L03: SQL: Intermediate

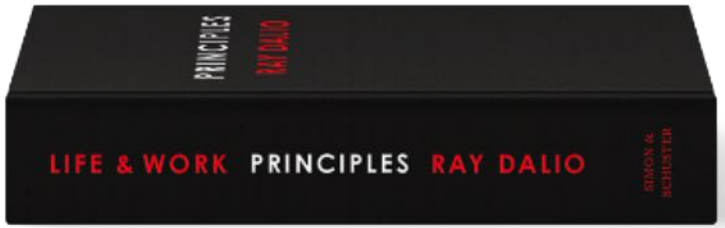
CS3200 Database design (fa18 s2)

https://northeastern-datalab.github.io/cs3200/

Version 9/13/2018

Announcements!

- It is ok to make mistakes in class. Making mistakes in class is the best thing that can happen to you. You learn and will never make it again.
  - "Create a Culture in Which It Is Okay to Make Mistakes and Unacceptable Not to Learn from Them" ... "Recognize that mistakes are a natural part of the evolutionary process." ... "Don't feel bad about your mistakes or those of others. Love them!"
    Ray Dalio. Principles. 2017



- Continue bringing your name tags
- Thanks for posting Q&As on Piazza!
- Make use of office hours (OHs)
- Steve Ballmer: another attempt

OHs (Office Hours)

We are here to help.
Reach out to us!

OFFICE HOURS

If you don't understand something, please ask questions. We love questions. One of the benefits of attending a university as opposed to reading a book is that you get to interact with faculty, TAs, and your peers. Please reach out to us with your questions on our Piazza site or stop by for office hours.

