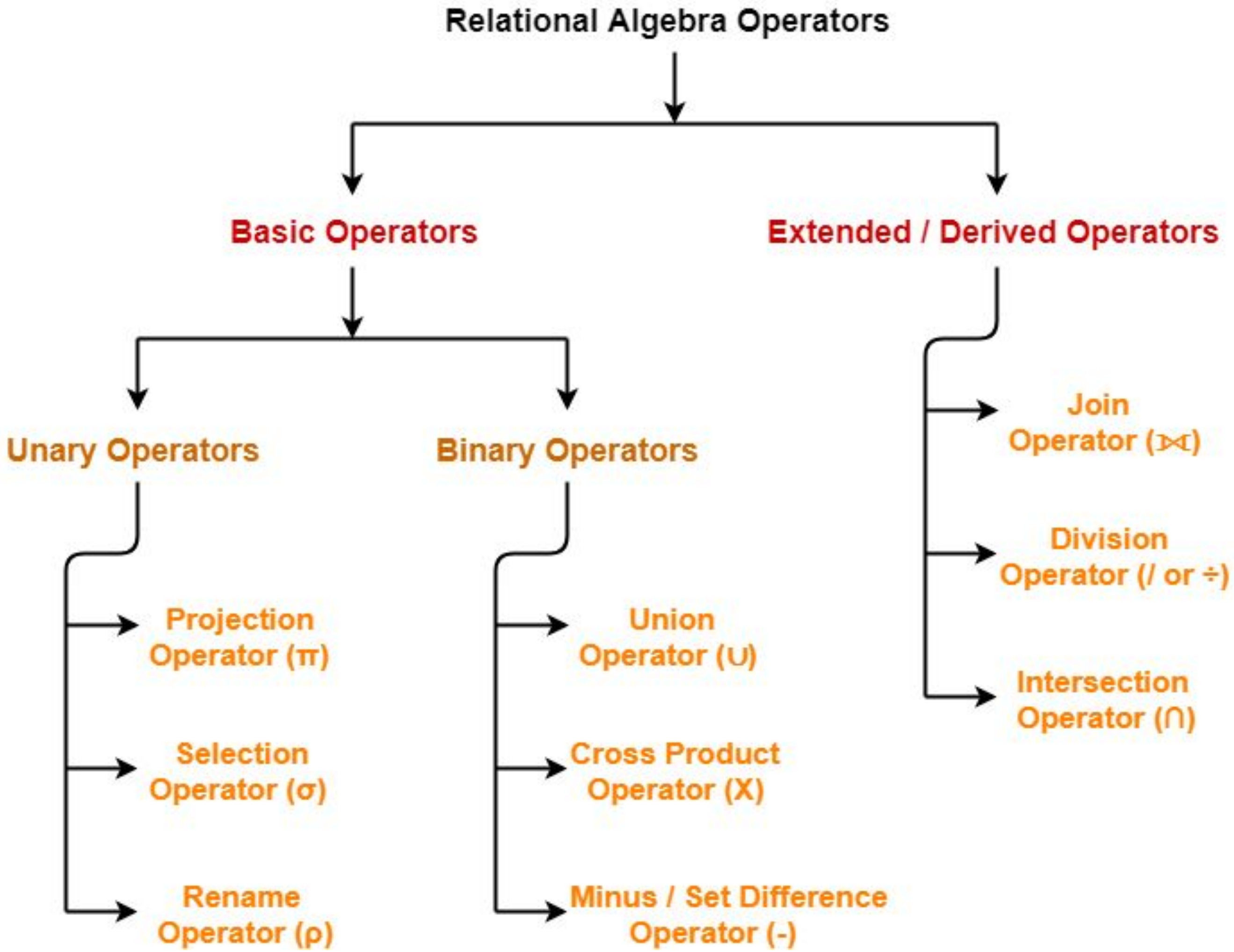
# Databases 2022

Darko Bozhinoski,
Ph.D. in Computer Science

# Agenda

- Data Definition Language (DDL) (Recap)
- Data Manipulation Language (DML)
- SQL Advanced Concepts



Relational Algebra Operators

Basic Operators

Extended / Derived Operators

Unary Operators

Binary Operators

Projection Operator (π)

Selection Operator (σ)

Rename Operator (ρ)

Union Operator (∪)

Cross Product Operator (X)

Minus / Set Difference Operator (-)

Join Operator (⋈)

Division Operator (/ or ÷)

Intersection Operator (∩)

