Introduction to Database Systems
IDBS – Fall 2024

- Week 2:
- SQL Select
- SQL Joins
- SQL Aggregations

Eleni Tzirita Zacharatou

Readings:

PDBM 7.3

SQL

Two in One Language

- **Data Definition Language (DDL)**
  - Used by the database administrator (DBA) to define the database's data model
  - Three common commands:
    - CREATE TABLE, ALTER TABLE, and DROP TABLE
- **Data Manipulation Language (DML)**
  - Used by applications and users to retrieve, insert, modify, and delete records
  - Four statements:
    - SELECT, INSERT, UPDATE, and DELETE

Today's focus

Database Schema

Running Example

- **Coffees(<u>name</u>, manufacturer)**
- **Coffeehouses(<u>name</u>, address, license)**
- **Drinkers(<u>name</u>, address, phone)**
- **Likes(<u>drinker</u>, <u>coffee</u>)**
- **Sells(<u>coffeehouse</u>, <u>coffee</u>, price)**
- **Frequents(<u>drinker</u>, <u>coffeehouse</u>)**

Note

Keys are underlined

User

declaratively

Query

DBMS Interface

SQL DML -- Select

Readings:

PDBM 1

Rationale & Example

- **SELECT** -- desired attributes
  FROM -- one or more relations
  WHERE -- condition about records of the relations

- **Which coffees are made by Ottolina?**

Coffees

name	manufacturer

```sql
SELECT name FROM Coffees
WHERE manufacturer = 'Ottolina';
```

name
Fortissima
Maracaibo
Buongiorno

- **Result**
  - a relation
  - a single attribute

The Same but Graphically

Coffees

name	manufacturer
…	…
…	…
Maracaibo	Ottolina
…	…

```sql
SELECT name FROM Coffees
WHERE manufacturer = 'Ottolina';
```

iterate

filter

Check if Ottolina

add

If so, project name

name
Fortissima
Maracaibo

Note

SQL uses bag semantics

Allows for duplicates

Star in SELECT

- What if the **SELECT** clause has ***** instead**?**

Coffees

name	manufacturer

```sql
SELECT * FROM Coffees
WHERE manufacturer = 'Ottolina';
```

name	manufacturer
Fortissima	Ottolina
Maracaibo	Ottolina
Buongiorno	Ottolina

- **This selects all attributes from the Coffees relation**

Attribute Renaming

- **What If you want the result to have different attribute name?**
  - Use then the keyword AS

Coffees

name	manufacturer

```sql
SELECT name AS coffee, manufacturer FROM Coffees
WHERE manufacturer = 'Ottolina';
```

coffee	manufacturer
Fortissima	Ottolina
Maracaibo	Ottolina
Buongiorno	Ottolina

Expressions in SELECT

- **Any expression that makes sense can appear as an element of a SELECT clause.**

```
SELECT coffeehouse, coffee, price * 0.965 AS priceInYen FROM Sells;
```

Sells

coffeehouse	coffee	price

coffeehouse	coffee	priceInYen
Sue's	Fortissima	342
Joe's	Maracaibo	285
Bob's	Buongiorno	149

Constants as Expressions in SELECT

- **Any constant as expression (example with the Likes relation)**

```sql
SELECT drinker,
       'likes Fortissima' AS whoLikesFort
FROM Likes
WHERE coffee = 'Fortissima';
```

Likes

drinker	coffee

drinker	whoLikesFort
Sally	Likes Fortissima
Fred	Likes Fortissima

Live Exercise

- **What should the following queries return?**
  - Assume the relation Coffees has two records only

coffee	manufacturer
Fortissima	Ottolina
Maracaibo	Ottolina

coffee	manufacturer
Fortissima	Ottolina
Maracaibo	Ottolina

```sql
SELECT *
FROM Coffees;
```

Query
42
42

```sql
SELECT 42 AS Query
FROM Coffees
```

Query
42

```sql
SELECT 42 AS Query;
```

Practice

- **Analog has a 50% discount on a second coffee of the same type. For each coffee sold there, show the price of two coffees (‘priceOfTwo’).**
- **A coffeehouse owner by the name of “Eleni” has passed out with caffeine shock. Write a query to find her home phone number.**

