SQL-99: Schema Definition, Constraints, and Queries and Views

Database Management Systems
Handout 7

Relational Languages

- SEQUEL (***S***tructured ***E***nglish ***Que***ry ***L***anguage) later became SQL (***S***tructured ***Q***uery ***L***anguage)
- QBE (***Q***uery ***B***y ***E***xample)
- QUEL (***Que***ry ***L***anguage)

SQL Statements

- Data definition statements

(CREATE TABLE, CREATE VIEWS, CREATE INDEXES, DROP TABLES)

- Data manipulation statements -Change the status of the database

  (INSERT, DELETE, UPDATE)

- Quires- Used to retrieve data from the database

Data Types

- INTEGER
- FLOAT
- DECIMAL(i,j)
- CHAR(n) : Padding with blank
- VARCHAR(n): No padding
- DATE: Made up of year-month-day in the format yyyy-mm-dd
- TIME:Made up of hour:minute:second in the format hh:mm:ss
- TIME(i):Made up of hour:minute:second plus i additional digits specifying fractions of a second
  - format is hh:mm:ss:ii...i
- TIMESTAMP: Data and Time

CREATE TABLE

- Specifies a new base relation by giving it a name, and specifying each of its attributes and their data types
- A constraint NOT NULL may be specified on an attribute

```
CREATE TABLE DEPARTMENT (
       DNAME          VARCHAR(10)          NOT NULL,
       DNUMBER INTEGER                     NOT NULL,
       MGRSSN              CHAR(9),
       MGRSTARTDATE    CHAR(9)
       PRIMARY KEY (DNUMBER),
       UNIQUE (DNAME),
       FOREIGN KEY (MGRSSN) REFERENCES EMP  );
```

DROP TABLE

- Used to remove a relation (base table) and its definition
- The relation can no longer be used in queries, updates, or any other commands since its description no longer exists
- Example:

DROP TABLE DEPENDENT;

ALTER TABLE

- Used to add an attribute to one of the base relations
  - The new attribute will have NULLs in all the tuples of the relation right after the command is executed; hence, the NOT NULL constraint is not allowed for such an attribute
- Example:
  ALTER TABLE EMPLOYEE ADD DOB VARCHAR(12);

- The database users must still enter a value for the new attribute DOB for each EMPLOYEE tuple.
  - This can be done using the UPDATE command.

Retrieval Queries in SQL

- SQL has one basic statement for retrieving information from a database; the **SELECT** statement
  - This is *not the same as* the SELECT operation of the relational algebra
- Important distinction between SQL and the formal relational model:
  - SQL allows a table (relation) to have two or more tuples that are identical in all their attribute values
  - Hence, an SQL relation (table) is a **multi-set** (sometimes called a **bag**) of tuples; it is *not* a set of tuples

Retrieval Queries in SQL (contd.)

- A **bag** or **multi-set** is like a set, but an element may appear more than once.
  - Example: {A, B, C, A} is a bag. {A, B, C} is also a bag that also is a set.
  - Bags also resemble lists, but the order is irrelevant in a bag.
- Example:
  - {A, B, A} = {B, A, A} as bags
  - However, [A, B, A] is not equal to [B, A, A] as lists

Retrieval Queries in SQL

- Basic form of the SQL SELECT statement

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

  - <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

Simple SQL Queries

- Basic SQL queries correspond to using the following operations of the relational algebra:
  - SELECT
  - PROJECT
  - JOIN
- All subsequent examples use the COMPANY database

Simple SQL Queries

- Example of a simple query on one relation
- **Query 0**: Retrieve the birthdate and address of the employee whose name is 'John B. Smith'.

```
SELECT     BDATE, ADDRESS
FROM            EMPLOYEE
WHERE      FNAME='John' AND MINIT='B’
AND        LNAME='Smith’
```

  – Similar to a SELECT-PROJECT pair of relational algebra operations:
    - The SELECT-clause specifies the projection attributes and the WHERE-clause specifies the selection condition

  – However, the result of the query may contain duplicate tuples

Simple SQL Queries

- **Query 1**: Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT    FNAME, LNAME, ADDRESS
FROM      EMPLOYEE, DEPARTMENT
WHERE     DNAME='Research' AND
     DNUMBER=DNO
```

  - Similar to a SELECT-PROJECT-JOIN sequence of relational algebra operations
  - (DNAME='Research') is a selection condition (corresponds to a SELECT operation in relational algebra)
  - (DNUMBER=DNO) is a join condition (corresponds to a JOIN operation in relational algebra)

Simple SQL Queries (contd.)

- **Exercise**: 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.

- **Exercise**: For every project located in 'Stafford', list the controlling department manager's last name, address, and birth date.

Aliases, * and DISTINCT, Empty WHERE-clause

- In SQL, we can use the same name for two (or more) attributes as long as the attributes are in *different relations*
- A query that refers to two or more attributes with the same name must *qualify* the attribute name with the relation name by *prefixing* the relation name to the attribute name
- Example:

  EMPLOYEE.LNAME, **DEPARTMENT.**DNAME

ALIASES

- Some queries need to refer to the same relation twice
  - In this case, *aliases* are given to the relation name
- **Query 8**: For each employee, retrieve the employee's name, and the name of his or her immediate supervisor.

