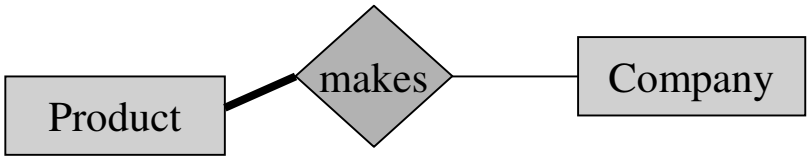
Lecture 3

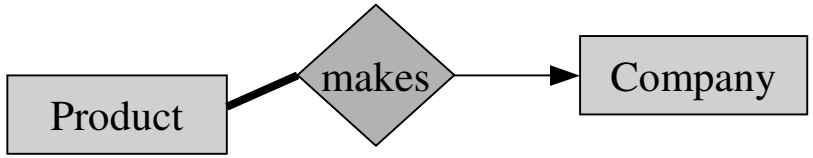
Monday, April 8, 2002

Review

What does this mean ?

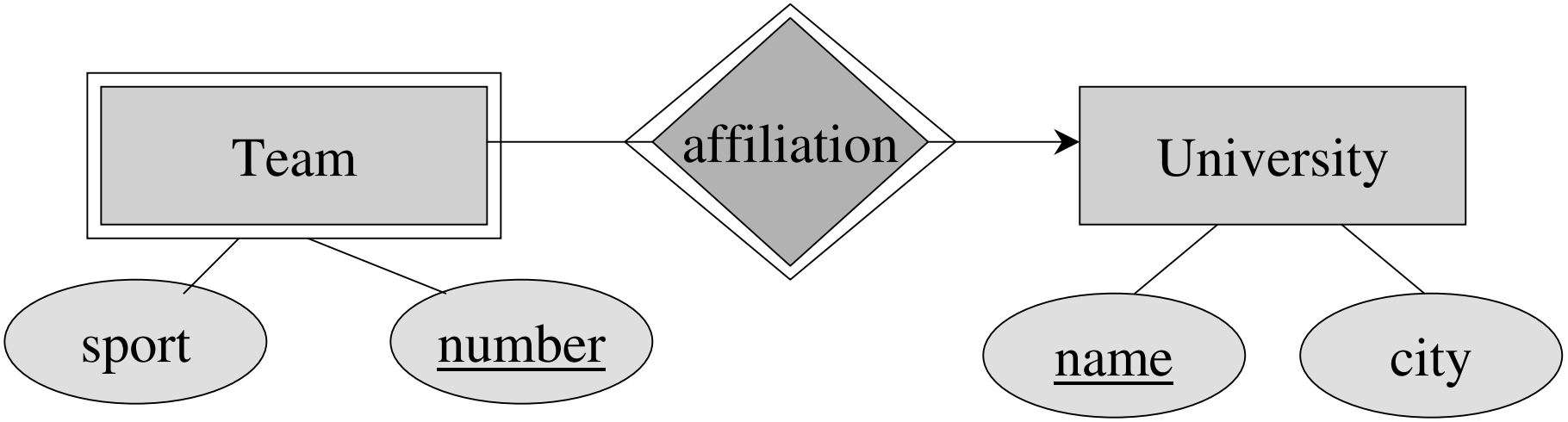


What does this mean ?



