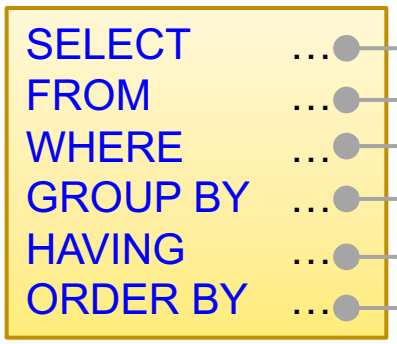
L04: SQL

Announcements!

- Polls on Piazza. Open for 2 days
- Outline today:
  - practicing more joins and specifying key and FK constraints
  - nested queries
- Next time: "witnesses" (traditionally students find this topic the most difficult)

Queries via SQL have multiple words: If you master this structure you know 50% about SQL Queries



- List of attributes to be included in final result (also called projection! ("*" selects all attributes)
- Indicates the table(s) from which data is to be retrieved
- Lists a comparison predicate, which restricts the rows returned by the query, e.g. “price < 20” or different join conditions
- Groups rows that have one more common values together into a smaller set of rows
- A comparison predicate used to restrict the rows resulting from the GROUP BY clause
- Identifies which columns are used to sort the resulting data, plus the direction each column is sorted by (ascending or descending)

Note1 : SQL is generally case insensitive, e.g. SELECT = Select = select

Note2 : The words always appear in this order – you CANNOT reorder them

How to specify Foreign Key constraints

- Suppose we have the following schema:

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

- And we want to impose the following constraint:
  - 'Only bona fide students may enroll in courses' i.e. a student must appear in the Students table to enroll in a class

Students

sid	name	gpa
101	Bob	3.2
123	Mary	3.8

Enrolled

student_id	cid	grade
123	564	A
123	537	A+

student_id alone is not a key- what is?

We say that student_id is a **foreign key** that refers to Students

Declaring Primary Keys

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

```
CREATE TABLE Students(
      sid  CHAR(20) PRIMARY KEY,
      name CHAR(20),
      gpa  REAL


)
```

Declaring Primary Keys

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

```
CREATE TABLE Students(
      sid  CHAR(20),
      name CHAR(20),
      gpa  REAL,
      PRIMARY KEY (sid)

)
```

Declaring Foreign Keys

```
Students(sid: string, name: string, gpa: float)
Enrolled(student_id: string, cid: string, grade: string)
```

```
CREATE TABLE Enrolled(
      student_id CHAR(20),
      cid        CHAR(20),
      grade      CHAR(10),
      PRIMARY KEY (student_id, cid),
      FOREIGN KEY (student_id) REFERENCES Students(sid)
)
```

An example of SQL semantics

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

Output

A
3
3

R

A
1
3

S

B	C
2	3
3	4
3	5

Cross Product

A	B	C
1	2	3
1	3	4
1	3	5
3	2	3
3	3	4
3	3	5

Apply Selections / Conditions

A	B	C
3	3	4
3	3	5

Apply Projection

Note the semantics of a join

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

1. Take **cross product**:

$$X = R \times S$$

Recall: Cross product (A X B) is the set of all unique tuples in A,B

Ex: {a,b,c} X {1,2}
= {(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)}

2. Apply **selections / conditions**:

$$Y = \{(r, s) \in X \mid r.A = r.B\}$$

= Filtering!

3. Apply **projections** to get final output:

$$Z = (y.A,) \; for \; y \in Y$$

= Returning only *some* attributes

Remembering this order is critical to understanding the output of certain queries (see later on…)

Note: we say “semantics” not “execution order”

- The preceding slides show what a join means
- Not actually how the DBMS executes it under the covers

Practicing more Joins

Quiz 4

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture at least two different products.

```
SELECT  DISTINCT cName
FROM
WHERE
```