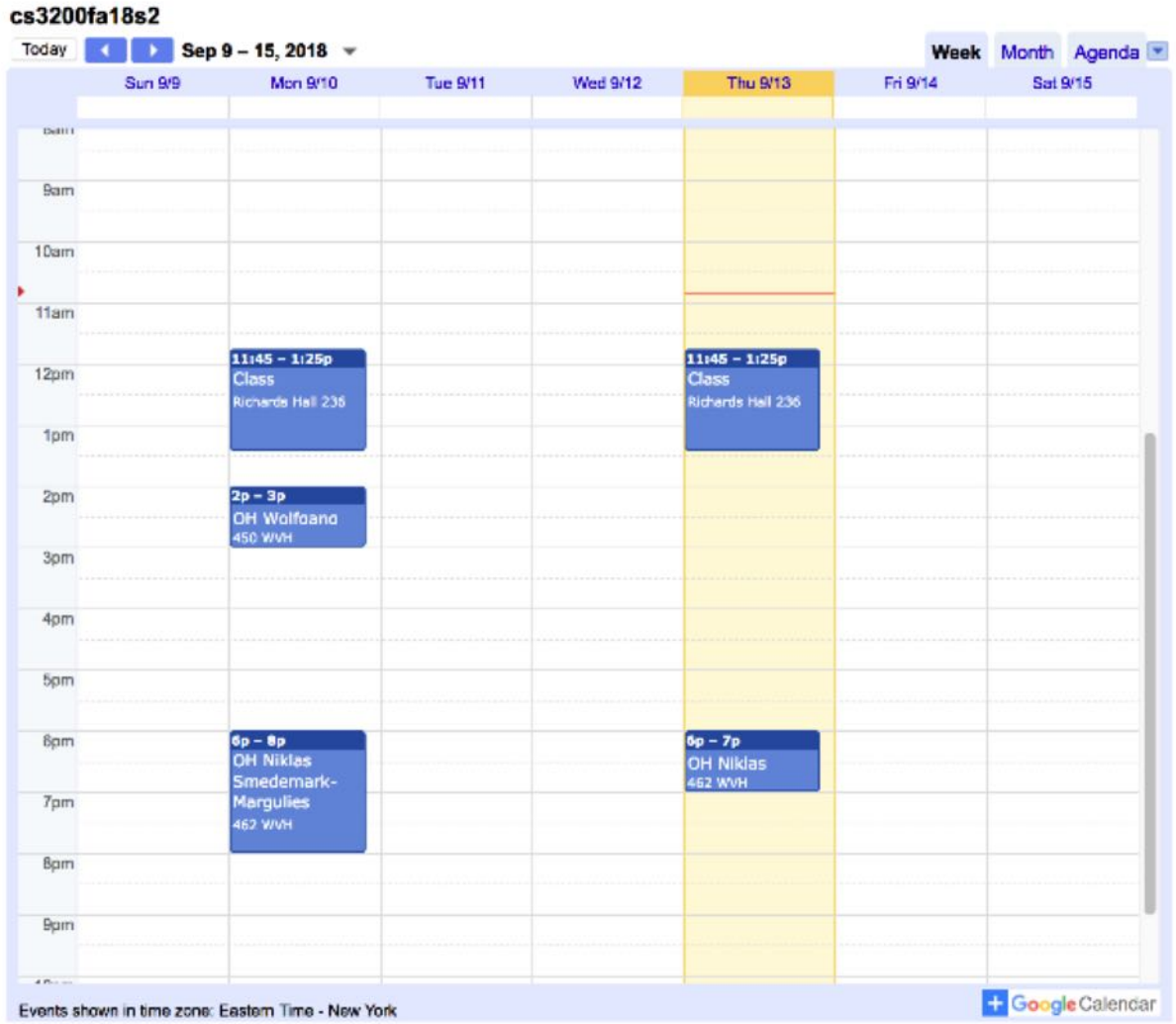


Table Alias (Tuple Variables)

Person (pName, address, works_for)
University (uName, address)

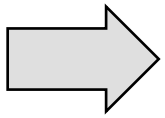
```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```

Table Alias (Tuple Variables)

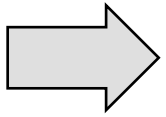
Person (pName, address, works_for)
University (uName, address)

which address?
Error!

```
SELECT  DISTINCT pName, address
FROM    Person, University
WHERE   works_for = uName
```



```
SELECT  DISTINCT Person.pName, University.address
FROM    Person, University
WHERE   Person.works_for = University.uName
```

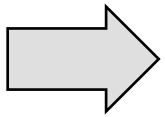


```
SELECT  DISTINCT X.pName, Y.address
FROM    Person as X, University Y
WHERE   X.works_for = Y.uName
```

Note that "as" is optional !!

Column Alias (rename attributes)

Person (pName, address, works_for)
University (uName, address)



```
SELECT  DISTINCT X.pName as name, Y.address adr
FROM    Person as X, University Y
WHERE   X.works_for = Y.uName
```

```
SELECT  DISTINCT X.pName, Y.address
FROM    Person as X, University Y
WHERE   X.works_for = Y.uName
```

Quiz

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

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 US companies that manufacture products in the 'Gadgets' category!

```
SELECT  cName
FROM
WHERE
```

Quiz

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

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 US companies that manufacture products in the 'Gadgets' category!

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

Cname
GizmoWorks
GizmoWorks

Quiz

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

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 US companies that manufacture products in the 'Gadgets' category!

