Chapter 7: More SQL

Database Systems CS203

Week 05

24th-26th Sep-2018

By: Anam Qureshi

Outline

- Variations of Insert Operation
- Importance of Foreign Key Constraint
- Three-value logic
- Nested Queries
- Aggregate Functions
- Grouping
- Case Statements

Insert Operation

```
INSERT INTO     EMPLOYEE
VALUES          ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );

INSERT INTO     EMPLOYEE (Fname, Lname, Dno, Ssn)
VALUES          ('Richard', 'Marini', 4, '653298653');

INSERT INTO     EMPLOYEE (Fname, Lname, Ssn, Dno)
VALUES          ('Robert', 'Hatcher', '980760540', 2);

INSERT INTO     EMPLOYEE (Fname, Lname, Dno)
VALUES          ('Robert', 'Hatcher', 5);
```

Insert Operation

```
CREATE TABLE    WORKS_ON_INFO
( Emp_name      VARCHAR(15),
  Proj_name     VARCHAR(15),
  Hours_per_week  DECIMAL(3,1) );

INSERT INTO     WORKS_ON_INFO ( Emp_name, Proj_name,
                Hours_per_week )
SELECT          E.Lname, P.Pname, W.Hours
FROM            PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE           P.Pnumber = W.Pno AND W.Essn = E.Ssn;
```

```
CREATE TABLE    D5EMPS LIKE EMPLOYEE
(SELECT         E.*
FROM            EMPLOYEE AS E
WHERE           E.Dno = 5) WITH DATA;
```

