CLASS ROOM LOG -7

E. LAXMI NIHARIKA
IMT2014017

Topics covered

- Aliasing
- Tables as sets
  - Set operators (UNION, INTERSECT, EXCEPT)
- UNION ALL, INTERSECT ALL, EXCEPT ALL

- Substring Pattern matching Operator
  - LIKE operator
  - _ operator
- Arithmetic operators
  - + , - , *, /
- BETWEEN, ORDER BY
- Nested queries
  - IN Operator
  - ALL Operator

- IS NULL Vs IS NOT NULL; EXISTS Vs NOT EXISTS
- JOIN
  - LEFT OUTER JOIN, RIGHT OUTER JOIN, NATURAL JOIN
- AGGREGATE FUNCTIONS
  - SUM, MIN, MAX, AVG, COUNT
- GROUP BY, HAVING clauses
- INSERT INTO, DELETE, UPDATE

Aliasing

- It is used when we want to have multiple references to the same relation/column within a query

Aliasing Syntax

SQL Alias Syntax for Columns

```
SELECT column_name AS alias_name
FROM table_name
WHERE [condition];
```

SQL Alias Syntax for Tables

```
SELECT column1, column2....
FROM table_name AS alias_name
WHERE [condition];
```

Consider the relation EMPLOYEE

EMPLOYEE(FNAME, LNAME, SSN, SUPER_SSN, DNO)

SUPER_SSN is FK of SSN

For each employee, retrieve the employee's last name and last name of his or her immediate supervisor

```
SELECT  E.LNAME, S.LNAME

FROM EMPLOYEE AS E, EMPLOYEE AS S

WHERE E.SUPER_SSN=S.SSN;
```

Here E,S are called Aliases

Why SQL considers tables as Multisets?

- Duplicate elimination is expensive
- User may want to see the duplicates as the result of the query
- When an aggregate function is applied to tuples, in most cases we do not want to eliminate duplicates

Tables as sets in SQL

- SQL treats tables(relations) as Multisets
- **DISTINCT :** Used to eliminate duplicates in a relation

Syntax for DISTINCT

```
SELECT DISTINCT column_name[s]
FROM table_name
WHERE [condition]
```

Consider the relation PRODUCT

PRODUCT(PNAME,PRICE, CATEORY,PRODUCT-ID, MANUFACTURER)

Select the distinct categories from PRODUCT relation

```
SELECT  DISTINCT CATEGORY

FROM    PRODUCT
```

Set operators

- UNION
- INTERSECT
- EXCEPT

These all operators remove duplicates after the query is performed.

UNION

Syntax :

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition];
```

Example for UNION

```
SELECT A
FROM R
UNION
SELECT A
FROM S
```

R UNION S

A
1
2
3
4
5

S

A	B
5	a1
4	a2
3	a3

R

A
1
2

INTERSECT

Syntax :

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

INTERSECT

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

Example for INTERSECT

```
SELECT A
FROM R
INTERSECT
SELECT A
FROM S
```

R INTERSECT S

a1

S

A	B
a1	1
a4	2
a3	3

R

A
a1
a2

EXCEPT

The SQL **EXCEPT** clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.

Syntax :

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

EXCEPT

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Example for EXCEPT

```
SELECT A
FROM R
EXCEPT
SELECT A
FROM S
```

R EXCEPT S

A
2

S

A	B
1	a1
4	a2
3	a3

R

A
1
2

UNION ALL, INTERSECT ALL, EXCEPT ALL

- They don't remove duplicates after these operations are performed

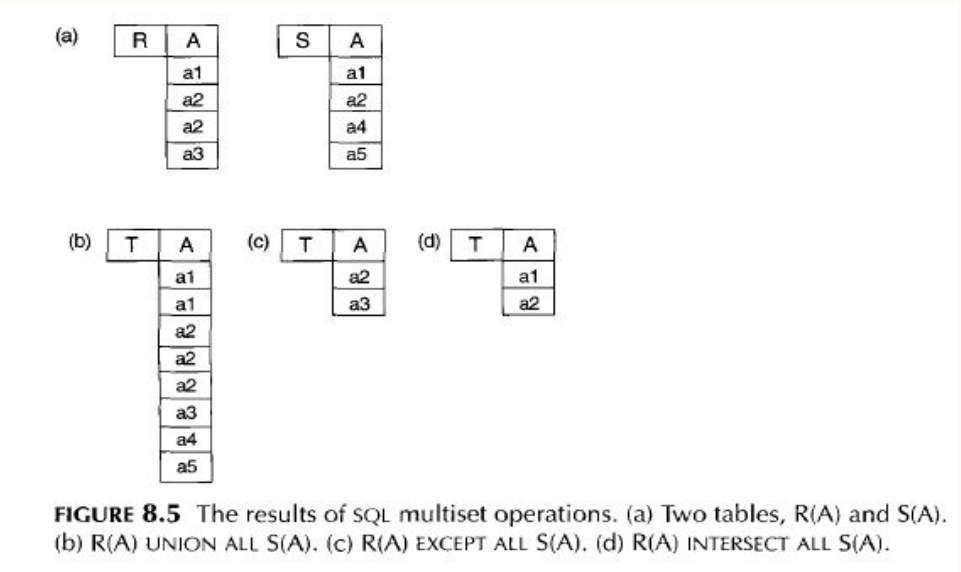

FIGURE 8.5 The results of SQL multiset operations. (a) Two tables, R(A) and S(A). (b) R(A) UNION ALL S(A). (c) R(A) EXCEPT ALL S(A). (d) R(A) INTERSECT ALL S(A).

LIKE Operator

- It is used as a Substring matching operator

Syntax:

