Example queries from lecture cse444

http://www.cs.washington.edu/education/courses/cse444/11wi/
version Jan 7, 2011

Motivation: this document should make it easy for you to get started and repeat (and tweak) some queries we see in class. Just copy and paste the grey areas into SQL server and execute. If you see something that bothers you, please let us or the TAs know.

Lecture 2: our first DISTINCT / ORDER BY queries

-- Create Company tables.

-- Comments in SQL are just two dashes.

-- First statements checks if table already defined (don't let yourself get confused, just ignore).
if exists (select table_name
            from information_schema.tables
            where table_name='Company') drop table Company;

create table Company (  
  CName char(20) PRIMARY KEY,  
  StockPrice int,  
  Country char(20));

insert into Company values ('GizmoWorks', 25, 'USA');
insert into Company values ('Canon', 65, 'Japan');
insert into Company values ('Hitachi', 15, 'Japan');

select * from Company;

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

-- Attempt a key violation.
insert into Company values ('Canon', 65, 'USA');

Msg 2627, Level 14, State 1, Line 1
Violation of PRIMARY KEY constraint 'PK__Company__85D445AB0519C6AF'. Cannot insert duplicate key in object 'dbo.Company'.
The duplicate key value is ('Canon').
The statement has been terminated.

-- Create Product tables.

if exists (select table_name
            from information_schema.tables
            where table_name='Product') drop table Product;

-- Alternative syntax to specify key constraint. Note that "constraint some_name" is optional,
create table Product (  
  PName char(20),  
  Price decimal(9, 2),  
  ...
Category   char(20),
Manufacturer char(20),
CONSTRAINT some_name PRIMARY KEY (PName));

insert into Product values ('Gizmo', 19.99, 'Gadgets', 'GizmoWorks');
insert into Product values ('PowerGizmo', 29.99, 'Gadgets', 'GizmoWorks');
insert into Product values ('SingleTouch', 149.99, 'Photography', 'Canon');
insert into Product values ('MultiTouch', 203.99, 'Household', 'Hitachi');

select * from Product;

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

-- We realize we forgot the foreign key constraints. Let's make up for that.
age table Product
ADD FOREIGN KEY (Manufacturer) REFERENCES Company(CName);

-- Here how we could have defined both key and foreign key constraint while defining the table. Remember SQL is not case sensitive.
create table Product (  
    PName   char(20) PRIMARY KEY,  
    Price   decimal(9, 2),  
    Category char(20),  
    Manufacturer char(20) FOREIGN KEY REFERENCES Company(CName));

-- Let's attempt to delete a tuple from Company. This is the default behavior. But could be defined differently (if interested book 7.1.2)
delete Company
where CName = 'Canon';
Msg 547, Level 16, State 0, Line 2
The DELETE statement conflicted with the REFERENCE constraint "FK__Product__Manufac__164452B1". The conflict occurred in
database "TestExamples", table "dbo.Product", column 'Manufacturer'.
The statement has been terminated.

-- Queries with DISTINCT and ORDER BY
select DISTINCT category
from Product
order by pName;
category
Gadgets
Household
Gadgets
Photography

-- This query creates a syntax error. (To be more specific, the error happens during the semantic analysis of the query)
select DISTINCT category
from Product
order by pName;
**Lecture 2: Conceptual query evaluation**

-- Create new tables

```
if exists (select table_name
    from information_schema.tables
    where table_name= 'R') drop table R;
if exists (select table_name
    from information_schema.tables
    where table_name= 'S') drop table S;
if exists (select table_name
    from information_schema.tables
    where table_name= 'T') drop table T;

create table R (a int);
create table S (a int);
create table T (a int);

insert into R values (1);
insert into R values (2);
insert into R values (3);
insert into R values (4);
insert into R values (5);
insert into S values (4);
insert into S values (5);
insert into S values (6);
insert into S values (7);
```

-- Look for intersection between R and S. Note that are two result tuples (the first line is the attribute name)
```
select DISTINCT R.a
from R, S
where R.a=S.a;
```

```
a
4
5
```

-- The following query delivers an empty result. Seems counterintuitive if we just think about the logics
```
select DISTINCT R.a
from R, S, T
where R.a=S.a
  or R.a=T.a
```

```
a
```

-- After inserting a single tuple into T (that has nothing to do with R and S), the query again gives the original 2 tuples.
Lecture 2: Nested queries in select clause

Product (pname, price, cid)
Company (cid, cname, city)

