CS 2451
Database Systems:
Intro to SQL … Part 2

http://www.seas.gwu.edu/~bhagiweb/cs2541

Spring 2020

Instructor: Dr. Bhagi Narahari & R. Leontie

Based on slides © Ramakrishnan&Gerhke, R. Lawrence

Basic SQL Query

```
SELECT    [DISTINCT]  attribute-list
FROM      relation-list
WHERE     qualification/predicate :
```

- *relation-list* A list of relation names (possibly with a *range-variable, i.e., tuple variable,* after each name).
- *attribute-list* A list of attributes of relations in *relation-list*
- *Qualification/predicate* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!
- To select all attributes in result, we use *

SQL and Relational Algebra

- The SELECT statement can be mapped directly to relational algebra.

SELECT $A_1, A_2, \ldots, A_n$ *this is projection π*

FROM $R_1, R_2, \ldots, R_m$ *this is Cartesian product ×*

WHERE P *this is the selection op σ*

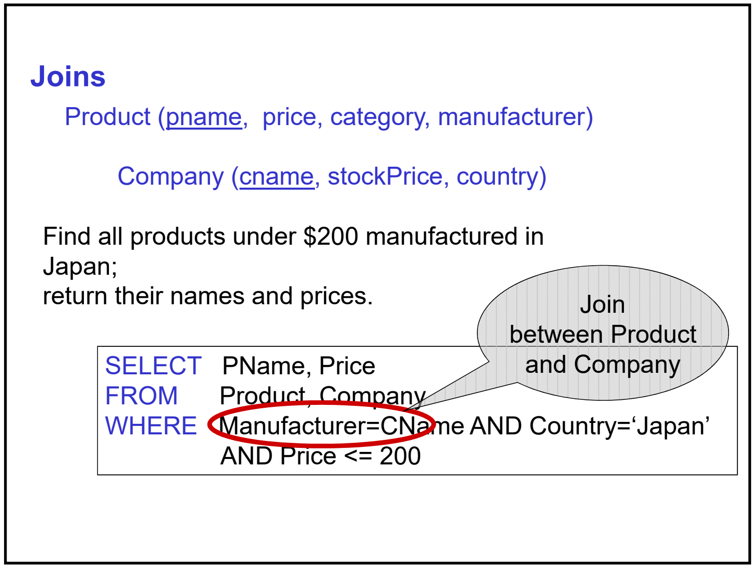
- is equivalent to:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

- If we don't want to project, then SELECT *

Cross products and Joins in SQL

- Multiple tables can be queried in a single SQL statement by listing them in the FROM clause.
  - Note that if you do not specify any join condition to relate them in the WHERE clause, you get a *cross product* of the tables.



Joins

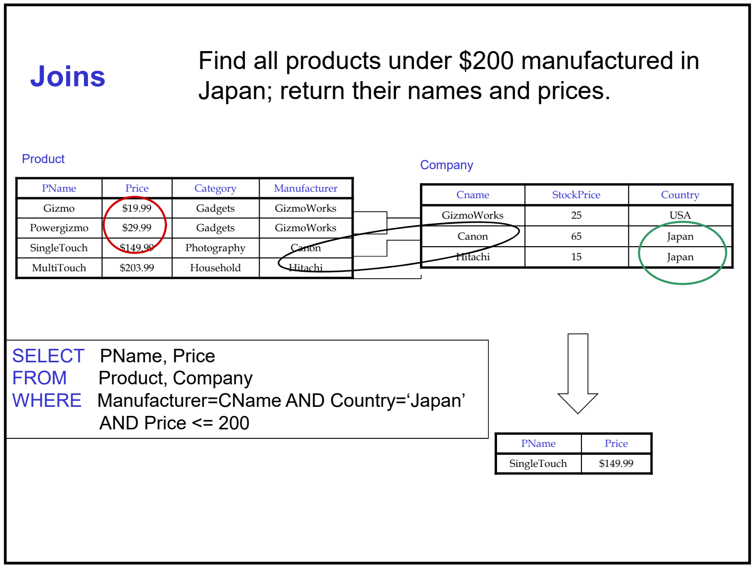
Product (pname, price, category, manufacturer)

Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan;
return their names and prices.

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

Join between Product and Company



Joins

Find all products under $200 manufactured in Japan; return their names and prices.

