COSC 311: Introduction to Data Visualization and Interpretation

(2) SQL

Dr. Shuangquan (Peter) Wang
(spwang@salisbury.edu)

Department of Computer Science

Salisbury University

About this note

- The contents of this note refer to:
  - ➢ Prof. Jeff Ullman and Prof. Jennifer Widom's Teaching Materials (Stanford University)
    - http://infolab.stanford.edu/~ullman/fcdb.html
  - ➢ Prof. Fang Li's Teaching Materials (Shanghai Jiao Tong University)
    - https://www.cs.sjtu.edu.cn/~li-fang/DB.htm
  - ➢ Prof. Tim Finin's Teaching Materials (UMBC)
    - https://www.csee.umbc.edu/courses/461/461.shtml
  - ➢ Prof. Matei Zaharia's Teaching Materials (Stanford University)
    - https://web.stanford.edu/class/cs245/
  - ➢ SQL Tutorial
    - https://www.w3schools.com/sql/default.asp
  - ➢ Some pictures are from Google search

Dissemination or sale of any part of this note is NOT permitted!

Content

1. Simple queries in SQL
2. Queries involving more than one relation
3. Subqueries
4. Aggregation

Language SQL

- SQL is a standard database language, adopted by many commercial systems
  - How to query the database
  - How to make modifications on database
  - Transactions in SQL

Why SQL?

- SQL is a very-high-level language.
  - Say “what to do” rather than “how to do it.”
  - Avoid a lot of data-manipulation details needed in procedural languages like C++ or Java.

- Database management system (DBMS) figures out the “best” way to execute query.
  - Called “query optimization”

SQL: structured query language

- Components of language:
  - Schema definition, Data retrieval, Data modification, Indexes, Constraints, Views, Triggers, Transactions, authorization, etc.

- DDL = data definition language
- DML = data manipulation language

- Two forms of usage:
  - Interactive SQL (GUI, prompt)
  - Embedded SQL (Python, C, Java)

Select-From-Where Statements

```
SELECT <desired attributes>
FROM <tuple variables or relation name>
WHERE <conditions>
GROUP BY <attributes>
HAVING <conditions>
ORDER BY < list of attributes>
```

Principal form

Our Running Example

- All our SQL queries will be based on the following database schema.
  - Underline indicates key attributes.

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

Example: Query on one relation

- Using Beers(name, manf), what beers are made by Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Note: single quotes for strings

name
Bud
Bud Lite
Michelob
. . .

The answer is a relation with a single attribute, name, and tuples with the name of each beer by Anheuser-Busch, such as Bud.