Conditions in WHERE

- **Comparisons =, <>, <, >, <=, >=.**
  - Many other operators that produce boolean-valued results.
- **Boolean operators AND, OR, NOT.**
- **Find the price Joe's coffeehouse charges for Maracaibo in Sells**

Sells

coffeehouse	coffee	price

```sql
SELECT price
FROM Sells
WHERE coffeehouse = 'Joe''s' AND coffee = 'Maracaibo';
```

Patterns

- **A condition can compare a string to a pattern by:**
  - <Attribute> LIKE <pattern>
  - <Attribute> NOT LIKE <pattern>
- **Pattern is a quoted string with**
  - % = “any string”
  - _ = “any character”
- **Escaping characters**
  - "\” default escape character
  - ESCAPE modifier
- **PostgreSQL supports regular expressions**
  - These are much more expressive!

```sql
SELECT * FROM Coffeehouses WHERE address LIKE '%1';
```

```sql
SELECT * FROM Coffeehouses WHERE address LIKE '%!_1' ESCAPE '!';
```

Practice

- **A coffeehouse patron by the last name of "Sivertsen" has passed out, again from caffeine shock. Show a query to find his home phone.**
- **Someone has recommended a coffeehouse that sells a coffee called "Blue"-something, that costs more than 100. Unfortunately, she doesn't remember the name of the coffeehouse. Find where that coffee is sold at such a high price.**

Sorting

- **SQL assumes bags semantics**
  - No order assumption
- **ORDER BY sorts the results**
  - Ascending (ASC) or descending (DESC)

```sql
SELECT coffeehouse, price
FROM Sells
ORDER BY coffeehouse, price DESC;
```

- **When should we use ORDER BY?**

User

SQL DML -- Joins

Readings:

PDBM 1

Multi-Relation Queries

- Most typically, queries combine data from more than one relation.
- Several relations in the FROM clause.
- Distinguish attributes of the same name by “<relation>.<attribute>”
- Find the coffees liked by at least one person who frequents Joe’s.

```sql
SELECT coffee
FROM Likes
JOIN Frequents
ON Likes.drinker = Frequents.drinker
WHERE Frequents.coffeehouse = 'Joe''s';
```

The Same but Graphically

```sql
SELECT coffee
FROM Likes
JOIN Frequents
ON Likes.drinker = Frequents.drinker
WHERE Frequents.coffeehouse = 'Joe''s';
```

Likes

drinker	coffee
…	…
…	…
Sally	Maracaibo
…	…

join

Frequents

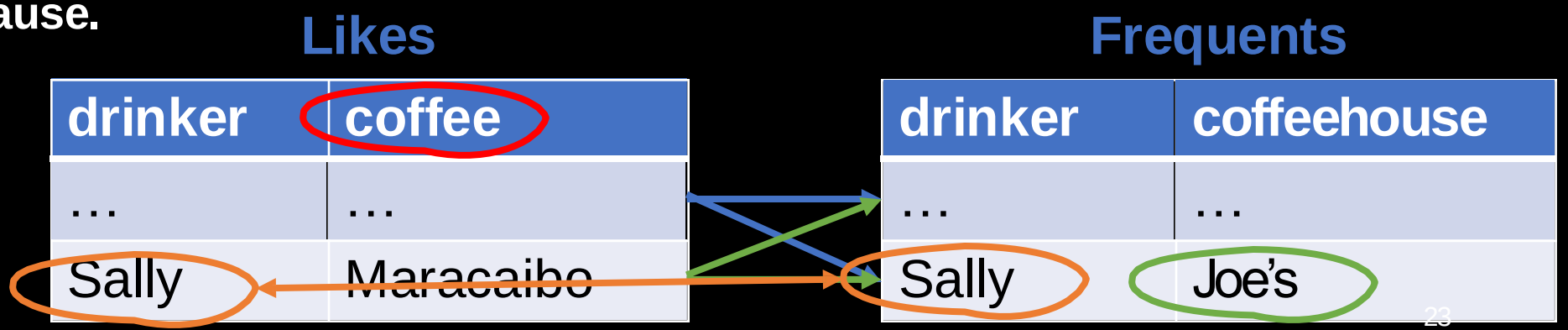
drinker	coffeehouse
…	…
…	…
Sally	Joe's
…	…

