CS 2451
Database Systems:
More SQL…

http://www.seas.gwu.edu/~bhagiweb/cs2541
Spring 2020
Instructor: Dr. Bhagi Narahari & R. Leontie

Based on slides © Ramakrishnan&Gerhke

This week….

- Today:
  - Wrap up basic SQL
    Aggregate functions on sets of tuples
    Max, sum,….
  - Operating on partitions of sets/relations
    GROUPBY
  - PHP+MySQL – lab exercise
- Next week….
  - How to design a good schema ?
  - 3-tier application design: Web + PHP + MySQL

Basic SQL and Relational Algebra

- The `SELECT` statement can be mapped directly to relational algebra.

- **SELECT** $A_1, A_2, \ldots, A_n$ */* this is projection*
- **FROM** $R_1, R_2, \ldots, R_m$ */* this is the selection op*
- **WHERE** P */* this is cartesian product*

- is equivalent to:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

SQL….review

- Select clause
  - Need to specify Join condition
- Concept of 'aliasing' to rename relation using AS keyword
  - Rename an attribute….in Select clause
- Tuple variables
- Nested queries
- Set operations:
  - Union
  - Set membership – IN, NOT IN
  - Existence of query results – EXISTS, NOT EXISTS
  - Compare values with values in set (generated by subquery)
    ALL, > ANY, <, >.....

What we've seen in SQL so far is equivalent in power to RA and TRC

Some MySQL goodies…

- INTO clause
  - Variation on aliasing
  - Pipe output of SELECT into another table
- SELECT in FROM clause…..use as derived table later in query

```
SELECT name
FROM deposit join
        (select name, custid
          from customer) dep-cust
ON deposit.custid= dep-cust.custid;
```

Derived table

- INNER JOIN
- LEFT (OUTER) JOIN, and RIGHT (OUTER) JOIN

Even more SQL….

- Functions on sets of tuples
  - Aggregate functions: Max, sum,….
- Operating on partitions of sets/relations
  - GROUPBY
- Update operations
- Security, Views, Transactions….not today
  - Maybe later!

SQL– Aggregate Operations

- Thus far SQL (and Relational Algebra/Calculus) only fetched data stored in database tables
- What if we need some basic ‘statistics’ on the data ?
  - Number of rows?
  - Maximum value in a field ?
- Aggregate Operators: apply a function to a set of tuples
  - Function defined on one (or more) field
  - Number of customers with loans
  - Average balance for a customer
  - Number of tuples in a relation
  - ……

Aggregate Operators

- Compute functions on set of tuples selected by where clause
  - Operate on a single column
- Semantics: if SELECT clause contains aggregate operations then it can contain *only* aggregate operations
  - Except when groupby construct is used
  - Functions on sets of values but result is single value
  - Average, minimum, maximum, sum, count(size)
- These functions operate on a single column of a table and return a single value.

Aggregate Operators

- Significant extension of relational algebra.

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

single column

Aggregate Functions

- The five basic aggregate functions are:
  - `COUNT` - returns the # of values in a column
  - `SUM` - returns the sum of the values in a column
  - `AVG` - returns the average of the values in a column
  - `MIN` - returns the smallest value in a column
  - `MAX` - returns the largest value in a column
- Notes:
  - 1) `COUNT`, `MAX`, and `MIN` apply to all types of fields, whereas `SUM` and `AVG` apply to only numeric fields.
  - 2) Except for `COUNT(*)` all functions ignore nulls. `COUNT(*)` returns the number of rows in the table.
  - 3) Use `DISTINCT` to eliminate duplicates.

Examples

Aggregate operators and computed columns
-arithmetic on column values

Purchase(product, date, price, quantity)

```
SELECT  Sum(price * quantity)
FROM    Purchase
```

What do they mean ?

```
SELECT  Sum(price * quantity)
FROM    Purchase
WHERE   product = 'bagel'
```

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM    Purchase
WHERE   product = 'bagel'
```

⇒ 50 (= 20+30)

Simple Aggregations

Purchase

Product	Date	Price	Quantity
Bagel	10/21	1	20
Banana	10/3	0.5	10
Banana	10/10	1	10
Bagel	10/25	1.50	20

```
SELECT  Sum(price * quantity)
FROM    Purchase
WHERE   product = 'bagel'
```

50 (= 20+30)

Aggregate Function Example

- Return the number of employees and their average salary.