**Two types of operations:**

- set operations from set theory

- operations developed specifically for relational databases

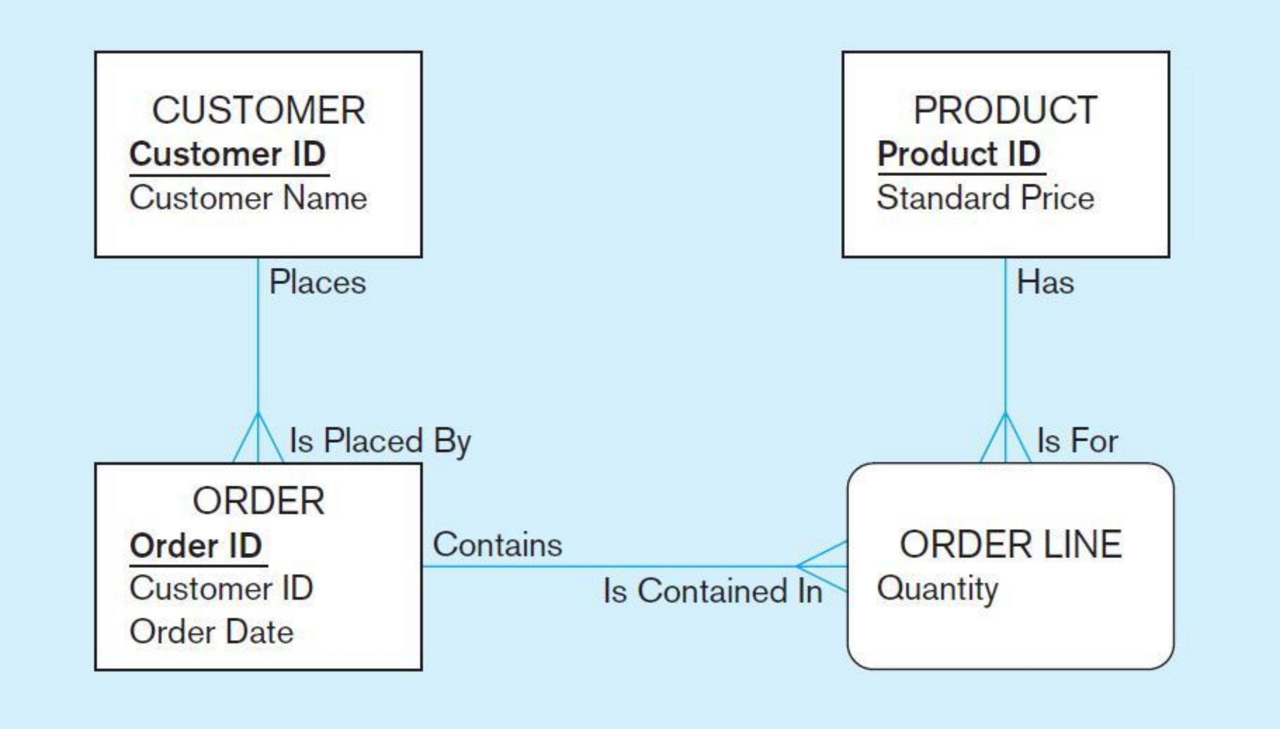
# SQL ENVIRONMENT

- **Catalog**
  - A set of schemas that constitute the description of a database
- **Schema**
  - The structure that contains descriptions of objects created by a user (base tables, views, constraints)
- **Data Definition Language (DDL)**
  - Commands that define a database, including creating, altering, and dropping tables and establishing constraints (CREATE, ALTER, DROP)
- **Data Manipulation Language (DML)**
  - Commands that maintain and query a database (INSERT, UPDATE, DELETE, SELECT)
- **Data Control Language (DCL)**
  - Commands that control a database, including administering privileges and committing data (GRANT, REVOKE)

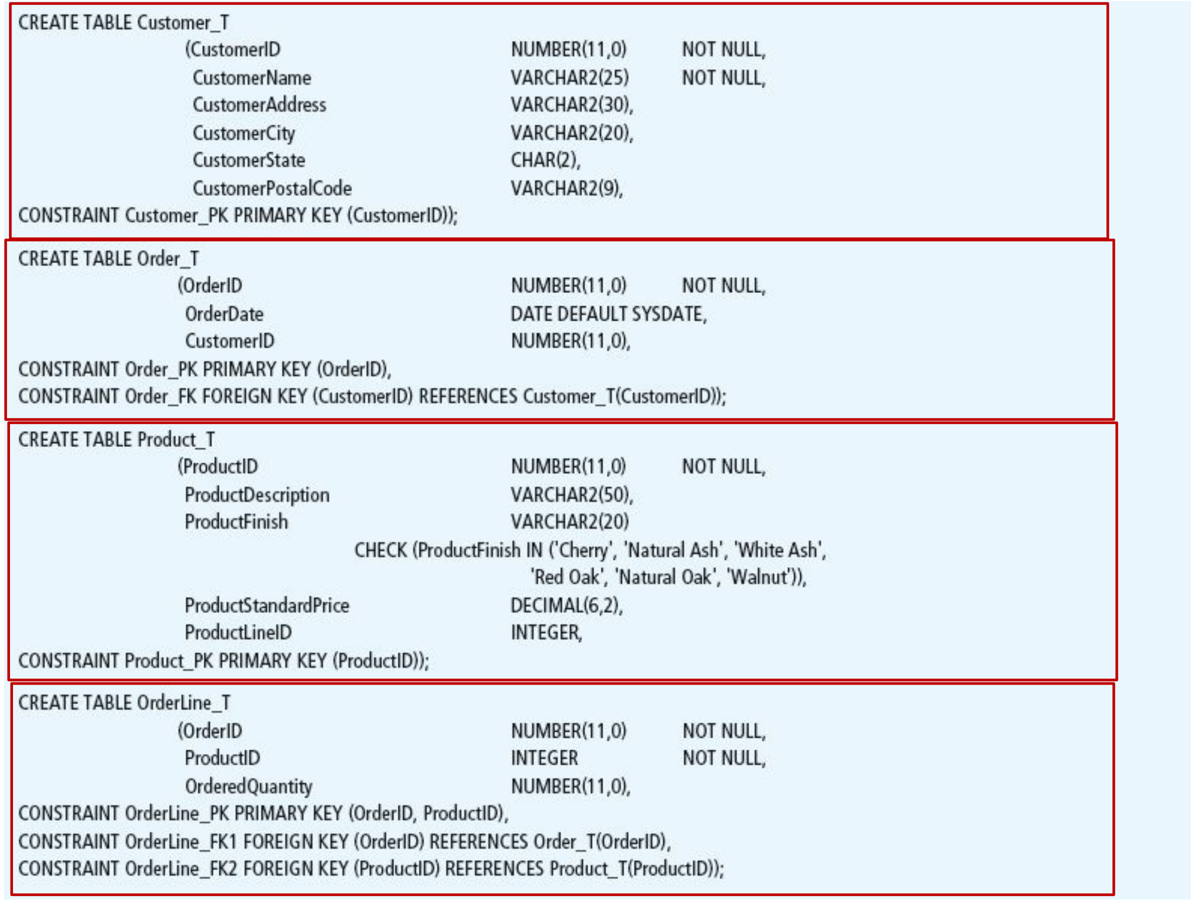
# STEPS IN TABLE CREATION

- Identify data types for attributes
- Identify columns that can and cannot be null
- Identify columns that must be unique (candidate keys)
- Identify primary key–foreign key mates
- Determine default values
- Identify constraints on columns (domain specifications)
- Create the table and associated indexes

