SQL Queries

Prepared By
Prof. Alok Haldar
Department of Computer Science
Kharagpur College

Ordered Results

- SQL query results can be ordered by particular attributes
- Two main categories of query results:
  - Not ordered by anything
    - Tuples can appear in *any* order
  - Ordered by attributes *A*1, *A*2, …
    - Tuples are sorted by specified attributes
    - Results are sorted by *A*1 first
    - Within each value of *A*1, results are sorted by *A*2
    - etc.
- Specify an **ORDER BY** clause at end of **SELECT**

Ordered Results

- **Find bank accounts with a balance under $700**

```
SELECT account_number, balance
 FROM account
 WHERE balance < 700  ;
```

```
+---------------------------------- +-----------------+
| account_number   | balance |
 +-------------------------------------------------------+
 |A-102                    | 400.00    |
 | A-101                   |  500.00  |
 | A-444                   |  625.00  |
 | A-305                   | 350.00   |
  +---------------------------------+-----------------+
```

- Order results in increasing order of bank balance:

```
SELECT account_number, balance
FROM account
WHERE balance < 700
ORDER BY balance;
```

- Default order is ascending order

```
+---------------------------------+----------------------+
| account_number | balance    |
 +--------------------------------+--------------------+
  | A-305                 | 350.00     |
 | A-102                  | 400.00     |
 | A-101                  | 500.00   |
 | A-444                  | 625.00   |
 +--------------------+------------+
```

Ordered Results

- Say **ASC** or **DESC** after attribute name to specify order

  ASC is redundant, but can improve readability in some cases
- Can list multiple attributes, each with its own order

  Retrieve a list of all bank branch details, ordered by branch city, with each city's branches listed in reverse order of holdings.

  SELECT * FROM branch

  ORDER BY branch_city ASC, assets DESC;

Aggregate Functions in SQL

- SQL provides grouping and aggregate operations, just like relational algebra
- Aggregate functions:

  SUM sums the values in the collection

  AVG computes average of values in the collection

  COUNT counts number of elements in the collection

  MIN returns minimum value in the collection

  MAX returns maximum value in the collection
- **SUM** and **AVG** require numeric inputs (obvious)

Aggregate Examples

- Find average balance of accounts at Perryridge branch

  SELECT AVG(balance) FROM account

  WHERE branch_name = 'Perryridge';

- Find maximum amount of any loan in the bank

  SELECT MAX(amount) AS max_amt FROM loan;

Aggregate Examples

- This query produces an error:
  SELECT branch_name, MAX(amount) AS max_amt
  FROM loan;
- Aggregate functions compute a *single value* from a multiset of inputs
  - Doesn't make sense to combine individual attributes and aggregate functions like this
- This does work:

```
SELECT MIN(amount) AS min_amt,
       MAX(amount) AS max_amt
FROM loan;
```

Eliminating Duplicates

- Sometimes need to eliminate duplicates in SQL queries
  - Can use **DISTINCT** keyword to eliminate duplicates
- Example:

  Find the number of branches that currently have loans.

  SELECT COUNT(branch_name) FROM loan;
  - Doesn't work, because branches may have multiple loans
  - Instead, do this:

  SELECT COUNT(DISTINCT branch_name) FROM loan;
  - Duplicates are eliminated from input multiset before aggregate function is applied

Computing Counts

- Can count individual attribute values
  COUNT(branch_name)
  COUNT(DISTINCT branch_name)
- Can also count the total number of tuples **COUNT(*)**
  - If used with grouping, counts total number of tuples in each group
  - If used without grouping, counts total number of tuples
- Counting a specific attribute is useful when:
  - Need to count (possibly distinct) values of a particular attribute
  - Cases where some values in input multiset may be **NULL**

    As before, **COUNT** ignores **NULL** values (more on this next week)

Grouping and Aggregates

- Can also perform grouping on a relation before computing aggregates
  - Specify a **GROUP BY A1,A2,...** clause at end of query
- Example:

  Find the average loan amount for each branch.

  SELECT branch_name, AVG(amount) AS avg_amt
  FROM loan GROUP BY branch_name;

  - First, tuples in **loan** are grouped by **branch_name**
  - Then, aggregate functions are applied to each group

Grouping and Aggregates

- Can group on multiple attributes
  - Each group has unique values for the entire set of grouping attributes
- Example:

  How many accounts does each customer have at each branch?
  - Group by both customer name *and* branch name
  - Compute count of tuples in each group
  - Can write the SQL statement yourself, and try it out

Grouping and Aggregates

- SQL syntax

  SELECT *G*1,*G*2,..., *F*1(*A*1),*F*2(*A*2),...

  FROM *r*1,*r*2,... WHERE P GROUP BY *G*1,*G*2,...

  - Frequently, grouping attributes are specified in both the **SELECT** clause and **GROUP BY** clause

Grouping and Aggregates

- SQL doesn't require that you specify the grouping attributes in the **SELECT** clause
  - Only requirement is that the grouping attributes are specified in the **GROUP BY** clause
  - e.g. if you only want the aggregated results, could do this:

    SELECT *F*1(*A*1),*F*2(*A*2),...

    FROM *r*1,*r*2,... WHERE P

    GROUP BY *G*1,*G*2,...
- Also, can use expressions for grouping and aggregates
  - Example (very uncommon, but also valid):

    SELECT MIN(a + b) – MAX(c)

    FROM t GROUP BY d * e;

Filtering Tuples

- The **WHERE** clause is applied *before* any grouping occurs

  SELECT *G*1,*G*2,..., *F*1(*A*1),*F*2(*A*2),...

  FROM *r*1,*r*2,... WHERE P

  GROUP BY *G*1,*G*2,...

  - A **WHERE** clause constrains the set of tuples that grouping and aggregation are applied to

Filtering Results

- To apply filtering to the results of grouping and aggregation, use a **HAVING** clause
  - Exactly like **WHERE** clause, except applied *after* grouping and aggregation

SELECT *G*1,*G*2,..., *F*1(*A*1),*F*2(*A*2),...
FROM *r*1,*r*2,... WHERE table
GROUP BY *G*1,*G*2,...
HAVING condition;

SQL continue …..

Thank You