Relational Model and Algebra
Introduction to Databases
CompSci 316 Fall 2020

Today’s plan
• Revisit relational model
• Simple SQL queries and its semantic
• Start relational algebra

The famous “Beers” database
Bar
Each has an address

Drinker Frequent Bars
“X” times a week

Drinker Likes Beer

Beer
Each has a brewer

Your database for HW1

“Beers” as a Relational Database

<table>
<thead>
<tr>
<th>Bar</th>
<th>Brewer</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>Anheuser-Busch Inc.</td>
<td>108 Morris Street</td>
</tr>
<tr>
<td>The Edge</td>
<td>Grupo Modelo</td>
<td>905 W. Main Street</td>
</tr>
</tbody>
</table>
| Satisfaction | Dixie Brewing | Drinker

<table>
<thead>
<tr>
<th>Drinker</th>
<th>Bar</th>
<th>Times_a_week</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
<td>The Edge</td>
<td>2</td>
</tr>
<tr>
<td>Dan</td>
<td>The Edge</td>
<td>1</td>
</tr>
<tr>
<td>Ben</td>
<td>Satisfaction</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Drinker</th>
<th>Beer</th>
<th>Likes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amy</td>
<td>Corona</td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td>Budweiser</td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td>Corona</td>
<td></td>
</tr>
<tr>
<td>Ben</td>
<td>Budweiser</td>
<td></td>
</tr>
</tbody>
</table>

Basic queries: SFW statement

\[
\text{SELECT } A_{1}, A_{2}, \ldots, A_{n} \\text{FROM } R_{1}, R_{2}, \ldots, R_{m} \\text{WHERE } \text{condition}
\]

In HW1, you can only use SFW

• SELECT, FROM, WHERE are often referred to as SELECT, FROM, WHERE “clauses”

• Each query must have a SELECT and a FROM

• WHERE is optional

Announcements (Thu. Aug. 20)
• Project details posted on Sakai
• Read it carefully!
• Think about fixed vs. open project (some project videos from last semester will be available on sakai soon – keep them private)
• Roster for discussion sessions available on sakai (teammates have to be from the same discussion session)
• You do not have to form your teams or decide fixed/open projects right now. Names of team members and project choice are due on 9/8, so you will have some time (and the class/discussion sections are still in flux)
• Survey has been sent – Due by tomorrow 08/21 night EDT
• To know about your time zones, expectations, available resources, project/team-member preference etc.
• Please respond on time – there is a 2% weight for communication!
• Monday’s discussion sessions: Installation and practice SQL
• Emails coming soon
Example: reading a table

- SELECT *
  FROM Serves

- Single-table query
- * is a shorthand for “all columns”

---

Example: ORDER BY

- SELECT *
  FROM Serves
  ORDER BY beer

- Equivalent to “ORDER BY beer asc” (asc is default option)
- For descending order, use “desc”
- Can combine multiple orders
- What does this return?
  - ORDER BY beer asc, price desc

---

Example: some columns and DISTINCT

- SELECT beer
  FROM Serves

- Only want unique values? Use DISTINCT

- SELECT DISTINCT beer
  FROM Serves

- Returns a set

---

Example: selecting few rows

- SELECT beer AS mybeer
  FROM Serves
  WHERE price < 2.75

- SELECT S.beer
  FROM Serves S
  WHERE bar = 'The Edge'

- What does these return?
  - SELECT list can contain expressions
    Can also use built-in functions such as SUBSTR, ABS, etc.
  - NOT EQUAL TO: Use <>
  - LIKE matches a string against a pattern
    % matches any sequence of zero or more characters

---

Example: Join

- Find addresses of all bars that ‘Dan’ frequents

- Which tables do we need?
Example: Join

• Find addresses of all bars that 'Dan' frequents

<table>
<thead>
<tr>
<th>Bar</th>
<th>name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>108 Morris Street</td>
<td></td>
</tr>
<tr>
<td>Satisfaction</td>
<td>905 W. Main Street</td>
<td></td>
</tr>
</tbody>
</table>