# Example DATA MODEL



# SQL database definition commands



```
CREATE TABLE Customer_T
                (CustomerID                    NUMBER(11,0)       NOT NULL,
                 CustomerName                  VARCHAR2(25)       NOT NULL,
                 CustomerAddress               VARCHAR2(30),
                 CustomerCity                  VARCHAR2(20),
                 CustomerState                 CHAR(2),
                 CustomerPostalCode            VARCHAR2(9),
CONSTRAINT Customer_PK PRIMARY KEY (CustomerID));
```

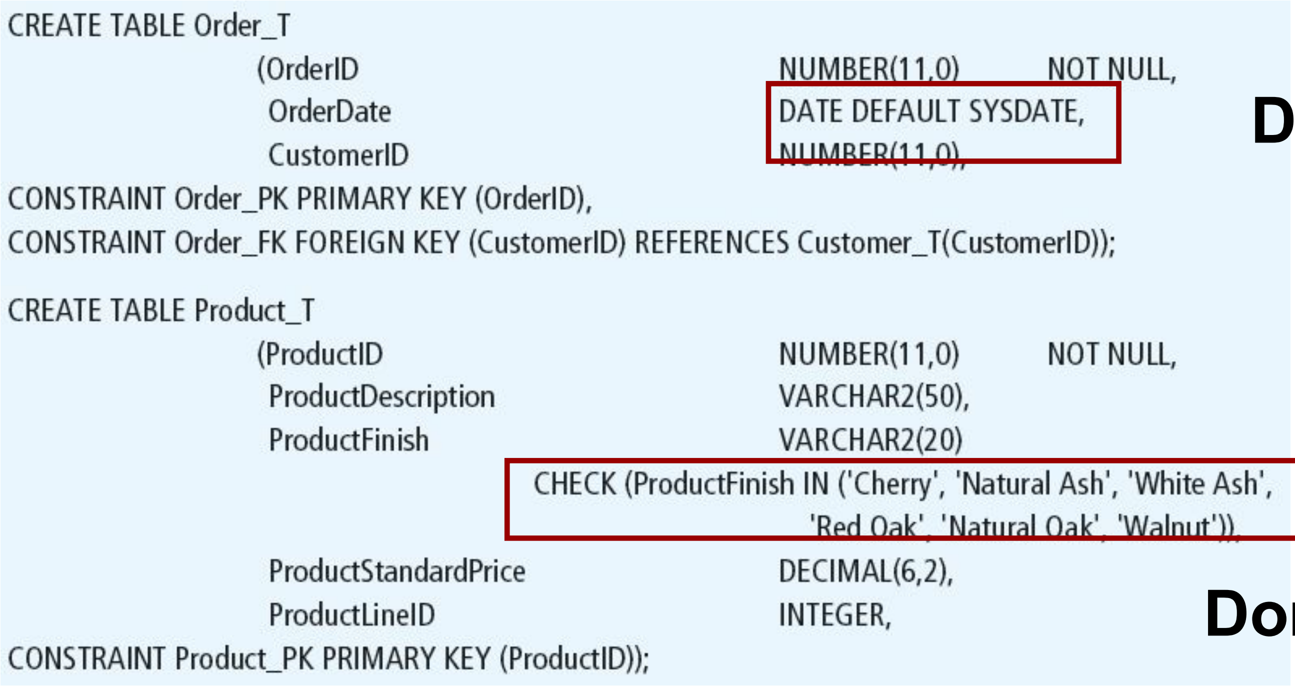
```
CREATE TABLE Order_T
                (OrderID                       NUMBER(11,0)       NOT NULL,
                 OrderDate                     DATE DEFAULT SYSDATE,
                 CustomerID                    NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));
```

```
CREATE TABLE Product_T
                (ProductID                     NUMBER(11,0)       NOT NULL,
                 ProductDescription            VARCHAR2(50),
                 ProductFinish                 VARCHAR2(20)
                              CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                                  'Red Oak', 'Natural Oak', 'Walnut')),
                 ProductStandardPrice          DECIMAL(6,2),
                 ProductLineID                 INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

```
CREATE TABLE OrderLine_T
                (OrderID                       NUMBER(11,0)       NOT NULL,
                 ProductID                     INTEGER            NOT NULL,
                 OrderedQuantity               NUMBER(11,0),
CONSTRAINT OrderLine_PK PRIMARY KEY (OrderID, ProductID),
CONSTRAINT OrderLine_FK1 FOREIGN KEY (OrderID) REFERENCES Order_T(OrderID),
CONSTRAINT OrderLine_FK2 FOREIGN KEY (ProductID) REFERENCES Product_T(ProductID));
```

**Overall table definitions**

# Controlling the values in attributes



```
CREATE TABLE Order_T
                    (OrderID                                   NUMBER(11,0)       NOT NULL,
                     OrderDate                                 DATE DEFAULT SYSDATE,
                     CustomerID                                NUMBER(11,0),
CONSTRAINT Order_PK PRIMARY KEY (OrderID),
CONSTRAINT Order_FK FOREIGN KEY (CustomerID) REFERENCES Customer_T(CustomerID));