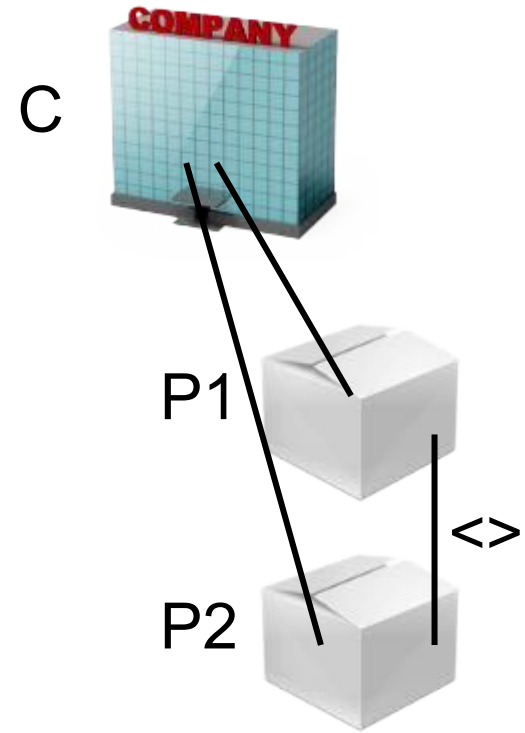
Quiz 4

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture at least two different products.

```
SELECT  DISTINCT cName
FROM    Product P1, Product P2, Company
WHERE   country = 'USA'
    and P1.manufacturer = cName
    and P2.manufacturer = cName
    and P1.pName <> P2.pName
```



Quiz 4

302

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
...	...	...	...

<>

P2

PName	Price	Category	Manufacturer
...	...	...	...
Powergizmo	$29.99	Gadgets	GizmoWorks

Company

CName	StockPrice	Country
GizmoWorks	25	USA
...	...	...

```
SELECT  DISTINCT cName
FROM    Product P1, Product P2, Company
WHERE   country = 'USA'
    and P1.manufacturer = cName
    and P2.manufacturer = cName
    and P1.pName <> P2.pName
```

Cname
GizmoWorks

Quiz 5

302

Product (pName, price, category, manufacturer)
Company (cName, stockPrice, country)

Q: Find all US companies that manufacture a product below $20 and a product above $15.

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 15
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

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

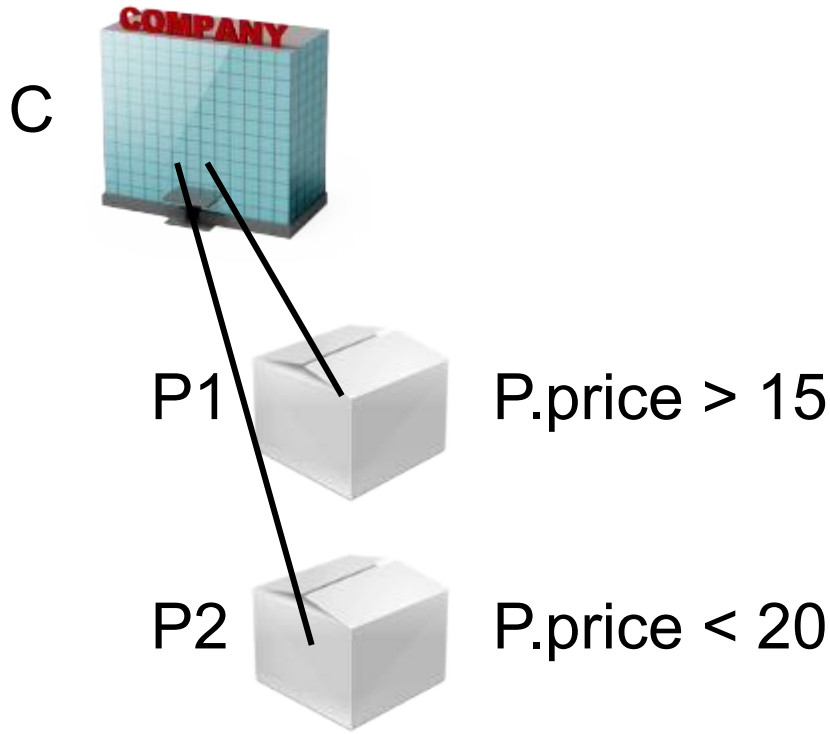
Quiz 5

302

Product (<u>pName</u>, price, category, manufacturer)
Company (<u>cName</u>, stockPrice, country)

Q: Find all US companies that manufacture a product below $20 and a product above $15.

