SQL: Nested Queries, Group By, Having

Prof. Hyuk-Yoon Kwon
https://sites.google.com/view/seoultech-bigdata

Most parts are based on slides used in Stanford (http://web.stanford.edu/class/cs145)

Today's Lecture

1. **Multiset operators & Nested queries**

2. **Aggregation & GROUP BY**

3. **Advanced SQL-izing**

INTERSECT: Still some subtle problems…

Company(name, hq_city)
Product(pname, maker, factory_loc)

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
       AND factory_loc = 'US'
INTERSECT
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
              AND factory_loc = 'China'
```

*"Headquarters of companies which make gizmos in US **AND** China"*

INTERSECT: Remember the semantics!

Company(name, hq_city) AS C
Product(pname, maker, factory_loc) AS P

```
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
  AND factory_loc='US'
INTERSECT
SELECT hq_city
FROM   Company, Product
WHERE  maker = name
AND factory_loc='China'
```

Example: C JOIN P on maker = name

C.name	C.hq_city	P.pname	P.maker	P.factory_loc
X Co.	Seattle	X	X Co.	U.S.
Y Inc.	Seattle	X	Y Inc.	China

X Co has a factory in the US (but not China)
Y Inc. has a factory in China (but not US)

But Seattle is returned by the query!

One Solution: Nested Queries

Company(name, hq_city)
Product(pname, maker, factory_loc)

```
SELECT DISTINCT hq_city
FROM   Company, Product
WHERE  maker = name
     AND name IN (
                    SELECT maker
                    FROM   Product
                    WHERE  factory_loc = 'US')
           AND name IN (
                    SELECT maker
                    FROM   Product
                    WHERE  factory_loc = 'China')
```

*"Headquarters of companies which make gizmos in US **AND** China"*

a **IN** {a, b, c} equals to
(a = a) or (a = b) or (a = c)

High-level note on nested queries

- **We can do nested queries because SQL is *compositional:***
  - Everything (inputs / outputs) is represented as multisets- the output of one query can thus be used as the input to another (nesting)!
- **This is <u>extremely</u> powerful!**

Nested queries: Sub-queries Return Relations

Another example:

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

```
SELECT c.city
FROM   Company c
WHERE  c.name  IN (
          SELECT pr.maker
         FROM   Purchase p, Product pr
         WHERE  p.product = pr.name
                AND p.buyer = 'Joe Blow')
```

"Cities where one can find companies that manufacture products bought by Joe Blow"

Nested Queries

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

```
SELECT c.city
FROM   Company c,
       Product pr,
       Purchase p
WHERE  c.name = pr.maker
  AND  pr.name = p.product
  AND  p.buyer = ‘Joe Blow’
```

Oracle Practice #8 – Nested Queries

1. Display the package number, internet speed and sector number for all packages whose sector number equals to the sector number of package number 10 (*Packages* table).

2. Display the first name, last name and join date for all customers who joined the company on the same month and on the same year as customer number 372 (*Customers* table).
   - How to extract Month from Date type in Oracle: **extract(month from join_date)**
   - How to extract Year from Date type in Oracle: **extract(year from join_date)**

3. Display the first name, city, state, birthdate and monthly discount for all customers who was born on the same date as customer number 179, and whose monthly discount is greater than the monthly discount of customer number 107 (*Customers* table)

Nested Queries Return Relations

You can also use operations of the form:

- s > ALL R
- s < ANY R
- EXISTS R

Ex: Product(name, price, category, maker)

```
SELECT name
FROM   Product
WHERE  price > ALL(
          SELECT price
     FROM   Product
     WHERE  maker = 'Gizmo-Works')
```

Find products that are more expensive than all those produced by "Gizmo-Works"

Nested Queries Returning Relations

You can also use operations of the form:

- s > ALL R
- s < ANY R
- EXISTS R

Ex: Product(name, price, category, maker)

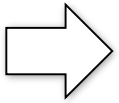
```
SELECT p1.name
FROM   Product p1
WHERE  p1.maker = 'Gizmo-Works'
  AND EXISTS(
          SELECT p2.name
     FROM   Product p2
     WHERE  p2.maker <> 'Gizmo-Works'
              AND p1.name = p2.name)
```

<> means !=

Find 'copycat' products, i.e. products made by competitors with the same names as products made by "Gizmo-Works"

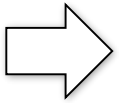
Nested queries as alternatives to INTERSECT and MINUS

```
(SELECT R.A, R.B
 FROM   R)
INTERSECT
(SELECT S.A, S.B
 FROM   S)
```



```
SELECT R.A, R.B
FROM   R
WHERE EXISTS(
            SELECT *
            FROM S
          WHERE R.A=S.A AND R.B=S.B)
```

```
(SELECT R.A, R.B
 FROM   R)
MINUS
(SELECT S.A, S.B
 FROM   S)
```



```
SELECT R.A, R.B
FROM   R
WHERE NOT EXISTS(
             SELECT *
          FROM S
          WHERE R.A=S.A AND R.B=S.B)
```

Correlated Queries Using External Vars in Nested Queries

Movie(title, year, director, length)