CREATE TABLE Product_T
                    (ProductID                                 NUMBER(11,0)       NOT NULL,
                     ProductDescription                        VARCHAR2(50),
                     ProductFinish                             VARCHAR2(20)
                                        CHECK (ProductFinish IN ('Cherry', 'Natural Ash', 'White Ash',
                                                       'Red Oak', 'Natural Oak', 'Walnut')),
                     ProductStandardPrice                      DECIMAL(6,2),
                     ProductLineID                             INTEGER,
CONSTRAINT Product_PK PRIMARY KEY (ProductID));
```

**Default value**

**Domain constraint**

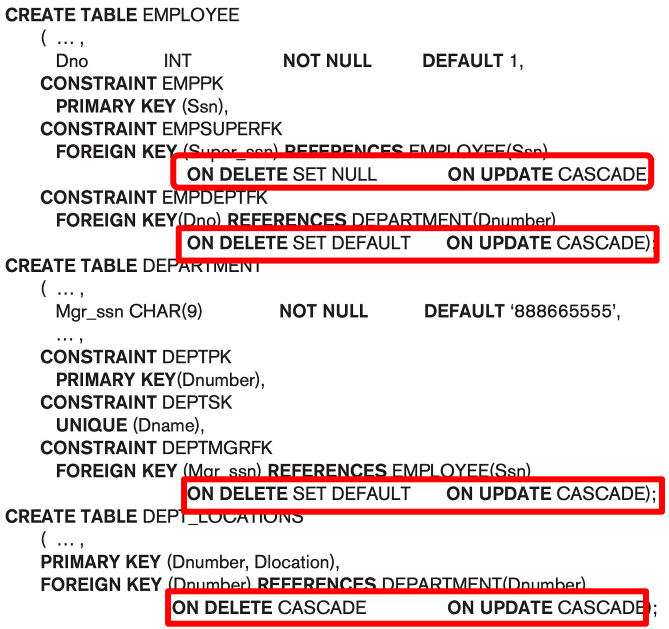
# DATA INTEGRITY CONTROLS

- ❖ Referential integrity–constraint that ensures that foreign key values of a table must match primary key values of a related table in 1:M relationships
- ❖ Restricting:
  - ➢ Deletes of primary records
  - ➢ Updates of primary records
  - ➢ Inserts of dependent records

# Key and Referential Integrity Constraints

The schema designer can specify an alternative action to be taken by attaching a referential triggered action clause to any foreign key constraint. The options include:
SET NULL, CASCADE, and SET DEFAULT.

An option must be qualified with either ON DELETE or ON UPDATE.



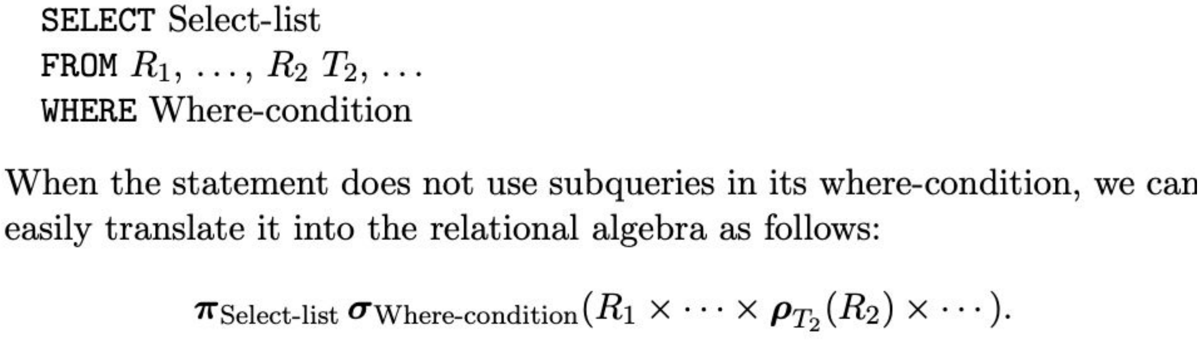
```
CREATE TABLE EMPLOYEE
    ( … ,
      Dno          INT           NOT NULL      DEFAULT 1,
    CONSTRAINT EMPPK
      PRIMARY KEY (Ssn),
    CONSTRAINT EMPSUPERFK
      FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn)
                  ON DELETE SET NULL        ON UPDATE CASCADE,
    CONSTRAINT EMPDEPTFK
      FOREIGN KEY(Dno) REFERENCES DEPARTMENT(Dnumber)
                  ON DELETE SET DEFAULT     ON UPDATE CASCADE);
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);
CREATE TABLE DEPT_LOCATIONS
    ( … ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                ON DELETE CASCADE           ON UPDATE CASCADE);
```

# SQL Query

# Basic SQL Query

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

# SQL Query to Relational Algebra Basic



SELECT Select-list
FROM $R_1$, $\ldots$, $R_2$ $T_2$, $\ldots$
WHERE Where-condition

When the statement does not use subqueries in its where-condition, we can easily translate it into the relational algebra as follows:

$$\boldsymbol{\pi}_{\text{Select-list}}\, \boldsymbol{\sigma}_{\text{Where-condition}}(R_1 \times \cdots \times \boldsymbol{\rho}_{T_2}(R_2) \times \cdots).$$

# Simple SQL Query

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT   *
FROM     Product
WHERE    category=‘Gadgets’
```

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |

“selection”