```
SELECT     E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM       EMPLOYEE E S
WHERE      E.SUPERSSN=S.SSN
```

  - In Q8, the alternate relation names E and S are called *aliases* or *tuple variables* for the EMPLOYEE relation
  - We can think of E and S as two different *copies* of EMPLOYEE; E represents employees in role of *supervisees* and S represents employees in role of *supervisors*

ALIASES

- Aliasing can also be used in any SQL query for convenience
- Can also use the AS keyword to specify aliases

```
Q8:    SELECT    E.FNAME, E.LNAME,
                 S.FNAME, S.LNAME
       FROM      EMPLOYEE AS E,
                 EMPLOYEE AS S
       WHERE     E.SUPERSSN=S.SSN
```

UNSPECIFIED WHERE-clause

- A *missing WHERE-clause* indicates no condition; hence, all tuples of the relations in the FROM-clause are selected
  - This is equivalent to the condition WHERE TRUE
- **Query 9**: Retrieve the SSN values for all employees.

```
SELECT      SSN
FROM        EMPLOYEE
```

- If more than one relation is specified in the FROM-clause *and* there is no join condition, then the *CARTESIAN PRODUCT* of tuples is selected

UNSPECIFIED WHERE-clause

X

A	B
a1	b1
a2	b2
a3	b3

Y

C
c1
c2

```
SELECT      A, C
FROM     X, Y
```

Result

A	C
a1	c1
a1	c2
a2	c1
a2	c2
a3	c1
a3	c2

USE OF *

- To retrieve all the attribute values of the selected tuples, a * is used, which stands for *all the attributes* Examples:

```
SELECT      *
FROM        EMPLOYEE
WHERE       DNO=5

SELECT      *
FROM        EMPLOYEE, DEPARTMENT
WHERE       DNAME='Research' AND
            DNO=DNUMBER
```

USE OF DISTINCT

- SQL does not treat a relation as a set; duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword **DISTINCT** is used

```
SELECT      SALARY
FROM        EMPLOYEE
```

```
SELECT      DISTINCT SALARY
FROM        EMPLOYEE
```

SET OPERATIONS

- SQL has directly incorporated some set operations
- There is a union operation (UNION), and in *some versions* of SQL there are set difference (MINUS) and intersection (INTERSECT) operations
- The resulting relations of these set operations are sets of tuples; *duplicate tuples are eliminated from the result*
- The set operations apply only to *union compatible relations*; the two relations must have the same attributes and the attributes must appear in the same order

SET OPERATIONS-Exercise

- Make a list of all project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.
- Identify employees who have worked in a project in ‘Stanford’ and ‘London’.
- Identify employees who have worked in no projects.

NESTING OF QUERIES

- A complete SELECT query, called a *nested query*, can be specified within the WHERE-clause of another query, called the *outer query*
  - Many of the previous queries can be specified in an alternative form using nesting
- Query 1: Retrieve the name and address of all employees who work for the 'Research' department.

```
SELECT     FNAME, LNAME, ADDRESS
FROM       EMPLOYEE
WHERE      DNO IN
           (SELECT  DNUMBER
           FROM  DEPARTMENT
           WHERE DNAME='Research' )
```

Inner query

Outer query

IN Operator

- IN operator can also compare a tuple of values in parenthesis with a set of union compatible tuples.

(name, age, address, dob)   (-,-,-,-,-)
(-,-,-,-,-)
(-,-,-,-,-)

Other comparison operators:

=ANY, >ANY, >=ANY, <ANY, <=ANY, < >ANY

=ALL, > ALL, >= ALL, < ALL, <= ALL, < > ALL

v* > ALL V** (returns true if ***v is greater than all values in **V**)

Exercise: Retrieve the names of employees whose salary is greater than the salary of all the employees in department 2

NESTING OF QUERIES

- The nested query selects the number of the 'Research' department
- The outer query select an EMPLOYEE tuple if its DNO value is in the result of either nested query
- The comparison operator IN compares a value v with a set (or multi-set) of values V, and evaluates to TRUE if v is one of the elements in V
- In general, we can have several levels of nested queries
- In this example, the nested query is *not correlated* with the outer query

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

CORRELATED NESTED QUERIES

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

```
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)
```

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, Q12 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
```

THE EXISTS FUNCTION

- EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.

- Exists(Q): Return true if there is at least one tuple in the query Q, otherwise false.
- Not Exists(Q): Return true if there no tuples in the query Q, otherwise false.

We can formulate Query 12 in an alternative form that uses EXISTS as Q12B

THE EXISTS FUNCTION

- Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.
  Q12B:

```
SELECT  FNAME, LNAME
FROM    EMPLOYEE
WHERE   EXISTS  (SELECT        *
                 FROM   DEPENDENT
                 WHERE  SSN=ESSN  AND
                 FNAME=DEPENDENT_NAME)
```

Exercises

- Retrieve the names of employees who have no dependents.
- List the names of managers who have at least one dependent.

