Data Science & Analytics

Hiren Deliwala & Dr. Jian ZHANG

Agenda

- Continue from last class
  - Review Table Structures
  - Review SQL Syntax
  - Review MySQL commands
- Run Queries from World Database & Employees Database
- Review Advanced Concepts
  - Joins,
  - Group Bys
  - Nested Queries
  - Nulls

Table Structures

- World Database
  - Country – Country identified by Code
  - City – Joins with Country using City.CountryCode = Country.Code
  - CountryLanguage – Joins with Country using CountryLanguage.CountryCode = Country.Code
- Employees Database
  - employees – Employees identified by emp_no
  - departments – department identified by dept_no
  - dept_manager
    - Joins with department using dept_manager.dept_no = departments.dept_no
    - Joins with employees using dept_manager.emp_no = employees.emp_no
  - dept_emp
    - Joins with department using dept_emp.dept_no = departments.dept_no
    - Joins with employees using dept_emp.emp_no = employees.emp_no
  - titles
    - Joins with employees using titles.emp_no = employees.emp_no

SQL Syntax

- SELECT <attributes>
- FROM <one or more relations>
- WHERE <conditions>

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```

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

MySQL Basics

show databases;	show which databases are available
select database();	show the name of the current database
select version();	show mysql version
show tables;	show the tables in your current database
help	shows help options and available commands
help select	(for example) shows help for the select command
\P	sets the Pager so that results are displayed one page at a time; if this is on, you need to type 'q' to exit the result display
quit	to exit mysql
describe [table name];	to describe the table with that name
use [database name];	to change to the database with that name
source [filename.sql];	execute the commands in the file with that name

SQL JOIN OPERATION

SQL: Join operation

- A join can be specified in the FROM clause which list the two input relations and the WHERE clause which lists the join condition. Example:

Emp

ID	State	Dept
1000	CA	
1001	MA	1001
1002	TN	1002

Dept

ID	Name
1001	IT
1002	Sales
1003	Biotech

SQL: Join operation (cont.)

- inner join = join

  SELECT *

  FROM emp join dept (or FROM emp, dept)

  on emp.id = dept.id;

Emp

ID	State	Dept
1000	CA	
1001	MA	1001
1002	TN	1002

Dept

ID	Name
1001	IT
1002	Sales
1003	Biotech

Emp.ID	Emp.State	Dept.ID	Dept.Name
1001	MA	1001	IT
1002	TN	1002	Sales

SQL: Join operation (cont.)

- left outer join = left join

```
SELECT *
FROM emp left join dept
on emp.id = dept.id;
```

Emp

ID	State	Dept
1000	CA	
1001	MA	1001
1002	TN	1002

Dept

ID	Name
1001	IT
1002	Sales
1003	Biotech

Emp.ID	Emp.State	Dept.ID	Dept.Name
1000	CA	null	null
1001	MA	1001	IT
1002	TN	1002	Sales

Show all employees even if they have no department assigned

SQL: Join operation (cont.)

- right outer join = right join

```
SELECT *
FROM emp right join dept
on emp.id = dept.id;
```

Emp

ID	State	Dept
1000	CA	
1001	MA	1001
1002	TN	1002

Dept

ID	Name
1001	IT
1002	Sales
1003	Biotech

Emp.ID	Emp.State	Dept.ID	Dept.Name
1001	MA	1001	IT
1002	TN	1002	Sales
null	null	1003	Biotech

Show all departments even if they have no employees

GROUPING AND AGGREGATION

Purchase

ProdName	Store	Date	Price	Quantity

Grouping and Aggregation

Purchase(product, date, price, quantity)

Find total sales after 01/1/2014 per product.

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > '01/1/2014'
GROUP BY  product
```

Let's see what this means…

Grouping and Aggregation

1. Compute the FROM and WHERE clauses.

2. Group by the attributes in the GROUPBY

3. Compute the SELECT clause: grouped attributes and aggregates.

HAVING Clause

Same query, except that we consider only products that had at least 30 buyers.

