Complex Queries and Constraints

COURSE : 23CAT- Database Management System
UNIT I : Introduction
CLASS : I Semester / I MCA

- More Complex SQL Retrieval Queries
- Specifying Semantic Constraints as Assertions and Actions as Triggers
- Views (Virtual Tables) in SQL
- Triggers
- Schema Modification in SQL

- Handling NULLs, 3-valued Logic in SQL
- Nested Queries
  - Correlated vs. uncorrelated
  - EXISTS function
- Joined Tables, Inner Joins, and Outer Joins
- Aggregate Functions and Grouping in SQL
  - COUNT, AVG, SUM, MIN, MAX functions
  - GROUP BY, HAVING clauses

Handling NULLs

- SQL allows queries that check if an attribute is NULL (missing or undefined or not applicable)
- SQL uses **IS** or **IS NOT** to compare an attribute to NULL because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate.
- Example: Retrieve the names of all employees who do not have supervisors.

```
SELECT fname, lname FROM employee WHERE supervisor IS NULL
```

3 Valued Logic in SQL

- Standard 2-valued logic assumes a condition can evaluate to either TRUE or FALSE
- With NULLs, a condition can evaluate to UNKNOWN, leading to 3-valued logic
- Combining individual conditions using AND, OR, NOT logical connectives must consider UNKNOWN

Example: Consider a condition EMPLOYEE.REGNO = 115

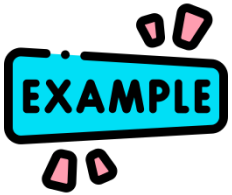
This evaluates for individual tuples in EMPLOYEE as follows:

- **TRUE** for tuples with REGNO=115
- **UNKNOWN** for tuples where DNO is NULL
- **FALSE** for other tuples in EMPLOYEE

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

p	NOT *p*
True	False
False	True
Unknown	Unknown

❑ Complete select-from-where blocks within WHERE clause of another query



Make a list of all project IDs for projects that involve employee Smith either as worker or as a manager of the department that controls the project

```
SELECT   DISTINCT Pnumber
FROM     PROJECT
WHERE    Pnumber IN
         ( SELECT     Pnumber
           FROM       PROJECT, DEPARTMENT, EMPLOYEE
           WHERE      Dnum=Dnumber AND
                      Mgr_ssn=Ssn AND Lname='Smith' )
         OR
         Pnumber IN
         ( SELECT     Pno
           FROM       WORKS_ON, EMPLOYEE
           WHERE      Essn=Ssn AND Lname='Smith' );
```

- Comparison operator IN
  - Compares value v with a set (or multiset) of values V
  - Evaluates to TRUE if v is one of the elements in

- Use tuples of values in comparisons
- Place them within parentheses

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

General Form of ALL, ANY, SOME

```
SELECT [column_name ]
FROM [table_name]
WHERE expression operator
                {ALL | ANY | SOME} (
subquery )
```