Product

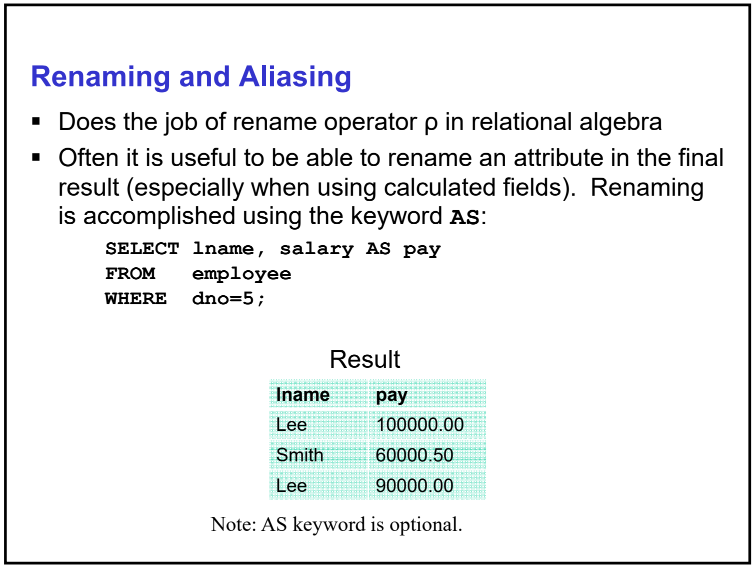
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

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

```
SELECT   PName, Price
FROM     Product, Company
WHERE    Manufacturer=CName AND Country='Japan'
         AND Price <= 200
```

PName	Price
SingleTouch	$149.99



Renaming and Aliasing

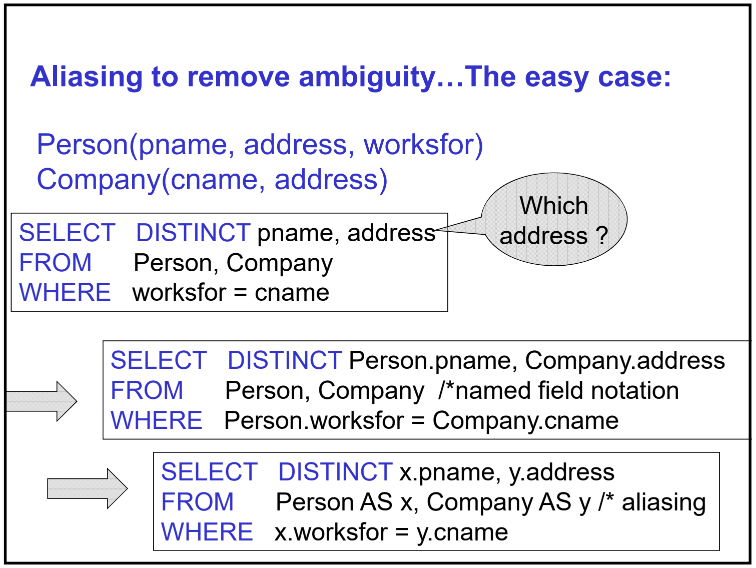
- Does the job of rename operator ρ in relational algebra
- Often it is useful to be able to rename an attribute in the final result (especially when using calculated fields). Renaming is accomplished using the keyword **AS**:

```
SELECT lname, salary AS pay
FROM   employee
WHERE  dno=5;
```

Result

lname	pay
Lee	100000.00
Smith	60000.50
Lee	90000.00

Note: AS keyword is optional.



Aliasing to remove ambiguity…The easy case:

Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

Which address ?

```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company  /*named field notation
WHERE    Person.worksfor = Company.cname
```

```
SELECT   DISTINCT x.pname, y.address
FROM     Person AS x, Company AS y /* aliasing
WHERE    x.worksfor = y.cname
```

Renaming…Using Tuple/Range variables

- Concept of tuple/range variables borrowed from relational calculus
  - Tuple *t* of type *R*: $t \in R$
  - *What about* $x \in R$, $y \in R$
- It performs the job of the rename operator from relational algebra
  - **One variable with name *x* and one with name *y, BOTH of type R***
- Need to worry about scope of tuple variables when we have nested queries

Tuple Variables

Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT P.pname,C.address
FROM     Person P, Company C
 WHERE   P.worksfor = C.cname;
```

x is a copy of Person, y is a copy of Company
P is a variable of ‘type’ Person C is a variable of ‘type’ Company

Renaming: Joining table with itself

- Aliases/Tuple variables must be used when relation has to be ‘joined’ with itself – i.e., two or more copies of the same table are needed. Using ***aliases*** allows you to uniquely identify what table you are talking about.

Example: Return last names of employees and their managers.