```
SELECT  DISTINCT cName
FROM    Product P, Company
WHERE   country = 'USA'
    and P.category = 'Gadgets'
    and P.manufacturer = cName
```

Cname
GizmoWorks

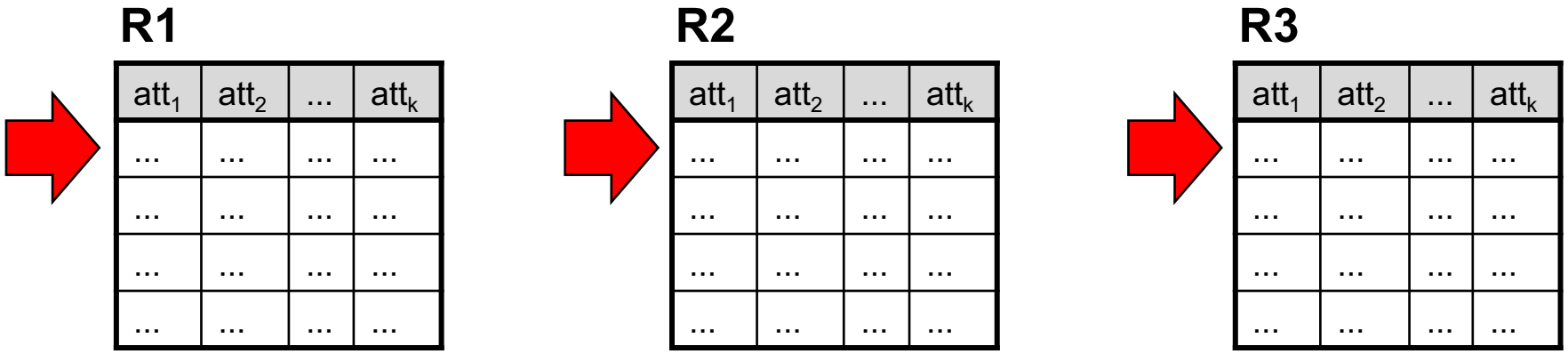
Meaning (Semantics) of conjunctive SQL Queries

SELECT $a_1, a_2, \ldots, a_k$
FROM R_1 as x_1, R_2 as x_2, …, R_n as x_n
WHERE Conditions

Conceptual evaluation strategy (nested for loops):

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

Meaning (Semantics) of conjunctive SQL Queries



Answer = {}
for x_1 **in** R_1 **do**
  for x_2 **in** R_2 **do**
    …..
      for x_n **in** R_n **do**
        if Conditions
          then Answer = Answer $\cup$ $\{(a_1,\ldots,a_k)\}$
return Answer

Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - FROM: Compute the cross-product of relation-list.
  - WHERE: Discard resulting tuples if they fail qualifications.
  - SELECT: Delete attributes that are not in target-list.
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute the same answers.

Quiz

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

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM
WHERE
```

Quiz

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

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

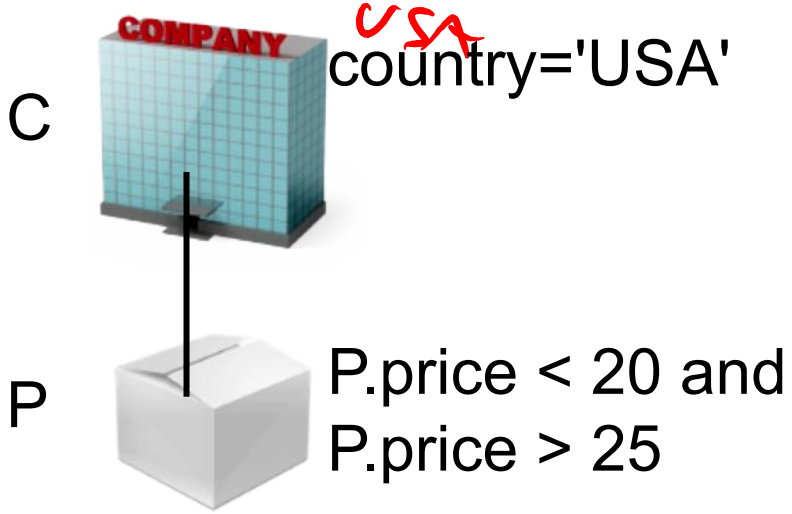
```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and P.price < 20
    and P.price > 25
    and P.manufacturer = cName
```

