SQL

SQL – Structured Query Language

SEQUEL – Structured English QUEry Language

Parts of SQL:
DDL
DML
Views
Security & Authorization
Transaction Control

CREATE SCHEMA

CREATE SCHEMA COMPANY **AUTHORIZATION** 'Jsmith';

Authorization Identifier – Who owns the Schema

In general, not all users are authorized to create schemas and schema elements. The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

Tables are generally created within the SQL Schema

Do you Find Something Weird In this ?

```
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),
          Super_ssn      CHAR(9),
          Dno            INT                NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

Foreign Keys can be specified later, through ALTER Command

Attributes are considered
To be ordered
In this order

```
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),
  Super_ssn      CHAR(9),
  Dno            INT                NOT NULL,
PRIMARY KEY (Ssn),
FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

4.2.2 Specifying Key and Referential Integrity Constraints

Dnumber INT **PRIMARY KEY**;

PRIMARY KEY = NOT NULL + UNIQUE

Is it possible to create a Table in SQL without a Primary Key ?

4.1.3 Attribute Data Types and Domains in SQL

- **Numeric** data types include integer numbers of various sizes (INTEGER or INT, and SMALLINT) and floating-point (real) numbers of various precision (FLOAT or REAL, and DOUBLE PRECISION). Formatted numbers can be declared by using DECIMAL(i,j)—or DEC(i,j) or NUMERIC(i,j)—where i, the *precision*, is the total number of decimal digits and j, the *scale*, is the number of digits after the decimal point. The default for scale is zero, and the default for precision is implementation-defined.

4.1.3 Attribute Data Types and Domains in SQL

Character String

Bit String

Boolean

Date

TImestamp

4.1.3 Attribute Data Types and Domains in SQL

```
CREATE DOMAIN SSN_TYPE AS CHAR(9);
```

4.2.1 Specifying Attribute Constraints and Attribute Defaults

NOT NULL

DEFAULT

```
Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);

CREATE DOMAIN D_NUM AS INTEGER
CHECK (D_NUM > 0 AND D_NUM < 21);
```

4.2.2 Specifying Key and Referential Integrity Constraints

Dnumber INT **PRIMARY KEY**;

```
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) );
```

Composite Primary Key

4.2.2 Specifying Key and Referential Integrity Constraints

Dname VARCHAR(15) **UNIQUE;**

Foreign Key

Syntax for F.Key you know already.

What about actions when referential integrity is violated ?

The default action is to “REJECT” the update operation.

Referential Triggered Action

```
CREATE TABLE DEPARTMENT
    (   . . . ,
        Mgr_ssn     CHAR(9)         NOT NULL            DEFAULT '888665555',
        . . . ,
    CONSTRAINT DEPTPK
        PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
        UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                        ON DELETE SET DEFAULT   ON UPDATE CASCADE);
```

Referential Triggered Action

```
CREATE TABLE DEPT_LOCATIONS
        (   . . . ,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                        ON DELETE CASCADE          ON UPDATE CASCADE);
```

Trigger

Action

4.2.4 Specifying Constraints on Tuples Using CHECK

CHECK (Dept_create_date <= Mgr_start_date);

This line is added at the end of the CREATE Table Command

5.4 Schema Change Statements in SQL

Called as **Schema Evolution Commands**

DROP SCHEMA COMPANY

DROP TABLE DEPENDENT

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

Question Can you use NOT NULL constraint with this new column ?

ALTER TABLE COMPANY.EMPLOYEE **DROP COLUMN** Address

Queries

Multisets vs Sets

To constrain a table to be a Set, use Key constraint or DISTINCT Command

```
SELECT    <attribute list>
FROM      <table list>
WHERE     <condition>;
```

where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Figure 3.5
Schema diagram for the COMPANY relational database schema.

Figure 3.6
One possible database state for the COMPANY relational database schema.

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

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Query 0. Retrieve the birth date and address of the employee(s) whose name is 'John B. Smith'.

```
Q0:   SELECT   Bdate, Address
      FROM     EMPLOYEE
      WHERE    Fname='John' AND Minit='B' AND Lname='Smith';
```

Query 1. Retrieve the name and address of all employees who work for the 'Research' department.

```
Q1:   SELECT   Fname, Lname, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    Dname='Research' AND Dnumber=Dno;
```

Query 2. For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

```
Q2:   SELECT   Pnumber, Dnum, Lname, Address, Bdate
      FROM     PROJECT, DEPARTMENT, EMPLOYEE
      WHERE    Dnum=Dnumber AND Mgr_ssn=Ssn AND
               Plocation='Stafford';
```

(c)

Pnumber	Dnum	Lname	Address	Bdate
10	4	Wallace	291Berry, Bellaire, TX	1941-06-20
30	4	Wallace	291Berry, Bellaire, TX	1941-06-20