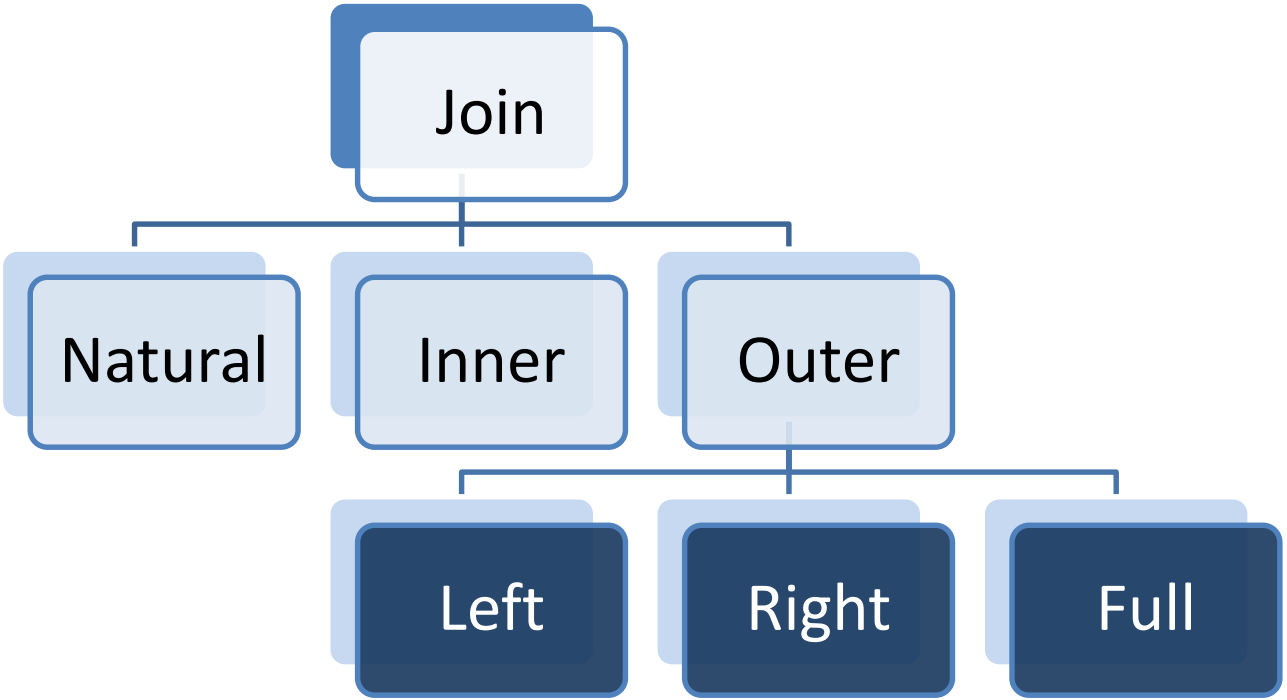
- other comparison operators to compare a single value *v*
  - = **ANY** (or = **SOME**) operator [equivalent to IN]
    - Returns TRUE if the value *v* is equal to some value in the set
  - Other operators that can be combined with **ANY** (or **SOME**): >, >=, <, <=, and <>
  - **ALL**: value must exceed all values from nested query

Aggregate functions

- sum()
- max()
- min()
- avg()

- 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

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


Join
Natural
Inner
Outer
Left
Right
Full

- **Partition** relation into subsets of tuples based on grouping attributes by **GROUP BY** clause
- **HAVING** clause provides a condition to select or reject an entire group

```
SELECT    Pnumber, Pname, COUNT (*)
                  FROM
          PROJECT, WORKS_ON
                  WHERE     Pnumber=Pno
                  GROUP BY  Pnumber, Pname
                  HAVING    COUNT (*) > 2;
```

