DM505 Database Design and Programming
DM576 Database Systems

Panagiotis Tampakis

ptampakis@imada.sdu.dk

Back to SQL

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

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:

```
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):

```
(SELECT drinker FROM Frequents)
   EXCEPT ALL
(SELECT drinker FROM Likes);
```

- Lists drinkers who frequent more bars than they like beers, and do so as many times as the difference of those counts

Ordering Results

- ORDER BY <list of attributes>
- ASC (Ascending - Default) or DESC (Descending)
- Example

```
SELECT bar, beer, price
FROM Sells
ORDER BY price DESC;
```

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

Only one of these

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 Odense Classic:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Od.Cl.';
```

Eliminating Duplicates in an Aggregation

- Use DISTINCT inside an aggregation
- Example: find the number of *different* prices charged for Bud:

```
SELECT COUNT(DISTINCT price)
FROM Sells
WHERE beer = 'Budweiser';
```

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-NULL values in a column, then the result of the aggregation is NULL
  - Exception: COUNT of an empty set is 0

Example: Effect of NULL's

```
SELECT count(*)
FROM Sells
WHERE beer = 'Od.Cl.';
```

The number of bars that sell Odense Classic

```
SELECT count(price)
FROM Sells
WHERE beer = 'Od.Cl.';
```

The number of bars that sell Odense Classic 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;
```

beer	AVG(price)
Od.Cl.	20
…	…

Example: Grouping

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

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

Compute all drinker-bar-bar-beer-price quintuples for Odense Cl.

Then group them 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 Odense Cl. the cheapest by:

```
SELECT bar, MIN(price)
FROM Sells
WHERE beer = ' Od.Cl.' ;
```

- But this query is illegal in SQL

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

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 Albani Bryggerierne

Solution

Beer groups with at least 3 non-NULL bars and also beer groups where the manufacturer is Albani.

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR
    beer IN (SELECT name
             FROM Beers
             WHERE manf = 'Albani');
```

Beers manu-factured by Albani.

Exercise

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

a) List all the manufacturers that make Laptops with a hard disk of at least 100 GB. Each manufacturer should appear once.
b) Find how many PC models each manufacturer makes. If a manufacturer doesn't make PC's he should appear in the result.
c) Find the screen size of laptops that has an average price of at least 10000 dkk.

Database Modifications

- A *modification* command does not return a result (as a query does), but changes the database in some way
- 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:

  INSERT INTO <relation>

  VALUES (<list of values>);

- Example: add to Likes(drinker, beer) the fact that Lars likes Odense Classic.

```
INSERT INTO Likes
VALUES('Lars', 'Od.Cl.');
```

Specifying Attributes in INSERT

- We may add to the relation name a list of attributes
- 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 Lars likes Odense Cl. to Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
VALUES('Od.Cl.', 'Lars');
```

Adding Default Values

- In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value
- When an inserted tuple has no value for that attribute, the default will be used

Example: Default Values

```
CREATE TABLE Drinkers (
   name CHAR(30) PRIMARY KEY,
   addr CHAR(50)
       DEFAULT 'Vestergade',
   phone CHAR(16)
);
```

Example: Default Values

```
INSERT INTO Drinkers(name)
VALUES('Lars');
```

Resulting tuple:

name	address	phone
Lars	Vestergade	NULL

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 Lars "potential buddies", i.e., those drinkers who frequent at least one bar that Lars also frequents

The other drinker

Solution

Pairs of Drinker tuples where the first is for Lars, the second is for someone else, and the bars are the same

```
INSERT INTO PotBuddies
(SELECT d2.drinker
 FROM Frequents d1, Frequents d2
 WHERE d1.drinker = 'Lars' AND
   d2.drinker <> 'Lars' AND
   d1.bar = d2.bar
);
```

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 Lars likes Odense Classic:

```
DELETE FROM Likes
WHERE drinker = 'Lars' AND
    beer = 'Od.Cl.';
```

Example: Delete all Tuples

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

Example: Delete Some Tuples

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

```
DELETE FROM Beers b
WHERE EXISTS (
    SELECT name FROM Beers
    WHERE manf = b.manf AND
        name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b

Semantics of Deletion

- Suppose Albani makes only Odense Classic and Eventyr
- Suppose we come to the tuple *b* for Odense Classic first
- The subquery is nonempty, because of the Eventyr tuple, so we delete Od.Cl.
- Now, when *b* is the tuple for Eventyr, do we delete that tuple too?

Semantics of Deletion

- Answer: we *do* delete Eventyr as well
- The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied
  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 Lars's phone number to 47 11 23 42:

```
UPDATE Drinkers
SET phone = '47 11 23 42'
WHERE name = 'Lars';
```

Example: Update Several Tuples

- Make 30 the maximum price for beer:

```
UPDATE Sells
SET price = 30
WHERE price > 30;
```

Summary 5

More things you should know:

- More joins
  - OUTER JOINS (FULL, LEFT and RIGHT)
- Aggregation
  - COUNT, SUM, AVG, MAX, MIN
  - GROUP BY, HAVING
- Database updates
  - INSERT, DELETE, UPDATE