Note that we did not specify any condition that P1 and P2 need to be distinct. An alternative interpretation is "...and <u>another</u> product above..."



Quiz 5

302

P1

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
...	...	...	...

P2

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
...	...	...	...

Company

CName	StockPrice	Country
GizmoWorks	25	USA
...	...	...

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 15
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Cname
GizmoWorks

Quiz 6

302

Product

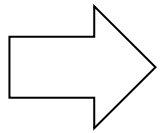
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all countries that have companies that manufacture some product in the 'Gadgets' category!

```
SELECT   country
FROM     Product, Company
WHERE    manufacturer = cName
   and   category = 'Gadgets'
```



?

Quiz 6

302

Product

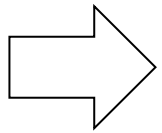
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

CName	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

Q: Find all countries that have companies that manufacture some product in the 'Gadgets' category!

```
SELECT   country
FROM     Product, Company
WHERE    manufacturer = cName
   and   category = 'Gadgets'
```



Country
USA
USA

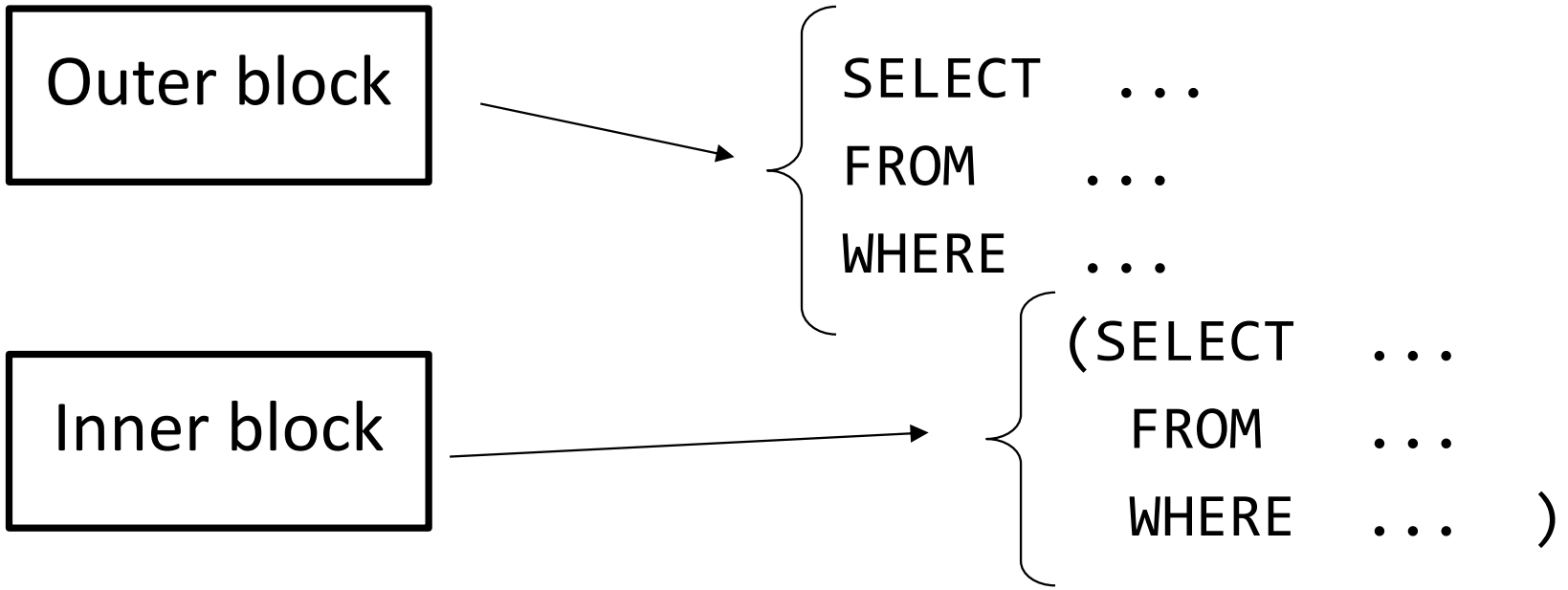
Joins can introduce duplicates -> remember to use DISTINCT!