This: also called *referential integrity constraint*

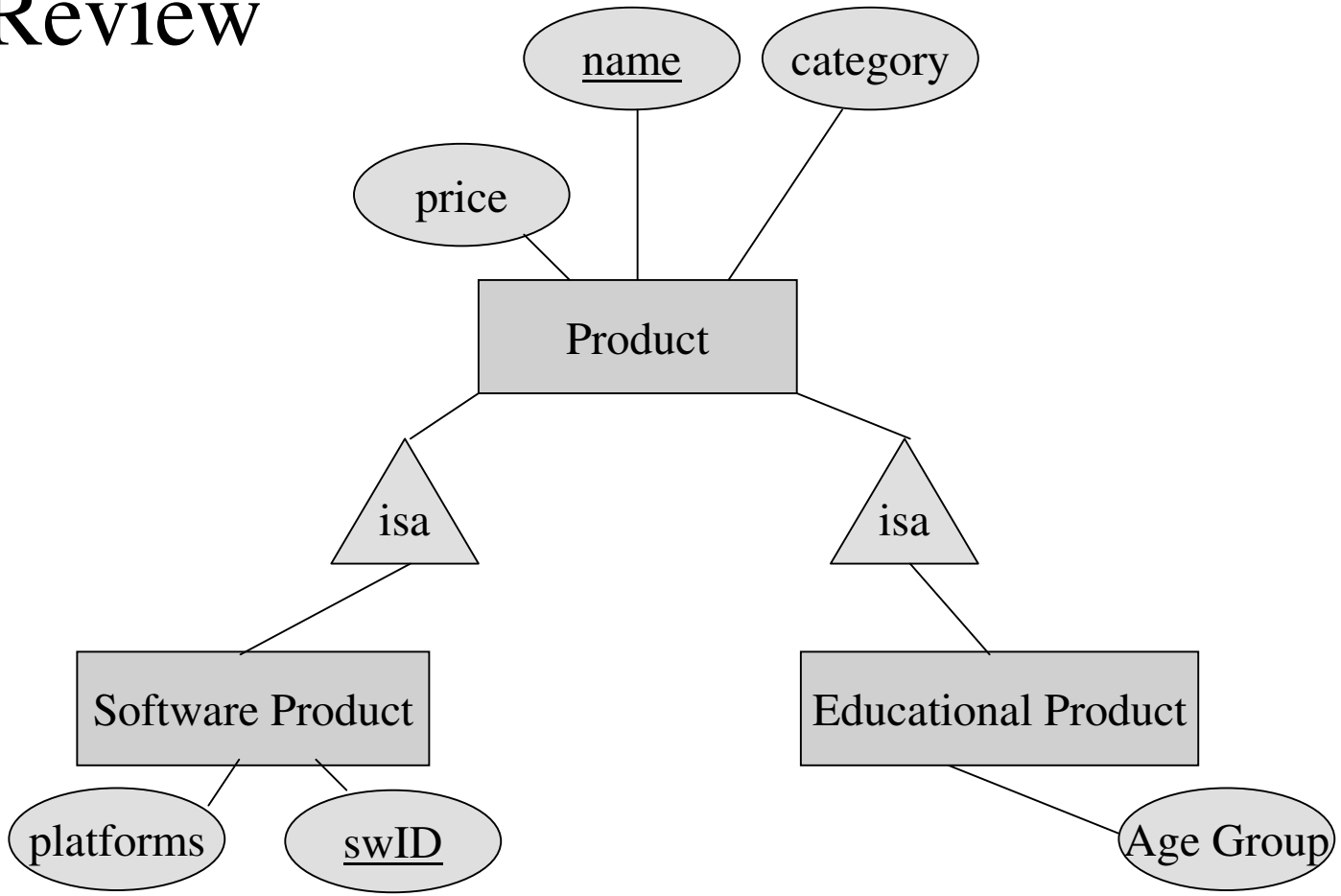
Review

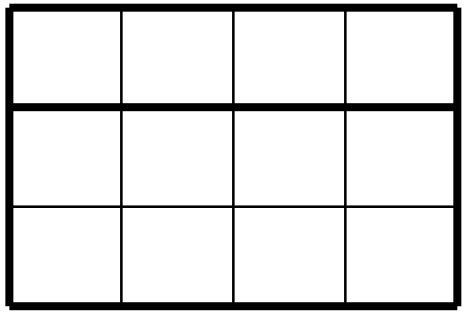


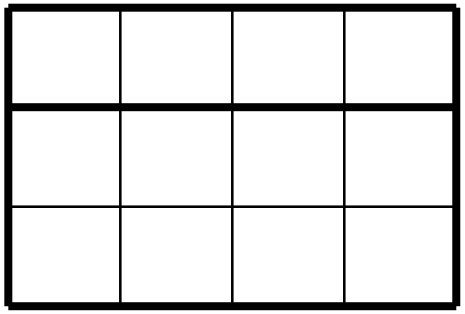
Team

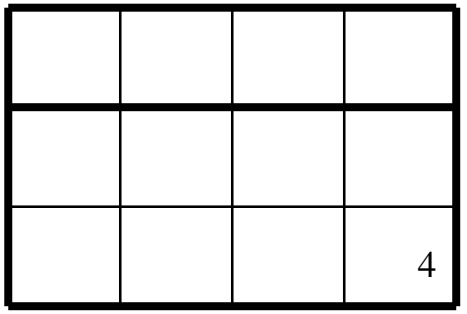
University

Review


name
category
price
Product
isa
isa
Software Product
Educational Product
platforms
swID
Age Group







Reading Assignment

1. A Relational Model of Data for Large Shared Data Banks

2. Of Objects and Databases: A Decade of Turmoil

Outline

- Basic SQL Queries 5.2
- Binary operators 5.3
- Subqueries 5.4
- Aggregates 5.5

SQL Introduction

Basic form: (many many more bells and whistles in addition)

```
Select   [attributes]
From     [relations]
Where    [conditions]
```

Projections

Select only a subset of the attributes

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(name, stock price)

Projections

Rename the attributes in the resulting table

```
SELECT   name AS company, stockprice AS price
FROM     Company
WHERE    country="USA" AND stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(company, price)

Eliminating Duplicates

Rename the attributes in the resulting table

```
SELECT DISTINCT country
FROM      Company
WHERE   stockPrice > 50
```

Input schema: Company(sticker, name, country, stockPrice)
Output schema: R(country)

Ordering the Results

```
SELECT   name, stockPrice
FROM     Company
WHERE    country="USA" AND stockPrice > 50
ORDERBY  country, name
```

Ordering is ascending, unless you specify the DESC keyword.

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

Joins