Semantics of SFW

• SELECT $E_1, E_2, ..., E_m$
  FROM $R_1, R_2, ..., R_m$
  WHERE condition

1. Apply “FROM” Form “cross-product” of $R_1, .., R_m$

   If condition is true over $t_1, t_2, ..., t_m$:

   1. Apply “WHERE”
     Only consider satisfying rows
   2. Apply “SELECT”
     Output the desired columns

Step 1: Illustration of Semantics of SFW

• NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” it outputs!

   Form a “Cross product” of two relations

   SELECT B.address
   FROM Bar B, Frequents F
   WHERE B.name = F.bar
   AND F.drinker = 'Dan'

Step 2: Illustration of Semantics of SFW

• NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” it outputs!

   Discard rows that do not satisfy WHERE condition

   SELECT B.address
   FROM Bar B, Frequents F
   WHERE B.name = F.bar
   AND F.drinker = 'Dan'

Step 3: Illustration of Semantics of SFW

• NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” it outputs!

   Output the “address” output of rows that survived

   SELECT B.address
   FROM Bar B, Frequents F
   WHERE B.name = F.bar
   AND F.drinker = 'Dan'

Final output: Illustration of Semantics of SFW

• NOTE: This is “NOT HOW” the DBMS outputs the result, but “WHAT” it outputs!

   Output the “address” output of rows that survived

   SELECT B.address
   FROM Bar B, Frequents F
   WHERE B.name = F.bar
   AND F.drinker = 'Dan'
SQL vs. C++, Java, Python...

SQL is declarative
- Programmer specifies what answers a query should return,
- but not how the query is executed
- DBMS picks the best execution strategy based on availability of indexes, data/workload characteristics, etc.
- Not a "Procedural" or "Operational" language like C++, Java, Python
- There are several ways to write a query, but equivalent queries always provide the same (equivalent) results
- SQL (+ its execution and optimizations) is based on a strong foundation of "Relational Algebra"

Relational algebra
A language for querying relational data based on "operators"

• Core operators:
  - Selection, projection, cross product, union, difference, and renaming
  - Additional, derived operators:
    - Join, natural join, intersection, etc.
  - Compose operators to make complex queries

More on selection
- Selection condition can include any column of \( R \), constants, comparison (\( =, \leq \), etc.) and Boolean connectives (\( \land \) and, \( \lor \) or, \( \neg \) not)
  - Example: Serves tuples for "The Edge" or price \( \geq 2.75 \)
  - Example: Serves tuples for "The Edge" \( \land \) price \( \leq 2.75 \)
- You must be able to evaluate the condition over each single row of the input table!
  - Example: the most expensive beer at any bar
    - Example: the most expensive beer at any bar

Selection
- Input: a table \( R \)
- Notation: \( \sigma_p R \)
  - \( p \) is called a selection condition (or predicate)
- Purpose: filter rows according to some criteria
- Output: same columns as \( R \), but only rows of \( R \) that satisfy \( p \) (set)
  - Example: Find beers with price < 2.75

Projection
- Input: a table \( R \)
- Notation: \( \pi_L R \)
  - \( L \) is a list of columns in \( R \)
- Purpose: output chosen columns
- Output: same rows, but only the columns in \( L \) (set)
  - Example: Find all the prices for each beer

### Tables
#### Serves
<table>
<thead>
<tr>
<th>bar</th>
<th>beer</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Edge</td>
<td>Budweiser</td>
<td>2.50</td>
</tr>
<tr>
<td>The Edge</td>
<td>Corona</td>
<td>3.00</td>
</tr>
<tr>
<td>Satisfaction</td>
<td>Budweiser</td>
<td>2.25</td>
</tr>
</tbody>
</table>

