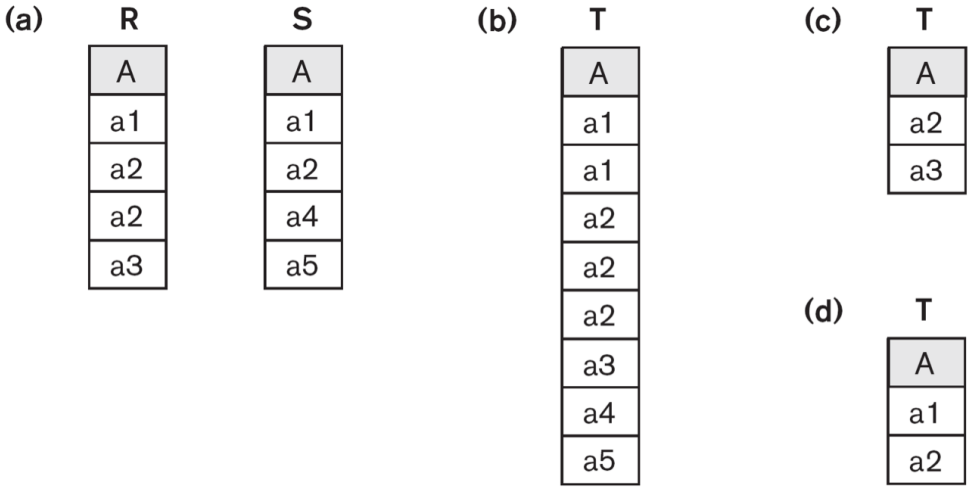
Set Operations in SQL

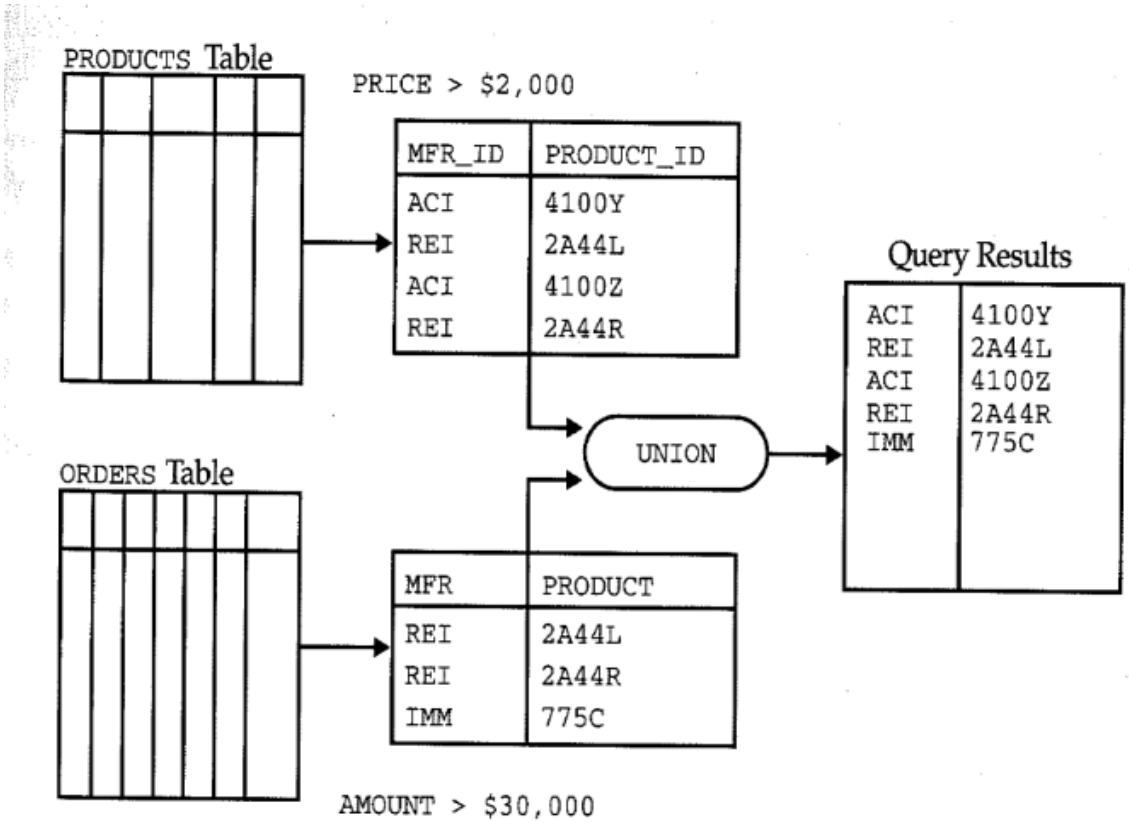
- SQL allows for set operations (set theory)
  - Set union (UNION)
  - Set difference (EXCEPT)
  - Set intersection (INTERSECT)
- Two involved relations must be union-compatible (have same attributes in the same order…)
- Output of these operations are sets of tuples
  - Duplicate tuples are removed from the result
  - Utilize keyword ALL to keep duplicates