Wrong! Gives empty result: There is no product with price <20 and >25

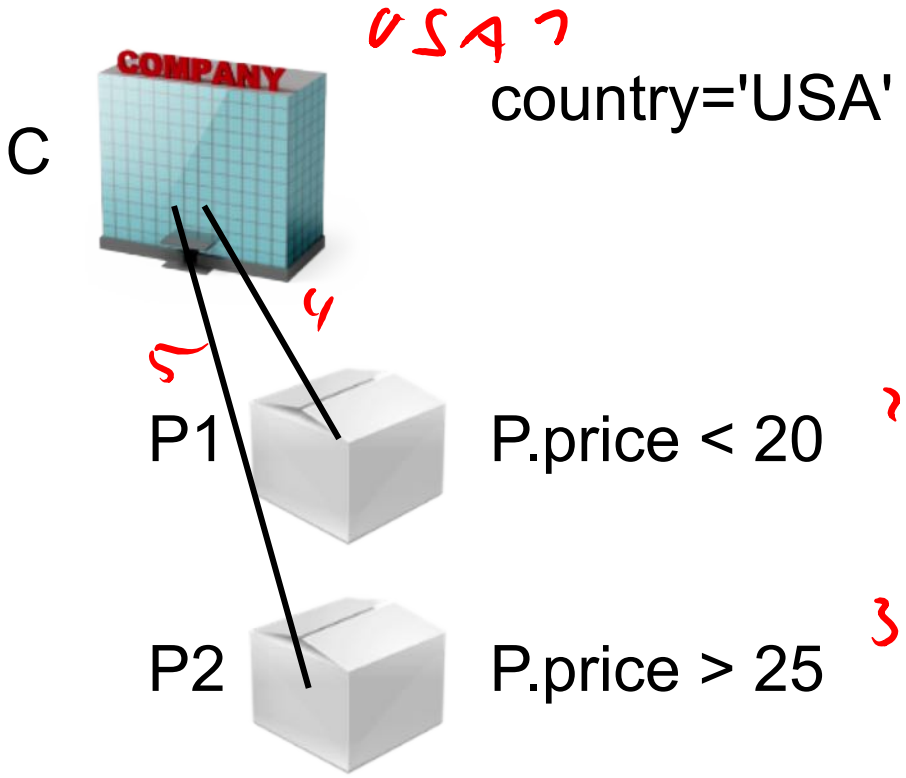
Quiz

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

Q: Find all US companies that manufacture both a product below $20 and a product above $25.



not possible!
-> Empty result



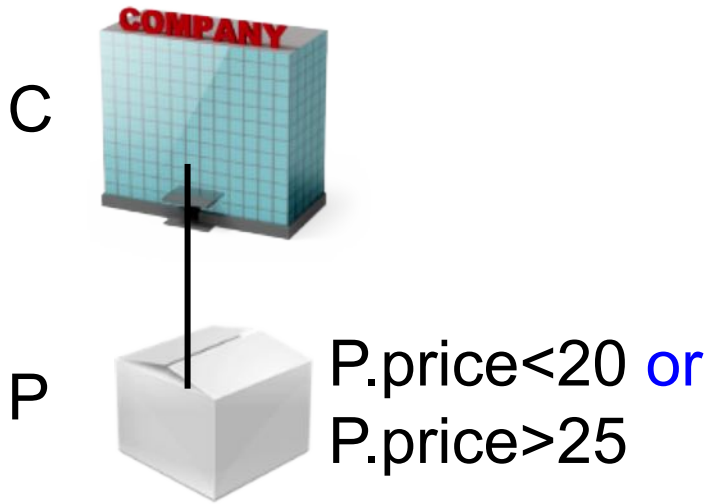
Quiz

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

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P, Company
WHERE   country = 'USA'
    and (P.price < 20
    or   P.price > 25)
    and  P.manufacturer = cName
```

Returns companies with single product w/price (<20 or >25)



Quiz

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

Q: Find all US companies that manufacture both a product below $20 and a product above $25.

```
SELECT  DISTINCT cName
FROM    Product as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Quiz

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 as P1, Product as P2, Company
WHERE   country = 'USA'
    and P1.price < 20
    and P2.price > 25
    and P1.manufacturer = cName
    and P2.manufacturer = cName
```

Cname
GizmoWorks

Inner Joins

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
WHERE   E.DepartmentID = D. DepartmentID
```

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Jones	33	33	Engineering
Smith	34	34	Clerical
Steinberg	33	33	Engineering
Rafferty	31	31	Sales

