CSE544
Data Management

Lectures 2: SQL

Relational Data Model: Recap

- Database: collection of relations

$$R_1, R_2, \ldots, R_m$$

- Relation (aka Table): set of tuples

$$R = \{t_1, t_2, \ldots, t_n\}$$

- Tuple (aka row, record):

$$\mathrm{t} \in \mathrm{Dom}_1 \times \cdots \times \mathrm{Dom}_\mathrm{k}$$

SQL

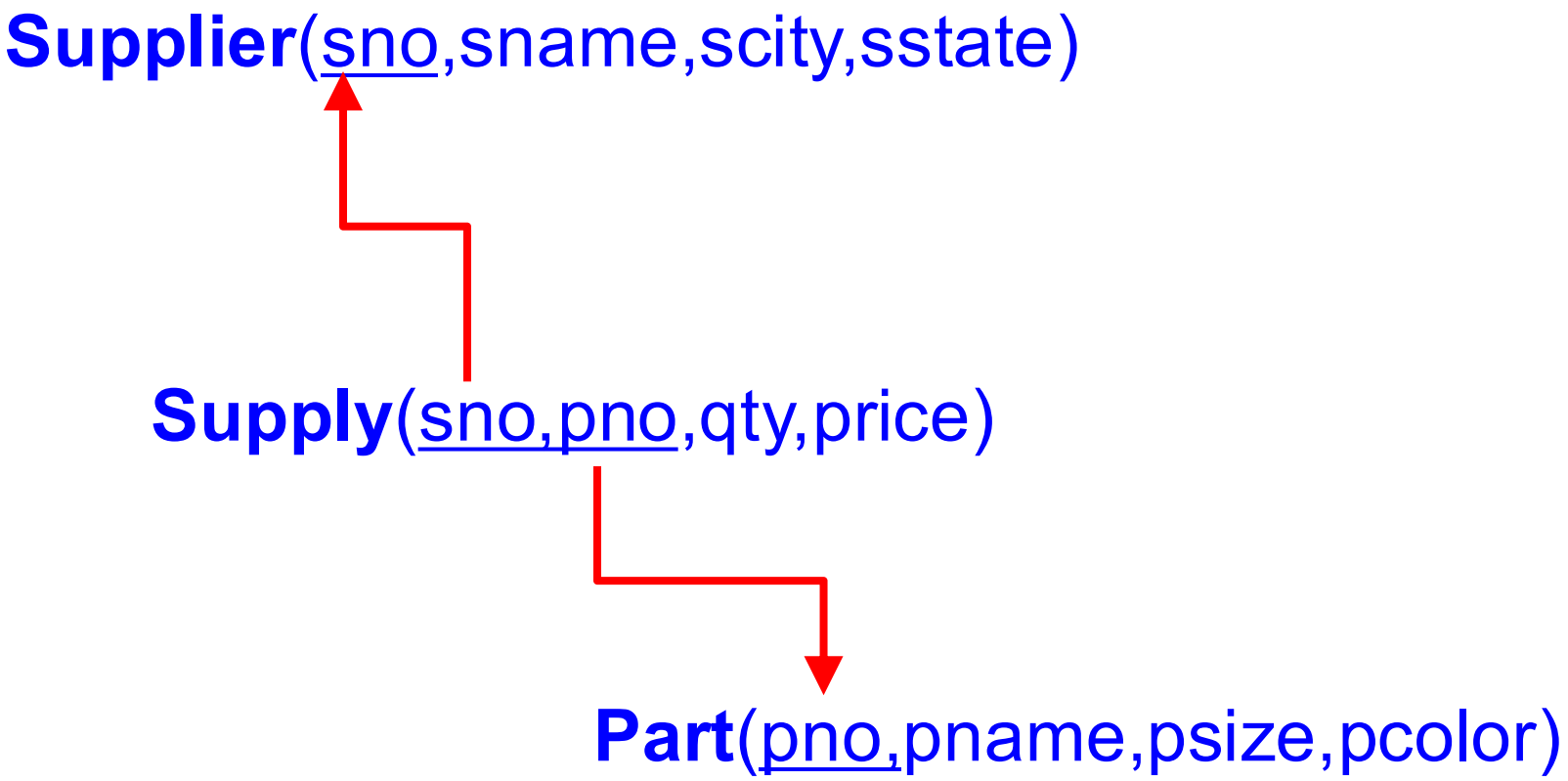
SQL

- Philosophy from the 70's:
  walk-up and read

- Data Definition Language (DDL):
  - Quick overview now, then on your own

- Data Manipulation Language (DML)

Relational Data Model



```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Relational Data Model

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

Relational Data Model

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

```
CREATE TABLE
    SUPPLIER(sno int,
             sname text,
             scity text,
             sstate text);
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

```
CREATE TABLE
    SUPPLIER(sno int,
             sname text,
             scity text,
             sstate text);
```

Creates an empty table:

sno	sname	scity	sstate

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text,
             scity text,
             sstate text);
```

Creates an empty table:

sno	sname	scity	sstate

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text,
             scity text,
             sstate text);
```

```
CREATE TABLE
    Part(pno int primary key,
         pname text,
         psize int,
         pcolor text);
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text
```

```
CREATE TABLE
    Supply(sno int,
           pno int,
           qty int,
           price int);
```

```
ary key,

           pcolor text);
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text
```

```
CREATE TABLE
    Supply(sno int references Supplier,
           pno int references Part,
           qty int,
           price int);
```

```
ary key,
           pcolor text);
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL DDL

```
CREATE TABLE
    SUPPLIER(sno int primary key,
             sname text
```

```
CREATE TABLE
    Supply(sno int references Supplier,
           pno int references Part,
           qty int,
           price int,
           primary key (sno, pno));
```

```
ary key,
           pcolor text);