EXPLICIT SETS

- It is also possible to use an **explicit (enumerated) set of values** in the WHERE-clause rather than a nested query
- Query 13: 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)
```

NULLS IN SQL QUERIES

- SQL allows queries that check if a value is **NULL** (missing or undefined or not applicable)
- 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 14**: Retrieve the names of all employees who do not have supervisors.

```
SELECT     FNAME, LNAME
FROM       EMPLOYEE
WHERE      SUPERSSN  IS  NULL
```

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

AGGREGATE FUNCTIONS

- Include **COUNT, SUM, MAX, MIN, and AVG**
- **Query 15:** Find the maximum salary, the minimum salary, and the average salary among all employees.

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

AGGREGATE FUNCTIONS

- **Query 16**: 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

- **Queries 17** and **18**: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:    SELECT   COUNT (*)
        FROM     EMPLOYEE


Q18:    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 20**: 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
```

  - In Q20, 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

Exercise

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

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)

Query 22: 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.

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

ORDER BY

- The **ORDER BY** clause is used to sort the tuples in a query result based on the values of some attribute(s)
- Query 28: 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
```

ORDER BY

- The default order is in ascending order of values
- We can specify the keyword **DESC** if we want a descending order; the keyword **ASC** can be used to explicitly specify ascending order, even though it is the default

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

Specifying Updates in SQL

- There are three SQL commands to modify the database: **INSERT**, **DELETE**, and **UPDATE**

- In its simplest form, it is used to add one or more tuples to a relation
- Attribute values should be listed in the same order as the attributes were specified in the **CREATE TABLE** command

INSERT

- Example:

```
INSERT INTO   EMPLOYEE
      VALUES ('Richard','K','Marini', '653298653', '30-
DEC-52',  '98 Oak Forest,Katy,TX', 'M',
37000,'987654321', 4 )
```

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
  - Attributes with NULL values can be left out
- Example: Insert a tuple for a new EMPLOYEE for whom we only know the FNAME, LNAME, and SSN attributes.

```
 INSERT INTO  EMPLOYEE (FNAME, LNAME,  SSN)
        VALUES ('Richard', 'Marini', '653298653')
```

INSERT

Another variation of INSERT allows insertion of *multiple tuples* resulting from a query into a relation

- Example: Suppose we want to create a temporary table that has the name, number of employees, and total salaries for each department.
  - A table DEPTS_INFO is created by U3A, and is loaded with the summary information retrieved from the database by the query in U3B.

```
U3A:        CREATE TABLE  DEPTS_INFO
                   (DEPT_NAME          VARCHAR(10),
                    NO_OF_EMPS         INTEGER,
                    TOTAL_SAL          INTEGER);


U3B:        INSERT INTO    DEPTS_INFO (DEPT_NAME,
                           NO_OF_EMPS, TOTAL_SAL)
            SELECT         DNAME, COUNT (*), SUM (SALARY)
            FROM           DEPARTMENT, EMPLOYEE
            WHEREDNUMBER=DNO
            GROUP BY       DNAME ;
```

DELETE

- Removes tuples from a relation
  - Includes a WHERE-clause to select the tuples to be deleted
  - Referential integrity should be enforced
  - Tuples are deleted from only *one table* at a time (unless CASCADE is specified on a referential integrity constraint)
  - A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
  - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause

DELETE

- Examples:

```
U4A:    DELETE FROM     EMPLOYEE
        WHERE           LNAME='Brown’

U4B:    DELETE FROM     EMPLOYEE
        WHERE           SSN='123456789’

U4C:    DELETE FROM     EMPLOYEE
        WHERE           DNO  IN
        (SELECT     DNUMBER
        FROM        DEPARTMENT
        WHERE       DNAME='Research')

U4D:    DELETE FROM     EMPLOYEE
```

UPDATE

- Used to modify attribute values of one or more selected tuples
- A WHERE-clause selects the tuples to be modified
- An additional SET-clause specifies the attributes to be modified and their new values
- Each command modifies tuples *in the same relation*
- Referential integrity should be enforced

UPDATE

- Example: Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively.

```
U5:     UPDATE    PROJECT
        SET       PLOCATION = 'Bellaire',
                          DNUM = 5
        WHERE     PNUMBER=10
```

UPDATE

- Example: Give all employees in the 'Research' department a 10% raise in salary.

```
U6:       UPDATE    EMPLOYEE
   SET          SALARY = SALARY *1.1
   WHERE        DNO  IN (SELECT       DNUMBER
                      FROM     DEPARTMENT
                      WHERE    DNAME='Research')
```

- In this request, the modified SALARY value depends on the original SALARY value in each tuple
  - The reference to the SALARY attribute on the right of = refers to the old SALARY value before modification
  - The reference to the SALARY attribute on the left of = refers to the new SALARY value after modification