Database Systems CSE 414

Lectures 4: Joins & Aggregation (Ch. 6.1-6.4)

CSE 414 - Fall 2017

Announcements

· Should now have seats for all registered

CSE 414 - Fall 2017

Outline

- Inner joins (6.2, review)
- Outer joins (6.3.8)
- Aggregations (6.4.3 6.4.6)

CSE 414 - Fall 2017

UNIQUE

- PRIMARY KEY adds implicit "NOT NULL" constraint while UNIQUE does not
 - you would have to add this explicitly for UNIQUE:

```
CREATE TABLE Company(
name VARCHAR(20) NOT NULL, ...
UNIQUE (name));
```

- · You almost always want to do this (in real schemas)
 - SQL Server behaves strangely with NULL & UNIQUE
 - otherwise, think through NULL for every query
 - you can remove the NOT NULL constraint later

CSE 414 - Fall 2017

(Inner) Joins

```
SELECT a1, a2, ..., an FROM R1, R2, ..., Rm WHERE Cond
```

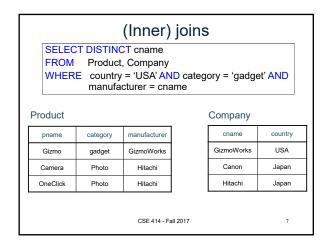
CSE 414 - Fall 2017

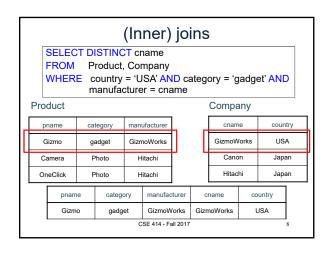
(Inner) joins

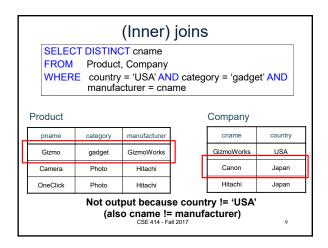
Company(<u>cname</u>, country)
Product(<u>pname</u>, price, category, manufacturer)
– manufacturer is foreign key

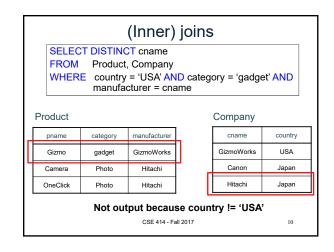
SELECT DISTINCT cname
FROM Product, Company
WHERE country = 'USA' AND category = 'gadget' AND manufacturer = cname

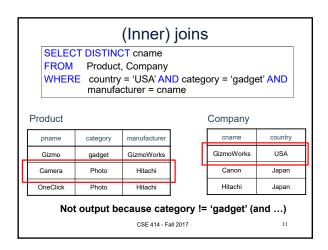
CSE 414 - Fall 2017

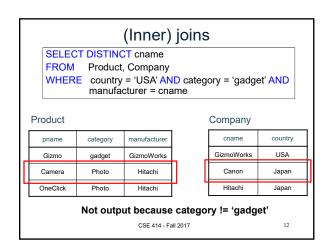


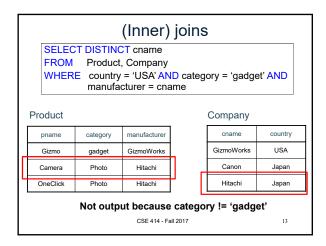


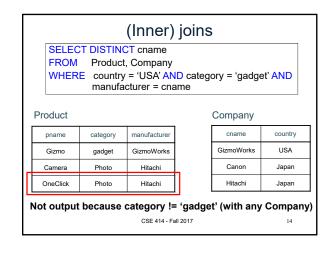


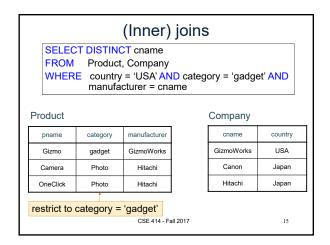


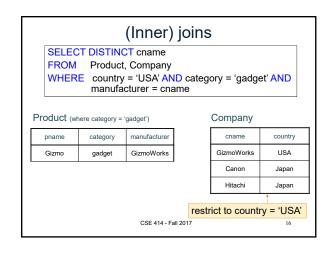


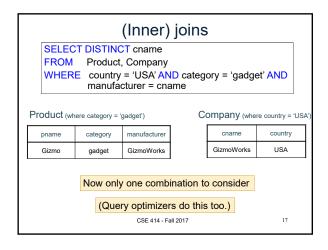


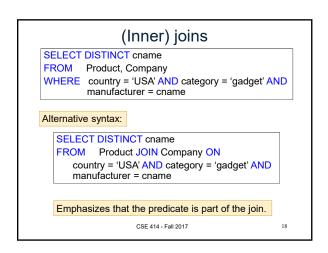












Self-Joins and Tuple Variables

- Ex: find companies that manufacture both products in the 'gadgets' category and in the 'photo' category
- Just joining Company with Product is insufficient: need to join Company with Product with Product

```
FROM Company, Product, Product
```

- When a relation occurs twice in the FROM clause we call it a self-join; in that case every column name in Product is ambiguous (why?)
 - are you referring to the tuple in the 2nd or 3rd loop?