# Simple SQL Query

Product

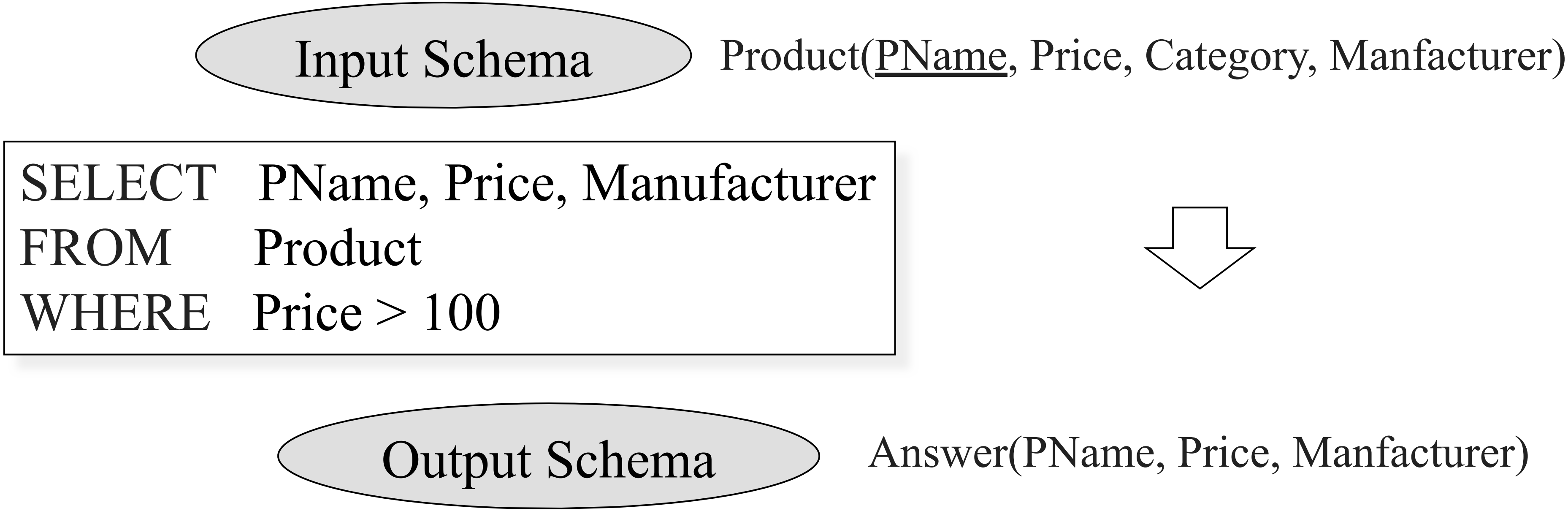
| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT   PName, Price,
Manufacturer
FROM     Product
WHERE    Price > 100
```

| PName | Price | Manufacturer |
|---|---|---|
| SingleTouch | $149.99 | Canon |
| MultiTouch | $203.99 | Hitachi |

“selection” and “projection”

# Notation



Product(PName, Price, Category, Manfacturer)

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

Answer(PName, Price, Manfacturer)

# The **LIKE** operator

```
SELECT   *
FROM     Products
WHERE    PName LIKE ‘%gizmo%’
```

- s **LIKE** p:  pattern matching on strings
- p may contain two special symbols:
  - %  = any sequence of characters
  - _   = any single character

# Eliminating Duplicates

```
SELECT   DISTINCT category
FROM     Product
```

→

| Category |
|---|
| Gadgets |
| Photography |
| Household |

Compare to:

```
SELECT   category
FROM     Product
```

→

| Category |
|---|
| Gadgets |
| Gadgets |
| Photography |
| Household |

# Ordering the Results

```
SELECT   pname, price, manufacturer
FROM     Product
WHERE    category=‘gizmo’ AND price > 50
ORDER BY  price, pname
```

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

⇒ ?

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

→ ?

```
SELECT   Category
FROM     Product
ORDER BY  PName
```

→ ?

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

```
SELECT   DISTINCT category
FROM     Product
ORDER BY category
```

⇨ ?

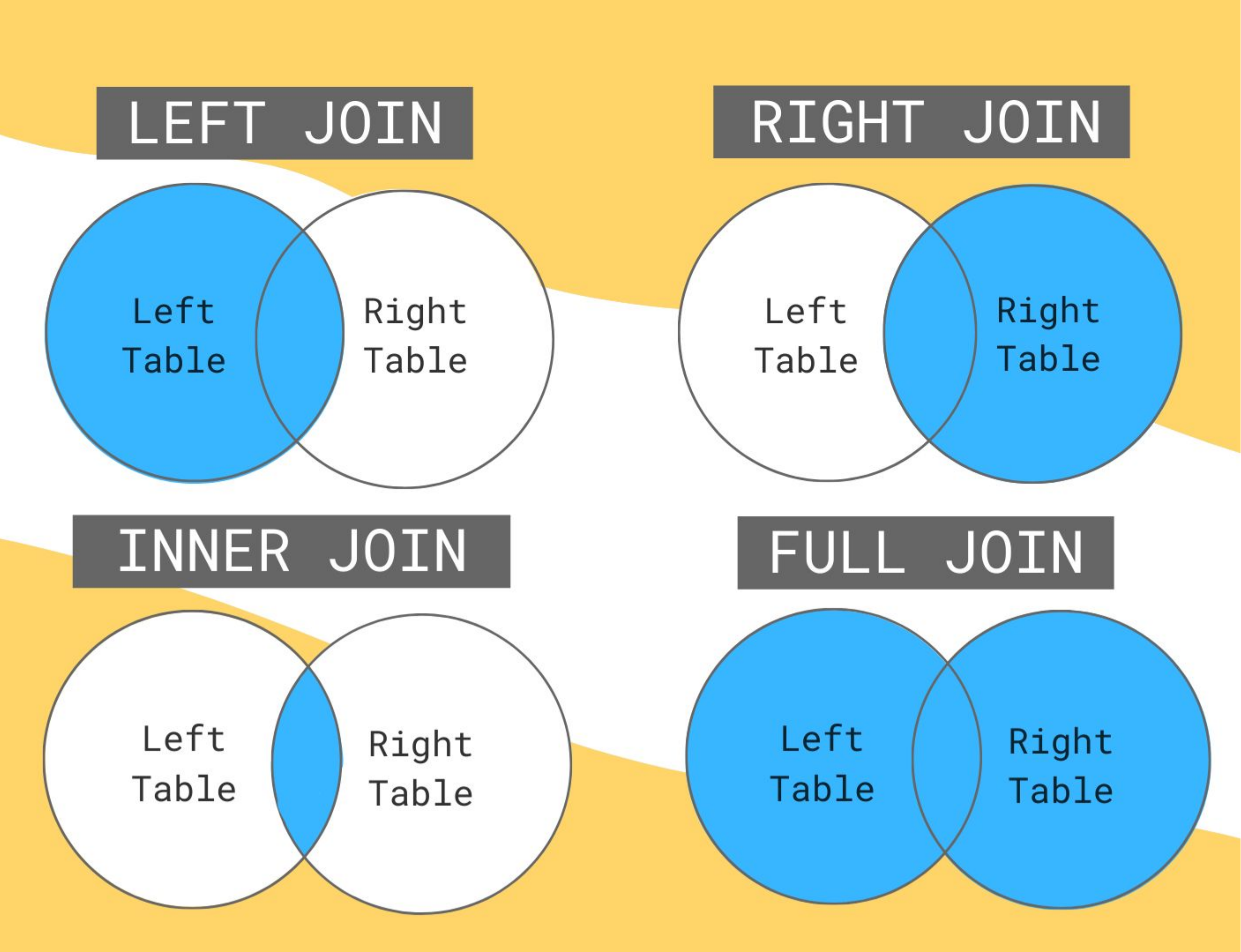
```
SELECT   Category
FROM     Product
ORDER BY  PName
```

⇨ ?

