CSE544
Data Management

Lectures 2: SQL

Announcements

- Lecture recordings are on zoom
- Wednesday, 1/10: review 1 is due
- Monday, 1/15: holiday, no class
- Wednesday, 1/17: canceled
- Friday, 1/19: makeup lecture, CSE2-371
- Also Friday, 1/19: review 2 is due

Recap

Relational model:

- Data is stored in flat relations
- No prescription of the physical storage
- Access to the data through high-level declarative language

SQL

SQL

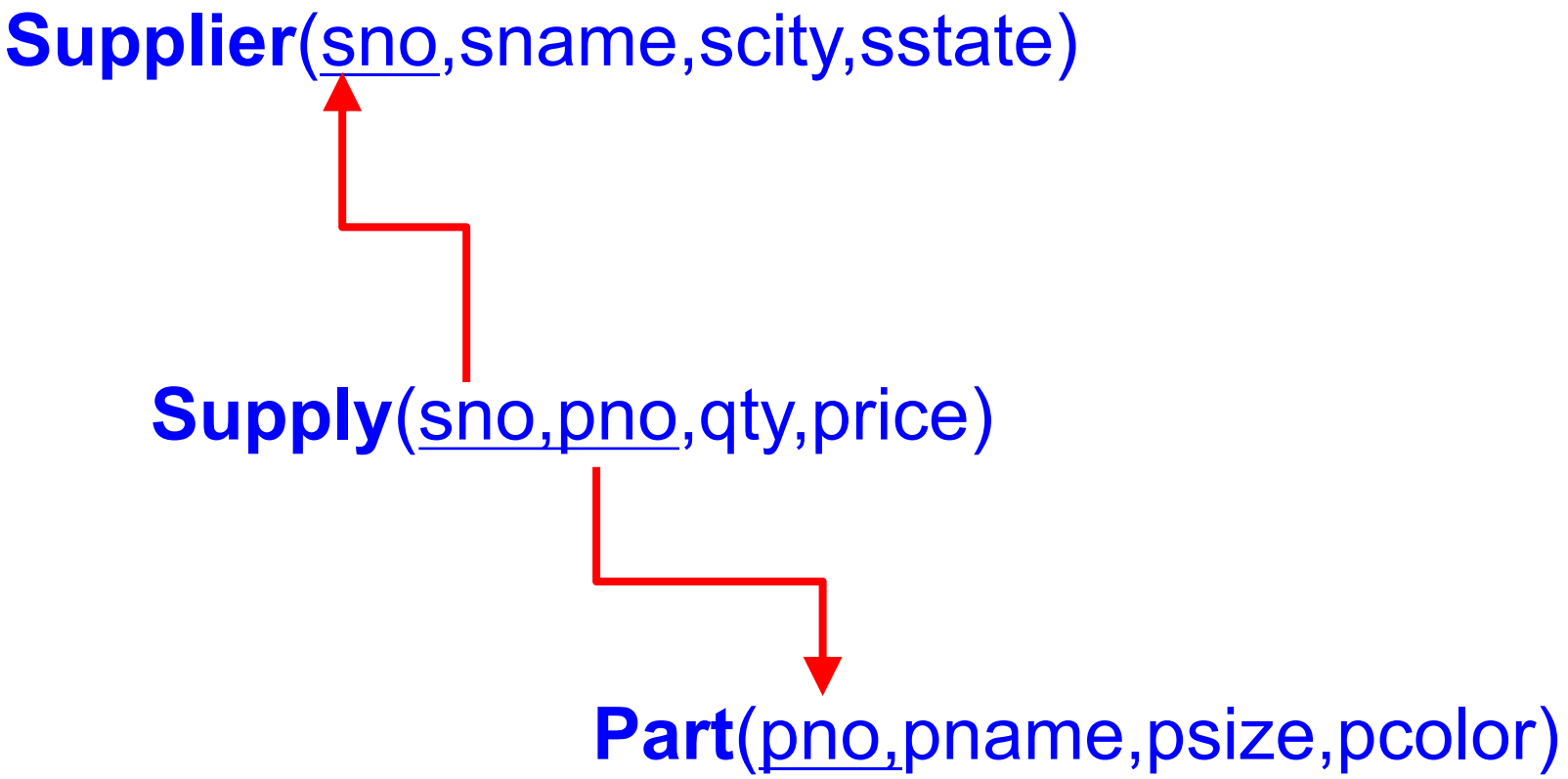
- Introduced in the late 70s
- Standard has been continuously evolving into a huge language
- SQL systems support various subsets
- We will study a core supported by all systems; this is all you need

