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. Le's make up for that.
alter 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;
```

```
Msg 145, Level 15, State 1, Line 1
ORDER BY items must appear in the select list if SELECT DISTINCT is specified.
```

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.
```

```
insert into T values (10);

select     DISTINCT R.a
from       R, S,T
where      R.a=S.a
        or R.a=T.a
```

a
4
5

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;
```

pname	price	cid
Gelato	11	1
Gelato	12	2
Baguette	3	3

cid	cname	city
1	Francesco	Roma
2	Frederico	Roma
3	Francois	Paris

```
-- This query can produce runtime errors, depending on the database instance. Over this instance ir runs.
select    P.pname, (          select    C.city
                              from      Company C
                              where     C.cid = P.cid)
from      Product P
```

pname	(No column name)
Gelato	Roma
Gelato	Roma
Baguette	Paris

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

pname	(No column name)
Baguette	Paris
Gelato	Roma

```
-- Now let's change one value ("update one tuple") in the database.
update    Company
set       city = 'Pisa'
where     cid= 2;

select * from Company;
```

cid	cname	city
1	Francesco	Roma
2	Frederico	Pisa
3	Francois	Paris

```
-- The query still executes fine
select    P.pname, (          select    C.city
                              from      Company C
                              where     C.cid = P.cid)
from      Product P
```

pname	(No column name)
Gelato	Roma
Gelato	Pisa
Baguette	Paris

```
-- Now let's change back to original 'Roma' value, but change the id (for whatever reason)
update    Company
set       city = 'Roma'
where     cid= 2;
update    Company
set       cid = 1
where     cid= 2;
```

```
select * from Company;
```

cid	cname	city
1	Francesco	Roma
1	Frederico	Roma
3	Francois	Paris

```
-- 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
```

pname	(No column name)
Gelato	Roma
Gelato	Roma
Baguette	Paris

```
-- 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!
```

pname	(No column name)
Gelato	Roma
Gelato	NULL
Baguette	Paris

```
-- 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),
            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;
```

pname	price	cid
Gelato	11	1
Gelato	12	2
Baguette	3	3
Fish Soup	29	4

cid	cname	city
1	Francesco	Roma
2	Frederico	Roma
3	Francois	Paris

```
-- 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
```

pname	(No column name)
Gelato	Roma
Gelato	Pisa
Baguette	Paris
Fish Soup	NULL

```
-- 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
```

pname	(No column name)
Gelato	Roma
Gelato	Pisa
Baguette	Paris

… to be continued. Or feel free to play around. Learning by doing. Learning by playing.

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.

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;
```

product	price	quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10

```
-- First, let’s look at a few very simple examples
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
select     sum(DISTINCT quantity)
from       Purchase
```

(No column name)
80

```
-- Simple Aggregate group by query
select     product, sum(quantity) as TotalSales
from       Purchase
where      price > 1
group by product
```

Product	TotalSales
Bagel	40
Banana	20

```
-- Nested query that is equivalent to aggregate group by query
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
```

Product	TotalSales
Bagel	40
Banana	20

```
-- 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.

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

select * from Purchase;
```

product	price	quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10
Bubble Gum	1	100

```
-- We issue the changed query. SUM here returns NULL
select     distinct x.product,
           ( select   sum(y.quantity)
             from     Purchase y
             where   x.product = y.product
             and       price > 1) as TotalSales
from       Purchase x
```

Product	TotalSales
Bagel	40
Banana	20
Bubble Gum	NULL

```
-- 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
```

Product	TotalSales
Bagel	2
Banana	2
Bubble Gum	0

```
-- 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
```

Product	TotalSales
Bagel	40
Banana	70

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

product	SumQuantity	MaxPrice
Banana	70	4
Bubble Gum	100	1

```
-- 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 hesiting in class. Answer should have been an unconditional: yes we can. The thing that should gude 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     -- we remove the sum (quantity)
from       Purchase
group by product
having   sum(quantity) > 50            -- but still keep it in the HAVING clause
```

product	MaxPrice
Banana	4
Bubble Gum	1

```
-- 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;
```

pname	price	cid
Gelato	11	1
Gelato	12	2
Baguette	3	3
Baklava	10	NULL

cid	cname	city
1	Francesco	Roma
2	Frederico	Roma
3	Francois	Paris
4	Luis	NULL
5	Greco	NULL
6	Thomas	Berlin

Q:Find the number of companies in each city.

```
-- Unnested version. SQLserver groups the NULLs together.
select     city, count(*)
from       Company
group by city
```

city	(No column name)
NULL	2
Berlin	1
Paris	1
Roma	2

```
-- 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).

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

city	(No column name)
NULL	0
Berlin	1
Paris	1
Roma	2

```
-- Joins ignore NULL (same reason as above)
select     *
from       Company X, Product Y
where      X.cid = Y.cid
```

cid	cname	city	pname	price	cid
1	Francesco	Roma	Gelato	11	1
2	Frederico	Roma	Gelato	12	2
3	Francois	Paris	Baguette	3	3

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

```
select     X.city, count(*)
```

```
from        Company X, Product Y
where       X.cid = Y.cid
group by X.city
```

city	(No column name)
Paris	1
Roma	2

```
-- COUNT initializes from 0.
select      DISTINCT X.city, (select  count(*)
                              from        Product Y, Company Z
                              where       Z.cid = Y.cid
                              and Z.city = X.city)
from        Company X
```

city	(No column name)
NULL	0
Berlin	0
Paris	1
Roma	2

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;
```

pname	category

Gelato	food
Baguette	food
Baklava	food

prodName	month	store
Gelato	September	Francesco
Baguette	September	Francois
Baguette	September	NULL

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
```

pname	(No column name)
Baguette	2
Gelato	1

```
-- 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
```

pname	(No column name)
Baguette	1
Baklava	0
Gelato	1

```
-- 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
```

pname	(No column name)
Baguette	2
Baklava	0
Gelato	1

```
-- 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
```

pname	(No column name)
Baguette	2

Baklava	1
Gelato	1

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
```

person
Alice
Alice
Charlie
Charlie

```
-- 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         -- alternatively use F2 here instead of F1
                    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.

```
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))
```

person
Charlie

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

…

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 ('Tracy', 'Bud')
insert into Likes values ('Mallory', '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 ('Brian', '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')
insert into Frequents values ('Tracy', 'Kaos')
insert into Frequents values ('Tracy', 'Cafeteria')
insert into Frequents values ('Mallory', 'Orange-and-Brew')
insert into Frequents values ('Alex', 'Orange-and-Brew')
insert into Frequents values ('Brian', 'Orange-and-Brew')
insert into Frequents values ('Brian', 'Purple Purpoise')

insert into Serves values ('Purple Purpoise', 'Michelob')
insert into Serves values ('Purple Purpoise', 'Natural')
insert into Serves values ('Purple Purpoise', 'Bud')
insert into Serves values ('Kaos', 'Bud')
insert into Serves values ('Orange-and-Brew', 'Natural')
insert into Serves values ('Orange-and-Brew', 'Michelob')
insert into Serves values ('Cafeteria', 'Root')
```

… 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.