-- Create tables for slightly changed schema.

if exists (select table_name
from information_schema.tables
where table_name='Product') drop table Product;
if exists (select table_name
from information_schema.tables
where table_name='Company') drop table Company;

create table Product (
    pname char(20),
    price int,
    cid int);
create table Company (
    cid int,
    cname char(20),
    city char(20));

insert into Product values ('Gelato', 11, 1);
insert into Product values ('Gelato', 12, 2);
insert into Product values ('Baguette', 3, 3);
insert into Company values (1, 'Francesco', 'Roma');
insert into Company values (2, 'Frederico', 'Roma');
insert into Company values (3, 'Francois', 'Paris');

select * from Product;
select * from Company;

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Gelato</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Baguette</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Francesco</td>
<td>Roma</td>
</tr>
<tr>
<td>2</td>
<td>Frederico</td>
<td>Roma</td>
</tr>
<tr>
<td>3</td>
<td>Francois</td>
<td>Paris</td>
</tr>
</tbody>
</table>
-- This query can produce runtime errors, depending on the database instance. Over this instance it runs.

```sql
select P.pname, (select C.city from Company C where C.cid = P.cid)
from Product P
```

<table>
<thead>
<tr>
<th>Pname</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Baguette</td>
<td>Paris</td>
</tr>
</tbody>
</table>

-- Slight variation.

```sql
select DISTINCT P.pname, (select C.city from Company C where C.cid = P.cid)
from Product P
```

<table>
<thead>
<tr>
<th>Pname</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Baguette</td>
<td>Paris</td>
</tr>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
</tbody>
</table>

-- Now let's change one value ("update one tuple") in the database.

```sql
update Company
set city = 'Pisa'
where cid = 2;
```

```sql
select * from Company;
```

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Francesco</td>
<td>Roma</td>
</tr>
<tr>
<td>2</td>
<td>Frederico</td>
<td>Pisa</td>
</tr>
<tr>
<td>3</td>
<td>Francois</td>
<td>Paris</td>
</tr>
</tbody>
</table>

-- The query still executes fine

```sql
select P.pname, (select C.city from Company C where C.cid = P.cid)
from Product P
```

<table>
<thead>
<tr>
<th>Pname</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Gelato</td>
<td>Pisa</td>
</tr>
<tr>
<td>Baguette</td>
<td>Paris</td>
</tr>
</tbody>
</table>

-- Now let's change back to original 'Roma' value, but change the id (for whatever reason)

```sql
update Company
set city = 'Roma'
where cid = 2;
```

```sql
update Company
set cid = 1
where cid = 2;
```
select * from Company;

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Francesco</td>
<td>Roma</td>
</tr>
<tr>
<td>1</td>
<td>Frederico</td>
<td>Roma</td>
</tr>
<tr>
<td>3</td>
<td>Francois</td>
<td>Paris</td>
</tr>
</tbody>
</table>

-- Now, the query does not execute. We get a runtime error.

```
select P.pname, 
     (select C.city
      from Company C
      where C.cid = P.cid)
from Product P
```

Msg 512, Level 16, State 1, Line 1
Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <=, >, >= or when the subquery is used as an expression.

-- Unnesting makes it work
```
select P.pname, C.city
from Product P, Company C
where C.cid = P.cid
```

<table>
<thead>
<tr>
<th>pname</th>
<th>(No column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Baguette</td>
<td>Paris</td>
</tr>
</tbody>
</table>

-- Let's just add a little DISTINCT in the nested query. What is happening here.
```
select P.pname, 
     (select DISTINCT C.city
      from Company C
      where C.cid = P.cid)
from Product P
```

-- Think about the conceptual evaluation strategy as follows: The query starts from the "FROM Product" clause. There is no "WHERE ..." clause, so all tuples are given to the "SELECT ..." clause. For the second tuple, the query can find a `pname = 'Gelato'`, but no matching result from the nested subquery. Hence a NULL.

-- In a side remark I said something different on Wednesday. Sorry!

