COS221
L10 - Advanced SQL
(Chapter 5 in Edition 6 and Chapter 7 in Edition 7)

Linda Marshall

17 March 2023

Comparisons involving NULL and three-valued logic

- Recall, NULL is used to represent missing values. A NULL value could mean value is unknown, unavailable or withheld, and not applicable.
- SQL has three-valued logic with values TRUE, FALSE and UNKNOWN. This means the boolean operators need to make provision for TRUE and FALSE in combination with UNKNOWN.

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

Comparisons involving NULL and three-valued logic

- In *select-project* queries, only tuples that result in a TRUE for the WHERE clause are selected. There is one exception and that is when considering outer joins.
- The keyword IS or IS NOT is used for comparisons of NULL values.

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

- SQL considers each NULL value as unique and therefore comparison operators cannot be used. Tuples with the comparison of NULL are therefore not included in the results.

Nested Queries (sub-queries), Tuples, and Set/Multiset Comparisons

- Some queries may require results from one query to be used as input to another query. Nested queries are used in such cases.
- The nested query is usually added in the WHERE clause of the outer query . The nested query is executed first and the result is used by the outer query

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

- Generally a nested query results in a table and not a scalar value.

Nested Queries (sub-queries), Tuples, and Set/Multiset Comparisons

- Other than IN, other comparisons can be used to compare a single value, v, to a set or multiset, V.
  - $=$ ANY (or $=$ SOME) is TRUE if v is equal some value in V. This makes it equivalent to IN
  - ANY and SOME can be combined with other operators (other than $=$). The keyword ALL can be used with these operators. For example: $v >$ ALLV returns TRUE if the value v is greater than all the values in V.

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

Nested Queries (sub-queries), Tuples, and Set/Multiset Comparisons

- Ambiguity of attributes when using nested joins can be a problem. In general it is a good idea to create aliases for all tables in a query.

```
SELECT E.Fname, E.Lame
FROM EMPLOYEE AS E
WHERE E.Ssn IN
    ( SELECT D.Essn
      FROM DEPENDENT AS D
      WHERE E.Fname = D.Dependent_name AND
            E.Sex =D.Sex );
```

Other Nested Query Related Concepts

- **Correlated Nested Query** - When the WHERE clause of a nested query references an attribute of the outer query, the queries are correlated.
- **EXISTS and UNIQUE functions** are used in the WHERE clause as they result in a TRUE/FALSE result. (NOT) EXISTS tests in the nested query is empty or not. UNIQUE returns TRUE if there are no duplicate tuples, otherwise it returns FALSE.
- **Explicit Sets** are specified in the WHERE clause and the value being selected must be in the set.
- Within the SELECT clause, attributes can be **renamed**. This will result in a table with attributes with the new naming.

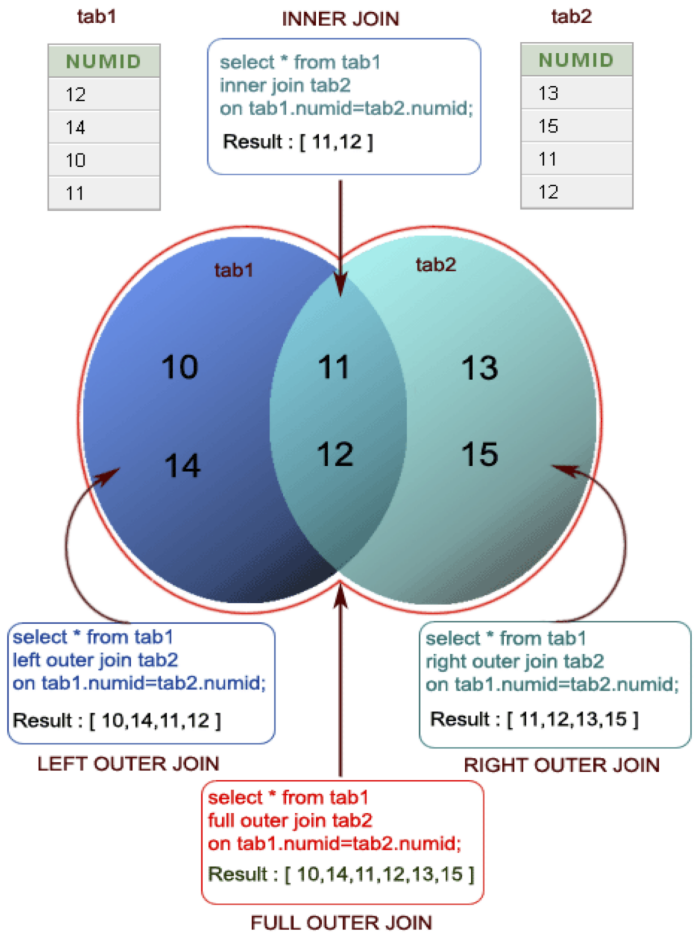
Other Nested Query Related Concepts

- **Joins** form a major part of SQL and where introduced so that joining is specified in the FROM clause.
  - A join is used to combine rows from two or more tables. Joins are left associative. That is A join B join C is completed as the result of A join B is joined to C.