Source: http://en.wikipedia.org/wiki/Join_(SQL)#Cross_join

Cross Joins: usually not what you want ☹

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT  *
FROM    Employee E, Department D
WHERE   E.DepartmentID = D. DepartmentID
```

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Rafferty	31	31	Sales
Jones	33	31	Sales
Steinberg	33	31	Sales
Smith	34	31	Sales
Robinson	34	31	Sales
Rafferty	31	33	Engineering
Jones	33	33	Engineering
Steinberg	33	33	Engineering
Smith	34	33	Engineering
Robinson	34	33	Engineering
Rafferty	31	34	Clerical
Jones	33	34	Clerical
Steinberg	33	34	Clerical
Smith	34	34	Clerical
Robinson	34	34	Clerical
Rafferty	31	35	Marketing
Jones	33	35	Marketing
Steinberg	33	35	Marketing
Smith	34	35	Marketing
Robinson	34	35	Marketing

Definitions (for job interviews?)

- An equi-join is a join in which the joining condition is based on equality between values in the common columns; common columns appear redundantly in the result table
- A natural join is an equi-join in which one of the duplicate columns is eliminated in the result table
- A cross join returns the Cartesian product of rows from tables in the join
  - (i.e. it will produce rows which combine each row from the first table with each row from the second table, that's usually *not* what you want)

Definitions (for job interviews?)

Equi-join

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Jones	33	33	Engineering
Smith	34	34	Clerical
Steinberg	33	33	Engineering
Rafferty	31	31	Sales

Natural join

E.LastName	DepartmentID	D.DepartmentName
Robinson	34	Clerical
Jones	33	Engineering
Smith	34	Clerical
Steinberg	33	Engineering
Rafferty	31	Sales

Cross join

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Rafferty	31	31	Sales
Jones	33	31	Sales
Steinberg	33	31	Sales
Smith	34	31	Sales
Robinson	34	31	Sales
Rafferty	31	33	Engineering
...	...	...	...

Alternative JOIN Syntax

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT    *
FROM      Employee E, Department D
WHERE     E.DepartmentID = D. DepartmentID
AND       E.DepartmentID = 34
```

```
SELECT    *
FROM      Employee E JOIN Department D
     ON   E.DepartmentID = D. DepartmentID
WHERE     E.DepartmentID = 34
```

E.LastName	E.DepartmentID	D.DepartmentID	D.DepartmentName
Robinson	34	34	Clerical
Smith	34	34	Clerical

NATURAL JOIN Syntax

Employee

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

Department

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

```
SELECT    *
FROM      Employee E, Department D
WHERE     E.DepartmentID = D. DepartmentID
AND       E.DepartmentID = 34
```

```
SELECT    *
FROM      Employee E NATURAL JOIN Department D

WHERE     E.DepartmentID = 34
```

LastName	DepartmentID	DepartmentName
Robinson	34	Clerical
Smith	34	Clerical

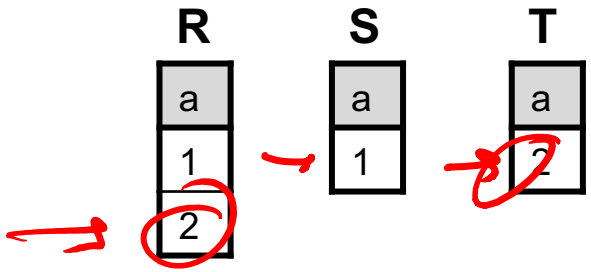
Syntax is not supported by all DBMS's and not recommended

(someone reading the query does not see the join keys. That makes is harder to debug queries or to modify & enhance them)

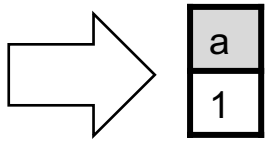
Using the Formal Semantics

R(a), S(a), T(a)

What do these queries compute?

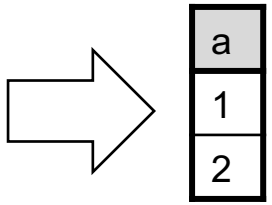



```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T
WHERE   R.a=S.a
    or  R.a=T.a
```

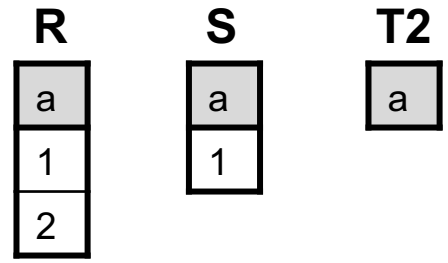


Returns R ∩ (S ∪ T)
if S ≠ ∅ and T ≠ ∅

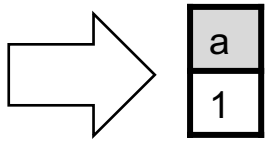
Using the Formal Semantics

R(a), S(a), T2(a)

What do these queries compute?

