SQL Part 2

4CCS1DBS - Database Systems

Author: Aaron Patrick Monte - 20059926

Set Operations

`UNION`

> 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. Which rows are retrieved?

```
(SELECT PNUMBER
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM = DNUMBER AND MGRSSN=SSN AND LNAME='SMITH')

UNION

(SELECT PNUMBER
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER=PNO AND ESSN=SSN AND LNAME='SMITH')
```

`EXCEPT (MINUS)`

> Example: List SSNs from all employees except those who are working on Project 1.

```
(SELECT SSN
FROM EMPLOYEE)

EXCEPT

(SELECT ESSN AS SSN
FROM WORKS_ON
WHERE PNO=1)
```

Arithmetic Operations

The standard arithmetic operators can be applied to numeric values in an SQL query result.

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

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

- Constants are allowed
- Note the use of **AS** to alias the result as an attribute
- `IF(<condition>, <True Value>, <Faluse Value>)`

- IF with more than 2 values? See CASE(), or nested IF statements.

```
SELECT FNAME, LNAME, (SALARY / 1000) AS SALARY_K, 1 AS ONE,
    IF(SALARY>30000, True, False) AS IS_LOADED,
    IF(SUPERSSN IS NULL, "BOSS", "WORKER") AS ETYPE
FROM EMPLOYEE
```

Use of CAST()

- Convert the Data Type of an attribute using CAST()
- CAST(<expression> AS <type>):

Manipulating data in SQL

Three SQL commands to modify the state of a database (part of the DML for SQL)

- INSERT
- DELETE
- UPDATE

They do not modify the SCHEMA of the database.

Note that SELECT is widely considered part of the DML because it clearly is not a DDL command.

INSERT

- In its simplest form, it is used to add one or more tuples in a relation

```
INSERT INTO <table name>
VALUES <tuple>
```

Attribute values should be listed in the same order as the attributes that were specified in the CREATE TABLE command.

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple.
- *Left out attributes will be default value or NULL*

```
INSERT INTO EMPLOYEE <table name> (<attribute list>)
VALUES <tuple>
```

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

  We want to create a temporary table that has the employee last name, project name, and hours per week for each employee working on a project.

1. Create the table, WORKS_ON_INFO
2. Load WORKS_ON_INFO with the results of a joined query:

```
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;
```

Values are mapped in the order they appear.

Using `CREATE TABLE ... AS`

- It is also possible to do the previous two queries in one `CREATE TABLE` command (note that **AS** is used in two different ways in the following query):

```
CREATE TABLE WORKS_ON_INFO AS
    E.Lname AS Emp_Name,
    P.Pname AS Proj_Name,
    W.Hours AS Hours_per_week
FROM PROJECT P, WORKS_ON W, EMPLOYEE E
WHERE P.Pnumber=W.Pno AND W.Essn=E.Ssn;
```

DELETE

- Removes tuples from a relation

```
DELETE FROM <table name>
WHERE <condition>
```

- 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` condition.

UPDATE

- Used to modify attribute values of one or more selected tuples

```
UPDATE <table name>
SET <attribute>=<value>, ...
WHERE <condition>
```

- 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.

Nested Queries

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

```
UPDATE EMPLOYEE
SET SALARY = SALARY*1.1
WHERE DNO IN (SELECT DNUMBER
```

```
    FROM DEPARTMENT
    WHERE DNAME='Research');
```

- Math Expression: 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 to the left of = refers to the new salary value post-modification
- 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
- Using the comparison operator IN: compares a value v with a set of values V and returns TRUE if v is one of the elements in V.

  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 D_NAME='Research');
```

Explanation of `IN`

- The nested query selects the number of the 'Research' department
- The outer query selects 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
- A reference to an unqualified attribute refers to the relation declared in the innermost nested 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) in the outer query.

  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)
```

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.

The `EXISTS` function

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

  Retrieve the name of each employee has a dependent with the same first name as the employee.

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE NOT EXISTS (SELECT *
    FROM DEPENDENT
    WHERE E.SSN=ESSN)
```

`ALL` Comparison Operator

- Comparison operators to compare a single value (as an attribute) to a set or multiset (a nested query)

  Retrieve the names of employees whose salary is greater than the department of all employees in department 5.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY
    FROM EMPLOYEE
    WHERE DNO = 5)
```

`NULL` 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.
- In join conditions, tuples with NULL values in these attributes are not included in result (i.e. DNUMBER = DNO, and both are NULL).

  Retrieve the names of all employees who do not have supervisors

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

Joins

- Using the `JOIN` keyword can specify 'joined relations'

- Two joined relations look like any other relation.
- There are many types.