```
SELECT E.lname, M.lname
FROM  employee E, employee M
WHERE E.superssn = M.ssn;
```

- E is a variable of type Employee, and denotes an employee
- M is a variable of type Employee, and denotes (will bind to) values of supervisor

Meaning (Semantics) of SQL Queries with tuple variables

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

```
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 ∪ {($a_1$,…,$a_k$)}
return Answer
```

Tuple variables

- Find students who are taking the same course as student with sid=1234.
- Need to access Takes table twice
  - Once to extract courses (with course id CID=X) taken by student with ID=1234
  - Second time to find students who are taking these X courses
- Define two “variables” A,B of ‘type’ Takes
  - B is variable that corresponds ID 1234 and its cid field is equal to “*X*”
  - A is a variable whose CID is equal to “*X*”
- SELECT A.sid
- FROM Takes A, Takes B
- WHERE A.cid = B.cid AND B.sid= 1234;

Outer Joins and Inner Joins..

- INNER JOIN
  - ‘standard’ join
- OUTER JOIN
  - Include tuples that don’t match

    To Keep track of tuples that don’t match

Inner Join Operation

Explicit joins in SQL = “inner joins”:

Product(name, category)

Purchase(prodName, store)

```
SELECT Product.name, Purchase.store
FROM    Product JOIN Purchase ON
                    Product.name = Purchase.prodName;
```

Same as:

```
SELECT Product.name, Purchase.store
FROM    Product, Purchase
WHERE   Product.name = Purchase.prodName;
```

```
SELECT Student.name
FROM    Student JOIN Takes
USING (sid);
```

MySQL: Also allows specifying common attribute for join by specifying a “using” keyword

Why provide Inner Join ?

- The semantics of the basic SQL query has cross product of the tables
  - Could be a very large intermediate result and impacts performance
  - Code optimizer (query processor) has to determine the join condition from the where clause
- Specifying join condition explicitly makes it easier for query optimizer to interpret
  - Creates the join instead of cross product
  - Smaller intermediate result, so better performance

Outer Joins

- Sometimes we may want to keep tuples that do not join with the other table
- 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 the both left and right tuples even if there's no match

Example of why outer joins…

Find sales of all products, including those that with no sales

Explicit joins in SQL = "inner joins":
Product(name, category)
Purchase(prodName, store)

```
SELECT Product.name, Purchase.store
FROM     Product JOIN Purchase ON
                     Product.name = Purchase.prodName
```

But Products that never sold will be lost !

Outerjoins

Find sales of all products, including those that with no sales

Left outer joins in SQL:
Product(name, category)
Purchase(prodName, store)

```
SELECT Product.name, Purchase.store
FROM     Product LEFT OUTER JOIN Purchase ON
                     Product.name = Purchase.prodName
```

Product

Name	Category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

ProdName	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Name	Store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

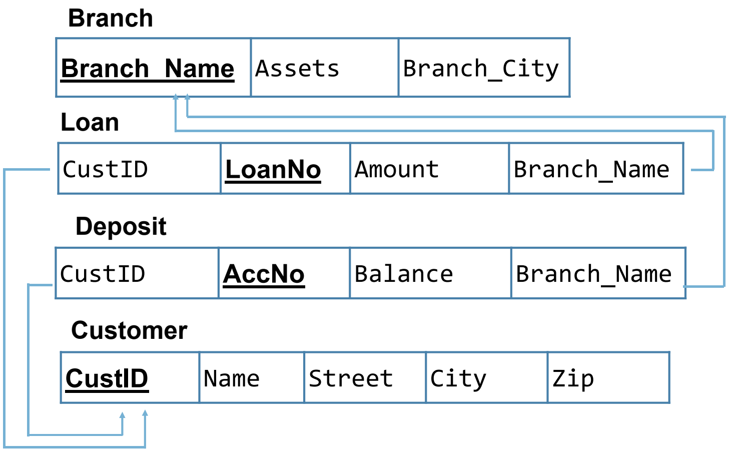
The result reveals that OneClick had no sales

Next: InClass exercises Test your querying skills!

- Step 1: 5 minutes
  - Do NOT code…
  - Work at your table to discuss solutions/queries – do not write down code

- Step 2: 15 minutes - Work individually and Code your queries
  - And submit query/output screenshot on github

Bank Database Schema



Branch

<u>Branch_Name</u>	Assets	Branch_City

Loan

CustID	**<u>LoanNo</u>**	Amount	Branch_Name

Deposit

CustID	**<u>AccNo</u>**	Balance	Branch_Name

Customer

<u>CustID</u>	Name	Street	City	Zip

Connecting to mySQL on gwupyterhub

- Use your GW netID to connect to the gwupyterhub.seas.gwu.edu server