filter

Join Semantics

- **Start with the product of all relations in the FROM clause**
  - Imagine one tuple-variable for each relation in the FROM clause.
  - Think of nested for-loops
- **Apply the selection conditions from the JOIN clauses**
  - Usually F.foreign_key = K.key
- **Apply the selection condition from the WHERE clause**
- **Apply the selection Project onto the list of attributes and expressions in the SELECT clause.**

```sql
SELECT coffee
FROM Likes
JOIN Frequents
ON Likes.drinker = Frequents.drinker
WHERE Frequents.coffeehouse = 'Joe''s';
```



Renaming Relations

```sql
SELECT coffee
FROM Likes
JOIN Frequents
ON Likes.drinker = Frequents.drinker
WHERE Frequents.coffeehouse = 'Joe''s';
```

```sql
SELECT L.coffee
FROM Likes L
JOIN Frequents F
ON L.drinker = F.drinker
WHERE F.coffeehouse = 'Joe''s';
```

Practice

- **For each person that "frequents" some coffeehouse, show the name of the person and the address of the coffeehouse.**
- **For each person that "frequents" some coffeehouse, show the address of the person and the address of the coffeehouse.**
- **For each person that "frequents" some coffeehouse, show the name of the person and the address of the coffeehouse.**
  - Why don't we need the Drinkers relation?
  - What if we want only the names of drinkers and coffeehouses?
  - What if we used IDs in our design, and wanted the names?
  - What if we want all drinkers, including those that don't frequent any coffeehouse?
  - What if we only want the name of drinkers?

Select + Join

```
SELECT      -- desired attributes
FROM        -- one or more relations
WHERE       -- condition about records of the relations
JOIN        -- connect records between relations
```

Duplicate Elimination

- **Force the result to be a set with SELECT DISTINCT . . .**
  - Can also do this with GROUP BY – but please don't!
- **From Sells(coffeehouse, coffee, price)**
  - find all the different prices charged for coffees:

Note

Sorting = O(n log n)

```sql
SELECT DISTINCT price
FROM Sells;
```

Note

Without DISTINCT, each price would be listed as many times as there were coffeehouse/coffee pairs at that price

User

declaratively

Query

DBMS Interface

SQL DML -- Aggregations

Readings:

PDBM 1

Relation-based Aggregations

- **SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause**
  - Produces that aggregation on the column
- **COUNT(*) counts the number of tuples.**
- **Find the average price of Maracaibo**

```sql
SELECT AVG(price)
FROM Sells
WHERE coffee = 'Maracaibo';
```

Duplicate Elimination

- **Use DISTINCT inside an aggregation.**

- **Find the number of different prices charged for Maracaibo**

```sql
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE coffee = 'Maracaibo';
```

Practice

- **Show the highest price of any coffee**

Nulls in Aggregations

Sells

coffeehouse	coffee	price

- **NULL never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.**
- **But if there are only NULL values in a column**
  - The result of the aggregation is NULL
- **Exception: COUNT of an empty set is 0.**

Note

The number of coffeehouses at a known price!

```sql
SELECT COUNT(coffeehouse)
FROM Sells
WHERE coffee = 'Maracaibo';
```

```sql
SELECT COUNT(price)
FROM Sells
WHERE coffee = 'Maracaibo';
```

Grouping-based Aggregations

- **SELECT-FROM-WHERE expression followed by GROUP BY**
- **The relation that results from the SELECT-FROM-WHERE is:**
  - grouped according to the values of the attributes in the **GROUP BY** clause
  - any aggregation is applied only within each group
- **How does the system perform GROUP BY?**
  - sort the relation OR hash it!
- **Find the average price for each coffee**

```sql
SELECT coffee, AVG(price)
FROM Sells
GROUP BY coffee;
```

coffee	AVG(price)
Maracaibo	2.33
Fortissima	3.51

Grouping-based Aggregations

- Find for each drinker the average price of Maracaibo at the coffeehouses they frequent

```sql
SELECT drinker, AVG(price)
FROM Frequents F
JOIN Sells S ON F.coffeehouse = S.coffeehouse
WHERE coffee = 'Maracaibo'
GROUP BY drinker;
```