4.3.2 Ambiguous Attribute Names, Aliasing, Renaming, and Tuple Variables

In SQL, the same name can be used for two (or more) attributes as long as the attributes are in *different relations*. If this is the case, and a multitable query refers to two or more attributes with the same name, we *must* **qualify** the attribute name with the relation name to prevent ambiguity. This is done by *prefixing* the relation name to the attribute name and separating the two by a period. To illustrate this, suppose that

```
Q1A:  SELECT   Fname, EMPLOYEE.Name, Address
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DEPARTMENT.Name='Research' AND
               DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice, as in the following example.

Query 8. For each employee, retrieve the employee's first and last name and the first and last name of his or her immediate supervisor.

```
Q8:   SELECT   E.Fname, E.Lname, S.Fname, S.Lname
      FROM     EMPLOYEE AS E, EMPLOYEE AS S
      WHERE    E.Super_ssn=S.Ssn;
```

One Level Recursive Query

E → Subordinates
S → Supervisor

Tuple Variables / Aliases / Copies

You can Rename/Alias without Explicitly using the AS keyword

```
Q1B:    SELECT    E.Fname, E.LName, E.Address
        FROM      EMPLOYEE E, DEPARTMENT D
        WHERE     D.DName='Research' AND D.Dnumber=E.Dno;
```

```
Q9:     SELECT    Ssn
        FROM      EMPLOYEE;

Q10:    SELECT    Ssn, Dname
        FROM      EMPLOYEE, DEPARTMENT;
```

Its equivalent to Project Operation in RA

```
Q1C:    SELECT    *
        FROM      EMPLOYEE
        WHERE     Dno=5;
```

*** Operator, returns all the Attributes**

Why SQL Allows Duplicates ?

- Duplicate elimination is an expensive operation. One way to implement it is to sort the tuples first and then eliminate duplicates.
- The user may want to see duplicate tuples in the result of a query.
- When an aggregate function (see Section 5.1.7) is applied to tuples, in most cases we do not want to eliminate duplicates.

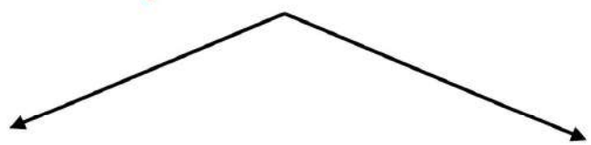
```
Q11:    SELECT    ALL Salary
        FROM      EMPLOYEE;
```

This ALL is typically not used,
But it is implicit

What if I want to eliminate Duplicates ?

```
Q11A:  SELECT    DISTINCT Salary
       FROM      EMPLOYEE;
```

Set Operations in SQL



No Duplicates
Union
Intersect …

Duplicates Allowed
Union ALL
Intersect ALL …

Pattern Matching in SQL

Query 12. Retrieve all employees whose address is in Houston, Texas.

```
Q12:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Address LIKE '%Houston,TX%';
```

Query 12A. Find all employees who were born during the 1950s.

```
Q12:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Bdate LIKE '_ _ 5 _ _ _ _ _ _ _';
```

What if I want to search for ‘%’ or ‘_’ itself

‘AB_CD\%EF’ ESCAPE ‘\’

Usage of Arithmetic operators within Queries

Query 13. Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
Q13:  SELECT  E.Fname, E.Lname, 1.1 * E.Salary AS Increased_sal
      FROM    EMPLOYEE AS E, WORKS_ON AS W, PROJECT AS P
      WHERE   E.Ssn=W.Essn AND W.Pno=P.Pnumber AND
              P.Pname='ProductX';
```

Between Operator

Query 14. Retrieve all employees in department 5 whose salary is between $30,000 and $40,000.

```
Q14:  SELECT   *
      FROM     EMPLOYEE
      WHERE    (Salary BETWEEN 30000 AND 40000) AND Dno = 5;
```

Order By

Query 15. Retrieve a list of employees and the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, then first name.

```
Q15:    SELECT      D.Dname, E.Lname, E.Fname, P.Pname
        FROM        DEPARTMENT D, EMPLOYEE E, WORKS_ON W,
                    PROJECT P
        WHERE       D.Dnumber= E.Dno AND E.Ssn= W.Essn AND
                    W.Pno= P.Pnumber
        ORDER BY    D.Dname, E.Lname, E.Fname;
```

Order By

ORDER BY D.Dname **DESC**, E.Lname **ASC**, E.Fname **ASC**

Structure of a Simple Query

Mandatory Clauses

```
SELECT      <attribute list>
FROM        <table list>
```

Optional Clauses

```
[ WHERE     <condition> ]
[ ORDER BY <attribute list> ];
```

Delete Command

```
U4A:  DELETE FROM  EMPLOYEE
      WHERE        Lname='Brown';

U4B:  DELETE FROM  EMPLOYEE
      WHERE        Ssn='123456789';

U4C:  DELETE FROM  EMPLOYEE
      WHERE        Dno=5;

U4D:  DELETE FROM  EMPLOYEE;
```

Update Command

```
UPDATE      PROJECT
SET         Plocation = 'Bellaire', Dnum = 5
WHERE       Pnumber=10;
```

```
U6:     UPDATE      EMPLOYEE
        SET         Salary = Salary * 1.1
        WHERE       Dno = 5;
```

MultiSets in SQL

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

A TABLE in SQL **may** have duplicates, if it has no Primary Key defined.

