP BY    Pnumber, Pname;
```

1.8 Grouping : The GROUP BY and HAVING CLAUSES

- **`HAVING`** clause
  - Provides a condition to select or reject an entire group:
- **Query 26.** 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.

Q26:

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

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> ];
```

Note that Section 7.2 on Creating Assertions and Triggers are left for advanced class or personal reading and not covered in this class.

2. Views (Virtual Tables) in SQL

- A view is a virtual table whose tuples can be derived from physically stored base tables(relations)
- Views are used for queries posed frequently.
  - For example, how many students are registered in a course? or
  - Get names of students registered in a course.
- A view can be defined using CREATE VIEW statement with the following format.

2. Views (Virtual Tables) in SQL

- CREATE VIEW <VIEW NAME>
- AS SELECT<attribute or function list>
- FROM <table list>
- WHERE <condition>
- [GROUP BY <grouping attributes>]
- [HAVING <group condition>]
- [ORDER BY <attributes list>];

2. Views (Virtual Tables) in SQL

- **`CREATE VIEW`** command
  - Give table name, list of attribute names, and a query to specify the contents of the view
  - In V1, attributes retain the names from base tables. In V2, attributes are assigned names

```
V1:   CREATE VIEW   WORKS_ON1
      AS SELECT     Fname, Lname, Pname, Hours
          FROM      EMPLOYEE, PROJECT, WORKS_ON
          WHERE     Ssn=Essn AND Pno=Pnumber;

V2:   CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
      AS SELECT     Dname, COUNT (*), SUM (Salary)
          FROM      DEPARTMENT, EMPLOYEE
          WHERE     Dnumber=Dno
          GROUP BY  Dname;
```

2. Views (Virtual Tables) in SQL

- Once the view has been defined or created as above, then we use select instructions on the view to retrieve all or some information in the view.
- To answer query V1, we can now use:
- QV1: Select Fname, Lname, Pname, Hours From WORKS_ON1;
-
- Views simplify the specification of certain queries as this query does not specify join of the three base tables needed.
- A view is supposed to be always up-to-date and all updates in the tuples of the base relations are automatically reflected in the views when the queries are specified
- the DBMS is responsible for keeping views up to date.
- We can drop a view when it is no longer needed using the DROP VIEW command. We can drop view CV1 with the following statements:
-
- DROP VIEW WORKS_ON1;

3. The Schema Change Statements in SQL

- 1. DROP command
  - Used to drop named schema elements, such as tables, domains, or constraints
- Drop behavior options:
  - CASCADE and RESTRICT
- Example:
  - DROP SCHEMA COMPANY CASCADE;
  - This removes the database schema and all its elements including tables,views, constraints, etc.
- The restrict option would drop the schema only if it has no elements in it, otherwise, the drop command will not be executed.
- Similarly, a base relation within a schema can be dropped with the DROP TABLE command as for example:
  DROP TABLE DEPENDENT CASCADE;
- Or on our system, use:
- DROP TABLE DEPENDENT CASCADE CONSTRAINTS;

3. The Schema Change Statements in SQL:
2. The Alter command

- The ALTER command can be used to change the definition of the base table or other named schema elements.
- For base tables, the possible Alter table actions are for:
  - i) adding or dropping a column(attribute)
  - ii) changing a column definition
  - iii) adding or dropping table contraints
- For example, to add an attribute job to the EMPLOYEE relation of the COMPANY schema in fig 6.1, we use:
-
- ALTER TABLE COMPANY.EMPLOYEE ADD JOB VARCHAR(12);
-

3.2 The Alter command

- With the employee relation having an additional attribute,
  - we need to enter a value for this attribute using the UPDATE command individually on each tuple.
- To drop a column we must choose either:
  - CASCADE or RESRTICT for drop behavior.
- For CASCADE, all constraints and views that reference the column are:
  - dropped automatically from the schema along with the column.
- IF RESTRICT is chosen, the command is successful:
  - only if no views or constraints (or other schema elements) reference the column.

3.2 The Alter command

- To remove the attributes address from the EMPLOYEE table, use;

- ALTER TABLE COMPANY.EMPLOYEE DROP Address CASCADE;
-
- We can drop or change the default of a column with the following statements:
-
- ALTER TABLE COMPANY.DEPARTMENT ALTER Mgr_ssn DROP DEFAULT;

- ALTER TABLE COMPANY.DEPARTMENT ALTER Mgr_ssn SET DEFAULT ‘333445555’;

3.2 The Alter command

- A constraint defined on a table can be dropped or added if it has a name.
- For example, to drop a constraint named EMPSUPERFK from the EMPLOYEE relation, we write
-
- ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT EMPSUPERFK CASCADE;
- We can also redefine a replacement constraint by adding a new constraint using the ADD CONSTRAINT keyword in the ATER TABLE statement.
- Summary of SQL syntax is given next and on Table 7.2

Table 7.2 (continued) 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> )
```

Table 7.2 (continued) 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.