```
SELECT column_name(s)

FROM table_name

WHERE column_name LIKE pattern;
```

- Consider the relation PRODUCT

  PRODUCT (PID,PNAME, PRICE, CATEGORY, MANUFACTURER)

Retrieve all PRODUCT details whose CATEGORY contains “ELEC”

```
SELECT *

FROM PRODUCT

WHERE CATEGORY LIKE '%ELEC%';
```

_ Wild card

- ‘_’ replaces single character
- Consider relation

Customers(CustID, CustName, Address, City, PostalCode, Country)

Select all customers with a City starting with any character, followed by "erlin

```
SELECT *
FROM Customers
WHERE City LIKE '_erlin';
```

ARITHMETIC OPERATORS

- ADDITION(+)
- SUBTRACTION(-)
- MULTIPLICATION(*)
- DIVISION(/)

- These operators can be applied on numeric values or attributes with numeric domains

Consider relation

PRODUCT (PNAME, PRICE, PID, CATEGORY, MANUFACTURER)

Display price of all the products if each product price is given a 10 percent hike.

```
SELECT 1.1*PRICE AS
INCREASED_PRICE

FROM PRODUCT
```

BETWEEN

- The **BETWEEN** operator selects values within a range. The values can be numbers, dates etc...

Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1
AND value2;
```

Consider relation

EMPLOYEE(ENAME, SALARY, DNO, SSN)

Retrieve all employees details in department 5 whose salary is between $30,000 and $40,000.

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

ORDER BY

- It is used to sort the result-set by one or more columns.

Syntax

```
SELECT column-list
FROM table_name
[WHERE condition]
[ORDER BY column1, column2, .. columnN] [ASC | DESC];
```

- **Order of selection :**

FROM -> Where -> SELECT -> ORDER BY

Consider relation

PRODUCT (PNAME, PID, PRICE, CATEGORY, MANUFACTURER)

Retrieve product-name, price, manufacturer details of PRODUCT relation ordered by price, name of the product

```
SELECT  PNAME, PRICE, MANUFACTURER
FROM    PRODUCT
ORDER BY  PRICE, PNAME
```

NESTED QUERIES

- Dynamically created tables from SELECT queries can be used within other SELECT queries:

- In (v **IN** V), Operator **IN**, compares a value v with a set (or multiset) of values V and evaluates to TRUE if v is one of the elements in V

- (**v > ALL V**) returns TRUE if the value v is greater than all the values in the set (or multiset) V.

NESTED QUERIES USING IN

CONSIDER THE FOLLOWING RELATIONS

Company(name, city)

Product(pname, maker)

Purchase(id, product, buyer)

Where **Purchase.product is FK to Product.pname** amd

Product.maker is FK to Company.name

Find city names of the companies that manufacture products bought by Jay

```
SELECT Company.city
FROM   Company
WHERE  Company.name IN
        ( SELECT Product.maker
          FROM   Purchase, Product
          WHERE
Product.pname=Purchase.product AND Purchase
.buyer =  'Jay');
```

EXAMPLE OF NESTED QUERIES USING ALL

Consider relation **EMPLOYEE**(ENAME, SALARY, DNO, SSN) and find the names of employees whose salary is greater than the salary of all the employees in department 5

```
SELECT E.ENAME
FROM EMPLOYEE AS E
WHERE E.SALARY> ALL (
SELECT  S.SALARY
FROM EMPLOYEE AS S
WHERE  S.DNO=5);
```