A TABLE in SQL **can** have duplicates, **If and only if** it has no Primary Key defined.

When storing application logs, Caches, or system events, duplicate entries are normal and may even be necessary

```
CREATE TABLE SystemLogs (

LogID INT AUTO_INCREMENT,
EventTime TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
EventType VARCHAR(50),

);
```

```
CREATE TABLE API_Cache (

APIEndpoint VARCHAR(255),
ResponseData TEXT,
Timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);
```

These are highly rare cases, Almost always tables will be created with Primary keys

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

NULL In SQL

- Unknown
- Not Available / Unavailable
- Not Applicable

1. **Unknown value.** A person's date of birth is not known, so it is represented by NULL in the database.
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.

NULL In SQL

It is often not possible to determine which of the meanings is intended; for example, a NULL for the home phone of a person can have any of the three meanings. Hence, SQL does not distinguish between the different meanings of NULL.

Three Valued Logic in SQL

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

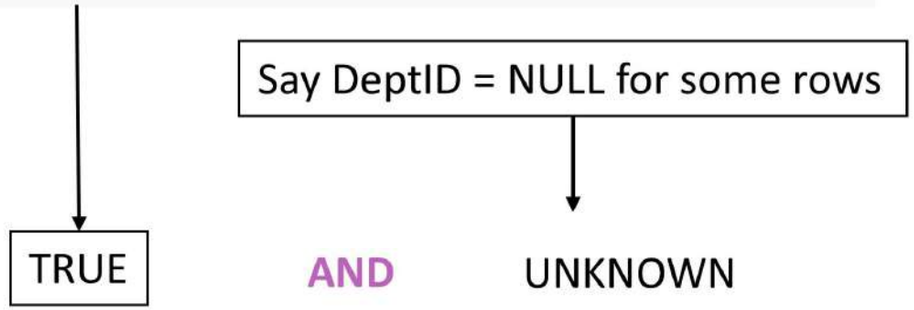
OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Three Valued Logic in Practice

```sql
SELECT * FROM Employee WHERE (Salary > 50000) AND (DeptID = 3);
```




```sql
SELECT * FROM Employee WHERE NOT (DeptID = 3);
```

Three Valued Logic in SQL

Query 18. Retrieve the names of all employees who do not have supervisors.

```
SELECT  names
FROM    EMPLOYEE
WHERE   Super_ssn = NULL
```

```
Q18:    SELECT    Fname, Lname
        FROM      EMPLOYEE
        WHERE     Super_ssn IS NULL;
```

Nested Queries

Nested Queries

Some queries require that existing values in the database be fetched and then used in a comparison condition. Such queries can be conveniently formulated by using **nested queries**, which are complete select-from-where blocks within the WHERE clause of another query. That other query is called the **outer query**.

Nested Queries

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  (SELECT    DISTINCT Pnumber
       FROM      PROJECT, DEPARTMENT, EMPLOYEE
       WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                 AND Lname='Smith' )
       UNION
      ( SELECT   DISTINCT Pnumber
       FROM      PROJECT, WORKS_ON, EMPLOYEE
       WHERE     Pnumber=Pno AND Essn=Ssn
                 AND Lname='Smith' );
```

Nested Queries

Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  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' );
```

Nested Queries

If a nested query returns a single attribute *and* a single tuple, the query result will be a single (scalar) value. In such cases, it is permissible to use = instead of IN for the comparison operator. In general, the nested query will return a **table** (relation), which is a set or multiset of tuples.

- **Nested** subquery runs only once for the entire nesting (outer) query. It does not contain any reference to the outer query row.

- **Correlated** subquery runs once for each row selected by the outer query. It contains a reference to a value from the row selected by the outer query.