<table>
<thead>
<tr>
<th>pname</th>
<th>(No column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Gelato</td>
<td>NULL</td>
</tr>
<tr>
<td>Baguette</td>
<td>Paris</td>
</tr>
</tbody>
</table>

-- Let's just add one more tuple into the original database. To keep track of the database instance, let's start all over from scratch.
```
if exists (select table_name
            from information_schema.tables
            where table_name='Product') drop table Product;
if exists (select table_name
            from information_schema.tables
            where table_name='Company') drop table Company;
create table Product (
    pname     char(20),
    ...)
```sql
price int,
cid int);
create table Company (cid int,
cname char(20),
city char(20));
insert into Product values ('Gelato', 11, 1);
insert into Product values ('Gelato', 12, 2);
insert into Product values ('Baguette', 3, 3);
insert into Product values ('Fish Soup', 29, 4); -- new tuple
insert into Company values (1, 'Francesco', 'Roma');
insert into Company values (2, 'Frederico', 'Roma');
insert into Company values (3, 'Francois', 'Paris');
select * from Product;
select * from Company;

<table>
<thead>
<tr>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>Gelato</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>Baguette</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Fish Soup</td>
<td>29</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Francesco</td>
<td>Roma</td>
</tr>
<tr>
<td>2</td>
<td>Frederico</td>
<td>Roma</td>
</tr>
<tr>
<td>3</td>
<td>Francois</td>
<td>Paris</td>
</tr>
</tbody>
</table>

-- The query still executes fine, but returns the NULL, because it considers each tuple from Product.
select P.pname, (select C.city from Company C where C.cid = P.cid) from Product P

<table>
<thead>
<tr>
<th>pname</th>
<th>(No column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Gelato</td>
<td>Pisa</td>
</tr>
<tr>
<td>Baguette</td>
<td>Paris</td>
</tr>
<tr>
<td>Fish Soup</td>
<td>NULL</td>
</tr>
</tbody>
</table>

-- Unnesting makes it work without the NULL. Now the conceptual evaluation strategy iterates over the crossproduct between both tables (both tables appear in the "FROM clause"). Only those joins pass the "WHERE clause", which finds mates through the join. No NULL returned.
select P.pname, C.city from Product P, Company C where C.cid = P.cid

<table>
<thead>
<tr>
<th>pname</th>
<th>(No column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gelato</td>
<td>Roma</td>
</tr>
<tr>
<td>Gelato</td>
<td>Pisa</td>
</tr>
<tr>
<td>Baguette</td>
<td>Paris</td>
</tr>
</tbody>
</table>
```
Lecture 3: Aggregates

Purchase (product, price, quantity)

-- Reason why we always use this conditional delete at the beginning is that it is just comfortable: If the table already exists, it gets deleted ("dropped"). That way one can execute the whole grayed out area over and over again. You do not need to know that. Stuff like that, one can look up.

```sql
if exists (select table_name
            from information_schema.tables
            where table_name= 'Purchase') drop table Purchase;

create table Purchase (  
    product char(20),
    price int,
    quantity int);

insert into Purchase values ('Bagel', 3, 20);
insert into Purchase values ('Bagel', 2, 20);
insert into Purchase values ('Banana', 1, 50);
insert into Purchase values ('Banana', 2, 10);
insert into Purchase values ('Banana', 4, 10);
```

select * from Purchase;

<table>
<thead>
<tr>
<th>product</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

-- First, let’s look at a few very simple examples

```sql
select count(product)
from Purchase

(No column name)
5

select count(DISTINCT product)
from Purchase

(No column name)
2
```
Following makes less sense, but still possible

```sql
select sum(DISTINCT quantity)
from Purchase

(No column name)
80
```

Simple Aggregate group by query

```sql
select product, sum(quantity) as TotalSales
from Purchase
where price > 1
group by product
```

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
</tbody>
</table>

Nested query that is equivalent to aggregate group by query

```sql
select distinct x.product,
           ( select sum(y.quantity)
             from Purchase y
             where x.product = y.product
             and price > 1 ) as TotalSales
from Purchase x
where price > 1
```

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
</tbody>
</table>

Why do we need twice the "price > 1" condition before: So let's insert one more product (with price not > 1) and see the problem if we leave out the outer price > 1 or the inner price > 1. This should be very revealing what is going on.

```sql
insert into Purchase values ('Bubble Gum', 1, 100);
```

```sql
select * from Purchase;
```

<table>
<thead>
<tr>
<th>product</th>
<th>price</th>
<th>quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Bagel</td>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>Banana</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>Banana</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Bubble Gum</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

We issue the changed query. SUM here returns NULL

```sql
select distinct x.product,
           ( select sum(y.quantity)
             from Purchase y
```
where x.product = y.product and price > 1) as TotalSales
from Purchase x

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>20</td>
</tr>
<tr>
<td>Bubble Gum</td>
<td>NULL</td>
</tr>
</tbody>
</table>

On a side remark, COUNT returns 0 here.