```
ssh -Y GWnetID@gwupyterhub.seas.gwu.edu
```

- Login into MySQL

```
mysql –u GWnetID -p
```

NOTE: use your GW NetID, WITH the password CSCI2541_sp20

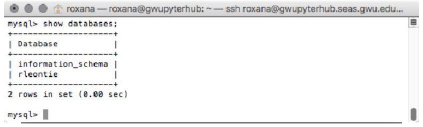
- Reset your password

```
SET PASSWORD FOR 'GWNetID'@'localhost'='NEWPASSWORD';
```

MySQL Database

- An existing database is available for your use

```
show databases;
```

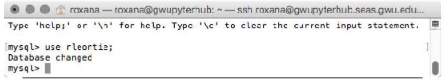

roxana — roxana@gwupyterhub: ~ — ssh roxana@gwupyterhub.seas.gwu.edu...

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| rleontie           |
+--------------------+
2 rows in set (0.00 sec)

mysql> 
```

- To use your database:

```
use database_name;
```


roxana — roxana@gwupyterhub: ~ — ssh roxana@gwupyterhub.seas.gwu.edu...

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use rleontie;
Database changed
mysql> 
```

NOTE: use your GW NetID for database name

More SQL stuff …

- IN operator
- NULLs
- Nested Queries
- Set operations
  - Membership
  - Union
  - Comparison

IN Operator

- To specify that an attribute value should be in a given set of values, the **IN** keyword is used.
  - Example: Return all employees who are in any one of the departments {'D1', 'D2', 'D3'}.

```
SELECT ename
FROM   emp
WHERE  dno IN ('D1','D2','D3')
```

- Note that this is equivalent to using OR:

```
SELECT ename
FROM   emp
WHERE  dno = 'D1' OR dno = 'D2' OR dno = 'D3'
```

- more practical uses of `IN` and `NOT IN` when we study nested subqueries.

Set Operations

- The set operations of union, intersection, and difference are used to combine the results of two SQL queries.
  - `UNION` , `INTERSECT`, `EXCEPT`
  - Note: `UNION ALL` returns all rows
- Example: Return the sid of students who are either taking course with cid=123 or course with cid=345.

```
(SELECT sid
FROM   students
WHERE  cid=‘123’)
UNION
(SELECT sid
FROM   students
WHERE  cid = ‘345’);
```

Set Operations

- MINUS (EXCEPT) – set difference
- INTERSECT
- CONTAINS (or NOT CONTAINS) - subset
- MySQL does NOT support any of these ☹
  - Have to implement using other operators

NULLS in SQL

- Whenever we don’t have a value, we can put a NULL
- Can mean many things:
  - Value does not exist
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if it can be null (*nullable* attribute) or not
  - NOT NULL after declaring attribute domain
- How does SQL cope with tables that have NULLs ?

Null Values

- If x= NULL then 4*(3-x)/7 is still NULL

- If x= NULL then x=“Joe”   is UNKNOWN
- In SQL there are three boolean values:

FALSE	=	0
UNKNOWN	=	0.5
TRUE	=	1

Null Values

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1

```
SELECT *
FROM Person
WHERE  (age < 25) AND
       (height > 6 OR weight > 190)
```

E.g.
age=20
height=NULL
weight=200

Rule in SQL: include only tuples that yield TRUE

Null Values

Unexpected behavior:

```
SELECT *
FROM    Person
WHERE  age < 25  OR  age >= 25
```

Some Persons are not included !

Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM    Person
WHERE  age < 25  OR  age >= 25 OR age IS NULL
```

Now it includes all Persons

Subqueries

- SQL allows a single query to have multiple subqueries nested inside of it. This allows for more complex queries to be written.
- When queries are nested, the outer statement determines the contents of the final result, while the inner `SELECT` statements are used by the outer statement (often to lookup values for `WHERE` clauses).
- Need to be careful about scope of tuple variables

  Scoping rules: local definition and then global

  In subquery – legal to use only tuple variables defined in subquery itself or in any query that contains the subquery

Nested Queries: Semantics and set operators

- Evaluate subquery at each reference
  - Construct cross product of tables in FROM clause
  - For each row when testing predicate conditions in WHERE clause

    Recompute subquery

    – Is this really necessary?
- Set membership operators provided to test results of subquery
  - IN, EXISTS, CONTAINS (subset), op ALL, op SOME ….(op is >, <, =)