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL DML

```
INSERT INTO Supplier VALUES
    (11,'ACME','Seattle','WA'),
    (12,'Walmart','Portland','OR'),
    (13,'Walmart','Seattle','WA');
```

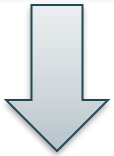
Or import from a CSV file, as in HW1

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL DML

```
INSERT INTO Supplier VALUES
    (11,'ACME','Seattle','WA'),
    (12,'Walmart','Portland','OR'),
    (13,'Walmart','Seattle','WA');
```

Or import from a CSV file, as in HW1



Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Walmart	Seattle	WA

Summary So Far

- SQL DDL:
  - CREATE TABLE
  - DROP TABLE
  - ALTER TABLE
  - CREATE INDEX (next lecture)
- SQL DML:
  - INSERT/DELETE/UPDATE
  - SELECT-FROM-WHERE: next!

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL

```
SELECT …columns…
FROM …tables…
WHERE …condition…
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Walmart	Seattle	WA

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL

Return all supplier names

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Walmart	Seattle	WA

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL

Return all supplier names

```
SELECT sname
FROM Supplier
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Walmart	Seattle	WA

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL

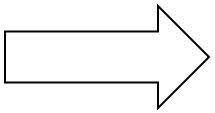
Return all supplier names

On your screen:

```
SELECT sname
FROM Supplier
```

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Walmart	Seattle	WA



sname
ACME
Walmart
Walmart

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL

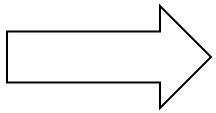
Return all supplier names

```
SELECT sname
FROM Supplier
```

This is a bag

Supplier

sno	sname	scity	sstate
11	ACME	Seattle	WA
12	Walmart	Portland	OR
13	Walmart	Seattle	WA



sname
ACME
Walmart
Walmart

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL

Remove duplicates

```
SELECT DISTINCT sname
FROM Supplier
```

sname
ACME
Walmart
~~Walmart~~

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL

Remove duplicates

```
SELECT DISTINCT sname
FROM Supplier
```

Now it's a set

sname
ACME
Walmart

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL

```
SELECT sname
FROM Supplier
```

```
SELECT sname, scity
FROM Supplier
```