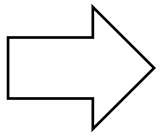



```
SELECT  R.a
FROM    R, S
WHERE   R.a=S.a
```



Returns R ∩ S (intersection)

```
SELECT  R.a
FROM    R, S, T2 as T
WHERE   R.a=S.a
   or   R.a=T.a
```



Returns ∅
if S = ∅ or T = ∅

Can seem counterintuitive! But remember conceptual evaluation strategy: Nested loops. If one table is empty -> no looping

Illustration with Python


```
'''
Created on 3/23/2015
Illustrates nested Loop Join in SQL
__author__ = 'gatt'
'''

print "--- 1st nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(2):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 2nd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(1):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print

print "\n--- 3rd nested loop ---"
for i in xrange(2):
    for j in xrange(3):
        for k in xrange(0):
            print "i=%d, j=%d, k=%d: " % (i, j, k),
            if i == j or i == k:
                print "TRUE",
            print
```

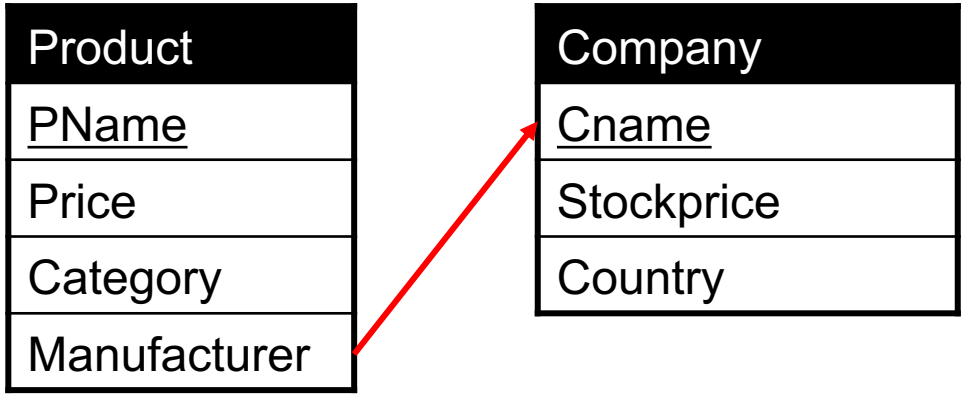
```
/Library/Frameworks/Python.framework/Versio
--- 1st nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=0, k=1:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=1, k=1:
i=0, j=2, k=0:  TRUE
i=0, j=2, k=1:
i=1, j=0, k=0:
i=1, j=0, k=1:  TRUE
i=1, j=1, k=0:  TRUE
i=1, j=1, k=1:  TRUE
i=1, j=2, k=0:
i=1, j=2, k=1:  TRUE

--- 2nd nested loop ---
i=0, j=0, k=0:  TRUE
i=0, j=1, k=0:  TRUE
i=0, j=2, k=0:  TRUE
i=1, j=0, k=0:
i=1, j=1, k=0:  TRUE
i=1, j=2, k=0:

--- 3rd nested loop ---

Process finished with exit code 0
```

The comparison gets never evaluated

"Premature optimization is the root of all evil."
Donald Knuth (1974)

CREATE Tables
&
INSERT Values

(Relational Database) Schema



"Schema": describes the structure of data in terms of the relational data model.

A schema includes tables, columns, PKs, FKs, and other constraints

Product(pname, price, category, manufacturer)
Company(cname, stockprice, country)

Product.manufacturer is FK to Company


```
-----------------------------
-- Create the tables
-----------------------------

create table Company (

                        );

create table Product (

                                                    );
```


```
-----------------------------
-- Create the tables
-----------------------------

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

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),

                                                  );
```


```
-----------------------------
-- Create the tables
-----------------------------

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

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),

                                                    );
```


```
-----------------------------
-- Create the tables
-----------------------------

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

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),
PRIMARY KEY (PName)
                                                  );
```

```
-----------------------------
-- Create the tables
-----------------------------

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

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),
PRIMARY KEY (PName),
FOREIGN KEY (Manufacturer) REFERENCES Company(CName) );
```


```
-----------------------------
-- Create the tables
-----------------------------

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

create table Product (
      PName char(20),
      Price decimal(9, 2),
      Category char(20),
      Manufacturer char(20),
PRIMARY KEY (PName),
FOREIGN KEY (Manufacturer) REFERENCES Company(CName) );
```

```
-----------------------------
-- Drop tables if they already exist
-----------------------------

DROP TABLE IF EXISTS Product;
DROP TABLE IF EXISTS Company;
```

```
-----------------------------
-- Populate the tables
-----------------------------

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

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');
```

