From Relational Algebra to the Structured Query Language

Rose-Hulman Institute of Technology

Curt Clifton

Review – Relational Integrity

- Entity Integrity Constraints:
  - Primary key values cannot be null
- Referential Integrity Constraints:
  - Foreign key values must either:
    - Match the primary key values of some tuple, or
    - Be null

Review – Relational Algebra

- Intersection: $R1 \cap R2$
- Union: $R1 \cup R2$
- Difference: $R1 - R2$
- Selection: $\sigma_{BDATE < 1970\text{-}1\text{-}1}(EMPLOYEE)$
- Projection: $\pi_{FNAME,\ BDATE}(EMPLOYEE)$
- Theta-Join: $DEPT \bowtie_{MGRSSN=SSN} EMP$
- Natural Join: $R1 * R2$

Why Relational Algebra?

- Foundational knowledge
- Used by query optimizers
  - Finer grained than SQL
  - Can be formally reasoned about
- Formal basis for *semantics* of SQL

Homework Problem 6.18

- Parts a–d and g
- Begin in class, may work in groups of 2–3
  - Please note your partners on the sheet

Sets versus Bags

- Sets
  - Order doesn't matter
  - No duplicates
- Examples
  - $\{1,2,3\} = \{2,1,3\}$
  - $\{1,2\} \cup \{2\} = \{1,2\}$

- Bags (or multi-sets)
  - Order doesn't matter
  - Duplicates allowed
- Examples
  - $\{1,2,3\} = \{2,1,3\}$
  - $\{1,2\} \cup \{2\} = \{1,2,2\}$
  - $\{1,2,3\} \neq \{1,2,2,3\}$

Why Bags? Efficiency!

- Eliminating duplicates can be expensive
- By default SQL uses bags

Bag Union

- "Just dump all the elements into a single bag"
- An element appears in the union of two bags the sum of the number of times it appears in each bag

Bag Intersection

- "Whichever bag has the fewest, has the answer"
- An element appears in the intersection of two bags the minimum number of times it appears in either.

Bag Difference

- "Take elements out of the first if they're in the second"
- An element appears in the difference of two bags as many times as it appears in the first, minus the number of times it appears in the second, but no less than 0 times

Bag Selection

- Like set selection
- But input and output can be bags

R

A	B
1	2
5	6
1	2

Bag Projection

- Unlike set project, can turn a set into a bag

R

A	B
1	2
5	6
1	2

Bag Theta-Join

- Pair each tuple of first table with each tuple of second
- Check condition
- Don’t eliminate duplicates

R

A	B
1	2
5	6
1	2

S

B	C
3	4
7	8

Introducing SQL

- Pronounced:
  - “ess queue ell”
  - Or “sequel”
- Benefits:
  - Designed for the relational model
  - Easily optimized by DBMS
  - Standard (well, sort of, a little bit, sometimes)

http://sqlfairy.sourceforge.net

Running Example – The SodaBase

- **Soda**(name, manf)
- **Rest**(name, addr, contract)
- **Customer**(name, addr, phone)
- **Likes**(customer, soda)
- **Sells**(rest, soda, price)
- **Frequents**(customer, rest)

The Basic SQL Query

- SELECT *attributes*
  FROM *table*
  WHERE *condition*
- Semantics: $\pi_{attributes}(\sigma_{condition}(table))$

Example

- Find all the names of all sodas made by PepsiCo

Example

- Find all the names of all sodas made by PepsiCo

-
  ```
  SELECT name
      FROM Soda
      WHERE manf = 'PepsiCo'
  ```
- (note single quotes)

Select *

- For getting all attributes…
- SELECT *
  FROM *table*
  WHERE *condition*
- Semantics: $\sigma_{condition}(table)$

Example

```
SELECT *
    FROM Soda
    WHERE manf = 'PepsiCo'
```

Renaming Attributes

- SELECT *attribute1* AS *newName1*, …
  FROM *table*
  WHERE *condition*
- Semantics:

  $\rho_{newName1,\ \ldots}(\pi_{attribute1,\ \ldots}(\sigma_{condition}(table)))$

Example

-
  ```
  SELECT name AS soda, manf AS maker
      FROM Soda
      WHERE manf = 'PepsiCo'
  ```

Expressions in SELECT Clauses

- Can use expressions on attributes in SELECT
- SELECT *f(attributes)*, …
  FROM *table*
  WHERE *condition*
- More powerful than the relational algebra we've seen
  - Would need functions on tuples

Example

- Show selling prices in Yen

Example

- Show selling prices in Yen
- SELECT rest, soda,

```
                price * 115 AS priceInYen
    FROM Sells
```

Another Example: Constants

-
  ```
  SELECT customer,
                   'likes Pepsi' AS promotion
     FROM Likes
     WHERE soda = 'Pepsi'
  ```

Example: Complex Conditions

- Find the price that Joe’s Sushi charges for Pepsi

Example: Complex Conditions

- Find the price that Joe’s Sushi charges for Pepsi

-
  ```
  SELECT price
      FROM Sells
      WHERE rest = 'Joe''s Sushi'
              AND soda = 'Pepsi'
  ```
- Note:
  - Double apostrophe inside string
  - AND, OR, NOT
  - Case insensitive

Pattern Matching

- WHERE clauses can compare string to pattern
  - *Attribute* LIKE *pattern*
  - *Attribute* NOT LIKE *pattern*
- Pattern syntax:
  - Pattern is a string
  - % in string represents any number of characters
  - _ in string represent any single character

Example

- Find the customers with exchange 555, regardless of area code

Example

- Find the customers with exchange 555, regardless of area code

-
  ```
  SELECT name
      FROM Customer
      WHERE phone LIKE '%555-____'
  -- That's four underscores
  ```

Dealing with Null

- Why might a tuple have a null value?
- SQL uses **three-valued logic** to handle null
  - A boolean expression can be **true**, **false**, or **unknown**
  - Comparison with null yields **unknown** instead of error
  - WHERE clause must be **true** to match

Three-Valued Logic

- True = 1
- False = 0
- Unknown = 1/2
- x AND y = min(x, y)
- x OR y = max(x, y)
- NOT x = 1 - x

Consider

-
  ```
  SELECT rest
      FROM Sells
      WHERE price < 2.00 OR price >= 2.00
  ```
- If the Sells relation has the value:

rest	soda	price
Joe's	Pepsi	null

- Beware of nulls!

Combining Relations

- List multiple tables in FROM
- Use *Relation.Attribute* to distinguish
- SELECT soda

```
    FROM Likes, Frequents
    WHERE
        Frequents.customer = Likes.customer
            AND rest = 'Joe''s'
```

- Semantics: $\pi_{\text{soda}}(\sigma_{condition}(\text{Likes} \times \text{Frequents}))$

Tuple Variables

- Can distinguish two copies of same relation
- Example: Find all pairs of sodas by the same manufacturer…
  - Omitting trivial pairs like (Pepsi, Pepsi)
  - Omitting permutations of same sodas by listing members of pair alphabetically

Solution

-
  ```
  SELECT s1.name, s2.name
      FROM Soda s1, Soda s2
      WHERE s1.manf = s2.manf
          AND s1.name < s2.name
  ```

The Story Thus Far

- SELECT … FROM … WHERE
- SELECT * …
- SELECT Foo AS Bar …
- SELECT *expression* …
- SELECT … FROM … WHERE … LIKE …
- SELECT … FROM Foo, Bar …
- SELECT … FROM Foo f1, Foo f2 …