```
select distinct x.product,
    ( select count(y.quantity)
      from Purchase y
      where x.product = y.product and price > 1 ) as TotalSales
from Purchase x

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>2</td>
</tr>
<tr>
<td>Banana</td>
<td>2</td>
</tr>
<tr>
<td>Bubble Gum</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Next let’s leave the inner “price > 1” condition away.

```
select distinct x.product,
    ( select sum(y.quantity)
      from Purchase y
      where x.product = y.product) as TotalSales
from Purchase x
where price > 1

<table>
<thead>
<tr>
<th>Product</th>
<th>TotalSales</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bagel</td>
<td>40</td>
</tr>
<tr>
<td>Banana</td>
<td>70</td>
</tr>
</tbody>
</table>
```

Aggregate with having.

```
select product,
    sum(quantity) as SumQuantity,
    max(price) as MaxPrice
from Purchase
group by product
having sum(quantity) > 50

<table>
<thead>
<tr>
<th>Product</th>
<th>SumQuantity</th>
<th>MaxPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banana</td>
<td>70</td>
<td>4</td>
</tr>
<tr>
<td>Bubble Gum</td>
<td>100</td>
<td>1</td>
</tr>
</tbody>
</table>
```

Aggregate with having. Question from class: Can we include an aggregate condition even if we do not include this aggregate in the SELECT clause. I was hesitating in class. Answer should have been an unconditional: yes we can. The thing that should guide the answer is the evaluation strategy on slide 12 of lecture 12. SQL checks the conditions on the grouping via HAVING before (!) it determines what to output through the SELECT.

```
select product,
    max(price) as MaxPrice
from Purchase
group by product
having sum(quantity) > 50
  -- we remove the sum (quantity)

select product,
    max(price) as MaxPrice
from Purchase
group by product
having sum(quantity) > 50
  -- but still keep it in the HAVING clause
```
<table>
<thead>
<tr>
<th>product</th>
<th>MaxPrice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Banana</td>
<td>4</td>
</tr>
<tr>
<td>Bubble Gumball</td>
<td>1</td>
</tr>
</tbody>
</table>

-- But we can make the DMBS easily unhappy by having an attribute in the SELECT which is not in the GROUP BY

```
select product,
       price
from Purchase
group by product
```

Msg 8120, Level 16, State 1, Line 2
Column 'Purchase.price' is invalid in the select list because it is not contained in either an aggregate function or the GROUP BY clause.

**Lecture 3: NULLS**

Product (pname, price, cid)

Company (cid, cname, city)

-- Create tables for slightly changed schema.

```
if exists (select table_name
            from information_schema.tables
            where table_name='Product') drop table Product;
if exists (select table_name
            from information_schema.tables
            where table_name='Company') drop table Company;

create table Product (
    pname char(20),
    price int,
    cid int);
create table Company (cid int,
                      cname char(20),
                      city char(20));

insert into Product values ('Gelato', 11, 1);
insert into Product values ('Gelato', 12, 2);
insert into Product values ('Baguette', 3, 3);
insert into Product values ('Baklava', 10, NULL);
insert into Company values (1, 'Francesco', 'Roma');
insert into Company values (2, 'Frederico', 'Roma');
insert into Company values (3, 'Francois', 'Paris');
insert into Company values (4, 'Luis', NULL);
insert into Company values (5, 'Greco', NULL);
insert into Company values (6, 'Thomas', 'Berlin');

select * from Product;
select * from Company;
```
<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Francesco</td>
<td>Roma</td>
<td>Gelato</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Frederico</td>
<td>Roma</td>
<td>Gelato</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Francois</td>
<td>Paris</td>
<td>Baguette</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>Luis</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>5</td>
<td>Greco</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>6</td>
<td>Thomas</td>
<td>Berlin</td>
<td>NULL</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>

Q: Find the number of companies in each city.