#	Correlated Subquery	Non-Correlated Subquery
Execution	Triggered for each row in the outer query	Executed only once, regardless of outer query rows
Dependency	Relies on values from the outer query	Independent of the outer query
Efficiency	Less efficient for large datasets	More efficient for large datasets

Nested Queries

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

Notice the parenthesis

Comparing Single Value to a Set of Values

In addition to the IN operator, a number of other comparison operators can be used to compare a single value *v* (typically an attribute name) to a set or multiset *v* (typically a nested query). The = ANY (or = SOME) operator returns TRUE if the value *v* is equal to *some value* in the set *V* and is hence equivalent to IN. The two keywords ANY and SOME have the same effect. Other operators that can be combined with ANY (or SOME) include >, >=, <, <=, and <>. The keyword ALL can also be combined with each of these operators. For example, the comparison condition (*v* > ALL *V*) returns TRUE if the value *v* is greater than *all* the values in the set (or multiset) *V*.

Comparison operators in Nested Queries

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

Query 16. Retrieve the name of each employee who has a dependent with the same first name and is the same sex as the employee.

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

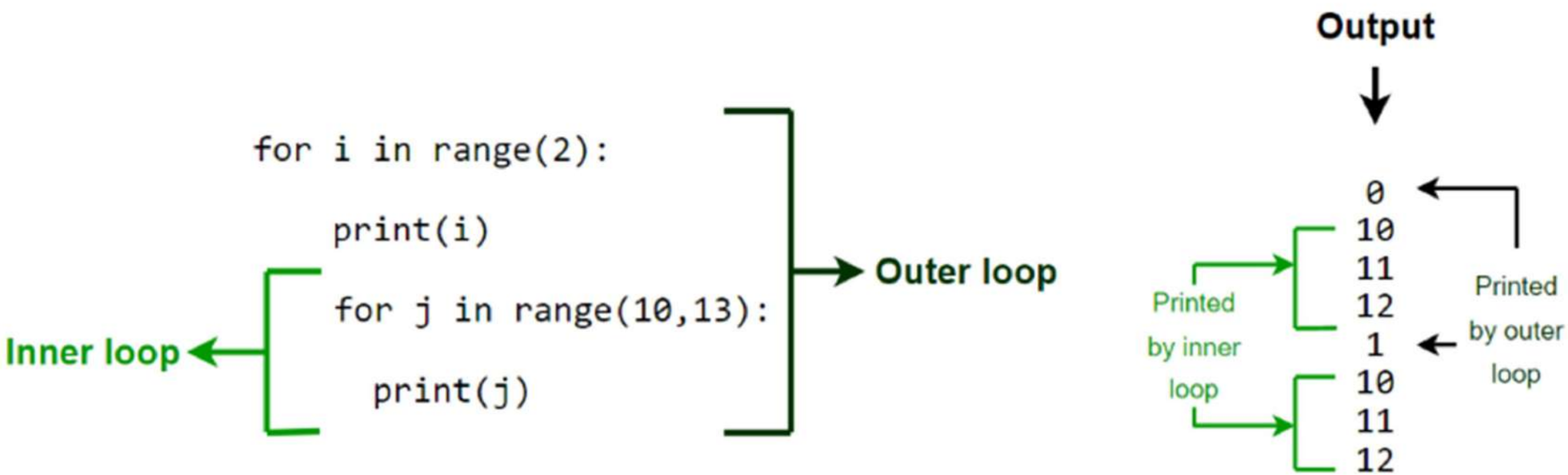
```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E
WHERE     E.Ssn IN  ( SELECT    Essn
                      FROM      DEPENDENT AS D
                      WHERE     E.Fname=D.Dependent_name
                      AND E.Sex=D.Sex );
```

In the nested query of Q16, we must qualify E.Sex because it refers to the Sex attribute of EMPLOYEE from the outer query, and DEPENDENT also has an attribute called Sex. If there were any unqualified references to Sex in the nested query, they would refer to the Sex attribute of DEPENDENT. However, we would not *have to* qualify the attributes Fname and Ssn of EMPLOYEE if they appeared in the nested query because the DEPENDENT relation does not have attributes called Fname and Ssn, so there is no ambiguity.

5.1.3 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**.

The Inner Query is evaluated once for each row of the outer query


for i in range(2):
print(i)
for j in range(10,13):
print(j)
Outer loop
Inner loop
Output
0
10
11
12
1
10
11
12
Printed by inner loop
Printed by outer loop

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

For each EMPLOYEE tuple:

retrieve the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple;

If the Ssn value of the EMPLOYEE tuple is in the result of the nested query, then select that EMPLOYEE tuple.

In general, 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.