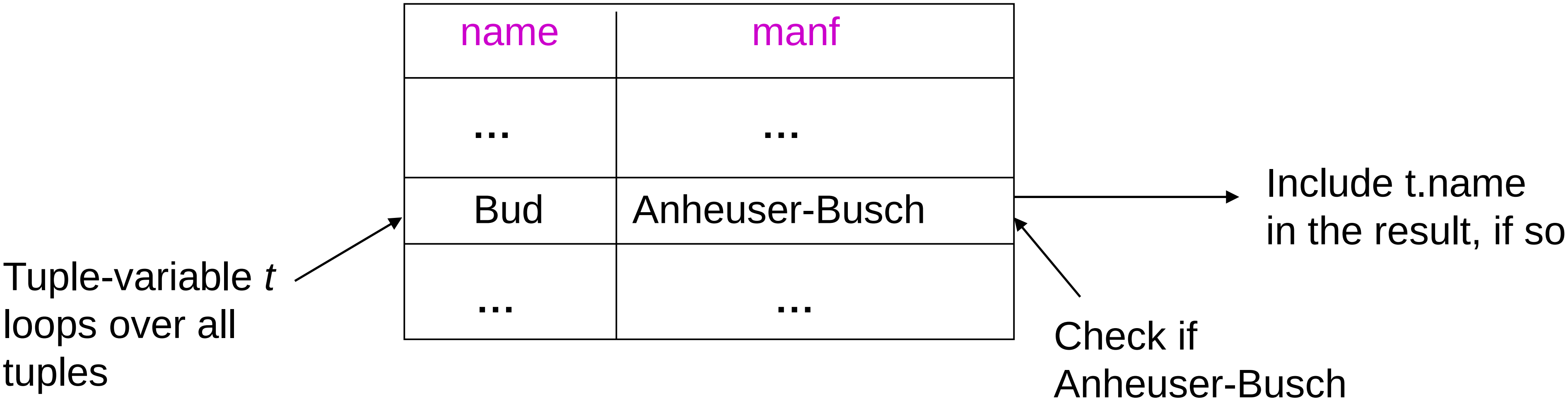
Meaning of Single-Relation Query

- Begin with the relation in the FROM clause.
- Apply the selection indicated by the WHERE clause.
- Apply the extended projection indicated by the SELECT clause.

```
SELECT <desired attributes>
FROM <tuple variables or relation name>
WHERE <conditions>
```

Operational Semantics

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```



name	manf
…	…
Bud	Anheuser-Busch
…	…

Tuple-variable *t* loops over all tuples

Check if Anheuser-Busch

Include t.name in the result, if so

Operational Semantics --- General

- Think of a *tuple variable* visiting each tuple of the relation mentioned in FROM.
- Check if the “current” tuple satisfies the WHERE clause.
- If so, compute the attributes or expressions of the SELECT clause using the components of this tuple.

* In SELECT clauses

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation."

- Example: Using Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

Result of Query:

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
. . .	. . .

Now, the result has each of the attributes of Beers.

Renaming Attributes

- If you want the result to have different attribute names, use “AS <new name>” to rename an attribute.

- Example: Using Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

Result of Query:

beer	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
. . .	. . .

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause.
- Example: Using Sells(bar, beer, price):

```
SELECT bar, beer,
       price*114 AS priceInYen
FROM Sells;
```

bar	beer	priceInYen
Joe’s	Bud	285
Sue’s	Miller	342
…	…	…

Complex Conditions in WHERE Clause

- Boolean operators AND, OR, NOT.
- Comparisons =, <>, <, >, <=, >=.
  - And many other operators that produce boolean-valued results.

Example: Complex Condition

- Using Sells(bar, beer, price), find the price Joe’s Bar charges for Bud:

```
SELECT price
FROM Sells
WHERE bar = ’Joe’’s Bar’ AND
      beer = ’Bud’;
```

Patterns

- WHERE clauses can have conditions in which a string is compared with a pattern, to see if it matches.
- Form:
  - <Attribute> LIKE <pattern> or
  - <Attribute> NOT LIKE <pattern>
- *Pattern* is a quoted string with % = "any string"; _ = "any character."

Note: SQL is case insensitive. Keywords like SELECT or AND can be written upper/lower case as you like. Only inside quoted strings does case matter.

Example: LIKE

- Using Drinkers(name, addr, phone) find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE ’%555-_ _ _ _’;
```

2. Queries involving more than one relation

- Interesting queries often combine data from more than one relation.
- We can address several relations in one query by listing them all in the FROM clause.
- Distinguish attributes of the same name by “<relation>.<attribute>” .

Example: Joining Two Relations

- Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents Joe’s Bar.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = ’Joe’’s Bar’ AND
   Frequents.drinker =
      Likes.drinker;
```