1. Aggregates
2. Groupings
3. Having

Aggregation

Car (name, price, maker)


```
SELECT  avg(price)
FROM    Car
WHERE   maker='Toyota'
```

```
SELECT  count(*)
FROM    Car
WHERE   price > 100
```

SQL supports several aggregation operations:

sum, count, min, max, avg

Except count, all aggregations apply to a single attribute

Aggregation


```
SELECT  avg(price)
FROM    Car
WHERE   maker='Toyota'
```

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
Prius	50	Toyota
Lexus1	75	Toyota
Lexus2	100	Toyota

Database creates new attribute name (for SQLserver)

(No column name)
75

Aggregation with rename

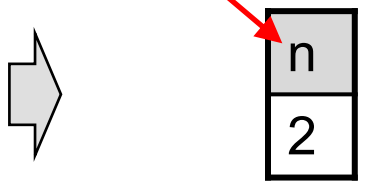
"as" optional

```
SELECT  count(*) as n
FROM    Car
WHERE   price > 100
```

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
~~Prius~~	~~50~~	~~Toyota~~
~~Lexus1~~	~~75~~	~~Toyota~~
~~Lexus2~~	~~100~~	~~Toyota~~

Database creates *our* new attribute name



Aggregation: Count Distinct


```
SELECT  count(maker)
FROM    Car
WHERE   price > 100
```

Same as count(*)

Car

Name	Price	Maker
M3	120	BMW
M5	150	BMW
Prius	50	Toyota
Lexus1	75	Toyota
Lexus2	100	Toyota

We probably want to ignore duplicates:

```
SELECT  count(DISTINCT maker)
FROM    Car
WHERE   price > 100
```

(No column name)
1

Simple Aggregation 1/3

Purchase (product, price, quantity)

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

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

What do these queries mean?

Simple Aggregation 2/3

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
~~Banana~~	~~2~~	~~10~~
~~Banana~~	~~4~~	~~10~~

3 * 20 = 60

2 * 20 = 40

sum: 100

Database creates new attribute name

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

(No column name)
100

Simple Aggregation 3/3

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
~~Banana~~	~~2~~	~~10~~
~~Banana~~	~~4~~	~~10~~

3		20
2		20
sum: 5	*	sum: 40 = 200

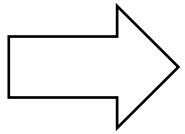
```
SELECT  sum(price) * sum(quantity)
FROM    Purchase
WHERE   product = 'Bagel'
```

(No column name)
200

Grouping and Aggregation

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10



Product	TotalSales
Bagel	40
Banana	20

Notice: we use "sales" for total number of products sold

Find total quantities for all purchases with price over $1 grouped by product.

From → Where → Group By → Select

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
~~Banana~~	~~1~~	~~50~~
Banana	2	10
Banana	4	10

Product	TotalSales
Bagel	40
Banana	20

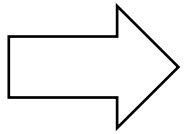
Select contains
- grouped attributes
- and aggregates

```
4  SELECT    product, sum(quantity) as TotalSales
1  FROM      Purchase
2  WHERE     price > 1
3  GROUP BY  product
```

Let's confuse the database engine

Purchase

Product	Price	Quantity
Bagel	3	20
Bagel	2	20
Banana	1	50
Banana	2	10
Banana	4	10



Product	Quantity
Bagel	?
Banana	?

What quantity should the DB return for Banana?

```
SELECT    product, quantity
FROM      Purchase
GROUP BY  product
```

The DB engine is confused, there is no single quantity for banana (it's an ill-defined query). It should thus return an error (only SQLite misbehaves and returns something, but which makes no sense). Please think this through carefully!