Set Operations

- Results of SQL Set operations:
  - (a) – Two tables R(A) and S(A)
  - (b) – R(A) UNION ALL S(A)
  - (c) – R(A) EXCEPT ALL S(A)
  - (d) – R(A) INTERSECT ALL S(A)



UNION Example



Set Operation Example

- 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.

```
(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');
```

Set Operation Example

- 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.

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

* Join can replace Union.

Substring Pattern Matching

- Comparison conditions on parts of a string using LIKE
  - Partial strings are specified using reserved characters:
    - % replaces an arbitrary number of zero or more characters
    - Underscore (_) replaces a single character
  - Examples:
    - Retrieve all employees whose address is in Houston, Texas.

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

    - Find all employees who were born during the 1950s.

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

Arithmetic Operations

- Standard arithmetic operators for addition (+), subtraction (-), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains.
- Example:
  - Suppose that we want to see the effect of giving a 10% raise to all employees.

```
SELECT Fname, Lname, 1.1*Salary
FROM EMPLOYEE;

SELECT Fname, 1.1*Salary AS RaiseSalary
 FROM EMPLOYEE;
```

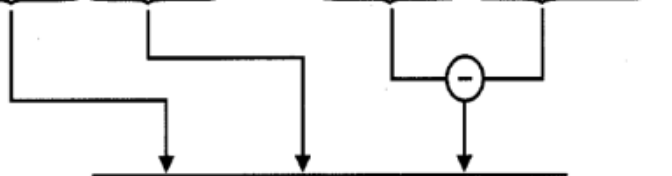
Arithmetic Operations

- Example:

```
SELECT CITY, REGION, (SALES-TARGET)
FROM OFFICES;
```

OFFICES Table

OFFICE	CITY	REGION	MGR	TARGET	SALES
22	Denver	Western	108	$300,000.00	$186,042.00
11	New York	Eastern	106	$575,000.00	$692,637.00
12	Chicago	Eastern	104	$800,000.00	$735,042.00
13	Atlanta	Eastern	NULL	$350,000.00	$367,911.00
21	Los Angeles	Western	108	$725,000.00	$835,915.00



Query Results

CITY	REGION	SALES-TARGET
Denver	Western	-$113,958.00
New York	Eastern	$117,637.00
Chicago	Eastern	-$ 64,958.00
Atlanta	Eastern	$ 17,911.00
Los Angeles	Western	$110,915.00

Queries with NULL

- SQL allows us to check if an attribute is NULL
- Using the comparison operators **IS** or **IS NOT**
  - Example: Retrieve the names of all employees who do not have supervisors

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

Dealing with NULL Values

- SQL uses three-value logic (TRUE, FALSE, UNKNOWN)
- Logical Connectives in Three-Value Logic

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

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

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Membership (IN)