Formal Semantics

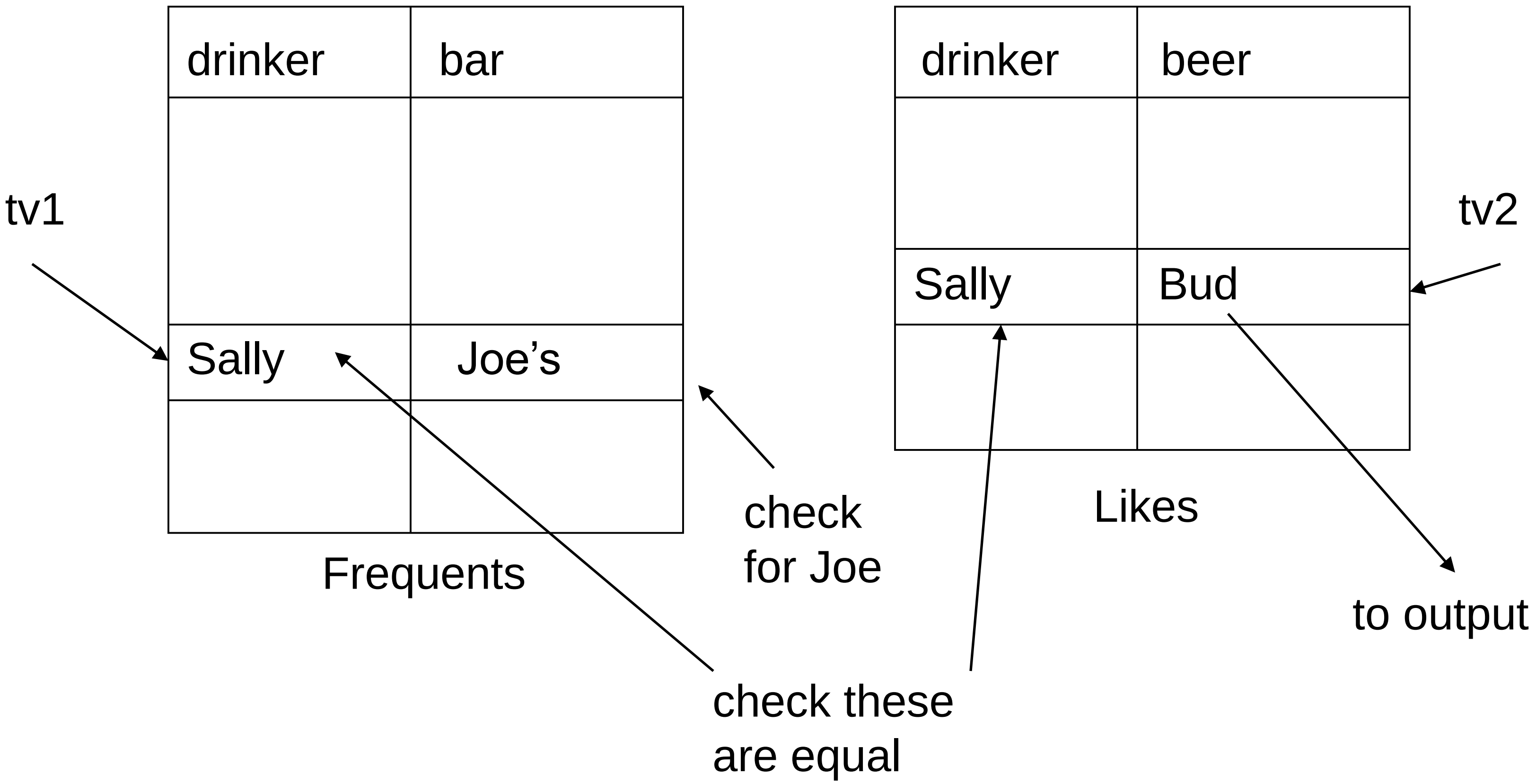
- Almost the same as for single-relation queries:
  1. Start with the product of all the relations in the FROM clause.
  2. Apply the selection condition from the WHERE clause.
  3. Project onto the list of attributes and expressions in the SELECT clause.

Operational Semantics

- Imagine one tuple-variable for each relation in the FROM clause.
  - These tuple-variables visit each combination of tuples, one from each relation.
- If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause.