Find names of people living in Seattle that bought gizmo products, and the names of the stores they bought from

```
SELECT   pname, store
FROM     Person, Purchase
WHERE    pname=buyer AND city="Seattle"
         AND product="gizmo"
```

Input schema: Person(pname, phoneNumber, city)
Purchase (buyer, seller, store, product)

Output Schema: R(pname, store)

Join

pname	phone	city
John	1111	Seattle
Fred	2222	Chicago
Mary	3333	Seattle
Mary	4444	Seattle
Mary	5555	Portland
John	6666	Seattle
Fred	7777	Seattle

buyer	. . . . .	product
Jane		Gizmo
John		Soap
Mary		Gizmo
Mary		Book
John		Book
John		Gizmo
Mary		Gizmo

Joins are a key operation in DBMS

Disambiguating Attributes

Find names of people buying telephony products:

```
SELECT   Person.name
FROM     Person, Purchase, Product
WHERE    Person.name=Purchase.buyer
     AND  Product=Product.name
     AND  Product.category="telephony"
```

Input schema: Product (name, price, category, maker)
Purchase (buyer, seller, store, product)
Person(name, phoneNumber, city)

Output schema: R(name)

Tuple Variables

Find pairs of companies making products in the same category

```
SELECT  product1.maker AS maker1, product2.maker AS maker2
FROM    Product AS product1, Product AS product2
WHERE  product1.category=product2.category
   AND  product1.maker <> product2.maker
```

Input schema: Product (name, price, category, maker)
Output schema: R(maker1, maker2)

Tuple Variables

Tuple variables introduced automatically by the system:

Product (name, price, category, maker)

```
SELECT  name
FROM    Product
WHERE  price > 100
```

Becomes:

```
SELECT Product.name
FROM   Product AS Product
WHERE Product.price > 100
```

Doesn't work when Product occurs more than once:
In that case the user needs to define variables explicitely.

Meaning (Semantics) of SQL Queries

SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions

1. Nested loops:

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

Meaning (Semantics) of SQL Queries

SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions

2. Parallel assignment

```
Answer = { }
for all assignments x1 in R1, …, xn in Rn do
        if Conditions then Answer = Answer U {(a1,…,ak)}
return Answer
```

Doesn't impose any order !
Like Datalog

Meaning (Semantics) of SQL Queries

SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions

3. Translation to Datalog: one rule

$$\text{Answer}(a_1,\ldots,a_k) \leftarrow R_1(x_{11},\ldots,x_{1p}),\ldots,R_n(x_{n1},\ldots,x_{np}), \text{Conditions}$$

Meaning (Semantics) of SQL Queries

SELECT a1, a2, …, ak
FROM R1 AS x1, R2 AS x2, …, Rn AS xn
WHERE Conditions

4. Translation to Relational algebra:

$\Pi_{a1,\ldots,ak}$ ($\sigma_{\text{Conditions}}$ (R1 x R2 x … x Rn))

Select-From-Where queries are precisely Select-Project-Join

First Unintuitive SQLism

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

Looking for $R \cap (S \cup T)$

But what happens if T is empty?

Union, Intersection, Difference

```
(SELECT DISTINCT name
 FROM      Person
 WHERE     City="Seattle")

    UNION

(SELECT DISTINCT name
 FROM      Person, Purchase
 WHERE     buyer=name AND store="The Bon")
```

Similarly, you can use INTERSECT and EXCEPT.
You must have the same attribute names (otherwise: rename).

Conserving Duplicates

The UNION, INTERSECTION and EXCEPT operators operate as sets, not bags.

```
(SELECT  name
 FROM      Person
 WHERE    City="Seattle")

    UNION  ALL



(SELECT  name
 FROM      Person, Purchase
 WHERE   buyer=name AND store="The Bon")
```

Exercises

Product (pname, price, category, maker)
Purchase (buyer, seller, store, product)
Company (cname, stock price, country)
Person(per-name, phone number, city)

Ex #1: Find people who bought telephony products.
Ex #2: Find names of people who bought American products
Ex #3: Find names of people who bought American products and did not buy French products
Ex #4: Find names of people who bought American products and they live in Seattle.
Ex #5: Find people who bought stuff from Joe or bought products from a company whose stock prices is more than $50.

Subqueries (or Nested Queries)

A subquery producing a single tuple:

```
SELECT DISTINCT Purchase.product
FROM     Purchase
WHERE  buyer =
            (SELECT  name
              FROM     Person
              WHERE   ssn = “123456789”);
```

In this case, the subquery returns one value.

If it returns more, it’s a run-time error.

Can say the same thing without a subquery:

```
SELECT DISTINCT Purchase.product
FROM     Purchase, Person
WHERE  buyer = name AND ssn = “123456789”
```

How do the two queries compare if we drop the DISTINCT keyword ?

Subqueries Returning Relations

Find companies who manufacture products bought by Joe Blow.

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

Here the subquery returns a set of values

Equivalent to:

```
SELECT DISTINCT Company.name
FROM      Company, Product, Purchase
WHERE   Company.name= Product.maker
          AND  Product.name  = Purchase.product
          AND  Purchase.buyer = “Joe Blow”
```

Subqueries Returning Relations

You can also use: s > ALL R
s > ANY R
EXISTS R

Product (pname, price, category, maker)
Find products that are more expensive than all those produced By “Gizmo-Works”

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

Question for Database Fans and their Friends

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

- Hint: show that all SFW queries are monotone (figure out what this means). A query with **ALL** is not monotone

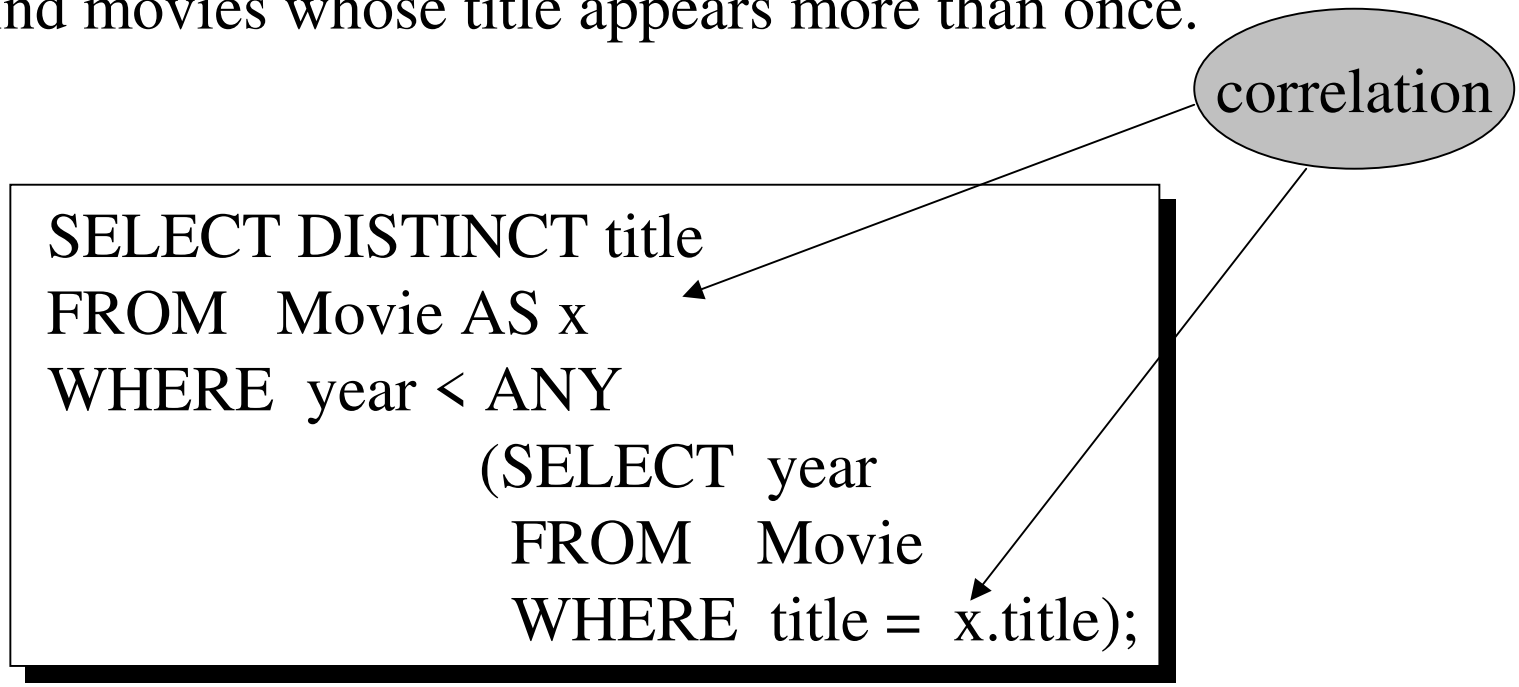
Conditions on Tuples

```
SELECT  DISTINCT Company.name
FROM      Company, Product
WHERE   Company.name= Product.maker
          AND  (Product.name,price)  IN
                  (SELECT Purchase.product, Purchase.price)
                    FROM   Purchase
                    WHERE Purchase.buyer = “Joe Blow”);
```

Correlated Queries

Movie (<u>title, year</u>, director, length)

Find movies whose title appears more than once.



Note (1) scope of variables (2) this can still be expressed as single SFW

Complex Correlated Query

Product (pname, price, category, maker, year)

- Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