```
SELECT DISTINCT title
FROM   Movie AS m
WHERE  year <> ANY(
                  SELECT  year
          FROM    Movie
          WHERE  title =  m.title)
```

Find movies whose title appears more than once.

Complex Correlated Query

Product(name, price, category, maker, year)

```
SELECT DISTINCT  x.name, x.maker
FROM   Product AS x
WHERE  x.price > ALL(
        SELECT y.price
        FROM   Product AS y
        WHERE  x.maker = y.maker
                    AND y.year < 1972)
```

Find products (and their manufacturers) that are more expensive than all products made by the same manufacturer before 1972

Oracle Practice #9 – Nested Queries

1. Display the first name, last name, city, state and package number for all customers whose internet speed is “5Mbps” (*Customers* and *Packages* table).
2. Display the first name, monthly discount, package number, main phone number and secondary phone number for all customers whose sector name is Business (*Customers*, *Packages* and *Sectors* tables).
3. Display the package number, internet speed, and monthly payment for all packages whose monthly payment is greater than the maximum monthly payment of packages with internet speed equals to “5Mbps” (*Packages* table).
   - You can use ALL operator
4. Display the package number, internet speed and monthly payment for all packages whose monthly payment is greater than the minimum monthly payment of packages with internet speed equals to “5Mbps” (*Packages* table).
   - You can use ANY operator

Basic SQL Summary

- SQL provides a high-level declarative language for manipulating data (DML)
- The main structure is the SFW block
- Powerful, nested queries also allowed.

2. Aggregation & GROUP BY

What you will learn about in this section

1. **Aggregation operators**

2. **GROUP BY**

3. **GROUP BY: with HAVING, semantics**

Aggregation

```
SELECT AVG(price)
FROM   Product
WHERE  maker = "Toyota"
```

```
SELECT COUNT(*)
FROM   Product
WHERE  year > 1995
```

- SQL supports several **aggregation** operations:
  - SUM, COUNT, MIN, MAX, AVG

Except COUNT, all aggregations apply to a single attribute

Aggregation: COUNT

- COUNT applies to duplicates, unless otherwise stated

```
SELECT COUNT(category)
FROM   Product
WHERE  year > 1995
```

Note: Same as COUNT(*). Why?

We probably want:

```
SELECT COUNT(DISTINCT category)
FROM   Product
WHERE  year > 1995
```

More Examples

```
Purchase(product, date, price, quantity)
```

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

```
SELECT SUM(price * quantity)
FROM   Purchase
WHERE  product = 'bagel'
```

What do these mean?

Oracle Practice # 10 - Aggregation

1. Display the lowest last name and the highest last name alphabetically (*Customers* table).
2. Display the total number of states (allowing redundancy) and the number of distinct states (*Customers* table).
3. Display the lowest, highest, and average monthly discount (*Customers* table).
4. Display the names of customers whose monthly discounts are larger than the average monthly discount (*Customer* table).

Grouping and Aggregation

Purchase(product, date, price, quantity)

```
SELECT   product,
         SUM(price * quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Find total sales after 10/1/2005 per product.

Let's see what this means…

Grouping and Aggregation

Semantics of the query:

1. Compute the FROM and WHERE clauses

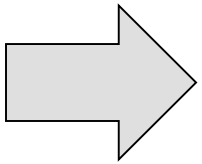
2. Group by the attributes in the GROUP BY

3. Compute the SELECT clause: grouped attributes and aggregates

1. Compute the FROM and WHERE clauses

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

FROM



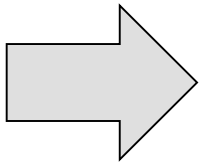
Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

2. Group by the attributes in the GROUP BY

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

GROUP BY



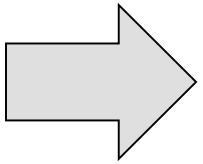
Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

3. Compute the SELECT clause: grouped attributes and aggregates

```
SELECT   product, SUM(price*quantity) AS TotalSales
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
```

Product	Date	Price	Quantity
Bagel	10/21	1	20
	10/25	1.50	20
Banana	10/3	0.5	10
	10/10	1	10

SELECT



Product	TotalSales
Bagel	50
Banana	15

Oracle Practice #11 – GROUP BY

1. Display the state and the number of customers for each state in the descending order by the number of customers (*Customers* table).
2. For each internet package (*Customers* table)
   A. Display the package number and the number of customers for each package number.
   B. Modify the query to display the package number and number of customers for each package number, only for the customers whose monthly discount is greater than 20.