--- Unnested version. SQL server groups the NULLs together.

```sql
select city, count(*)
from Company
group by city
```

<table>
<thead>
<tr>
<th>city</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>2</td>
</tr>
<tr>
<td>Berlin</td>
<td>1</td>
</tr>
<tr>
<td>Paris</td>
<td>1</td>
</tr>
<tr>
<td>Roma</td>
<td>2</td>
</tr>
</tbody>
</table>

--- Nested version. It still outputs the NULL, but the inner loop cannot match anything to NULL, because “NULL = NULL” is always false (this was also a short student question in lecture 2).

```sql
select DISTINCT city, (select count(*)
from Company Y
where X.city = Y.city)
from Company X
```

<table>
<thead>
<tr>
<th>city</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>Berlin</td>
<td>1</td>
</tr>
<tr>
<td>Paris</td>
<td>1</td>
</tr>
<tr>
<td>Roma</td>
<td>2</td>
</tr>
</tbody>
</table>

--- Joins ignore NULL (same reason as above)

```sql
select *
from Company X, Product Y
where X.cid = Y.cid
```

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>city</th>
<th>pname</th>
<th>price</th>
<th>cid</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Francesco</td>
<td>Roma</td>
<td>Gelato</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>Frederico</td>
<td>Roma</td>
<td>Gelato</td>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>Francois</td>
<td>Paris</td>
<td>Baguette</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Q: Find the number of products made in each city.

```sql
select X.city, count(*)
```
from Company X, Product Y
where X.cid = Y.cid

group by X.city

<table>
<thead>
<tr>
<th>city</th>
<th>(No column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paris</td>
<td>1</td>
</tr>
<tr>
<td>Roma</td>
<td>2</td>
</tr>
</tbody>
</table>

-- COUNT initializes from 0.

<table>
<thead>
<tr>
<th>city</th>
<th>(No column name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>0</td>
</tr>
<tr>
<td>Berlin</td>
<td>0</td>
</tr>
<tr>
<td>Paris</td>
<td>1</td>
</tr>
<tr>
<td>Roma</td>
<td>2</td>
</tr>
</tbody>
</table>

Optional: how COUNT and SUM are initialized

Product(pname, category)
Purchase(prodName, month, store)

-- Create tables for slightly changed schema.

if exists (select table_name
    from information_schema.tables
    where table_name= 'Product') drop table Product;
if exists (select table_name
    from information_schema.tables
    where table_name= 'Purchase') drop table Purchase;

create table Product (    
    pname char(20),
    category char(20));
create table Purchase (    
    prodName char(20),
    month char(20),
    store char(20));

insert into Product values ('Gelato', 'food');
insert into Product values ('Baguette', 'food');
insert into Product values ('Baklava', 'food');

insert into Purchase values ('Gelato', 'September', 'Francesco');
insert into Purchase values ('Baguette', 'September', 'Francois');
insert into Purchase values ('Baguette', 'September', NULL);

select * from Product;
select * from Purchase;

<table>
<thead>
<tr>
<th>pname</th>
<th>category</th>
</tr>
</thead>
</table>
### Q: Compute, for each product, the total number of sales in 'September'

```
SELECT Product.pname, count(*)
FROM Product, Purchase
WHERE Product.pname = Purchase.prodName
    and Purchase.month = 'September'
GROUP BY Product.pname
```

<table>
<thead>
<tr>
<th>pname</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baguette</td>
<td>2</td>
</tr>
<tr>
<td>Gelato</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**First with count(store)**

```
SELECT Product.pname, count(store)
FROM Product LEFT OUTER JOIN Purchase ON
    Product.pname = Purchase.prodName
    and Purchase.month = 'September'
GROUP BY Product.pname
```

<table>
<thead>
<tr>
<th>pname</th>
<th>count(store)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baguette</td>
<td>1</td>
</tr>
<tr>
<td>Baklava</td>
<td>0</td>
</tr>
<tr>
<td>Gelato</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Then with count(store)**

```
SELECT Product.pname, count(month)
FROM Product LEFT OUTER JOIN Purchase ON
    Product.pname = Purchase.prodName
    and Purchase.month = 'September'
GROUP BY Product.pname
```

<table>
<thead>
<tr>
<th>pname</th>
<th>count(month)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baguette</td>
<td>2</td>
</tr>
<tr>
<td>Baklava</td>
<td>0</td>
</tr>
<tr>
<td>Gelato</td>
<td>1</td>
</tr>
</tbody>
</table>

---

**Then with count(*)**

```
SELECT Product.pname, count(*)
FROM Product LEFT OUTER JOIN Purchase ON
    Product.pname = Purchase.prodName
    and Purchase.month = 'September'
GROUP BY Product.pname
```

<table>
<thead>
<tr>
<th>pname</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baguette</td>
<td>2</td>
</tr>
</tbody>
</table>
Person-bar-drink