```
SELECT DISTINCT  pname, maker
FROM     Product AS x
WHERE  price > ALL  (SELECT  price
                              FROM    Product AS y
                              WHERE  x.maker = y.maker AND y.year < 1972);
```

Powerful, but much harder to optimize !

Aggregation

```
SELECT   Sum(price)
FROM     Product
WHERE    maker="Toyota"
```

SQL supports several aggregation operations:

SUM, MIN, MAX, AVG, COUNT

```
SELECT  Count(*)
FROM    Product
WHERE   year > 1995
```

Except COUNT, all aggregations apply to a single attribute

Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```
SELECT  Count(name, category)
FROM    Product
WHERE   year > 1995
```

same as Count(*)

Better:

```
SELECT  Count(DISTINCT name, category)
FROM    Product
WHERE   year > 1995
```

Simple Aggregation

Purchase(product, date, price, quantity)

Example 1: **find total sales for the entire database**

```
SELECT  Sum(price * quantity)
FROM    Purchase
```

Example 1': **find total sales of bagels**

```
SELECT  Sum(price * quantity)
FROM    Purchase
WHERE   product = 'bagel'
```

Simple Aggregations

Product	Date	Price	Quantity
Bagel	10/21	0.85	15
Banana	10/22	0.52	7
Banana	10/19	0.52	17
Bagel	10/20	0.85	20

Grouping and Aggregation

Usually, we want aggregations on certain parts of the relation.

Purchase(product, date, price, quantity)

Example 2: **find total sales after 10/1 per product.**

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > "10/1"
GROUPBY   product
```

Grouping and Aggregation

1. Compute the relation (I.e., the FROM and WHERE).
2. Group by the attributes in the GROUPBY
3. Select one tuple for every group (and apply aggregation)

SELECT can have (1) grouped attributes or (2) aggregates.

First compute the relation (date > “9/1”) then group by product:

Product	Date	Price	Quantity
Banana	10/19	0.52	17
Banana	10/22	0.52	7
Bagel	10/20	0.85	20
Bagel	10/21	0.85	15

Then, aggregate

Product	TotalSales
Bagel	$29.75
Banana	$12.48

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > "9/1"
GROUPBY   product
```

Another Example

Product	SumSales	MaxQuantity
Banana	$12.48	17
Bagel	$29.75	20

For every product, what is the total sales and max quantity sold?

```
SELECT    product, Sum(price * quantity) AS SumSales
                   Max(quantity) AS MaxQuantity
FROM      Purchase
GROUP BY product
```

HAVING Clause

Same query, except that we consider only products that had at least 100 buyers.

```
SELECT      product, Sum(price * quantity)
FROM        Purchase
WHERE       date > "9/1"
GROUP BY    product
HAVING      Sum(quantity) > 30
```

HAVING clause contains conditions on aggregates.

General form of Grouping and Aggregation

SELECT	S
FROM	$R_1,\ldots,R_n$
WHERE	C1
GROUP BY	$a_1,\ldots,a_k$
HAVING	C2

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

C2 = is any condition on aggregate expressions

General form of Grouping and Aggregation

SELECT	S
FROM	$R_1,\ldots,R_n$
WHERE	C1
GROUP BY	$a_1,\ldots,a_k$
HAVING	C2

Evaluation steps:

1. Compute the FROM-WHERE part, obtain a table with all attributes in $R_1,\ldots,R_n$
2. Group by the attributes $a_1,\ldots,a_k$
3. Compute the aggregates in C2 and keep only groups satisfying C2
4. Compute aggregates in S and return the result

More Aggregation Examples

Author(login,name)

Document(url, title)

Wrote(login,url)

Mentions(url,word)

- Find all authors who wrote at least 10 documents:

```
Select author.name
From author, wrote
Where author.login=wrote.login
Groupby author.name
Having count(wrote.url) > 10
```

- Find all authors who have a vocabulary over 10000:

```
Select author.name
From author, wrote, mentions
Where author.login=wrote.login and wrote.url=mentions.url
Groupby author.name
Having count(distinct mentions.word) > 10000
```