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E, DEPENDENT AS D
WHERE     E.Ssn=D.Essn AND E.Sex=D.Sex
          AND E.Fname=D.Dependent_name;
```

5.1.4 The EXISTS and UNIQUE Functions in SQL

The EXISTS function in SQL is used to check whether the result of a correlated nested query is empty (contains no tuples) or not.

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

For each Employee:
If there exist atleast one dependent with
Same sex and same name, then
Return that employee

If at least one tuple EXISTS in the result of the nested query, then select that EMPLOYEE tuple

```
SELECT     Fname, Lname
FROM       EMPLOYEE
WHERE      NOT EXISTS ( SELECT     *
                        FROM       DEPENDENT
                        WHERE      Ssn=Essn );
```

Query 6. Retrieve the names of employees who have no dependents.

If and only if "NO" Tuple EXISTS in the result of the nested query, then select that EMPLOYEE tuple

Query 7. List the names of managers who have at least one dependent.

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     EXISTS ( SELECT    *
                   FROM      DEPENDENT
                   WHERE     Ssn=Essn )

    AND
    EXISTS ( SELECT    *
             FROM      DEPARTMENT
             WHERE     Ssn=Mgr_ssn );
```

Selects Employees With Dependents

Selects Employees Who are managers

For Each Employee tuple: You return that tuple If they have both dependents And Also they are managers

Query 7. List the names of managers who have at least one dependent.

```
SELECT DISTINCT E.Fname, E.Lname
FROM EMPLOYEE E, DEPARTMENT D, DEPENDENT DEP
WHERE E.Ssn = D.Mgr_ssn
AND E.Ssn = DEP.Essn;
```

Query 3. Find the names of employees who work on *all* the projects controlled by department number 5.

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     NOT EXISTS ( ( SELECT    Pnumber
                         FROM      PROJECT
                         WHERE     Dnum=5)
                       EXCEPT  ( SELECT    Pno
                                 FROM      WORKS_ON
                                 WHERE     Ssn=Essn) );
```

Not Correlated with the Outer Query

Selects all projects that the particular employee being considered works on

For each employee:
IF (First Subquery – Second Subquery) = Empty
It means that the employee works on all the projects and is therefore selected

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

For each EMPLOYEE tuple:

Retrieve the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple;

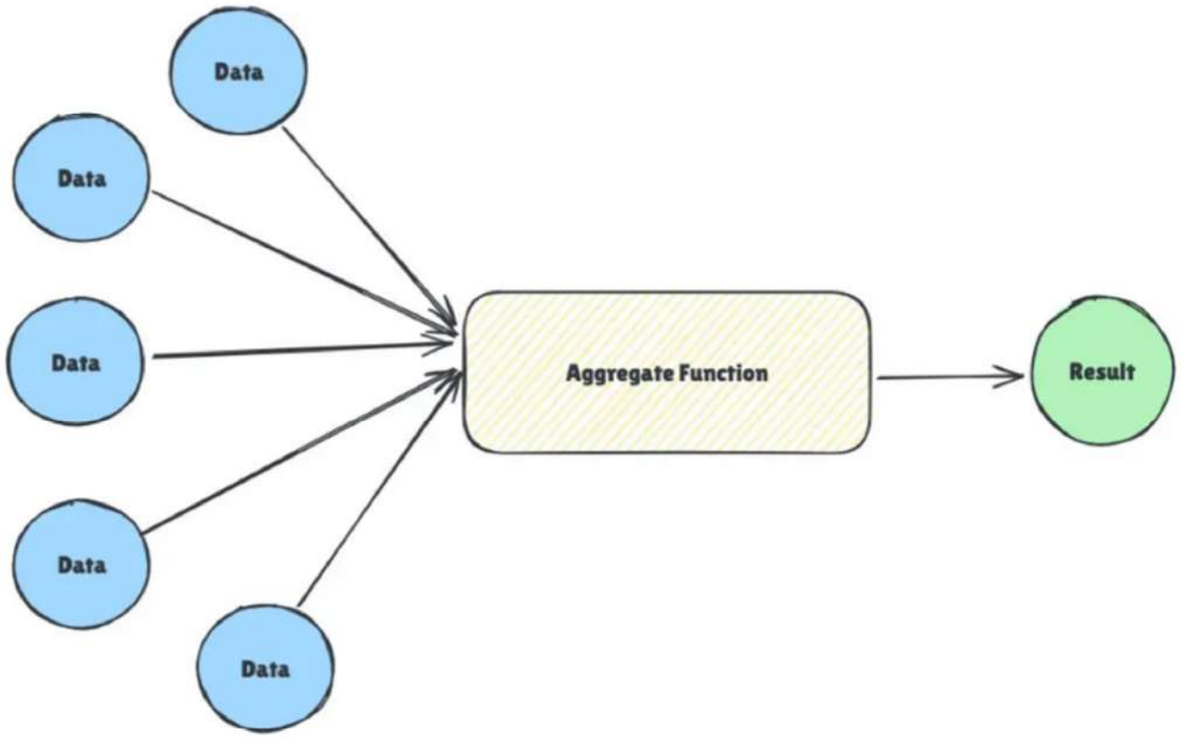
If the Ssn value of the EMPLOYEE tuple matches with the any of the Essn Value, then return the F.name and L.Name