```
SELECT    product, Sum(price * quantity)
FROM      Purchase
WHERE     date > '01/1/2014'
GROUP BY  product
HAVING    Sum(quantity) > 3
```

HAVING clause contains conditions on aggregates.

General form of Grouping and Aggregation

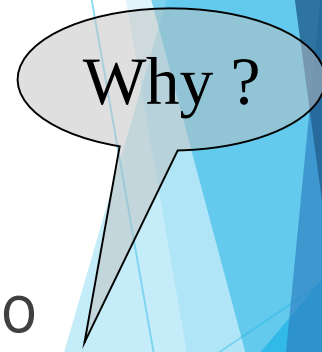
SELECT S

FROM $R_1,\ldots,R_n$

WHERE C1

GROUP BY $a_1,\ldots,a_k$

HAVING C2



S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregates but NO OTHER ATTRIBUTES

C1 = is any condition on the attributes in $R_1,\ldots,R_n$

C2 = is any condition on 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 C1
2. Group by the attributes $a_1,\ldots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

NESTED QUERIES

Details

We will use these instances of relations in our examples.

Reserves

sid	bid	day
22	101	10/10/96
95	103	11/12/96

Sailors

sid	sname	rating	age
22	Dustin	7	45.0
31	Lubber	8	55.5
95	Bob	3	63.5

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Nested Queries

- Powerful feature of SQL: WHERE clause can itself contain an SQL query!
  - Actually, so can FROM and HAVING clauses.

Names of sailors who've reserved boat #103:

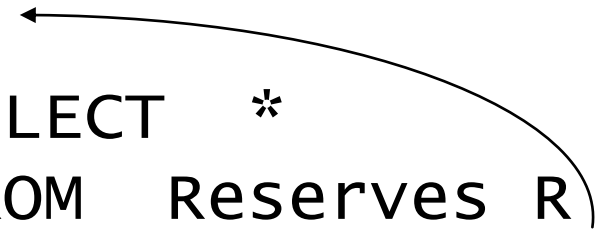
```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN (SELECT R.sid
                 FROM  Reserves R
                 WHERE  R.bid=103)
```

- To find sailors who've **not** reserved #103, use NOT IN.
- To understand semantics of nested queries:
  - think of a nested loops evaluation: For each Sailors tuple, check the qualification by computing the subquery.

Nested Queries with Correlation

Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM  Sailors S
WHERE EXISTS (SELECT  *
              FROM  Reserves R
              WHERE R.bid=103 AND
S.sid=R.sid)
```



- **EXISTS** is another set comparison operator, like IN.
- Can also specify **NOT EXISTS**
- If UNIQUE is used, and * is replaced by R.bid, finds sailors with at most one reservation for boat #103.
  - UNIQUE checks for duplicate tuples in a subquery;
- Subquery must be recomputed for each Sailors tuple.
  - Think of subquery as a function call that runs a query!

Subqueries producing One Value

Usually subqueries produce a relation as an answer. However, sometimes we expect them to produce single values

```
SELECT Purchase.product
FROM     Purchase
WHERE  buyer =
   (SELECT  buyer
   FROM     Person
   WHERE   Home_Zip = '70810' );
```

In this case, the subquery returns one value.
If it returns more, it's a run-time error.

GROUP BY v.s. Nested Quereis

```
SELECT      product, Sum(price*quantity) AS TotalSales
FROM        Purchase
WHERE       date > '10/1/2014'
GROUP BY    product
```

```
SELECT DISTINCT  x.product, (SELECT Sum(y.price*y.quantity)
                             FROM     Purchase y
                             WHERE x.product = y.product
                                   AND y.date > '10/1/2014')
                            AS TotalSales
FROM        Purchase x
WHERE       x.date > '10/1/2010'
```

Group-by v.s. Nested Query

Author(login,name)

Wrote(login,url)

- Find authors who wrote ≥ 10 documents:
- Attempt 1: with nested queries

```
SELECT DISTINCT Author.name
FROM        Author
WHERE       count(SELECT 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
```

No need for DISTINCT: automatically from GROUP BY

Group-by v.s. Nested Query

Author(login,name)

Wrote(login,url)

Mentions(url,word)

Find authors with vocabulary ≥ 10000 words:

```
SELECT    Author.name
FROM      Author, Wrote, Mentions
WHERE     Author.login=Wrote.login AND Wrote.url=Mentions.url
GROUP BY  Author.name
HAVING    count(distinct Mentions.word) > 10000
```

Two Examples

Store(sid, sname)
Product(pid, pname, price, sid)

Find all stores that sell *only* products with price > 100

same as:

Find all stores s.t. all their products have price > 100)

```
SELECT Store.name
FROM    Store, Product
WHERE  Store.sid = Product.sid
GROUP BY  Store.sid, Store.name
HAVING 100 < min(Product.price)
```

```
SELECT Store.name
FROM    Store
WHERE  Store.sid NOT IN
                (SELECT Product.sid
                  FROM Product
                  WHERE  Product.price <= 100)
```

Two Examples

Store(sid, sname)
Product(pid, pname, price, sid)

For each store,
find its most expensive product

Two Examples

This is easy but doesn't do what we want:

```
SELECT Store.sname, max(Product.price)
FROM    Store, Product
WHERE  Store.sid = Product.sid
GROUP BY  Store.sid, Store.sname
```

Better:

```
SELECT Store.sname, x.pname
FROM    Store, Product x
WHERE  Store.sid = x.sid and
            x.price >=
                ALL (SELECT y.price
                        FROM Product y
                        WHERE Store.sid = y.sid)
```

But may return multiple product names per store

Two Examples

Finally, choose some pid arbitrarily, if there are many with highest price:

```
SELECT Store.sname, max(x.pname)
FROM   Store, Product x
WHERE  Store.sid = x.sid and
       x.price >=
            ALL (SELECT y.price
                 FROM Product y
                 WHERE Store.sid = y.sid)
GROUP BY Store.sname
```

NULLS in SQL

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

- 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
heigth=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
```

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