Nested queries (Subqueries)

High-level note on nested queries

- We can do nested queries because SQL is compositional:
  - Everything (inputs / outputs) is represented as multisets- the output of one query can thus be used as the input to another (nesting)!
- This is extremely powerful!
- High-level idea: subqueries return relations (yet sometimes just values)

Subqueries = Nested queries



Subqueries

- A subquery is a SQL query nested inside a larger query
- Such inner-outer queries are called nested queries
- A subquery may occur in a:
  - SELECT clause
  - FROM clause
  - WHERE clause
  - HAVING clause

important!

- Rule of thumb: avoid writing nested queries when possible; keep in mind that sometimes it's impossible

1. Subqueries in SELECT

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each product return the city where it is manufactured!

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company2 C
                  WHERE   C.cid = P.cid)
FROM    Product2 P
```

What happens if the subquery returns more than one city ?

Runtime error

→ "Scalar subqueries"

1. Subqueries in SELECT

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: For each product return the city where it is manufactured!

```
SELECT  P.pname, (SELECT  C.city
                  FROM    Company2 C
                  WHERE   C.cid = P.cid)
FROM    Product2 P
```

"unnesting the query"

```
SELECT  P.pname, C.city
FROM    Product2 P, Company2 C
WHERE   C.cid = P.cid
```

Whenever possible,
don't use nested queries

1. Subqueries in SELECT

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: Compute the number of products made by each company!

```
SELECT  C.cname, ( SELECT count (*)
                   FROM   Product2 P
                   WHERE P.cid = C.cid)
FROM    Company2 C
```

Better: we can unnest by using a GROUP BY:

```
SELECT   C.cname, count(*)
FROM     Company2 C, Product2 P
WHERE    C.cid=P.cid
GROUP BY C.cname
```

2. Subqueries in FROM clause

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Q: Find all products whose prices are > 20 and < 30!

```
SELECT  X.pname
FROM    ( SELECT *
          FROM    Product2 as P
          WHERE   price >20 ) as X
WHERE   X.price < 30
```

unnesting

```
SELECT  pname
FROM    Product2
WHERE   price > 20 and price < 30
```

X

PName	Price	cid
Powergizmo	$29.99	1
~~MultiTouch~~	~~$203.99~~	~~3~~

Subqueries in WHERE clause

IN, ANY, ALL

3. Subqueries in WHERE

305

What do these queries compute?

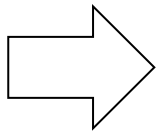
R

a
1
2

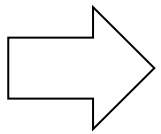
U

a
2
3
4

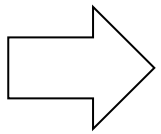
```
SELECT  a
FROM    R
WHERE   a IN
        (SELECT * from U)
```

 ?

```
SELECT  a
FROM    R
WHERE   a < ANY
        (SELECT * from U)
```

 ?

```
SELECT  a
FROM    R
WHERE   a < ALL
        (SELECT * from U)
```

 ?

3. Subqueries in WHERE

305

What do these queries compute?

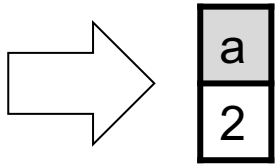
R

a
1
2

U

a
2
3
4

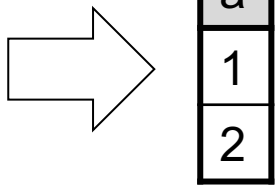
```
SELECT  a
FROM    R
WHERE   a IN
        (SELECT * from U)
```



a
2

Since 2 is in the set (2, 3, 4)

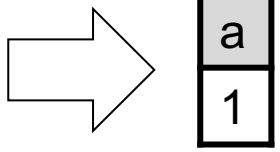
```
SELECT  a
FROM    R
WHERE   a < ANY
        (SELECT * from U)
```



a
1
2

Since 1 and 2 are < than at least one ("any") of 2, 3 or 4

```
SELECT  a
FROM    R
WHERE   a < ALL
        (SELECT * from U)
```



a
1

Since 1 is < than each ("all") of 2, 3, and 4

Something tricky about Nested Queries

Are these queries equivalent?

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
SELECT pr.maker
FROM   Purchase p, Product pr
WHERE  p.name = pr.product
   AND p.buyer = 'Joe B')
```

```
SELECT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Joe B'
```

Beware of duplicates!