Correlated Nested Queries

- Whenever a condition in the WHERE clause of a nested query references some attribute of a relation declared in the outer query, the two queries are said to be correlated

EXISTS, NOT EXISTS, NULL, NOT NULL

- EXISTS is true if result of the select operation is not empty
- EXISTS is false if result of the selection operation is empty
- NOT EXISTS is true if result of the selection operation is empty
- NOT EXISTS is false if result of the selection operation is not empty
- NOT NULL is true when result of the selection operation is not null
- NOT NULL is false when result of the selection operation is null
- NULL is true when result of the selection operation is null
- NULL is false when result of the selection operation is not null

Example of correlated Nested queries

1. Consider relation **EMPLOYEE**(ENAME, SALARY, DNO, SSN),

DEPARTMENT(DEPT_ID, DNAME, MANAGED-BY, LOCATIONS)

Write a query to find the departments which do not have employees at all?

```
SELECT DEPT_ID,DNAME
FROM DEPARTMENT AS D
WHERE NOT EXISTS(
SELECT  *
FROM EMPLOYEES AS E
WHERE E.DNO = D.DEPT_ID)
```

EXISTS

1) **Consider relations customers(customer_id, name, favourite-website), orders(order_id, customer_id, order_date).** Find all of the records from the customers table where there is at least one record in the orders table with the same customer_id.

```
SELECT *
FROM customers
WHERE EXISTS
  (SELECT *
   FROM orders
   WHERE customers.customer_id = orders.customer_id);
```

JOIN

- It performs join operation in the **FROM** clause of the query and outputs join table

Example: Consider relations

EMPLOYEE(ENAME, SALARY, ADDRESS, DNO, SSN),

DEPARTMENT(DNUMBER, DNAME, MANAGED-BY, LOCATIONS)

Here DNO is FK to DNUMBER

Retrieve the name and address of every employee who works for the 'Research' department

```
SELECT NAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT
ON DNO=DNUMBER)
WHERE DNAME='Research';
```

NATURAL JOIN

CONSIDER RELATIONS

foods(ITEM_ID,ITEM_NAME,
ITEM_UNIT, COMPANY_ID)

company(COMPANY_ID,
COMPANY_NAME, COMPANY_CITY)

NATURAL JOIN ON foods,company

```
SELECT *
FROM  foods  NATURAL JOIN
company;
```

LEFT OUTER JOIN

It returns all rows from the left table, even if there are no matches in the right table

Syntax

```
SELECT column_name(s)

FROM table1 LEFT OUTER JOIN table2 ON
table1.column_name=table2.column_name;
```

RIGHT OUTER JOIN

It returns all rows from the right table, even if there are no matches in the left table

Syntax

```
SELECT column_name(s)

FROM table1 RIGHT OUTER JOIN table2 ON
table1.column_name=table2.column_name;
```

Example: Employee ⟕ Department

```
SELECT *

FROM Employee RIGHT OUTER

JOIN Department ON EmployeeID= Manager
```

INPUT RELATIONS

Employee

Employee ID	Name	Department
E1	A	D1
E2	B	D2
E3	C	D1

Department

Dept ID	Dept Name	Manager
D1	X	E3
D2	Y	E2
D3	Z	null

AGGREGATE OPERATORS

- COUNT, SUM, MAX, MIN, AVG
- COUNT returns the number of tuples or values as specified in a query.
- The functions SUM, MAX, MIN, and AVG are applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values.

For example consider relation

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY

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;
```

Grouping: GROUP BY and HAVING Clauses

- **GROUP BY :** When we want aggregate functions to be applied to groups.
  - So, to mention grouping attributes we use GROUP BY

- **HAVING** : Used when we want to put condition on aggregation operation

GROUP BY and HAVING Clauses

Consider relation

EMPLOYEE(ENAME, SALARY, DNO, SSN) and compute each department, count the number of employees and their average salaries

```
SELECT DNO, COUNT(*), AVG(SALARY)

FROM  EMPLOYEE GROUP BY   DNO;
```

Consider relation

EMPLOYEE(ENAME, SALARY, DNO, SSN) and return all departments having more than twenty employees, and show the number of employees and their average salary.