Background: The original and commonly used schema is
Likes (drinker, beer)
Frequents (drinker, bar)
Serves (bar, beer)

We use here instead
Likes (person, drink)
Frequents (person, bar)
Serves (bar, drink)

to ensure that all attributes start with different letters (nothing personal against beer). That allows to abbreviate the schema in the logical representation (not relevant for now) as
L(p,d)
F(p,b)
S(b,d)
and we have unique letters. That simplifies stuff. Thus, we have (d,b,b) -> (p,b,d). Not relevant.

create table Likes(person varchar(20), drink varchar(20))
create table Frequents(person varchar(20), bar varchar(20))
create table Serves(bar varchar(20), drink varchar(20))

insert into Likes values ('Alice', 'Whitebeer');
insert into Likes values ('Bob', 'Brownbeer');
insert into Likes values ('Charlie', 'Whitebeer');
insert into Likes values ('Charlie', 'Blackbeer');
insert into Serves values ('Groundbar', 'Whitebeer');
insert into Serves values ('Seabar', 'Whitebeer');
insert into Serves values ('Seabar', 'Blackbeer');
insert into Serves values ('Skybar', 'Whitebeer');
insert into Serves values ('Skybar', 'Brownbeer');
insert into Serves values ('Skybar', 'Blackbeer');
insert into Frequents values ('Alice', 'Seabar');
insert into Frequents values ('Alice', 'Skybar');
insert into Frequents values ('Bob', 'Groundbar');
insert into Frequents values ('Bob', 'Seabar');
insert into Frequents values ('Charlie', 'Seabar');

1 Find persons that frequent some bar that serves some drink they like.

Note we ignore here the DISTINCT to see what is happening. In any "real" example, you should use DISTINCT. Please don’t forget 😊

-- Find persons that frequent some bar that serves some drink they like.
select  F.person
from Frequents F, Likes L, Serves S
where F.person = L.person
and F.bar = S.bar
and L.drink = S.drink

-- Above is unnested version of this here.
select F.person
from Frequents F
where exists
  (select *
  from Serves S
  where S.bar = F.bar
  and exists
    (select *
    from Likes L
    where L.person = F.person
    and S.drink = L.drink))

person
Alice
Alice
Charlie

2 Find persons that frequent only bars that serve some drink they like.

-- Find persons that frequent only bars that serve some drink they like:
select F1.person
from Frequents F1
where not exists
  (select *
  from Frequents F2
  where F2.person = F1.person
  and not exists
    (select *
    from Serves S3, Likes L4
    where L4.person = F1.person
    and S3.drink = L4.drink
    and S3.bar = F2.bar)))

person
Alice
Alice
Charlie
3 Find persons that frequent some bar that serves only drinks they like.

\[
\text{select F.person from frequents F where not exists (select * from serves S where F.bar = S.bar and not exists (select * from Likes L where L.person = F.person and S.drink = L.drink))}
\]

<table>
<thead>
<tr>
<th>person</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charlie</td>
</tr>
</tbody>
</table>

4 Find persons that frequent only bars that serve only drinks they like

\[
\text{\ldots}
\]

OUT: example database from Chakravarthy PDF

Schema:
Likes (person, drink)
Frequents (person, bar)
Serves (bar, drink)

SQL inserts:
create table Likes(person varchar(20), drink varchar(20))
create table Frequents(person varchar(20), bar varchar(20))
create table Serves(bar varchar(20), drink varchar(20))

insert into Likes values ('Charles', 'Michelob')
insert into Likes values ('Charles', 'Bud')
insert into Likes values ('Mickey', 'Michelob')
insert into Likes values ('Tracy', 'Natural')
insert into Likes values ('Mallory', 'Michelob')
insert into Likes values ('Mallory', 'Root')
insert into Likes values ('Alex', 'Natural')
insert into Likes values ('Alex', 'Michelob')
insert into Likes values ('Alex', 'Bud')
insert into Likes values ('Alex', 'Michelob')

insert into Frequents values ('Charles', 'Purple Purpoise')
insert into Frequents values ('Charles', 'Orange-and-Brew')
insert into Frequents values ('Charles', 'Kaos')
insert into Frequents values ('Mickey', 'Kaos')
insert into Frequents values ('Tracy', 'Orange-and-Brew')
try to find some instance that allows you to check all 4 different queries from before and see the difference in behavior (some example where the answer illustrates the query is correct. Not trivial, right.