More SQL

Relations as Bags
Grouping and Aggregation
Database Modification

Union, Intersection, and Difference

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:
  - (subquery) UNION (subquery)
  - (subquery) INTERSECT (subquery)
  - (subquery) EXCEPT (subquery)

Example

- From relations Likes(drinker, beer), Sells(bar, beer, price) and Frequents(drinker, bar), find the drinkers and beers such that:
  1. The drinker likes the beer, and
  2. The drinker frequents at least one bar that sells the beer.

Solution

(SELECT * FROM Likes)
  INTERSECT

Bag Semantics

- Although the SELECT-FROM-WHERE statement uses bag semantics, the default for union, intersection, and difference is set semantics.
  - That is, duplicates are eliminated as the operation is applied.

Motivation: Efficiency

- When doing projection in relational algebra, it is easier to avoid eliminating duplicates.
  - Just work tuple-at-a-time.
- When doing intersection or difference, it is most efficient to sort the relations first.
  - At that point you may as well eliminate the duplicates anyway.
Controlling Duplicate Elimination

- Force the result to be a set by `SELECT DISTINCT ...`
- Force the result to be a bag (i.e., don’t eliminate duplicates) by `ALL`, as in `... UNION ALL ...`

Example: DISTINCT

- From Sells(bar, beer, price), find all the different prices charged for beers:
  ```sql
  SELECT DISTINCT price
  FROM Sells;
  ```
- Notice that without `DISTINCT`, each price would be listed as many times as there were bar/beer pairs at that price.

Example: ALL

- Using relations Frequents(drinker, bar) and Likes(drinker, beer):
  ```sql
  (SELECT drinker FROM Frequents)
  EXCEPT ALL
  (SELECT drinker FROM Likes);
  ```
- Lists drinkers who frequent more bars than they like beers, and does so as many times as the difference of those counts.

Join Expressions

- SQL provides a number of expression forms that act like varieties of join in relational algebra.
  - But using bag semantics, not set semantics.
- These expressions can be stand-alone queries or used in place of relations in a `FROM` clause.

Products and Natural Joins

- Natural join is obtained by:
  ```sql
  R NATURAL JOIN S;
  ```
- Product is obtained by:
  ```sql
  R CROSS JOIN S;
  ```
- Example:
  ```sql
  Likes NATURAL JOIN Serves;
  ```
- Relations can be parenthesized subexpressions, as well.

Theta Join

- R JOIN S ON <condition> is a theta-join, using `<condition>` for selection.
- Example: using Drinkers(name, addr) and Frequents(drinker, bar):
  ```sql
  Drinkers JOIN Frequents ON
  name = drinker;
  ```
  gives us all (d, a, d, b) quadruples such that drinker d lives at address a and frequents bar b.
Outerjoins
◆ R OUTER JOIN S is the core of an outerjoin expression. It is modified by:
   1. Optional NATURAL in front of OUTER.
   2. Optional ON <condition> after JOIN.
   3. Optional LEFT, RIGHT, or FULL before OUTER.
      ◆ LEFT = pad dangling tuples of R only.
      ◆ RIGHT = pad dangling tuples of S only.
      ◆ FULL = pad both; this choice is the default.

Aggregations
◆ 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
◆ DISTINCT inside an aggregation causes duplicates to be eliminated before the aggregation.
◆ Example: find the number of different prices charged for Bud:
   
   SELECT COUNT(DISTINCT price)
   FROM Sells
   WHERE beer = 'Bud';

NULL’s Ignored in Aggregation
◆ NULL never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.
◆ But if there are no non-NUL values in a column, then the result of the aggregation is NULL.

Example: Effect of NULL’s
◆ The number of bars that sell Bud.
◆ The number of bars that sell Bud at a known price.
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;
```

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 drinker = bar AND beer = 'Bud'
GROUP BY drinker;
```

Restriction on SELECT Lists With Aggregation

◆ If any aggregation is used, then each element of the SELECT list must be either:
  1. Aggregated, or
  2. An attribute on the GROUP BY list.

Illegal Query Example

◆ You might think you could find the bar that sells Bud the cheapest by:

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = 'Bud';
```
◆ But this query is illegal in SQL.
  • Why? Note bar is neither aggregated nor on the GROUP BY list.

HAVING Clauses

◆ HAVING <condition> may follow a GROUP BY clause.
◆ If so, the condition applies to each group, and groups not satisfying the condition are eliminated.
Requirements on HAVING Conditions

- These conditions may refer to any relation or tuple-variable in the FROM clause.
- They may refer to attributes of those relations, as long as the attribute makes sense within a group; i.e., it is either:
  1. A grouping attribute, or
  2. Aggregated.

Example: HAVING

- From Sells(bar, beer, price) and Beers(name, manf), find the average price of those beers that are either served in at least three bars or are manufactured by Pete’s.

Solution

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

/*Beer groups with at least 3 non-NULL bars and also beer groups where the manufacturer is Pete’s.*/
```

Database Modifications

- A modification command does not return a result as a query does, but it changes the database in some way.
- There are three kinds of modifications:
  1. Insert a tuple or tuples.
  2. Delete a tuple or tuples.
  3. Update the value(s) of an existing tuple or tuples.

Insertion

- To insert a single tuple:
  ```sql
  INSERT INTO <relation>
  VALUES ( <list of values> );
  ```
- Example: add to Likes(drinker, beer) the fact that Sally likes Bud.
  ```sql
  INSERT INTO Likes
  VALUES(‘Sally’, ‘Bud’);
  ```

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes.
- There are two reasons to do so:
  1. We forget the standard order of attributes for the relation.
  2. We don’t have values for all attributes, and we want the system to fill in missing components with NULL or a default value.
Example: Specifying Attributes

- Another way to add the fact that Sally likes Bud to Likes(drinker, beer):

  INSERT INTO Likes(beer, drinker)
  VALUES('Bud', 'Sally');

Inserting Many Tuples

- We may insert the entire result of a query into a relation, using the form:

  INSERT INTO <relation>
  ( <subquery> );

Example: Insert a Subquery

- Using Frequents(drinker, bar), enter into the new relation PotBuddies(name) all of Sally’s “potential buddies,” i.e., those drinkers who frequent at least one bar that Sally also frequents.

Example: Deletion

- To delete tuples satisfying a condition from some relation:

  DELETE FROM <relation>
  WHERE <condition>;

Example: Deletion

- Delete from Likes(drinker, beer) the fact that Sally likes Bud:

  DELETE FROM Likes
  WHERE drinker = 'Sally' AND beer = 'Bud';
Example: Delete all Tuples

◆ Make the relation Likes empty:

DELETE FROM Likes;

◆ Note no WHERE clause needed.

Example: Delete Many Tuples

◆ Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

DELETE FROM Beers b
WHERE EXISTS (  
  )

Semantics of Deletion -- 1

◆ Suppose Anheuser-Busch makes only Bud and Bud Lite.
◆ Suppose we come to the tuple b for Bud first.
◆ The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
◆ Now, When b is the tuple for Bud Lite, do we delete that tuple too?

Semantics of Deletion -- 2

◆ The answer is that we do delete Bud Lite as well.
◆ The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied in the original relation.
  2. Delete the marked tuples.

Updates

◆ To change certain attributes in certain tuples of a relation:

UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;

Example: Update

◆ Change drinker Fred’s phone number to 555-1212:

UPDATE Drinkers
SET phone = ‘555-1212’
WHERE name = ‘Fred’;
Example: Update Several Tuples

◆ Make $4 the maximum price for beer:
  UPDATE Sells
  SET price = 4.00
  WHERE price > 4.00;