HAVING Clause

```
SELECT   product, SUM(price*quantity)
FROM     Purchase
WHERE    date > '10/1/2005'
GROUP BY product
HAVING   SUM(quantity) > 100
```

Same query as before, except that we consider only products that have more than 100 buyers

HAVING clauses contains conditions on **aggregates**

*Whereas WHERE clauses condition on **individual tuples…***

General form of Grouping and Aggregation

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

- S = Can ONLY contain attributes $a_1,\ldots,a_k$ and/or aggregates over other attributes
- C_1 = is any condition on the attributes in $R_1,\ldots,R_n$
- C_2 = is any condition on the aggregate expressions

General form of Grouping and Aggregation

```
SELECT    S
FROM      R1,…,Rn
WHERE     C1
GROUP BY  a1,…,ak
HAVING    C2
```

Evaluation steps:

1. Evaluate FROM-WHERE: apply condition C_1 on the attributes in $R_1,\ldots,R_n$
2. GROUP BY the attributes $a_1,\ldots,a_k$
3. Apply condition C_2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Group-by v.s. Nested Query

Author(login, name)

Wrote(login, url)

- **Find authors who wrote at least 10 documents:**
- **Attempt 1: with nested queries**

```
SELECT DISTINCT Author.name
FROM   Author
WHERE
    (SELECT COUNT(Wrote.url)
    FROM   Wrote
    WHERE  Author.login = Wrote.login) > 10
```

Group-by v.s. Nested Query

- **Find all authors who wrote at least 10 documents:**
- **Attempt 2: SQL style (with GROUP BY)**

```
SELECT   Author.name
FROM     Author, Wrote
WHERE    Author.login = Wrote.login
GROUP BY Author.name
HAVING   COUNT(Wrote.url) > 10
```

Group-by vs. Nested Query

Which way is more efficient?

- **Attempt #1- *With nested:* How many times do we do a SFW query over all of the Wrote relations?**

- **Attempt #2- *With group-by*: How about when written this way?**

With GROUP BY can be **much** more efficient!

Oracle Practice #12 - HAVING

1. For each internet package, Display the package number and number of customers for each package number, only for the packages with more than 100 customers (*Customers* table)
2. States and the lowest monthly discount (*Customers* table)
   A. Display the state and the lowest monthly discount for each state.
   B. Display the state and lowest monthly discount for each state, only for states where the lowest monthly discount is greater than 10
3. Display the internet speed and number of package for each internet speed where monthly payment is larger than $50 (i.e., > 50), only for the internet speeds with more than 3 packages.

3. Advanced SQL-izing

What you will learn about in this section

1. **Quantifiers**

2. **NULLs**

3. **Outer Joins**

Quantifiers

Product(name, price, company)
Company(name, city)

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

Find all companies that make some products with price < 100

An **existential quantifier** is a logical quantifier (roughly) of the form “there exists”

Existential: easy ! ☺

Quantifiers

Product(name, price, company)
Company(name, city)

Find all companies with products all having price < 100

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

Equivalent

Exclude that all companies that make some products with price >= 100

A **universal quantifier** is of the form “for all”

Universal: hard ! ☹

NULLS in SQL

- **Whenever we don’t have a value, we can put a NULL**

- **Can mean many things:**
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.

- **The schema specifies for each attribute if can be null (*nullable* attribute) or not**

- **How does SQL cope with tables that have NULLs?**

Null Values

- ***For numerical operations,* NULL -> NULL:**
  - If x = NULL then 4*(3-x)/7 is still NULL

- ***For boolean operations,* in SQL there are three values:**

FALSE = 0

UNKNOWN = 0.5

TRUE = 1

Null Values

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

A tuple (age=20 height=NULL weight=200) ?

Rule in SQL: include only tuples that yield TRUE (1)

Null Values

Unexpected behavior:

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

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!

RECAP: Inner Joins

By default, joins in SQL are "inner joins":

```
Product(name, category)
Purchase(prodName, store)
```

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

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

Both equivalent:
Both INNER JOINS!

Inner Joins + NULLS = Lost data?

By default, joins in SQL are "inner joins":

```
Product(name, category)
Purchase(prodName, store)
```

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

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

However: Products that never sold (with no Purchase tuple) will be lost!

Outer Joins

- **An outer join returns tuples from the joined relations that don't have a corresponding tuple in the other relations**
  - i.e., If we join relations A and B on a.X = b.X, and there is an entry in A with X=5, but none in B with X= 5…
    - A LEFT OUTER JOIN will return a tuple (a, NULL)!

- **Left outer joins in SQL:**

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

Now we'll get products even if they didn't sell

INNER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

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

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

Note: another equivalent way to write an INNER JOIN!

LEFT OUTER JOIN:

Product

name	category
Gizmo	gadget
Camera	Photo
OneClick	Photo

Purchase

prodName	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz

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

name	store
Gizmo	Wiz
Camera	Ritz
Camera	Wiz
OneClick	NULL

Other Outer Joins

- **Left outer join:**
  - Include the left tuple even if there's no match
- **Right outer join:**
  - Include the right tuple even if there's no match
- **Full outer join:**
  - Include the both left and right tuples even if there's no match

Oracle Practice #13 – Outer Join

- Customers and internet packages (*Customers* and *Packages* tables)
  A. Display the first name, last name, internet speed and monthly payment for all customers. Use INNER JOIN to solve this exercise.
  B. Modify last query to display all customers, including those without any internet package.
  C. Modify last query to display all packages, including those without any customers.
  D. Modify last query to display all packages and all customers.