Something tricky about Nested Queries

Are these queries equivalent?

```
SELECT DISTINCT c.city
FROM   Company c
WHERE  c.name  IN (
SELECT pr.maker
FROM   Purchase p, Product pr
WHERE  p.name = pr.product
   AND p.buyer = 'Joe B')
```

```
SELECT DISTINCT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = 'Joe B'
```

Now they are equivalent (both use set semantics)

Correlated subqueries

- In all previous cases, the nested subquery in the inner select block could be entirely evaluated before processing the outer select block.
- This is no longer the case for correlated nested queries.
- Whenever a condition in the WHERE clause of a nested query references some column of a table declared in the outer query, the two queries are said to be correlated.
- The nested query is then evaluated once for each tuple (or combination of tuples) in the outer query.

Correlated Queries (Using External Vars in Internal Subquery)

```
Movie(title, year, director, length)
```

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
           SELECT  year
           FROM    Movie
           WHERE   title =  m.title)
```

Find movies whose title appears in more than one year.

Note the scoping of the variables!

Note also: this can still be expressed as single SFW query…

Complex Correlated Query

```
Product(name, price, category, maker, year)
```

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

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

Can be very powerful (also much harder to optimize)

3. Subqueries in WHERE (existential)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Using IN:

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN ( 1, 2 )
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

3. Subqueries in WHERE (existential)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Using IN:

"Set membership"

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product2 P
                   WHERE   P.price < 25)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

3. Subqueries in WHERE (existential)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

Using EXISTS:

"Test for empty relations"

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   EXISTS ( SELECT *
                 FROM   Product2 P
                 WHERE  C.cid = P.cid
                    and P.price < 25)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery

3. Subqueries in WHERE (existential)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Using ANY (also some):

"Set comparison"

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   25 > ANY ( SELECT  price
                   FROM    Product2 P
                   WHERE   P.cid = C.cid)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery

SQLlite does not support "ANY" ☹

3. Subqueries in WHERE (existential)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Now, let's unnest:

```
SELECT   DISTINCT C.cname
FROM     Company2 C, Product2 P
WHERE    C.cid = P.cid
    and  P.price < 25
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Existential quantifiers are easy ! ☺

3. Subqueries in WHERE (universal)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make only products with price < 25!

same as:

Q: Find all companies for which all products have price < 25!

Universal quantifiers are more complicated ! ☹
(Think about the companies that should not be returned)

3. Subqueries in WHERE (exist not -> universal)

315

Q: Find all companies that make only products with price < 25!

1. Find the other companies: i.e. they have some product ≥ 25!

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product2 P
                   WHERE   P.price >= 25)
```

2. Find all companies s.t. all their products have price < 25!

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid NOT IN ( SELECT  P.cid
                       FROM    Product2 P
                       WHERE   P.price >= 25)
```

3. Subqueries in WHERE (exist not -> universal)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make only products with price < 25!

Using NOT EXISTS:

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   NOT EXISTS ( SELECT  *
                     FROM    Product2 P
                     WHERE   C.cid = P.cid
                         and   P.price >= 25)
```

3. Subqueries in WHERE (exist not -> universal)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make only products with price < 25!

Using ALL:

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   25 > ALL (  SELECT  price
                    FROM    Product2 P
                    WHERE   P.cid = C.cid)
```

SQLite does not support "ALL" ☹

Question for Database Fans & Friends

This topic goes beyond the course objectives; only for those who are really interested

- How can we unnest the universal quantifier query ?

