Relational Database: Additional Operations on Relations; SQL

Greg Plaxton
Theory in Programming Practice, Spring 2005
Department of Computer Science
University of Texas at Austin
Overview

• The course packet describes several simple aggregation operators
  – The count operator
  – Operators for numerical data: sum, average, minimum, maximum

• The course packet also describes the notion of grouping in the context of aggregation

• We will give a brief introduction to SQL in this lecture
  – We will present examples of some key constructs
  – As part of this introduction, we will see how aggregation and grouping are handled in SQL
Table Manipulation

- SQL includes a number of basic commands creating, removing, and maintaining tables
  - CREATE TABLE: Create a table with the specified attributes
  - DROP TABLE: Remove a table
  - TRUNCATE TABLE: Remove the contents of a table, but not the table itself
  - INSERT INTO: Add specified rows to a table
  - UPDATE: Alter values within a table
  - DELETE FROM: Delete rows satisfying a specified condition
SELECT

• The SELECT command is arguably the most important command in SQL
  – It encompasses the selection, projection, and join operations that we discussed earlier in the context of relational algebra

• We will present a series of examples demonstrating some of the possible forms of a SELECT command
  – Our examples will assume the existence of a “parts” relation with attributes “partnum” (integer), “desc” (string), “price” (float), and “quantity” (integer).
  – We will follow the convention of capitalizing all the letters in SQL keywords
Projection via SELECT

- The following command performs a projection but retains duplicates:
  SELECT price, quantity FROM parts

- The following variation yields the projection operator we discussed earlier in the context of relational algebra:
  SELECT DISTINCT price, quantity FROM parts

- The asterisk symbol may be used as a shorthand to request all attributes:
  SELECT * FROM parts
A WHERE clause can be used to perform selection within a SELECT command:

- SELECT partnum, quantity FROM parts WHERE price > 10
- SELECT partnum, 2 * quantity FROM parts WHERE price = 10
- SELECT * FROM parts WHERE price > 10
  AND price * quantity <= 1000
- SELECT * FROM parts WHERE price BETWEEN 5 AND 10
- SELECT * FROM parts WHERE desc IN ('nut', 'bolt')
- SELECT * FROM parts WHERE desc LIKE '%nut%'
Specifying an Output Ordering

- An ORDER BY clause can be used to specify the output ordering
  - SELECT * FROM parts ORDER BY partnum
  - SELECT * FROM parts ORDER BY partnum DESC, price
  - SELECT partnum, price * quantity FROM parts ORDER BY 2
Aggregation

- The operators MIN, MAX, AVG, and SUM can be used to aggregate numerical values
  - `SELECT MIN(price) FROM parts`
  - `SELECT * FROM parts WHERE price > 0.25 * MAX(price)`

- The operator COUNT can be used on any type of data
  - `SELECT COUNT(desc) FROM parts`
  - `SELECT COUNT(DISTINCT desc) FROM parts WHERE price > 10`
Grouped Aggregation

- Sometimes we would like to partition the tuples of a relation table into groups based on the values of certain attributes, and then aggregate over the groups
  - SELECT desc, SUM(quantity) FROM parts GROUP BY desc
HAVING Clause

- A HAVING clause can be used to further filter the output of a grouped aggregation

  - SELECT desc, SUM(quantity) FROM parts GROUP BY desc
    HAVING SUM(quantity)>20
Column Aliasing

- We can rename an attribute of the output relation by aliasing
- SELECT partnum "num", price * quantity "total" FROM parts
Operations Involving Multiple Relations

• Up to this point we have discussed SELECT commands involving a single relation (parts)

• Suppose now that we have an orders database with attributes “customer” (string), “partnum” (integer), “quantity” (integer)
Cross Product

- The cross product of two relations may be obtained using a SELECT command
  - `SELECT * FROM parts, orders`
  - `SELECT customer, price, parts.quantity FROM parts, orders`

- The above examples are somewhat unnatural

- More typically, we perform some form of a join on such a pair of relations

- Remark: Complex queries involving multiple tables are sometimes easier to read if each attribute name is prefixed with the associated table name (e.g., “orders.customer” rather than “customer”)

Theory in Programming Practice, Plaxton, Spring 2005
Inner Join (a.k.a. Equijoin)

- Quite commonly, a WHERE clause containing an equality constraint is used to join two relations
  - SELECT customer, price FROM parts, orders
    WHERE parts.partnum = orders.partnum

- Here is an alternative way to express the same query
  - SELECT customer, price FROM parts JOIN orders
    ON parts.partnum = orders.partnum
Table Aliasing

- It is often convenient to alias the name of a table within an SQL command involving multiple tables
  
  ```sql
  SELECT b.customer, a.price FROM parts a, orders b
  WHERE a.partnum = b.partnum
  ```
Natural Join

- Natural join is also defined within SQL, along with other variants
- Note that natural join can also be accomplished using an inner join
- In the case of our parts/orders relations, natural join is somewhat “unnatural”; why?
Nested SELECT Commands

- The tables passed as arguments to a SELECT command can be produced by nested SELECT commands
- Arbitrary nesting is allowed
- Thus, a single SQL query can be quite complex