```
SELECT    E.Fname, E.Lname
FROM      EMPLOYEE AS E, DEPENDENT AS D
WHERE     E.Ssn=D.Essn AND E.Sex=D.Sex
          AND E.Fname=D.Dependent_name;
```

In general, 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. For exam-

5.1.7 Aggregate Functions in SQL

Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary.



Function Name	Meaning	Example
SUM(column name)	Total sum of the values in a numeric column	`SUM(salary)`
AVG(column name)	Average of the values in a column	`AVG(salary)`
MAX(column name)	Largest value in a column	`MAX(salary)`
MIN(column name)	Smallest value in a column	`MIN(salary)`
COUNT(*)	Count of the number of rows selected	`COUNT(*)`

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.

```
SELECT    SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
FROM      (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
WHERE     Dname='Research';
```

Count the number of distinct salary values in the database.

```
SELECT    COUNT (DISTINCT Salary)
FROM      EMPLOYEE;
```

What if someone's salary is NULL, will it be counted ?

In **general**, NULL values are discarded when aggregate functions are applied to a particular column (attribute).

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

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

```
SELECT    Lname, Fname
FROM      EMPLOYEE
WHERE     ( SELECT    COUNT (*)
            FROM  DEPENDENT
            WHERE     Ssn=Essn ) >= 2;
```

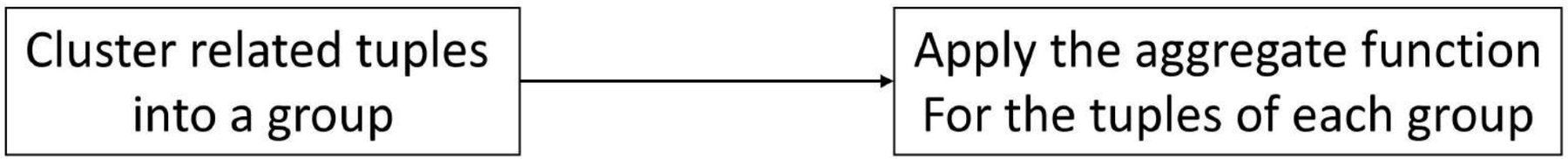
This correlated nested query counts the number of dependents that each employee has; if this is greater than or equal to two, the employee tuple is selected

Consider queries like:

Find average salary of each department.

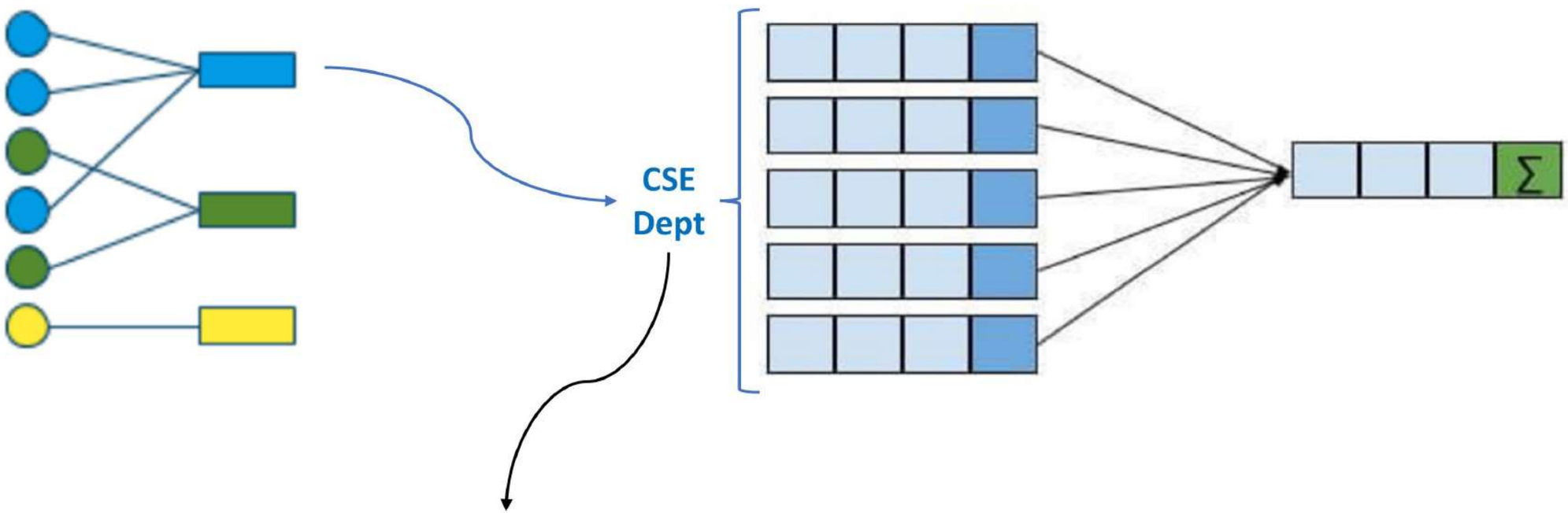
Find the number of employees in each project.

Find the number of employees managed by each manager.



Cluster related tuples into a group

Apply the aggregate function For the tuples of each group



Grouping Attribute: Has same value for a particular group

Query 24. For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
Q24:    SELECT      Dno, COUNT (*), AVG (Salary)
        FROM        EMPLOYEE
        GROUP BY    Dno;
```