- Testing set membership with the IN

$A = \{x, y, z\}$

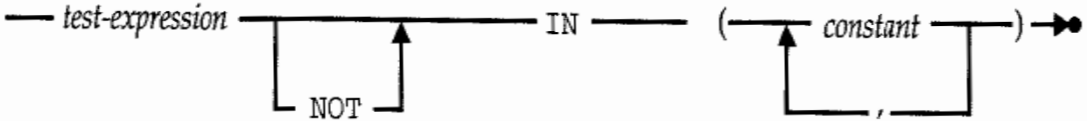
$x \in A$ IN

$i \notin A$ NOT IN

List the salespeople who work in New York, Atlanta, or Denver.

```
SELECT NAME, QUOTA, SALES
  FROM SALESREPS
 WHERE REP_OFFICE IN (11, 13, 22);
```

NAME	QUOTA	SALES
Bill Adams	$350,000.00	$367,911.00
Mary Jones	$300,000.00	$392,725.00
Sam Clark	$275,000.00	$299,912.00
Nancy Angelli	$300,000.00	$186,042.00



Nested Queries

- Sometimes we need to retrieve data from the DB and then use that data in a comparison condition.
- Utilize a nested query with the WHERE is another query
- Example:
  - 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.
    - Same query as in the UNION example
    - Reformulate with nested queries!

$$P\# \in \{ \Pi_{Pno} (\sigma_{lname='smith' \wedge essn=ssn}) \}$$

or

$$P\# \in \{ \Pi_{Pnumber} (\sigma_{mgrssn=ssn \wedge lname=smith \wedge dno=Dnumber}) \}$$

Nested Query Example

```
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 and Attribute Names

- Attribute ambiguity can be an issue with nested queries
- Create aliases for all tables that share attribute names
- Example:
  - Retrieve the name of each employee who has a dependent with the same first name and 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  );
```

EXISTS Function

- Usage: To Check whether the result of a nested query is empty or not $\exists x \in A \quad s.t \quad \langle condition \rangle$
- EXISTS returns TRUE (not empty) or FALSE (empty)
- Example: (Same as previous) $\exists x \in A \; s.t \; \langle condition \rangle$

```
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);
```

What does this query do?

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

NOT EXISTS Function

- Usage: To Check whether the result of a nested query is empty or not
- NOT EXISTS returns TRUE (empty) or FALSE (not empty)
- Example:
  - Retrieve the names of employees who have no dependents

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

Nested, nested queries…..

- Example: Retrieve employees who work on all the projects controlled by department number 5.

Compare this solution to the 4 solutions I showed in class.

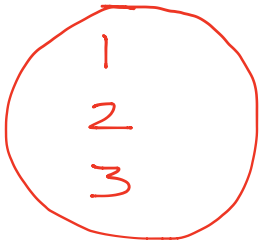
```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE  NOT EXISTS ( SELECT *
                    FROM WORKS_ON AS B
                    WHERE (B.Pno IN ( SELECT Pnumber
                                      FROM  PROJECT
                                      WHERE Dnum=5)
                    AND
                    NOT EXISTS ( SELECT *
                                 FROM WORKS_ON C
                                 WHERE  C.Essn=Ssn AND C.Pno=B.Pno)));
```

A

B

$\nexists (x \in A \text{ and } \nexists (y \in B \text{ and } y = x))$

A: set of projects controlled by dep 5



B: Projects involving emp 'John Smith'

is $A \subseteq B$?

If $A \subseteq B$ then 'John Smith' should be selected.

How to check whether $A \subseteq B$

if $x \in A \Rightarrow x \in B$

Equivalently, $\nexists (x \in A \text{ and } x \notin B)$

$\nexists$: NOT EXISTS; $\in$: IN; $\notin$: NOT IN

Explicit Set of Values

- You are able to use a set of values in the WHERE clause
- Example:
  - Retrieve the Social Security numbers of all employees who work on project numbers 1, 2, or 3.

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     Pno IN (1, 2, 3);
```

Joined Tables

- We've been joining tables together in our queries already. Now we'll examine the JOIN process in more detail.
- In order to JOIN two tables together, we've specified a join condition. (Matching attributes from the two tables)
  - Example from first SQL lecture:

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

Dnumber=Dno is the join condition

JOIN ON

- We can specify the JOIN conditions in the FROM clause instead of specifying them in the WHERE clause.
- Same Example:

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

- We are explicitly stating that we are joining two tables.
- Also allows us to specify different types of joins.