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE JOIN DEPARTMENT ON Dno = Dnumber)
WHERE Dname = 'Research';
```

Other Nested Query Related Concepts

- **Joins** - There are different types of joins: NATURAL JOIN (also referred to as an inner join) and various types of OUTER JOINs. With a NATURAL join there must be identical column names (attributes) in each table being joined.



Other Nested Query Related Concepts

- The attributes of the joined table are all the attributes of the first table followed by all the attributes of the second table.
- When a natural join is used, no join condition is specified. The condition is an implicit Equijoin.

```
SELECT Fname, Lname, Address
FROM (EMPLOYEE NATURAL JOIN
(DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
WHERE Dname = 'Research';
```

Other Nested Query Related Concepts

- Inner joins only pair the tuples that match the join condition.
- If all tuples in one relation are required then an outer join is required. There are therefore two types of outer join, where all tuples in the table to the left of the join condition and to the right of the join condition are included.

Other Nested Query Related Concepts

- **Aggregate functions** summerise information from multiple tuples. Examples of aggregate functions are COUNT, SUM, AVG, MIN, MAX...

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

This query returns a single row table with 4 columns. Renaming the columns will make the table more readable.

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

Other Nested Query Related Concepts

- **Grouping** enables an aggregate to be applied to a subgroup of tuples in a relation.

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

- If NULL exists in the table, then a separate group for all NULL values in the grouping attribute is created.
- Adding a HAVING clause, provides a condition on the grouping information.

Specifying Constraints as Assertions and Actions as Triggers

Constraints as Assertions

An assertion is a named constraint that may relate to the content of individual rows of a table, to the entire contents of a table, or to a state required to exist among a number of tables (entire schema or database).

```
CREATE ASSERTION <constraint name>
CHECK (<search condition>)
```

For example:

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK ( NOT EXISTS (
        SELECT  *
        FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
        WHERE  E.Salary>M.Salary AND E.Dno=D.Dnumber
        AND D.Mgr_ssn=M.Ssn ) );
```

Specifying Constraints as Assertions and Actions as Triggers

Actions as Triggers

A trigger is a set of actions that are run automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table.

Uses for triggers:

- Enforce business rules
- Validate input data
- Generate a unique value for a newly-inserted row in a different file.
- Write to other files for audit trail purposes
- Replicate data to different files to achieve data consistency

Specifying Constraints as Assertions and Actions as Triggers

Actions as Triggers

```
CREATE
[DEFINER = { user | CURRENT_USER }]
TRIGGER trigger_name {All triggers must have unique names w
ON tbl_name FOR EACH ROW
trigger_body {the statement to execute when the trigger act
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
```

Specifying Constraints as Assertions and Actions as Triggers

Actions as Triggers For example to check whenever an employee's salary is greater than the salary of his or her direct supervisor in the COMPANY database

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN ON EMPLOY
FOR EACH ROW
WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
WHERE SSN = NEW.SUPERVISOR_SSN ) )
INFORM_SUPERVISOR(NEW.Supervisor_ssn, NEW.Ssn );
```

Views (virtual tables)

A view is a single table that is derived from other tables and is a virtual table. It is usually defined for queries that are run frequently.
To create and remove a view:

```
CREATE VIEW Viewname
AS defining_query;
```

```
DROP view Viewname;
```

A view is up to date, modifying the values in tuples in the base table will update the values in the view. A view is not realised when it is created, but when a query is run on the view.

Views (virtual tables)

Examples of views:

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

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;
```

Schema change statements

The DROP command can be used on SCHEMA and TABLEs. The ALTER command can be used on TABLES.

```
DROP SCHEMA COMPANY CASCADE;

DROP TABLE DEPENDENT CASCADE;

ALTER TABLE COMPANY.EMPLOYEE
ADD COLUMN Job VARCHAR(12);

ALTER TABLE COMPANY.DEPARTMENT
ALTER COLUMN Mgr_ssn DROP DEFAULT;

ALTER TABLE COMPANY.DEPARTMENT
ALTER COLUMN Mgr_ssn SET DEFAULT ?333445555?;
```

Summary

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>)
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.