What do these queries return?

```
SELECT *
FROM Supplier
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: WHERE

```
SELECT *
FROM Supplier
WHERE sstate = 'WA'
```

Returns only suppliers in Washington State

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Discussion

- Keywords, table/attribute names are case insensitive:
  - SELECT, select, sElEcT
  - Supplier, SUPPLIER, …
- Strings are case sensitive:
  - ‘WA’ different from ‘wa’
- WHERE conditions can use complex predicates
  - WHERE psize>15 and (pcolor=‘red’ or pcolor=‘blue’)
- SQL has lots of built-in predicates; look them up!
  - WHERE sname LIKE ‘%mart%’

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in 'WA' that supply 'red' parts

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT
FROM
WHERE
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT
FROM    Supplier x, Supply y, Part z
WHERE
```

Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)

SQL: Joins

Find all suppliers in 'WA' that supply 'red' parts

```
SELECT
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT
FROM    Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = ‘WA’
        and z.pcolor = ‘red’;
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in ‘WA’ that supply ‘red’ parts

```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM      Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = ‘WA’
        and z.pcolor = ‘red’;
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

SQL: Joins

Find all suppliers in 'WA' that supply 'red' parts

```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM     Supplier x, Supply y, Part z
WHERE    x.sno = y.sno
         and y.pno = z.pno
         and x.sstate = 'WA'
         and z.pcolor = 'red';
```

What happens if we don't include these conditions?

Discussion

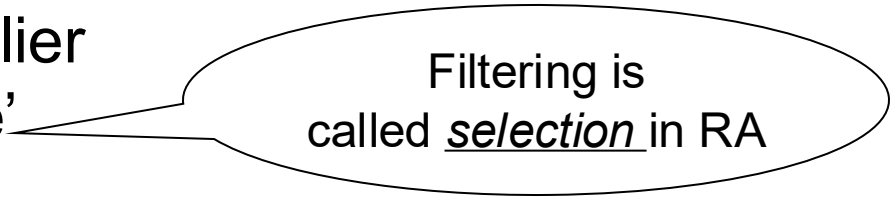
- Keys and Foreign Keys:
  - Used only to enforce data integrity
  - Not used in SELECT-FROM-WHERE

- Tuple variables `Supplier` **x**

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Terminology

- Selection/filter: return a subset of the rows:
  - SELECT * FROM Supplier
    WHERE scity = 'Seattle'



- Projection: return subset of the columns:
  - SELECT DISTINCT scity FROM Supplier;

- Join: refers to combining two or more tables
  - SELECT * FROM Supplier, Supply, Part …

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle and suppliers in Portland

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle and suppliers in Portland

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT
FROM    Supplier x, Supply y
WHERE
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

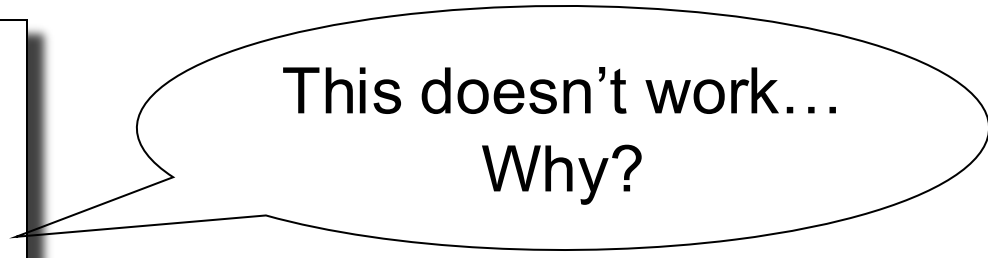
```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    x.scity = 'Seattle'
         and x.scity = 'Portland'
         and x.sno = y.sno
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    x.scity = 'Seattle'
         and x.scity = 'Portland'
         and x.sno = y.sno
```



```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    (x.scity = 'Seattle'
             or x.scity = 'Portland')
          and x.sno = y.sno
```

Does this work?