Note that: Select clause includes only the Grouping Attribute and the Aggregate functions to be applied on each Group of tuples

Q24: **SELECT** Dno, **COUNT** (*), **AVG** (Salary)
FROM EMPLOYEE
GROUP BY Dno;

(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

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

Result of Q24

What if there were some employees whose Dno = NULL

Query 25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
Q25:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber=Pno
        GROUP BY    Pnumber, Pname;
```

Query 25. For each project, retrieve the project number, the project name, and the number of employees who work on that project.

Say I'm adding one more condition to this query
I want only projects with more than two employees to appear in the result

Query 26. 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.

```
Q26:    SELECT      Pnumber, Pname, COUNT (*)
        FROM        PROJECT, WORKS_ON
        WHERE       Pnumber=Pno
        GROUP BY    Pnumber, Pname
        HAVING      COUNT (*) > 2;
```

WHERE helps to choose tuples to which the aggregate functions are applied.

HAVING clause serves to choose whole groups.

WHERE Clause	HAVING Clause
Applicable without GROUP BY clause	Cannot be used without GROUP BY Clause
Can be used with SELECT, UPDATE, DELETE statement.	Can only be used with SELECT statement
Cannot contain aggregate function	Can contain aggregate function
Used to filter the records from the table based on the specified condition.	Used to filter record from the groups based on the specified condition.

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

IN Vs Exists

IN is used when comparing a column against a list of values or a subquery result set.

It checks if a value is present in the specified list or subquery result.

The subquery returns a list of values that the main query checks against

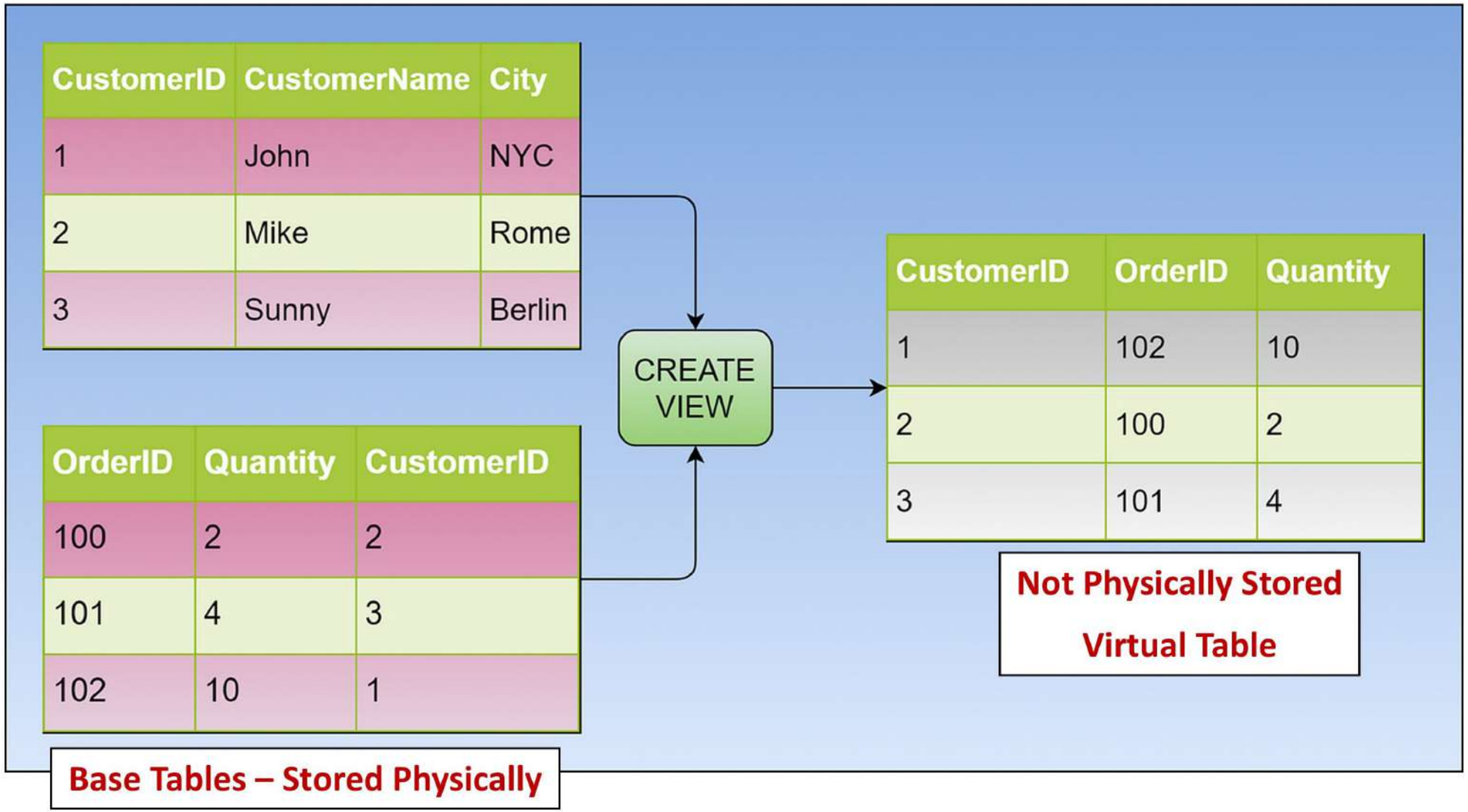
EXISTS checks whether the subquery returns any rows.

It does not return values, only TRUE or FALSE.

Stops execution as soon as a match is found (more efficient for large datasets).

Best suited when checking if a condition is met rather than retrieving values.

Feature	IN	EXISTS
Works on	List of values	Subquery returning rows
Returns	Values from subquery	TRUE or FALSE
Performance	Slower for large data	Faster for large data
Stops early?	No, checks all values	Yes, stops at first match
Use case	When checking for specific values	When checking for existence


CustomerID
CustomerName
City
1
John
NYC
2
Mike
Rome
3
Sunny
Berlin
OrderID
Quantity
CustomerID
100
2
2
101
4
3
102
10
1
CREATE VIEW
CustomerID
OrderID
Quantity
1
102
10
2
100
2
3
101
4
Not Physically Stored
Virtual Table
Base Tables – Stored Physically