```
SELECT   DISTINCT category
FROM     Product
ORDER BY PName
```

⇨ ?

# JOINS



LEFT JOIN

RIGHT JOIN

INNER JOIN

FULL JOIN

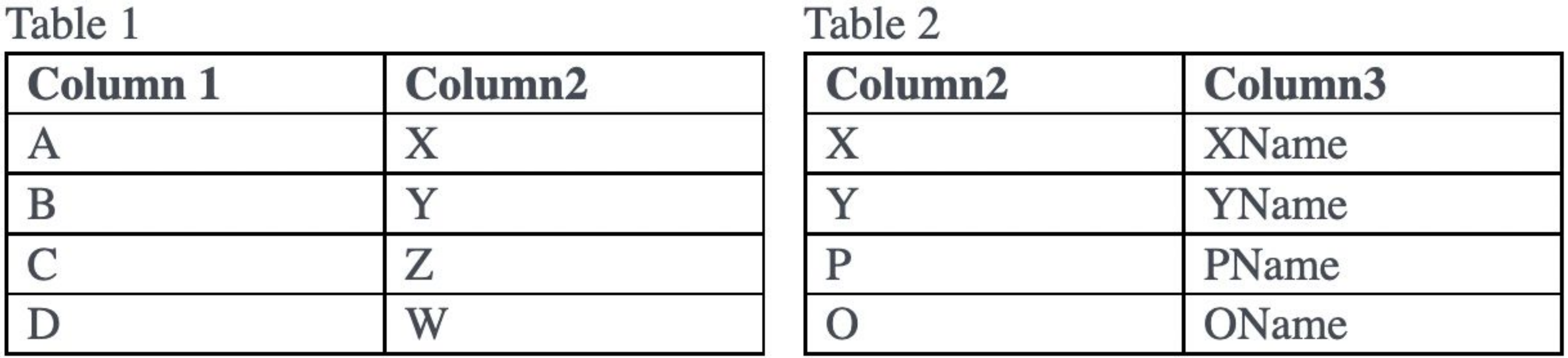


Table 1

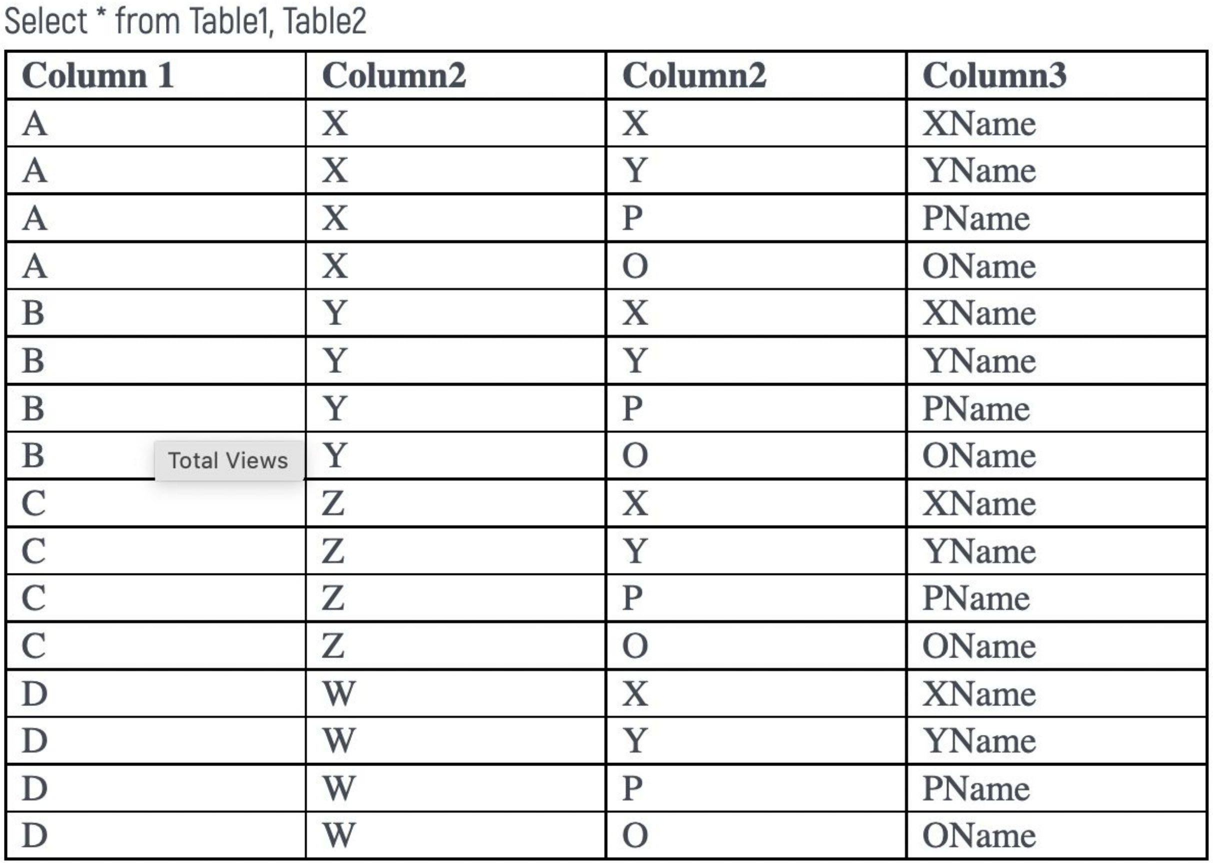
| Column 1 | Column2 |
|---|---|
| A | X |
| B | Y |
| C | Z |
| D | W |

Table 2

| Column2 | Column3 |
|---|---|
| X | XName |
| Y | YName |
| P | PName |
| O | OName |

**Difference between Cross Product and Full Outer Join?**

# Cartesian Product

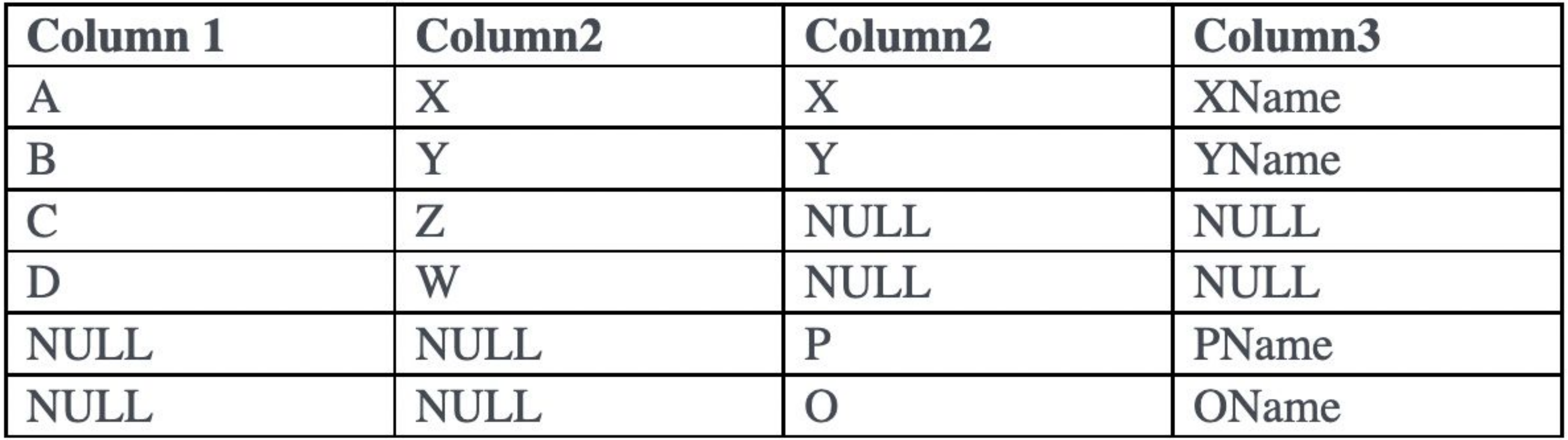