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y.pno
FROM      Supplier x, Supply y
WHERE    (x.scity = 'Seattle'
             or x.scity = 'Portland')
          and x.sno = y.sno
```

Does this work?

Nope!

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

Need TWO Suppliers and TWO Supplies

```
SELECT DISTINCT  y1.pno
FROM     Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE    x1.scity = 'Seattle'
         and x1.sno = y1.sno
         and x2.scity = 'Portland'
         and x2.sno = y2.sno
         and y1.pno = y2.pno
```

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y1.pno
FROM     Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE    x1.scity = 'Seattle'
         and x1.sno = y1.sno
         and x2.scity = 'Portland'
         and x2.sno = y2.sno
         and y1.pno = y2.pno
```

Need TWO Suppliers and TWO Supplies

one in Seattle the other in Portland

```
Supplier(sno,sname,scity,sstate)
Supply(sno,pno,qty,price)
Part(pno,pname,psize,pcolor)
```

Self-Joins

Find the Part numbers available both from suppliers in Seattle, and suppliers in Portland

```
SELECT DISTINCT  y1.pno
FROM     Supplier x1, Supplier x2, Supply y1, Supply y2
WHERE    x1.scity = 'Seattle'
         and x1.sno = y1.sno
         and x2.scity = 'Portland'
         and x2.sno = y2.sno
         and y1.pno = y2.pno
```

Need TWO Suppliers and TWO Supplies

one in Seattle the other in Portland

the SAME part

Recap

Syntax:

- FROM clause: cartesian product
- WHERE clause: filter out
- SELECT clause: says what to return

Semantics: next

Semantics

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**
  for x_2 **in** R_2 **do**
    …..
      for x_n **in** R_n **do**
        if Conditions
          then Answer = Answer $\cup$ $\{(a_1,\ldots,a_k)\}$
return Answer

Nested-Loop Semantics of SQL

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 AS x_1, R_2 AS x_2, …, R_n AS x_n
WHERE Conditions

This is SEMANTICS! It is NOT how the engine computes the query!

```
Answer = {}
for x1 in R1 do
    for x2 in R2 do
        …..
            for xn in Rn do
                if Conditions
                    then Answer = Answer ∪ {(a1,…,ak)}
return Answer
```

Discussion

- SQL engines choose much more efficient plans than nested loops

Discussion

- SQL engines choose much more efficient plans than nested loops
- Discuss possible execution plans for:

```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM      Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.sstate = 'WA'
        and z.pcolor = 'red';
```

NULLs

NULLs in SQL

- A NULL value means missing, or unknown, or undefined, or inapplicable

Part(pno,pname,price,psize,pcolor)

NULLs in WHERE Clause

Boolean predicate:

- Atomic: Expr1 op Expr2
- AND / OR / NOT

price < 100 and (pcolor='red' or psize=2)

Part(pno,pname,price,psize,pcolor)

NULLs in WHERE Clause

Boolean predicate:

- Atomic: Expr1 op Expr2
- AND / OR / NOT

price < 100 and (pcolor='red' or psize=2)

How do we compute the predicate when values are NULL?

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL

Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL

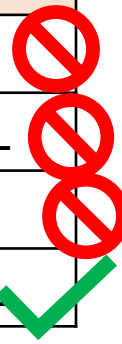
Part(pno,pname,price,psize,pcolor)

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
select *
from Part
where price < 100
 and (psize=2 or pcolor='red')
```

pno	pname	price	psize	pcolor
1	iPad	500	13	blue
2	Scooter	99	NULL	NULL
3	Charger	NULL	NULL	red
1	iPad	50	2	NULL



Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
-- problem: (A or not(A)) ≠ true
-- does NOT return all Products
select *
from Product
where (price <= 100) or (price > 100)
```