SELECT ... with a JOIN condition in the WHERE clause:

```
SELECT DLOCATION, MGRSSN
FROM DEPARTMENT, DEPT_LOCATIONS
WHERE DNAME='Research' AND DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER;
```

Using JOIN ... ON as an 'equi-join'

```
SELECT DLOCATION, MGRSSN
FROM DEPARTMENT
JOIN DEPT_LOCATIONS ON DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER WHERE DNAME='Research';
```

NATURAL JOIN

```
SELECT DLOCATION, MGRSSN
FROM DEPARTMENT NATURAL JOIN DEPT_LOCATIONS
WHERE DNAME='Research';
```

Distinguishing between `JOIN` functions

NATURAL JOIN

- No join condition may be specified, implicit condition to join on attributes with the same name

INNER JOIN

- Tuple is included in the result only if a matching tuple exists in the other relation (default type of JOIN)

OUTER JOIN

- All matching tuples are returned (depending on type of OUTER JOIN):
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
  - FULL OUTER JOIN

Consider:

```
FROM EMPLOYEE E, EMPLOYEE S
WHERE E.SUPERSSN = S.SSN
```

A left outer join...

```
SELECT E.NAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E LEFT OUTER JOIN EMPLOYEE AS S ON E.SUPERSSN=S.SSN
```

...would return the same as above, but include employees without supervisors.

A right outer join...

```
SELECT E.NAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E RIGHT OUTER JOIN EMPLOYEE AS S ON E.SUPERSSN=S.SSN
```

...would return the same as above, but include employees who do not supervise anyone.

A full outer join...

```
SELECT E.NAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E FULL OUTER JOIN EMPLOYEE AS S ON E.SUPERSSN=S.SSN
```

...would return both of these.

CROSS JOIN

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM (EMPLOYEE E CROSS JOIN EMPLOYEE S)
```

is equivalent to:

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E, EMPLOYEE S
```

```
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E JOIN EMPLOYEE S
```

Multiway Joins

> For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address and birthdate.

```
SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATRE
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM = DNUMBER AND MGRSSN = SSN AND LOCATION = 'Stafford'
```

Is equivalent to specifying a multiway join:

```
SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATRE
FROM ((PROJECT JOIN DEPARTMENT ON DNUM=DNUMBER
    JOIN EMPOLOYEE ON MGR_SSN=SSN)
WHERE PLOCATION = 'Stafford'
```

Aggregate Queries

We can use aggregate queries to find values that are derived from the set of values in a table.

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

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

Functions include `COUNT`, `SUM`, `MAX`, `MIN`, `AVG`.

- Some SQL implementations may not allow more than one function in the SELECT-cause statement.

Find the max, min and avg among employees who work for the 'Research' department.

```
SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNO=DNUMBER AND DNAME='Research'
```

Count

Retrieve the total number of employees in the company, and the number of employees in the 'Research' department

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

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

Select the names of all employees who have two or more dependents.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE (SELECT COUNT(*)
    FROM DEPENDENT
    WHERE SSN=ESSN) >= 2;
```

- Note that when the result is one attribute and one tuple, it becomes a **SCALAR**

Count the number of distinct salary values in the Employees table;

```
SELECT COUNT (DISTINCT SALARY)
FROM EMPLOYEE
```

Note that NULL values are not counted as part of the aggregate.

On the contrary, `COUNT(*)` counts null values too.

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

```
SELECT <Attribute list, including grouping attributes>
FROM <table list>
[WHERE <condition>]
GROUP BY <grouping attributes>
```

Grouping with aggregate functions

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
```

- 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.

Dno	Count(*)	Avg(Salary)
5	4	33250
4	3	31000
1	1	55000

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

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

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)

```
SELECT <Attribute list, including grouping attributes>
FROM <table list>
[WHERE <condition>]
GROUP BY <grouping attributes>
HAVING <condition>
```

Assertions

General constraints: constraints that do not fit in the basic SQL categories.

- Useful for Schema Assertions - Outside the scope of the built-in relational model constraints.
- Defines whether the State of the Database is VALID at any given point of time.

- `CREATE ASSERTION`, components include a constraint name followed by a CHECK keyword followed by a condition clause.
- enforcing the assertion is up to the Database implementation - i.e. rejecting a query will violate the Check ASSERTION.

  The salary of an employee must not be greater than the salary of the manager of the department that the employee works for.

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (
    SELECT *
    FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
    WHERE E.SALARY > M.SALARY AND
    E.DNO=D.NUMBER AND D.MGRSSN=M.SSN))
```

Views in SQL

A view is a 'virtual' table that is derived from other tables. There are two ways they are implemented in implementation:

- Query modification - copy and paste queries
- View materialization - short term physical implementation

They are limited for UPDATE operations. You are unable to update views which are:

- Derived from Multiple Tables with JOINs
- Views defined with GROUP BY and aggregate functions. Allows full query operations

These are a convenience for expressing certain operations, useful for security and authorization and prevents redundant storage.

A 'friendlier' view of WORKS_ON

```
CREATE VIEW WORKS_ON1 AS
SELECT FNAME, LNAME, PNAME, HOURS
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE SSN=ESSN AND PNO=PNUMBER
```

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

```
SELECT FNAME, LNAME
FROM WORKS_ON1
WHERE PNAME='ProductX';
```

and when no longer needed, a view can be dropped:

```
DROP WORKS_ON1;
```

Dropping a View DOES NOT modify the data!