```
Select * from Table1, Table2
```

| Column 1 | Column2 | Column2 | Column3 |
|---|---|---|---|
| A | X | X | XName |
| A | X | Y | YName |
| A | X | P | PName |
| A | X | O | OName |
| B | Y | X | XName |
| B | Y | Y | YName |
| B | Y | P | PName |
| B | Y | O | OName |
| C | Z | X | XName |
| C | Z | Y | YName |
| C | Z | P | PName |
| C | Z | O | OName |
| D | W | X | XName |
| D | W | Y | YName |
| D | W | P | PName |
| D | W | O | OName |

A cross join produces a cartesian product between the two tables, returning all possible combinations of all rows.

# FULL OUTER JOIN



| Column 1 | Column2 | Column2 | Column3 |
|---|---|---|---|
| A | X | X | XName |
| B | Y | Y | YName |
| C | Z | NULL | NULL |
| D | W | NULL | NULL |
| NULL | NULL | P | PName |
| NULL | NULL | O | OName |

**FULL OUTER JOIN:**
- It includes all the rows from both the tables.
- Assigns NULL for unmatched fields.
- A combination of both left and right outer joins.

# JOINS

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and prices.

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

# JOINS (2)

Product

| PName | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

Company

| Cname | StockPrice | Country |
|---|---|---|
| GizmoWorks | 25 | USA |
| Canon | 65 | Japan |
| Hitachi | 15 | Japan |

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country=‘Japan’
         AND Price <= 200
```

| PName | Price |
|---|---|
| SingleTouch | $149.99 |

# JOINS (3)

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all Russian companies that manufacture products both in the ‘electronic’ and ‘toy’ categories

```
SELECT   cname

FROM

WHERE
```

# JOINS (4)

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all countries that manufacture some product in the ‘Gadgets’ category.

```
SELECT   Country
FROM     Product, Company
WHERE    Manufacturer=CName AND Category=‘Gadgets’
```

# JOINS (5)

Product

| Name | Price | Category | Manufacturer |
|---|---|---|---|
| Gizmo | $19.99 | Gadgets | GizmoWorks |
| Powergizmo | $29.99 | Gadgets | GizmoWorks |
| SingleTouch | $149.99 | Photography | Canon |
| MultiTouch | $203.99 | Household | Hitachi |