```
SELECT COUNT(eno) AS numEmp, AVG(salary) AS avgSalary
FROM   emp
```

Result

numEmp	avgSalary
8	38750

Just a little bit more SQL…..

- Grouping
  - Operating on groups (partitions) of tuples

Motivation for Grouping

- So far, we've applied aggregate operators to all (qualifying) tuples. Sometimes, we want to apply them to each of several *groups* of tuples.
- Consider: *Find the average balance for each branch in the bank.*
  - In general, we don't know how many branches exist, and what the balances are!
  - Suppose we know that 10 branchnames exists; then we can write 10 queries that look like this (!):

For x = 1, 2, ... , 10:

```
SELECT  AVG(balance)
FROM  Deposit D
WHERE  D.branchname='x'
```

Oops…no For loops in SQL !!

GROUP BY Clause

- Aggregate functions are most useful when combined with the GROUP BY clause. The **GROUP BY** clause groups the tuples based on the values of the attributes specified.
- When used in combination with aggregate functions, the result is a table where each tuple consists of unique values for the group by attributes and the result of the aggregate functions applied to the tuples of that group.

Grouping and Aggregation

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product.

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > '10/1/2005'
GROUP BY  product
```

Let's see what this means…

Grouping and Aggregation

1. Compute the FROM and WHERE clauses.
2. Group by the attributes in the GROUPBY
3. Compute the SELECT clause: grouped attributes and aggregates.

1&2. FROM-WHERE-GROUPBY

Product	Date	Price	Quantity	
Bagel	10/21	1	20	Group1
Bagel	10/25	1.50	20	
Banana	10/3	0.5	10	
Banana	10/10	1	10	Group 2

3. SELECT

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

Product	TotalSales
Bagel	50
Banana	15

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > '10/1/2005'
GROUP BY  product
```

GROUP BY Example

- For each employee title, return the number of employees with that title, and the minimum, maximum, and average salary.

```
SELECT   title, COUNT(eno) AS numEmp,
         MIN(salary) as minSal,
         MAX(salary) as maxSal, AVG(salary) AS avgSal
FROM     emp
GROUP BY title
```

Result

title	numEmp	minSal	maxSal	avgSal
EE	2	30000	30000	30000
SA	3	50000	50000	50000
ME	2	40000	40000	40000
PR	1	20000	20000	20000

GROUP BY Clause Rules

- There are a few rules for using the `GROUP BY` clause:
  - 1) A column name cannot appear in the `SELECT` part of the query unless it is part of an aggregate function or in the list of group by attributes.
    Note that the reverse is allowed: a column can be in the `GROUP BY` without being in the `SELECT` part.
  - 2) Any `WHERE` conditions are applied before the `GROUP BY` and aggregate functions are calculated.

Condition on the Groups

- What if we are only in interested in groups that satisfy a condition ?

HAVING Clause

- The **HAVING** clause is applied ***AFTER*** the `GROUP BY` clause and aggregate functions are calculated.
- It is used to filter out entire *groups* that do not match certain criteria.
- The **HAVING** clause can contain any condition that references aggregate functions and the group by attributes themselves.
  - However, any conditions on the `GROUP BY` attributes should be specified in the `WHERE` clause if possible due to performance reasons.

Grouping and Aggregation: Evaluation Steps

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product.

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > '10/1/2005'
GROUP BY  product
```

Let's see what this means…

Grouping and Aggregation

Product	Date	Price	Quantity
Bagel	10/21	1	20
Bagel	10/25	1.50	20
Banana	10/3	0.5	10
Banana	10/10	1	10

Product	TotalSales
Bagel	50
Banana	15

What if we are only interested in products that sold quantity >30?

```
SELECT    product, Sum(price*quantity) AS TotalSales
FROM      Purchase
WHERE     date > '10/1/2005'
GROUP BY  product
```

HAVING Clause

Purchase(product, date, price, quantity)

Find total sales after 10/1/2005 per product, except that we consider only products that had at least 30 buyers.

```
SELECT    product, Sum(price * quantity)
FROM      Purchase
WHERE     date > '10/1/2005'
GROUP BY  product
HAVING    Sum(quantity) > 30
```

HAVING clause contains conditions on aggregates.