### Questions
- Equivalent SQL query?
- No actual deletion!
- Output of \( \pi_{\text{beer}, \text{price}} \) Serves?
Cross product

- Input: two tables $R$ and $S$
- Notation: $R \times S$
- Purpose: relates rows from two tables
- Output: for each row $r$ in $R$ and each $s$ in $S$, output a row $rs$ (concatenation of $r$ and $s$)
Union

- Input: two tables $R$ and $S$
- Notation: $R \cup S$
- $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows in $R$ and all rows in $S$ (with duplicate rows removed)

Example on board

Difference

- Input: two tables $R$ and $S$
- Notation: $R - S$
- $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows in $R$ that are not in $S$

Example on board

Derived operator: intersection

- Input: two tables $R$ and $S$
- Notation: $R \cap S$
- $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows that are in both $R$ and $S$
  - How can you write it using other operators?

Expression tree notation

- Find addresses of all bars that 'Dan' frequents

The Edge
108 Morris Street
Satisfaction
905 W. Main Street

Bar
name
The Edge
Satisfaction

Frequents
name
Frequent

Wrong!

Using the same relation multiple times

- Find drinkers who frequent both "The Edge" and "Satisfaction"

Using the same relation multiple times

Renaming

- Input: a table $R$
- Notation: $\rho_f R$, $\rho_{(A_1A_2\ldots)} R$, or $\rho_{(A_1A_2\ldots)} R$
- Purpose: "rename" a table and/or its columns
- Output: a table with the same rows as $R$, but called differently
- Used to:
  - Avoid confusion caused by identical column names
  - Create identical column names for natural joins
  - As with all other relational operators, it doesn't modify the database
  - Think of the renamed table as a copy of the original

Example on board
Summary of core operators

• Selection: $\sigma_R$
• Projection: $\pi_L R$
• Cross product: $R \times S$
• Union: $R \cup S$
• Difference: $R - S$
• Renaming: $\rho_{A_1, A_2, ...} R$
  • Does not really add “processing” power

Summary of derived operators

• Join: $R \bowtie_p S$
• Natural join: $R \bowtie S$
• Intersection: $R \cap S$
• Many more
  • Semijoin, anti-semijoin, quotient, ...

Exercise

• Bars that drinkers in address “300 N. Duke Street” do not frequent

Exercise

• Bars that drinkers in address “300 N. Duke Street” do not frequent

A trickier exercise

• For each bar, find the drinkers who frequent it max no. times a week
  • Who do NOT visit a bar max no. of times!
  • Whose times_of_weeks is lower than somebody else’s for a given bar

A trickier exercise

• For each bar, find the drinkers who frequent it max no. times a week

A deeper question:
When (and why) is “−” needed?
Monotone operators

- If some old output rows may need to be removed
  - Then the operator is non-monotone
- Otherwise the operator is monotone
  - That is, old output rows always remain “correct” when more rows are added to the input
- Formally, for a monotone operator \( \text{op} \): \( R \subseteq R' \) implies \( \text{op}(R) \subseteq \text{op}(R') \) for any \( R, R' \)

Which operators are non-monotone?

- Selection: \( \sigma_p R \) Monotone
- Projection: \( \pi_i R \) Monotone
- Cross product: \( R \times S \) Monotone
- Join: \( R \bowtie_p S \) Monotone
- Natural join: \( R \bowtie S \) Monotone
- Union: \( R \cup S \) Monotone
- Difference: \( R - S \) Monotone w.r.t. \( R \); non-monotone w.r.t. \( S \)
- Intersection: \( R \cap S \) Monotone

Why is “−” needed for “highest”?

- Composition of monotone operators produces a monotone query
  - Old output rows remain “correct” when more rows are added to the input
- Is the “highest” query monotone?

Extensions to relational algebra

- Duplicate handling (“bag algebra”)
- Grouping and aggregation
- “Extension” (or “extended projection”) to allow new column values to be computed


(Coming later)