CSE 414 - Fall 2017

19

Name Conflicts

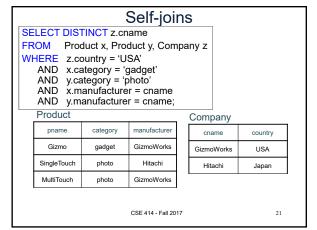
we used cname / pname to avoid this problem

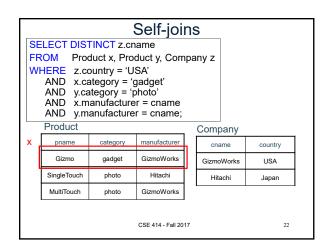
- When a name is ambiguous, qualify it:
 - WHERE Company.name = Product.name AND ...
- · For self-join, we need to distinguish tables:

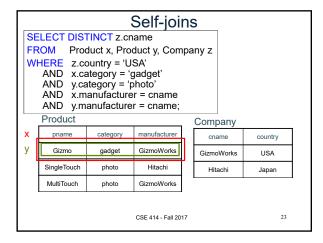
FROM Product x, Product y, Company

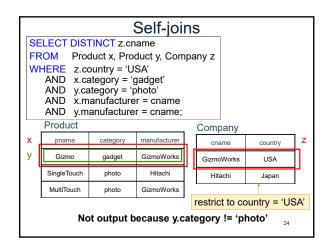
- · These new names are called "tuple variables"
 - can think of as name for the variable of each loop
 - can also write "Company AS C" etc.
 - can make SQL query shorter: C.name vs. Company.name