Subqueries Returning Relations and Set Membership operators

Company(<u>name</u>, city)
Product(<u>pname</u>, maker)
Purchase(<u>id</u>, product, buyer)

Return cities of companies that manufacture
products bought by Joe Plumber

```
SELECT  Company.city
FROM    Company
WHERE  Company.name  IN
                (Set of Companies that manufacture
                          products bought by Joe Blow);
/* write a SELECT query to obtain this set */
```

Subqueries Returning Relations

Company(name, city)
Product(pname, maker)
Purchase(id, product, buyer)

Return cities of companies that manufacture
products bought by Joe Plumber

```
SELECT  Company.city
FROM    Company
WHERE  Company.name  IN
              (SELECT Product.maker
                FROM   Purchase , Product
                WHERE Product.pname=Purchase.product
                     AND Purchase .buyer =  ‘Joe Blow ‘);
```

Set Membership Operations: (a)

- Can check for set membership using IN and NOTIN
  - x IN A or x NOTIN A

    Implements Relational Calculus operators
  - IN connective tests for membership in the set A

    Set A may be produced by a SELECT
  - NOTIN tests for absence of tuples
  - Can test using multiple attribute element
- Set existence using EXISTS
  - Returns true if the argument subquery is nonempty (the converse for the NOT EXISTS) thus checking for empty relations

Set Membership: Quantifiers

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

Find all companies that make some products with price < 100

```
SELECT DISTINCT  Company.cname
FROM     Company, Product
WHERE  Company.cname = Product.company and Product.price < 100
```

Existential: easy ! ☺

Set Membership: Quantifiers

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

Find all companies that make only products with price < 100

same as:

Find all companies such that all of their products have price < 100

Recall equivalence: Forall x P(x) = Not Exists x (Not P(x))

Universal: hard ! ☹

Set Membership: Quantifiers

1. Find *the other* companies: i.e. s.t. some product >= 100

```
SELECT DISTINCT  Company.cname
FROM     Company
WHERE  Company.cname IN (SELECT Product.company
                          FROM Product
                          WHERE Produc.price >= 100
```

Set Membership: Quantifiers

1. Find *the other* companies: i.e. s.t. some product ≥ 100

```
SELECT DISTINCT  Company.cname
FROM     Company
WHERE  Company.cname IN (SELECT Product.company
                          FROM Product
                          WHERE Produc.price >= 100
```

2. Find all companies s.t. all their products have price < 100

```
SELECT DISTINCT  Company.cname
FROM     Company
WHERE  Company.cname NOT IN (SELECT Product.company
                              FROM Product
                              WHERE Produc.price >= 100
```

Solving the query using EXISTS operator

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

Find companies that only make products with price <100

For a company C, the set of tuples with price >=100 is the empty set – i.e., NOT EXISTS

```
SELECT DISTINCT  C.cname
FROM     Company C
WHERE  NOT EXISTS (SELECT *
                    FROM Product P
                    WHERE P.price >= 100
                    AND P.company=C.cname ) ;
```

More Set Membership Operations

- ❖ Previous operators allowed checking for existence
- ❖ SQL provides operators to test elements of one set A with elements on another set B
  - SOME: *op* SOME

    Also called as ANY in some versions
  - ALL: *op* ALL
  - *op* can be >=, >, <, <=, =, not=
- ❖ Test single value against members of an entire set
  - *X > ALL (R)*

Comparing value with values in a set

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

Find products (names) which do not have the lowest price

```
SELECT product name where price is not the minimum of all prices
All prices given by subquery:
                              (SELECT PRICE
                                   FROM Product P ) ;
```

Comparing value with values in a set

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

Find products (names) which do not have the lowest price
   = Price is greater than price of some other product!

```
SELECT pname
FROM Product
WHERE price  > ANY
                              (SELECT PRICE
                                   FROM Product P ) ;
```

Other Set Operations….

- ❖ INTERSECTION
- ❖ MINUS (set difference)
- ❖ SUBSET Check if one set (query result) contains another set (query result)
  - Is A subset of B?
  - Is A not a subset of B ?
  - ❖**Contains** and **not contains** operators

❖ **Too bad MySQL does not support these ☹**

Next: more InClass exercises Test your querying skills!

- Step 1: 7 minutes
  - Do NOT code…
  - Work at your table to discuss solutions/queries – do not write down code
- Step 2: Work individually and Code your queries
  - And submit query/output screenshot on github