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Why ?

S = may contain attributes $a_1,\ldots,a_k$ and/or any aggregate function but NO OTHER ATTRIBUTES
C1 = is any condition on the attributes in $R_1,\ldots,R_n$
C2 = is any condition on aggregate expressions

General form of Grouping and Aggregation

SELECT S
FROM $R_1,\ldots,R_n$
WHERE C1
GROUP BY $a_1,\ldots,a_k$
HAVING C2

Evaluation steps:

1. Evaluate FROM-WHERE, apply condition C1
2. Group by the attributes $a_1,\ldots,a_k$
3. Apply condition C2 to each group (may have aggregates)
4. Compute aggregates in S and return the result

Generalized SELECT:Queries With GROUP BY and HAVING

SELECT [DISTINCT] *attribute-list*
FROM *relation-list*
WHERE *qualification/predicate*
GROUP BY *grouping-list*
HAVING *group-qualification/predicate*

- The *attribute-list* contains (i) attribute names (ii) terms with aggregate operations (e.g., MIN (balance)).
  - The attribute list must be a subset of *grouping-list*. Intuitively, each answer tuple corresponds to a *group*, and these attributes must have a single value per group. (A *group* is a set of tuples that have the same value for all attributes in *grouping-list*.)

Conceptual Evaluation

- The cross-product of *relation-list* is computed, tuples that fail *qualification* in WHERE clause are discarded, `*unnecessary*' fields are deleted, and the remaining tuples are partitioned into groups by the value of attributes in *grouping-list*.
- The *group-qualification* specified in the HAVING clause is then applied to eliminate some groups. Expressions in *HAVING* clause must have a *single value per group*!
  - In effect, an attribute in *group-qualification* that is not an argument of an aggregate op also appears in *grouping-list*. (SQL does not exploit primary key semantics here!)
- One answer tuple is generated per qualifying group.
- Any aggregate function can be applied to a group
  - Final SELECT can have function over each selected group

A quick Note: Group-by v.s. Nested Query

Author(login,name)

Wrote(login,url)

- Find authors who wrote ≥ 10 documents:
- Attempt 1: with nested queries

This is SQL by a novice

```
SELECT DISTINCT Author.name
FROM        Author
WHERE       count(SELECT Wrote.url
                  FROM Wrote
                  WHERE Author.login=Wrote.login)
                > 10
```

Group-by v.s. Nested Query

- Find all authors who wrote at least 10 documents:
- Attempt 2: SQL style (with GROUP BY)

This is SQL by an expert

```
SELECT      Author.name
FROM        Author, Wrote
WHERE       Author.login=Wrote.login
GROUP BY Author.name
HAVING      count(wrote.url) > 10
```

No need for DISTINCT: automatically from GROUP BY

….more SQL

- Finally, updates/modifications to database
- INSERT, DELETE and UPDATE can be result of queries!

INSERT

- Give all customers with a Loan at Downtown branch a $200 savings account with same account number as Loan number

```
INSERT INTO Deposit
SELECT CustID, Loan-number, Branch-name, 200
FROM Loan
WHERE branch-name = 'Downtown';
```

DELETE r
WHERE P
Predicate in P can be as complex as any select clause
Delete all accounts located in New York
DELETE Deposit
WHERE branchname in (SELECT branchname
FROM Branch
WHERE branchcity='New York');

How about this query ?

DELETE Deposit
WHERE balance < (SELECT avg(balance)
FROM Deposit);

Delete anomalies

- If delete/update request contains embedded select (sub-query) that references relation where deletions/update take place
- SQL standard disallows such requests
  - Alternate implementation: mark tuples in first round, and actual delete in second round

- INSERT
  - Can insert tuple with specified values
  - Can insert set of tuples resulting from query
- UPDATE
  - Change a value in tuple without changing all values in the tuple
  - Can update set of tuples by using query to select the set

Now, we are done…..kind of!

- More components to SQL:
- Views
- Constraints, Triggers
- …will get to these!

- Next....building 3-tier (full stack) application: PHP+MySQL