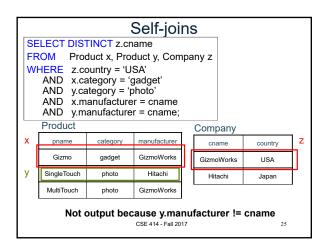
F 414 - Fall 2017

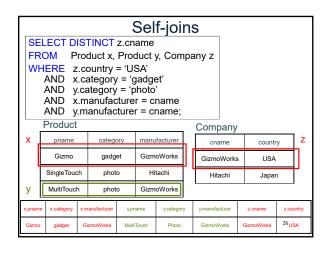


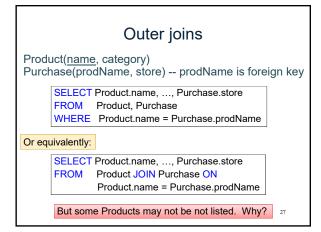




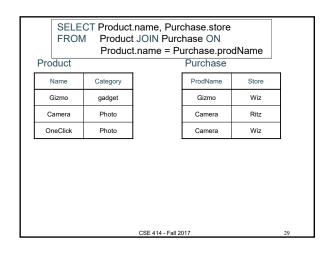


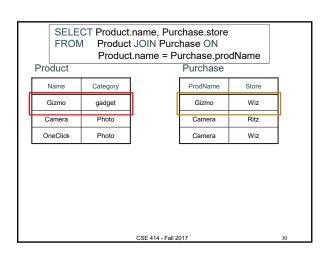


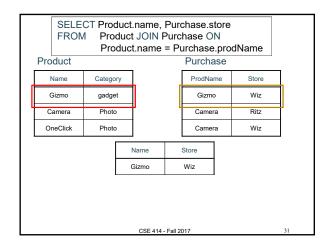


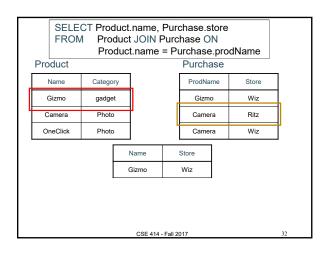


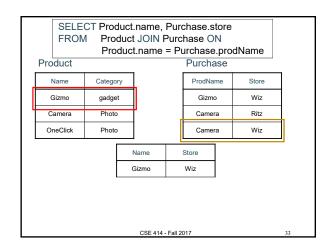


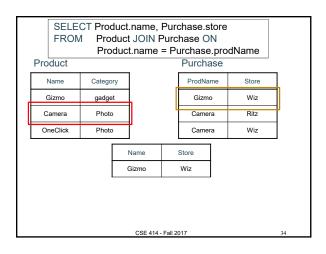


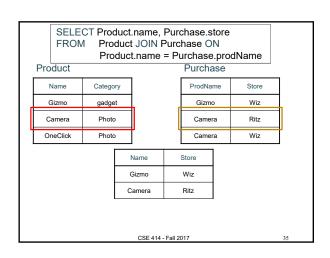


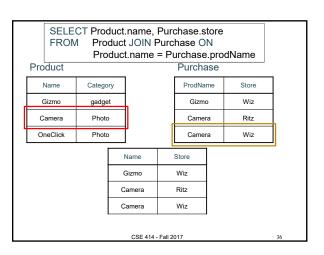


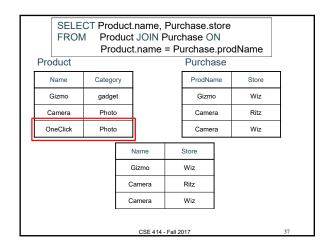


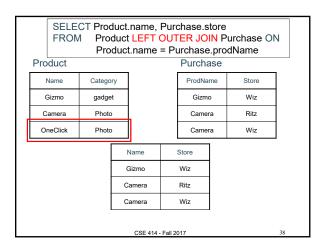


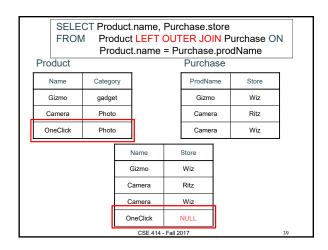


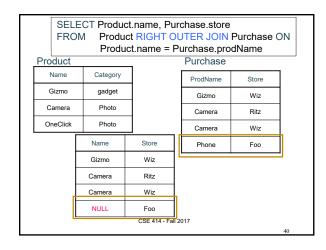


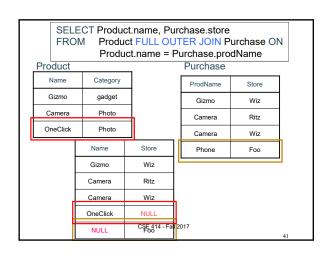












Outer Joins

- · Left outer join:
 - Include the left tuple even if there's no match
- · Right outer join:
 - Include the right tuple even if there's no match
- Full outer join:
 - Include both left and right tuples even if there's no match
- (Also something called a UNION JOIN, though it's rarely used.)
- (Actually, all of these are used much more rarely than inner joins.)

CSE 414 - Fall 2017

Outer Joins Example

See lec04-sql-outer-joins.sql...

CSE 414 - Fall 2017

43

45

Aggregation in SQL

Comment about SQLite

- One cannot load NULL values such that they are actually loaded as null values
- · So we need to use two steps:
 - Load null values using some type of special value
 - Update the special values to actual null values

```
update Purchase
  set price = null
  where price = 'null'
```

CSE 414 - Fall 2017

Simple Aggregations

Five basic aggregate operations in SQL

```
select count(*) from Purchase
select sum(quantity) from Purchase
select avg(price) from Purchase
select max(quantity) from Purchase
select min(quantity) from Purchase
```

Except count, all aggregations apply to a single value

CSE 414 - Fall 2017

46

Aggregates and NULL Values

```
Null values are not used in aggregates
```

```
insert into Purchase
values(12, 'gadget', NULL, NULL, 'april')
Let's try the following
select count(*) from Purchase
select count(quantity) from Purchase
select sum(quantity) from Purchase
select sum(quantity)
from Purchase
where quantity is not null;
```

CSE 414 - Fall 2017

Aggregates and NULL Values

```
Null values are not used in aggregates
```

```
insert into Purchase
values(12, 'gadget', NULL, NULL, 'april')

Let's try the following

select count(*) from Purchase

-- NULL is counted in count(*)
select count(quantity) from Purchase

-- NULL is ignored in count(quantity)

select sum(quantity) from Purchase

select sum(quantity) from Purchase

select sum(quantity)
from Purchase

where quantity is not null;

-- "is not null;

-- "is redundant

SE414-Fall 2017 48
```

Counting Duplicates

COUNT applies to duplicates, unless otherwise stated:

SELECT Count(product) FROM Purchase WHERE price > 4.99

same as Count(*) if no nulls

49

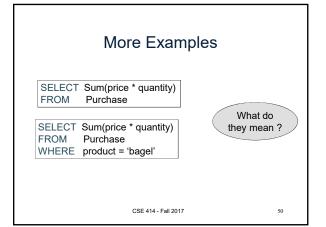
We probably want:

SELECT Count(DISTINCT product)

FROM Purchase

WHERE price> 4.99

CSE 414 - Fall 2017



Simple Aggregations

Purchase

Product Price Quantity Bagel 3 20 1.50 20 Bagel Banana 0.5 50 2 10 Banana 4 10 Banana

SELECT Sum(price * quantity) Purchase FROM

WHERE product = 'Bagel'

90 (= 60+30)

51 CSE 414 - Fall 2017

Simple Aggregations

Product Price Quantity Purchase. Bagel 3 20 1.50 Bagel 20 Banana 0.5 50 2 10 Banana

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

Banana

90 (= 60+30)

10

CSE 414 - Fall 2017

4

More Examples

How can we find the average revenue per sale?

SELECT sum(price * quantity) / count(*)

FROM Purchase

WHERE product = 'bagel'

How can we find the average price of a bagel sold?

SELECT sum(price * quantity) / sum(quantity)

FROM Purchase

WHERE product = 'bagel'

CSE 414 - Fall 2017

More Examples

SELECT sum(price * quantity) / count(*)

FROM Purchase

WHERE product = 'bagel'

SELECT sum(price * quantity) / sum(quantity)

FROM Purchase

WHERE product = 'bagel'

What happens if there are NULLs in price or quantity?

Lesson: disallow NULLs unless you need to handle them

CSE 414 - Fall 2017