Example

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joe''s Bar' AND
      Frequents.drinker = Likes.drinker;
```



drinker	bar
Sally	Joe's

Frequents

drinker	beer
Sally	Bud

Likes

3. Subqueries

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including FROM and WHERE clauses.

- Example: in place of a relation in the FROM clause, we can use a subquery and then query its result.
  - Must use a tuple-variable to name tuples of the result.

Example: Subquery in FROM

- Find the beers liked by at least one person who frequents Joe’s Bar.

Drinkers who frequent Joe’s Bar

```
SELECT beer
FROM Likes, (SELECT drinker
    FROM Frequents
    WHERE bar = 'Joe''s Bar')JD
WHERE Likes.drinker = JD.drinker;
```

Note:

- JD is a tuple-variable to name tuples of the result
- Parentheses around subquery are essential

Subqueries That Return One Tuple

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.
  - Usually, the tuple has one component.
  - A run-time error occurs if there is no tuple or more than one tuple.

Example: Single-Tuple Subquery

- Using Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.
- Two queries would surely work:
  1. Find the price Joe charges for Bud.
  2. Find the bars that serve Miller at that price.

Query + Subquery Solution

```
SELECT bar
FROM Sells
WHERE beer = ’Miller’ AND
      price = (SELECT price
               FROM Sells
               WHERE bar = ’Joe”s Bar’
                  AND beer = ’Bud’);
```

The price at which Joe sells Bud

Note the scoping rule: an attribute refers to the most closely nested relation with that attribute.

The IN Operator

- <tuple> IN (<subquery>) is true if and only if the tuple is a member of the relation produced by the subquery.
  - Opposite: <tuple> NOT IN (<subquery>).
- IN-expressions can appear in WHERE clauses.

Example: IN

- Using Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker = ’Fred’);
```

The set of beers Fred likes

Exercise: What is the difference?

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

a	b
1	2
3	4

R

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

b	C
2	5
2	6

S

Exercise: What is the difference?

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

Double loop, over the tuples of R and S

(1,2) with (2,5) and (1,2) with (2,6) both satisfy the condition; 1 is output twice.

a	b
1	2
3	4

R

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

One loop, over the tuples of R

Two 2's

(1,2) satisfies the condition; 1 is output once.

b	C
2	5
2	6

S

The Exists Operator

- EXISTS(<subquery>) is true if and only if the subquery result is not empty.
- Example: From Beers(name, manf) , find those beers that are the unique beer by their manufacturer.

Example: EXISTS

```
SELECT name
FROM Beers b1
WHERE NOT EXISTS (
    SELECT *
    FROM Beers
    WHERE manf = b1.manf  AND
          name <> b1.name);
```

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute.

Set of beers with the same manf as b1, but not the same beer

Notice the SQL “not equals” operator

Note: A subquery that refers to values from a surrounding query is called a correlated subquery.

4. Aggregation

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

Example: Aggregation

- From Sells(bar, beer, price), find the average price of Bud:

```
SELECT AVG(price)
FROM Sells
WHERE beer = ’Bud’;
```

Eliminating Duplicates in an Aggregation

- Use DISTINCT inside an aggregation.
- Example: From Sells(bar, beer, price), find the number of *different* prices charged for Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = ’Bud’;
```

Note: DISTINCT may be used in any aggregation, but typically only makes sense with COUNT.

Grouping

- We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes, and any aggregation is applied only within each group.

Example: Grouping

- From Sells(bar, beer, price), find the average price for each beer:

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

beer	AVG(price)
Bud	2.33
…	…

Example: Grouping

- From Sells(bar, beer, price) and Frequents(drinker, bar), find for each drinker the average price of Bud at the bars they frequent:

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = ’Bud’ AND
      Frequents.bar = Sells.bar
GROUP BY drinker;
```

Compute all drinker-bar-price triples for Bud.

Then group them by drinker.

Thanks

Reading textbook

- A First Course in Database systems (Third Edition), by Jeff Ullman and Jennifer Widom.
  - Chapter 6