Three-Valued Logic

- False=0, Unknown=0.5, True=1
- A = B (or A > B or …)
  - **False** or **True** when both A, B are not null
  - **Unknown** otherwise
- AND, OR, NOT are **min**, **max**.
- Return only tuples whose condition is **True**

```
-- problem: (A or not(A)) ≠ true
-- does NOT return all Products
select *
from Product
where (price <= 100) or (price > 100)
```

```
-- returns ALL Products
select *
from Product
where (price <= 100) or (price > 100)
        or isNull(price)
```

Discussion

- NULLs: weird behavior
- ”A Case Against SQL”
- Try to avoid NULLs in your database:

```
CREATE TABLE Product
                    (… pcolor int NOT NULL…)
```

- But very useful for OUTER JOINS. Next

Outer Joins

Outer Joins

- A join returns tuples that have a match in both input tables
- An outer joins returns tuples that have a match in at least one input table

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

Outer joins

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

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

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

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

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

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

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

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

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

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

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

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

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

missing

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz

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

prodName is foreign Key

Outer joins

Retrieve all product names, categories, and stores where they were purchased.
Include products that never sold

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

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Output

Name	Category	Store
Gizmo	gadget	Wiz
Camera	Photo	Ritz
Camera	Photo	Wiz
OneClick	Photo	NULL

Now it's present

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

1. Compute cross product R×S
2. Filter on C1
3. Add all R records without a match
4. Filter on C2

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

```
Tmp = {}
for x in R do                          // left outer join using C1
    for y in S do
        if C1 then Tmp = Tmp ∪ {(x,y)}
```

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

```
Tmp = {}
for x in R do                          // left outer join using C1
    for y in S do
        if C1 then Tmp = Tmp ∪ {(x,y)}
for x in R do
    if not (x in Tmp) then Tmp = Tmp ∪ {(x,NULL)}
```

Left Outer Join (Details)

```
select ...
from   R left outer join S on C1
where  C2
```

```
Tmp = {}
for x in R do                          // left outer join using C1
    for y in S do
        if C1 then Tmp = Tmp ∪ {(x,y)}
for x in R do
    if not (x in Tmp) then Tmp = Tmp ∪ {(x,NULL)}

Answer = {}                            // apply condition C2
for (x,y) in Tmp if C2 then Answer = Answer ∪ {(x,y)}
return Answer
```

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

ON v.s. WHERE

- Retrieve all products, together with the stores where they sold with price < 10

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

ON v.s. WHERE

- Retrieve all products, together with the stores where they sold with price < 10
- Which query does the job?

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

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

prodName is foreign Key

ON v.s. WHERE

- Retrieve all products, together with the stores where they sold with price < 10
- Which query does the job?

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

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

prodName is foreign Key

ON v.s. WHERE

- Retrieve all products, together with the stores where they sold with price < 10
- Which query does the job?

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, *then* checks price <10

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

prodName is foreign Key

ON v.s. WHERE

- Retrieve all products, together with the stores where they sold with price < 10
- Which query does the job?

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Includes products that were never purchased with price < 10

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

Includes products that were never purchased, *then* checks price <10

Same as inner join!

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

prodName is foreign Key

ON v.s. WHERE

- Retrieve all products, together with the stores where they sold with price < 10
- Which query does the job? This one:

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
   AND y.price < 10
```

Correct query

```
SELECT x.name, y.store
FROM   Product x
LEFT OUTER JOIN Purchase y
ON     x.name = y.prodName
WHERE y.price < 10
```

Joins: Recap

- **Inner join** = includes only matching tuples (i.e. regular join)
- **Left outer join** = includes everything from the left
- **Right outer join** = includes everything from the right
- **Full outer join** = includes everything

Summary so Far

- SELECT-FROM-WHERE
- Joins, self-joins, outer-joins
- NULLs

Next lectures: logical design, more SQL