Queries that must be nested

This topic goes beyond the course objectives; only for those who are really interested

- Definition: A query Q is monotone if:
  - Whenever we add tuples to one or more of the tables…
  - … the answer to the query cannot contain fewer tuples
- Fact: all unnested queries are monotone
  - Proof: using the "nested for loops" semantics
- Fact: Query with universal quantifier is not monotone
  - Add one tuple violating the condition. Then "all" returns fewer tuples
- Consequence: we cannot unnest a query with a universal quantifier

The drinkers-bars-beers example

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Challenge: write these in SQL.
Solutions: http://queryviz.com/online/

Find drinkers that frequent some bar that serves some beer they like.

x: $\exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$

Find drinkers that frequent only bars that serve some beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$

Find drinkers that frequent some bar that serves only beers they like.

x: $\exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

Find drinkers that frequent only bars that serve only beer they like.

x: $\forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$

Basic SQL Summary

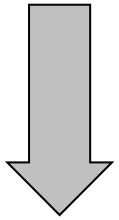
- SQL provides a high-level declarative language for manipulating data (DML)
- The workhorse is the SFW block
- Set operators are powerful but have some subtleties
- Powerful, nested queries also allowed.

WITH clause

WITH clause: temporary relations

315

```
SELECT  pname, price
FROM    Product2
WHERE   price =
        (SELECT  max(price)
        FROM     Product2)
```



```
WITH      max_price(value) as
          (SELECT  max(price)
          FROM     Product2)
SELECT    pname, price
FROM      Product2, max_price
WHERE     price = value
```

Product (pname, price, cid)

The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times

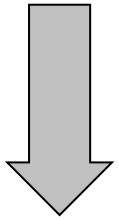
WITH clause: temporary relations

315

```
SELECT  pname, price
FROM    Product2
WHERE   price =
        (SELECT  max(price)
        FROM     Product2)
```

Product (pname, price, cid)

The **WITH** clause defines a temporary relation that is available only to the query in which it occurs. Sometimes easier to read. Very useful for queries that need to access the same intermediate result multiple times



```
WITH    max_price as
        (SELECT  max(price) as value
        FROM     Product2)
SELECT  pname, price
FROM    Product2, max_price
WHERE   price = value
```

Witnesses

Motivation: What are these queries supposed to return?

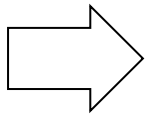
Product2

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company2

cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products



?

Motivation: What are these queries supposed to return?

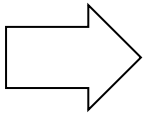
Product2

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company2

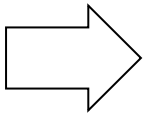
cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products



cid	mp
1	20
2	300

Find for each company id, the product with maximum price amongst its products



?

Motivation: What are these queries supposed to return?

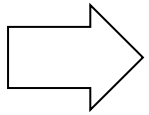
Product2

PName	Price	cid
Gizmo	15	1
SuperGizmo	20	1
iTouch1	300	2
iTouch2	300	2

Company2

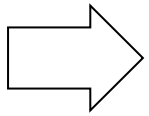
cid	cname	city
1	GizmoWorks	Oslo
2	Apple	MountainView

Find for each company id, the maximum price amongst its products



cid	mp
1	20
2	300

Find for each company id, the product with maximum price amongst its products (Recall that "group by cid" can just give us one single tuple per cid)



cid	mp	pname
1	20	SuperGizmo
2	300	iTouch1
2	300	iTouch2

Witnesses: simple (1/4)

Product2 (pname, price, cid)

315

Q: Find the most expensive product + its price

(Finding the maximum price alone would be easy)

Witnesses: simple (2/4)

Product2 (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery

1.
```
SELECT   max(P1.price)
FROM     Product2 P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (3/4)

Product2 (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...

2.
```
SELECT  P2.pname, P2.price
FROM    Product2 P2
```

1.
```
SELECT   max(P1.price)
FROM     Product2 P1
```

But we want the "witnesses," i.e. the product(s) with the max price. How do we do that?

Witnesses: simple (4/4)

Product2 (pname, price, cid)

Q: Find the most expensive product + its price
(Finding the maximum price alone would be easy)

Our Plan:

- 1. Compute max price in a subquery
- 2. Compute each product and its price...
  and compare the price with the max price

```
SELECT  P2.pname, P2.price
FROM    Product2 P2
WHERE   P2.price =
        (SELECT  max(P1.price)
        FROM     Product2 P1)
```