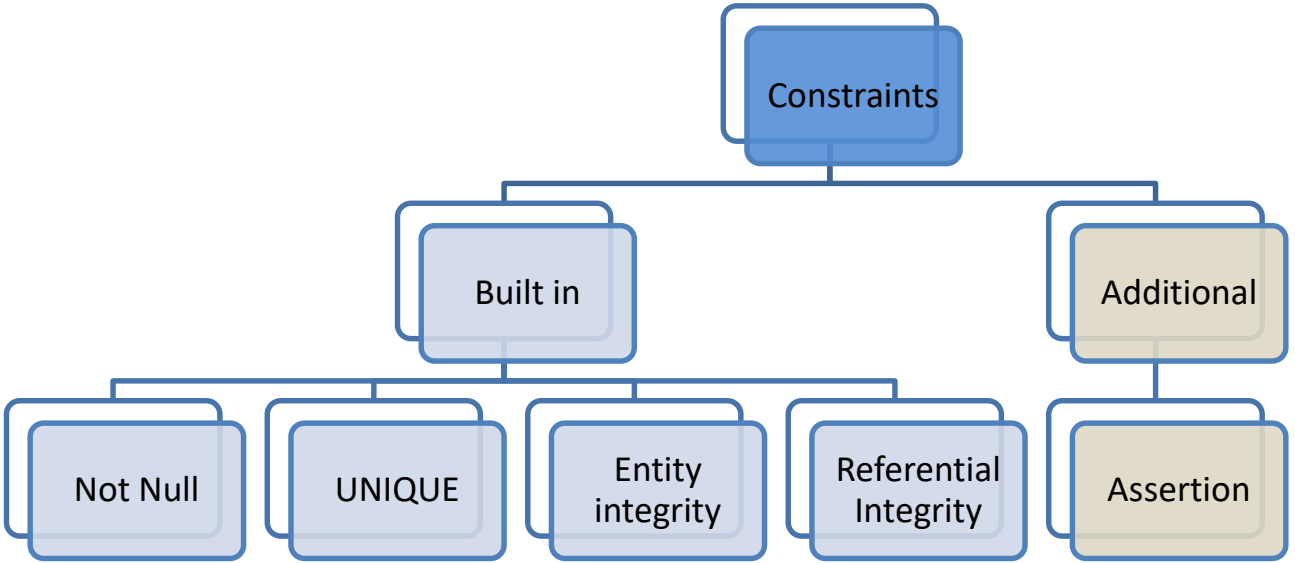
Expanded basic block of SQL Query

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

Constraints as Assertions

COURSE : 23CAT- Database Management System
UNIT I : Introduction
CLASS : I Semester / I MCA

Triggers


Constraints
Built in
Additional
Not Null
UNIQUE
Entity integrity
Referential Integrity
Assertion

- **Semantic Constraints**: The following are beyond the scope of the EER and relational model

- **CREATE ASSERTION**
  Specify additional types of constraints outside scope of built-in relational model constraints

- **CREATE TRIGGER**
  Specify automatic actions that database system will perform when certain events and conditions occur

Assertion

- Specify a query that selects any tuples that violate the desired condition
- Use only in cases where it goes beyond a simple CHECK which applies to individual attributes and domains

EXAMPLE

Condition:

Salary of an Employee must not be greater than the salary of the Manager of the corresponding department

```
CREATE ASSERTION SALARY _ CONSTRAINT
    CHECK ( NOT EXISTS ( SELECT * EMPLOYEE E, EMPLOYEE M,
                         DEPARTMENT D E . SAL > M . SAL AND
                         D E . SAL > M . SAL AND
                         D MAR . SSN = M . SSN ) )
```

❑ Schema evolution commands

- DBA may want to change the schema while the database is operational
- Does not require recompilation of the database schema

❑ **DROP** command
- drop named schema elements, such as tables, domains, or constraint
- Options: CASCADE and RESTRICT
- CASCADE removes the schema and all its elements including tables, views, constraints, etc.
- RESTRICT: drops only nothing in it

DROP SCHEMA COMPANY CASCADE

Alter table actions include:

- Adding or dropping a column (attribute)
- Changing a column definition
- Adding or dropping table constraints

Example:

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

Change constraints specified on a table

- Add or drop a named constraint

```
ALTER TABLE COMPANY.EMPLOYEE
DROP CONSTRAINT EMPSUPERFK CASCADE;
```

Trigger Example

```
SQL>CREATE OR REPLACE TRIGGER derive_commission_trg
  2 BEFORE UPDATE OF sal ON emp
  3 FOR EACH ROW
  4 WHEN (new.job = 'SALESMAN')
  5 BEGIN
  6   :new.comm := :old.comm * (:new.sal/:old.sal);
  7 END;
  8 /
```

Trigger name:	derive_commission_trg
Timing:	BEFORE executing the statement
Triggering event:	UPDATE of sal column
Filtering condition:	job = 'SALESMAN'
Target:	emp table
Trigger parameters:	old, new
Trigger action:	calculate the new commission to be updated

```
CREATE OR REPLACE TRIGGER display_salary_changes
BEFORE DELETE OR INSERT OR UPDATE ON customers
FOR EACH ROW
WHEN (NEW.ID > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :NEW.salary  - :OLD.salary;
  dbms_output.put_line('Old salary: ' || :OLD.salary);
  dbms_output.put_line('New salary: ' || :NEW.salary);
  dbms_output.put_line('Salary difference: ' || sal_diff);
END;
/
```

Triggering Event

INSERT INTO CUSTOMERS (ID,NAME,AGE,ADDRESS, SALARY) VALUES (7, 'Kriti', 22, 'HP', 15000.00);

Output

```
Old salary:
New salary: 15000
Salary difference:
```

UPDATE customers SET salary = salary + 3000
WHERE id = 7;

Output

```
Old salary: 15000
New salary: 18000
Salary difference: 3000
```