Company

| Cname | StockPrice | Country |
|---|---|---|
| GizmoWorks | 25 | USA |
| Canon | 65 | Japan |
| Hitachi | 15 | Japan |

```
SELECT   Country
FROM     Product, Company
WHERE    Manufacturer=CName AND Category='Gadgets'
```

What is
the problem ?
What's the
solution ?

| Country |
|---|
| ?? |
| ?? |
| |

# OUTER JOINS

Explicit joins in SQL = “inner joins”:

Product(name, category)
Purchase(prodName, store)

```
SELECT Product.name, Purchase.store
FROM     Product JOIN Purchase ON
                   Product.name = Purchase.prodName
```

Same as:

```
SELECT Product.name, Purchase.store
FROM     Product, Purchase
WHERE   Product.name = Purchase.prodName
```

But Products that never sold will be lost !

# OUTER JOINS

Left outer joins in SQL:

Product(name, category)  
Purchase(prodName, store)

```
SELECT Product.name, Purchase.store
FROM    Product LEFT OUTER JOIN Purchase ON
                  Product.name = Purchase.prodName
```

```
SELECT Product.name, Purchase.store
FROM    Product LEFT OUTER JOIN Purchase ON
                     Product.name = Purchase.prodName
```

Product

| Name | Category |
|---|---|
| Gizmo | gadget |
| Camera | Photo |
| OneClick | Photo |

Purchase

| ProdName | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |

| Name | Store |
|---|---|
| Gizmo | Wiz |
| Camera | Ritz |
| Camera | Wiz |
| OneClick | NULL |

# Example

Compute, for each product, the total number of sales in ‘September’

Product(name, category)

Purchase(prodName, month, store)

```
 SELECT Product.name, count(*)
 FROM     Product, Purchase
 WHERE   Product.name =
Purchase.prodName
         and  Purchase.month = ‘September’
 GROUP BY Product.name
```

What’s wrong ?

# Example

Compute, for each product, the total number of sales in ‘September’

Product(name, category)

Purchase(prodName, month, store)

```
SELECT Product.name, count(*)
FROM     Product LEFT OUTER JOIN Purchase ON
                   Product.name = Purchase.prodName
               and  Purchase.month = ‘September’
GROUP BY Product.name
```

Now we also get the products who sold in 0 quantity

# Tuple Variables

Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

Which address is this?

# Tuple Variables

Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

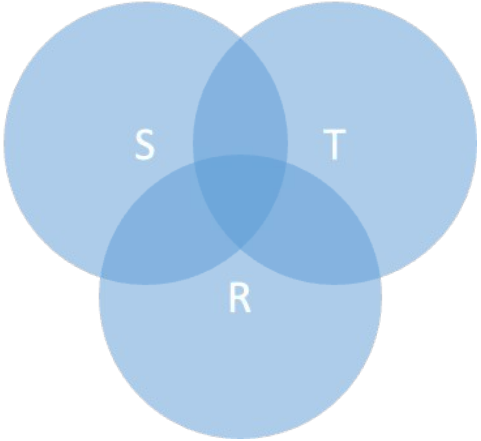
```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company
WHERE    Person.worksfor = Company.cname
```

```
SELECT  DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A    OR   R.A=T.A
```

**What does it compute ?**

```
SELECT  DISTINCT R.A
FROM   R, S, T
WHERE  R.A=S.A    OR   R.A=T.A
```

**What does it compute ?**



**Computes** R ∩ (S ∪ T)

# Subqueries Returning Relations

**Company(name, city)**
**Product(pname, maker)**
**Purchase(id, product, buyer)**

Return cities where one can find companies that manufacture products bought by Ivan Ivanov

```
SELECT  Company.city
FROM    Company
WHERE   Company.name  IN
            (SELECT Product.maker
              FROM   Purchase , Product
              WHERE Product.pname=Purchase.product
                  AND Purchase .buyer = ‘Ivan Ivanov‘);
```

# Subqueries Returning Relations

```
SELECT  Company.city
FROM       Company, Product, Purchase
WHERE   Company.name= Product.maker
           AND  Product.pname  = Purchase.product
           AND  Purchase.buyer = ‘Ivan Ivanov’
```

# Subqueries Returning Relations

```
SELECT  Company.city
FROM      Company, Product, Purchase
WHERE   Company.name= Product.maker
          AND  Product.pname  = Purchase.product
          AND  Purchase.buyer = ‘Ivan Ivanov’
```

Beware of duplicates !

# Removing Duplicates

```
SELECT DISTINCT Company.city
FROM     Company
WHERE  Company.name  IN
                  (SELECT Product.maker
                   FROM   Purchase , Product
                   WHERE Product.pname=Purchase.product
                       AND Purchase .buyer = ‘Joe Blow‘);
```

```
SELECT DISTINCT Company.city
FROM      Company, Product, Purchase
WHERE   Company.name= Product.maker
          AND  Product.pname  = Purchase.product
          AND  Purchase.buyer = ‘Joe Blow’
```

# Subqueries Returning Relations

You can also use: **s > ALL R**<br>
**s > ANY R**<br>
**EXISTS R**

Product ( pname, price, category, maker)

Find products that are more expensive than all those produced By "Gizmo-Works"

```
SELECT  name
FROM    Product
WHERE   price >  ALL (SELECT price
                      FROM    Purchase
                      WHERE   maker='Gizmo-Works')
```

# Question

- Can we express this query as a single SELECT-FROM-WHERE query, without subqueries ?

# Reading Material

- Fundamentals of Database Systems. Ramez Elmasri and Shamkant B. Navathe. Pearson. **Chapter 6. and Chapter 7.**
- SQL Tutorial: **https://www.w3schools.com/sql/default.asp**

Q
&
A