Frequents

drinker	coffeehouse

Sells

coffeehouse	coffee	price

SELECT Attributes with Aggregations

- **If any aggregation is used, then each element of the SELECT list must be either:**
  - An attribute on the GROUP BY list
  - Aggregation – COUNT, AVG, MAX, …
- **All GROUP BY attributes must be in SELECT**
  - Caveat: SQL standard vs. Implementation
    - Some systems are more flexible!
    - PostgreSQL allows omitting functionally dependent attributes (see week 6!)

Example Query

Sells

coffeehouse	coffee	price

- For each coffeehouse, show the average price of all coffees in that coffeehouse.

```sql
SELECT coffeehouse, AVG(price) AS average_price
FROM Sells
GROUP BY coffeehouse;
```

coffeehouse	average_price

What About this One?

- **Show the price of the most expensive coffee (from any coffeehouse) and the name of the coffeehouse that sells it**

sql

```sql
SELECT coffeehouse, MAX(price)
FROM Sells
GROUP BY coffeehouse;
```

sql

```sql
SELECT coffeehouse, MAX(price)
FROM Sells
```

sql

```sql
SELECT coffeehouse, price
FROM Sells
WHERE price = (SELECT MAX(price) FROM Sells);
```

Practice

- For each coffeehouse, show the number of coffees sold in that coffeehouse and the average price.
- For each coffeehouse, show the price of the most expensive coffee sold in that coffeehouse.
- Show the name and price of the least expensive coffee (from any coffeehouse).

HAVING Clause

- **HAVING <condition> may follow a GROUP BY clause.**
  - If so, the condition applies to each group, and groups not satisfying the condition are eliminated.
- **Like WHERE but for groups!**
- **Find the average price of those coffees that are served in at least two coffeehouses**

Sells

coffeehouse	coffee	price

```sql
SELECT coffee, AVG(price)
FROM Sells
GROUP BY coffee
HAVING COUNT(coffeehouse) > 1
```

HAVING Conditions

- **Anything goes in a sub-query – later…**

- **Outside sub-queries, they may refer to attributes only if they are either**
  - A grouping attribute, or
  - Aggregated

  (same condition as for SELECT clauses with aggregation)

Practice

- For each coffeehouse that sells more than two coffees, show the number of coffees sold in that coffeehouse and their average price.
- For each coffeehouse, show the number of drinkers that frequent the coffeehouse.
- For each drinker that frequents more than one coffeehouse, show the number of coffeehouses that he/she frequents.

GROUP BY + HAVING Implementation

depid	sal
1	39
3	21
2	15
9	97
1	41
.	
.	
.	
.	

sort →

depid	sal
1	39
1	41
.	.
2	15
.	
3	21
.	
.	
9	97

aggregate →

depid	AVG	COUNT
1	40	2
2	20	12
3	.	6
.	.	.
9	120	11

→

depid	AVG
2	20
9	120

sql

```sql
SELECT depid, AVG(sal)
FROM Emp
GROUP BY depid
HAVING COUNT(*) > 10
```

Takeaways

- **SQL universal relational query language**
  - Result set (bag) is a table
- **1 Block**
  - select, from, where, group by, having, sort by clauses
- **Joins**
  - Combine data from many tables
- **Aggregations**
  - Group data
- **SQL is code**
  - Treat it accordingly

What is next?

- **Next week: Advanced SQL**
- **Exercise 2 is out (SQL)**
  - Remember: No submission – but very important to do it!
  - It has some queries that you did not learn how to do yet – just think about potential solutions for now, we will cover them next week.
- **Homework 1 description is out**
  - SQL queries, some being more complex
  - PDF and SQL script are already available
  - Quiz on LearnIT will open at the end of the week
  - **For help:** Talk to TAs in exercise class
  - Due Sep 23 at **23:59** – individual submissions!
  - **Remember:** We only accept and give feedback to HWs submitted before the deadline.
  - **Do the exercises before the homework!**