SQL

Two parts:

- Data Definition Language (DDL):
  - CREATE TABLE, …
  - You will mostly read on your own
- Data Manipulation Language (DML)
  - SELECT-FROM-WHERE
  - INSERT/DELETE/UPDATE

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

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

SQL

```
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

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

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

SQL

```
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

```
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

```
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

```
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));
```

```
ry key,
                pcolor text);
```

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

SQL

```
INSERT INTO Supplier VALUES
    (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

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

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

SQL

```
SELECT sname
FROM Supplier
```

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

SQL

```
SELECT sname
FROM Supplier
```

sname
ACME
Walmart
Costco
Walmart
…

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

SQL

```
SELECT sname
FROM Supplier
```

This is a bag

sname
ACME
Walmart
Costco
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
Costco
~~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
Costco
…

```
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

```
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

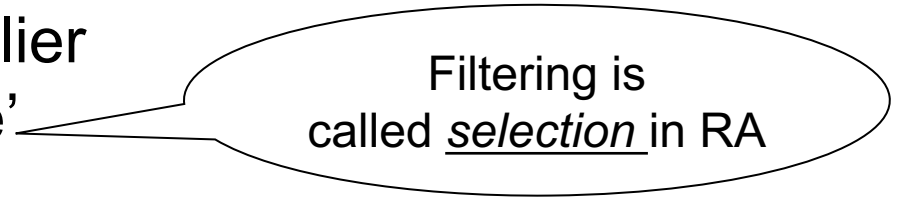
```
SELECT DISTINCT  z.pno, z.pname, x.scity
FROM     Supplier x, Supply y, Part z
WHERE    x.sno = y.sno
         and y.pno = z.pno
         and x.sstate = 'WA'
         and z.pcolor = 'red';
```

What happens if we don't include these conditions?

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

Operations

- 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 Parts 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 Parts 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 Parts 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
```

This doesn't work…
Why?

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

Self-Joins

Find the Parts 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 Parts 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 Parts 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 Parts 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
```

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 Parts 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

Discussion

SELECT-FROM-WHERE

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

The results can be described as a set of nested loops. Next.

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 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

Data independence:

- SQL engines do NOT compute the query using nested loop semantics
- Instead they choose between a variety of execution plans

How would you execute it?

```
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 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)

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
- A op B is
  - **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 op B is
  - **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 op B is
  - **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 op B is
  - **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 op B is
  - **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 op B is
  - **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 op B is
  - **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 op B is
  - **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

- So far we have seen only SELECT-FROM-WHERE queries
- Most data analysis requires some form of aggregation
- An aggregate operator takes a set of values and returns a single value
  - sum, min, max, avg, count

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

More SQL: Aggregates

```
SELECT count(*)
FROM   Part
```

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

More SQL: Aggregates

```
SELECT count(*)
FROM   Part
```

```
SELECT x.scity, avg(psize)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
```

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

More SQL: Aggregates

```
SELECT count(*)
FROM   Part
```

```
SELECT x.scity, avg(psize)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
```

```
SELECT x.scity, avg(psize)
FROM   Supplier x, Supply y, Part z
WHERE  x.sno = y.sno  and y.pno = z.pno
GROUP BY x.scity
HAVING count(*) > 200
```

Discussion

- SQL Aggregates = simple data analytics
- Semantics:
  1. FROM-WHERE (nested-loop semantics)
  2. Group answers by GROUP BY attrs
  3. Apply HAVING predicates on groups
  4. Apply SELECT aggregates on groups
- Aggregate functions:
  - count, sum, min, max, avg
- DISTINCT same as GROUP BY

Outer Joins

- A join returns only those outputs that have a tuple from each of the input tables
- Sometimes we want to include tuples from one table without a match from the other table:

Outer Join

```
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)

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)}
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

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
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

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
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

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
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

- Outer join condition in the ON clause
- Different from the WHERE clause
- Compare:

```
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!

Joins

- **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