```
SELECT  DNO, COUNT(*), AVG(SALARY)
FROM  EMPLOYEE
GROUP BY  DNO
HAVING  COUNT(*) > 20;
```

INSERT INTO, DELETE, AND UPDATE

```
CREATE TABLE EMPLOYEE
        ( FNAME                 VARCHAR(15)             NOT NULL ,
          MINIT                 CHAR ,
          LNAME                 VARCHAR(15)             NOT NULL ,
          SSN                   CHAR(9)                 NOT NULL ,
          BDATE                 DATE,
          ADDRESS               VARCHAR(30) ,
          SEX                   CHAR ,
          SALARY                DECIMAL(10,2) ,
          SUPERSSN              CHAR(9) ,
          DNO                   INT                     NOT NULL ,
    PRIMARY KEY (SSN) ,
    FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
    FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE DEPARTMENT
        ( DNAME                 VARCHAR(15)             NOT NULL ,
          DNUMBER               INT                     NOT NULL ,
          MGRSSN                CHAR(9)                 NOT NULL ,
          MGRSTARTDATE          DATE ,
        PRIMARY KEY (DNUMBER) ,
        UNIQUE (DNAME) ,
        FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN) ) ;
CREATE TABLE DEPT_LOCATIONS
        ( DNUMBER               INT                     NOT NULL ,
          DLOCATION             VARCHAR(15)             NOT NULL ,
        PRIMARY KEY (DNUMBER, DLOCATION) ,
        FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE PROJECT
        ( PNAME                 VARCHAR(15)             NOT NULL ,
          PNUMBER               INT                     NOT NULL ,
          PLOCATION             VARCHAR(15) ,
          DNUM                  INT                     NOT NULL ,
        PRIMARY KEY (PNUMBER) ,
        UNIQUE (PNAME) ,
        FOREIGN KEY (DNUM) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE WORKS_ON
        ( ESSN                  CHAR(9)                 NOT NULL ,
          PNO                   INT                     NOT NULL ,
          HOURS                 DECIMAL(3,1)            NOT NULL ,
        PRIMARY KEY (ESSN, PNO) ,
        FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ,
        FOREIGN KEY (PNO) REFERENCES PROJECT(PNUMBER) ) ;
CREATE TABLE DEPENDENT
        ( ESSN                  CHAR(9)                 NOT NULL ,
          DEPENDENT_NAME        VARCHAR(15)             NOT NULL ,
          SEX                   CHAR ,
          BDATE                 DATE ,
          RELATIONSHIP          VARCHAR(8) ,
        PRIMARY KEY (ESSN, DEPENDENT_NAME) ,
        FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ) ;
```

INSERT INTO

- By using Insert data tuples can be added to tables
- Constraints that can be violated
  - Domain constraint
  - Key constraint
  - Referential integrity constraint
  - Entity integrity constraint
- Insertion fails if any of the above constraints is violated

1) Insert a tuple into EMPLOYEE table

```
INSERT INTO   EMPLOYEE
VALUES  ('1002', 002, 'Bharath Kumar', 'M',
'9-5-1973', 'Rajajinagar Bangalore 10',
300000, 007, 5);
```

2) It allows the user to specify explicit attribute names that correspond to the values provided in the INSERT command. Remaining attribute values are set to their default/NULL

```
INSERT INTO EMPLOYEE (FNAME,
LNAME, DNO, SSN)
VALUES ('Richard', 'Marini', 4, '653298653');
```

DELETE

- The DELETE command removes tuples from a relation. It includes a WHERE to select the tuples to be deleted

- Referential Integrity may get violated
  -delete operation is rejected

- **In table2 put this constraint**

foreign-key attr-name references Table1name on delete{cascade/restrict}

```
DELETE FROM EMPLOYEE

WHERE LNAME = 'BROWN'
```

UPDATE

- By using Insert data tuples can be added to tables
- Constraints that can be violated
  - Domain constraint
  - Key constraint
  - Referential integrity constraint
  - Entity integrity constraint
- Updation fails if any of the above constraints is violated

Change the location and controlling department number of project number 10 to 'BANGALORE' and 5, respectively

```
UPDATE  PROJECT
SET PLOCATION = 'BANGALORE', DNUM = 5
WHERE PNUMBER=10;
```

REFERENCES

1. SQL Tutorial | W3Schools. (n.d.). from http://www.w3schools.com/SQl/default.asp
2. SQL Tutorial – TutorialsPoint. (n.d.). Retrieved from http://www.tutorialspoint.com/sql/
3. Elmasri, R., & Navathe, S. B. (2000). Fundamentals of database systems. Reading: Wesley.

Thank you