```
V1:     CREATE VIEW    WORKS_ON1
        AS SELECT      Fname, Lname, Pname, Hours
             FROM      EMPLOYEE, PROJECT, WORKS_ON
             WHERE     Ssn=Essn AND Pno=Pnumber;
```

VIEWS are not generally subjected to Updation.

VIEWS can be queried just like any other table.

```
SELECT    Fname, Lname
FROM      WORKS_ON1
WHERE     Pname='ProductX';
```

```
V2:     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;
```


DEPT_INFO

Dept_name	No_of_emps	Total_sal

Querying the VIEWS

```
QV1:  SELECT  Fname, Lname
      FROM    WORKS_ON1
      WHERE   Pname='ProductX';
```

```
SELECT  Fname, Lname
FROM    EMPLOYEE, PROJECT, WORKS_ON
WHERE   Ssn=Essn AND Pno=Pnumber
        AND Pname='ProductX';
```

VIEW Materialization

Updating the VIEWS

- A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, as well as all attributes with the NOT NULL constraint *that do not have* default values specified.
- Views defined on multiple tables using joins are generally not updatable.
- Views defined using grouping and aggregate functions are not updatable.

V1A: **DROP VIEW** WORKS_ON1;

Assertions – Specifying General Constraints

```
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 ) );
```

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

The basic technique for writing such assertions is to specify a query that selects any tuples that violate the desired condition.

By including this query inside a NOT EXISTS clause, the assertion will specify that the result of this query must be empty so that the condition will always be TRUE

Thus, the assertion is violated if the result of the query is not empty

SQL triggers are a critical feature in **database management systems (DBMS)** that provide automatic execution of a set of SQL statements when specific database events, such as **INSERT**, **UPDATE**, or **DELETE** operations, occur.

Triggers are commonly used to maintain **data integrity**, **track changes**, and **enforce business rules** automatically, without needing manual input.

For example, a trigger can be invoked when a row is inserted into a specified table or when specific table columns are updated

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
FOR EACH ROW
BEGIN
END;
```

Key Terms

- **trigger_name:** The name of the trigger to be created.
- **BEFORE | AFTER:** Specifies whether the trigger is fired **before** or **after** the triggering event (INSERT, UPDATE, DELETE).
- **{INSERT | UPDATE | DELETE}:** Specifies the operation that will activate the trigger.
- **table_name:** The name of the table the trigger is associated with.
- **FOR EACH ROW:** Indicates that the trigger is row-level, meaning it executes once for each affected row.
- **trigger_body:** The SQL statements to be executed when the trigger is fired.