Reading and Practice Assignment

- 6.4.2 &6.4.3
- Solve Review Questions
- Solve Exercise Questions

Importance of a Foreign Key

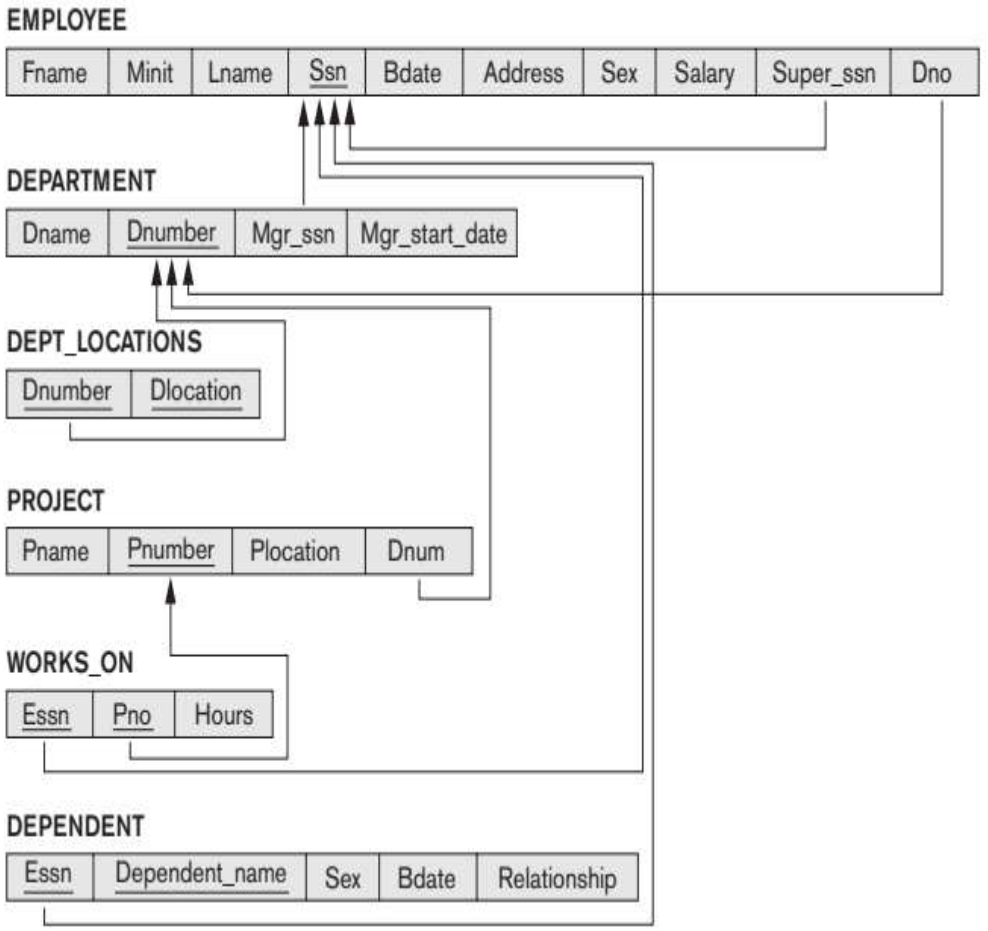
- Within Same Relation/Table

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

Importance of a Foreign Key

- In different Relations/Tables
  - Foreign key maintains referential integrity
  - One relation may contain single or multiple foreign keys
  - Used to create relationship among tables so hat join can be applied
  - To avoid referential integrity constraints violation, Use foreign key constraints as modification part (i.e., Alter table..)



NULL Constraint

1. **Unknown value.** A person's date of birth is not known, so it is represented by NULL in the database. An example of the other case of unknown would be NULL for a person's home phone because it is not known whether or not the person has a home phone.
2. **Unavailable or withheld value.** A person has a home phone but does not want it to be listed, so it is withheld and represented as NULL in the database.
3. **Not applicable attribute.** An attribute LastCollegeDegree would be NULL for a person who has no college degrees because it does not apply to that person.

Three-Value Boolean Logic

- SQL uses comparison operators IS or IS NOT rather than = or <>.
- Because SQL considers each NULL value distinct, hence equality comparison is not appropriate.
- In case of join, tuples with NULL values for the join attributes are not included in the result (unless it is an OUTER JOIN)

Table 7.1 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

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

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

Nested Queries OR Sub Queries

Nested Queries

- **Sub queries**
  - •Execution process: First Inner queries are executed then their values are matched using comparison operators (=,IN,NOT IN, SOME, ALL, and others) with outer queries.
  - •The result of an inner query can be scalar, row, or multiple rows.
  - •Equal '=' comparison operator is only used when output of an inner query is scalar.
  - •Attributes of sub queries can not be used in Select Statement of outer queries.
  - • Sub queries can be nested inside select, where, update, insert and delete statements

Nested Queries

- **Correlated Sub queries**
  - In correlated sub query the nested query is executed once for each tuple of an outer query.
  - If a condition in 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 EXISTS and UNIQUE Functions in SQL for correlating queries

`EXISTS` function

- Check whether the result of a correlated nested query is empty or not. They are Boolean functions that return a TRUE or FALSE result.

`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

Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary.
- Built-in aggregate functions
  - COUNT,SUM,MAX,MIN, and AVG
- Grouping
  - 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.

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';
```

Grouping: The GROUP BY Clause

- Partition relation into subsets of tuples
  - Based on grouping attribute(s)
  - Apply function to each such group independently
- GROUP BY clause
  - Specifies grouping attributes
- COUNT(*) counts the number of rows in the group.

Examples of GROUP BY

•The grouping attribute must appear in the SELECT clause:

```
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.

•GROUP BY may be applied to the result of a JOIN

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

Grouping: The GROUP BY and HAVING Clauses

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

Combining WHERE and HAVING Clause

•Note: the WHERE clause applies tuple by tuple whereas HAVING applied to entire group of tuples.

Query 28. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
Q28:  SELECT   Dnumber, COUNT (*)
      FROM     DEPARTMENT, EMPLOYEE
      WHERE    Dnumber=Dno AND Salary>40000 AND
               ( SELECT       Dno
                 FROM         EMPLOYEE
                 GROUP BY Dno
                 HAVING       COUNT (*) > 5)
```

SIX Clauses of SQL

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];

Use of CASE

- SQL also has a CASE construct
- Used when a value can be different based on certain conditions.
- Can be used in any part of an SQL query where a value is expected.
- Applicable when querying, inserting, or updating tuples

Example of use of CASE

- The following examples shows that employees are receiving different raises in different departments

```
UPDATE    EMPLOYEE
SET       Salary =
CASE      WHEN  Dno = 5 THEN   Salary + 2000
          WHEN  Dno = 4 THEN   Salary + 1500
          WHEN  Dno = 1 THEN   Salary + 3000
```

Summary

- NULL constraint
- Three-value logic
- Nested Queries
- Class Activity on Nested Queries
- Aggregate Functions
- Group By and Having Clause
- CASE construct