CSCI-GA.2433-001

Database Systems

Lecture 6: SQL II

Mohamed Zahran (aka Z)
mzahran@cs.nyu.edu
http://www.mzahran.com
Aggregate Operators

• Powerful class of constructs to calculate aggregate values.
• Useful when we want to do computations or summarizations on retrieved data.
• The aggregate operators can be applied on any column
Aggregate Operators

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

A column

More aggregate functions are added with every new standard
Aggregate Operators: Examples

Example: Find the average age of all sailors

```
SELECT AVG (S.age)
FROM   Sailors S
```

Example: Find the average age of all sailors with rating of 10

```
SELECT AVG (S.age)
FROM   Sailors S
WHERE  S.rating = 10
```
Aggregate Operators: Examples

Sailors (sid: integer, sname: string, rating: integer, age: real)
Boats (bid: integer, bname: string, color: string)
Reserves (sid: integer, bid: integer, day: date)

Q: Find the name and age of the oldest sailor

SELECT S.sname, MAX (S.age) 
FROM Sailors S 

Illegal in SQL!!
If the SELECT clause uses aggregate operation, it must ONLY use aggregate operations
Aggregate Operators: Examples

Q: Find the name and age of the oldest sailor

```
SELECT S.sname, S.age
FROM   Sailors S
WHERE  S.age = ( SELECT MAX (S2.age) 
                  FROM   Sailors S2 )
```
Aggregate Operators: Examples

Sailors (\texttt{sid}: integer, \texttt{sname}: string, \texttt{rating}: integer, \texttt{age}: real)
Boats (\texttt{bid}: integer, \texttt{bname}: string, \texttt{color}: string)
Reserves (\texttt{sid}: integer, \texttt{bid}: integer, \texttt{day}: date)

Q: Count the number of Sailors

\begin{verbatim}
SELECT COUNT (*)
FROM Sailors S
\end{verbatim}

Shorthand for all columns
Aggregate Operators: Examples

Sailors (sid: integer, sname: string, rating: integer, age: real)
Boats (bid: integer, bname: string, color: string)
Reserves (sid: integer, bid: integer, day: date)

Q: Count the number of distinct sailor names

SELECT COUNT ( DISTINCT S.sname )
FROM Sailors S

Without DISTINCT Count(*) is equivalent to Count (x) where x is any set of attributes. Can we prove/disprove this statement?
Q: Find the names of sailors who are older than the oldest sailor with a rating of 10

\[
\text{SELECT} \ S.\text{name} \\
\text{FROM} \ \text{Sailors} \ S \\
\text{WHERE} \ S.\text{age} > ( \ \text{SELECT} \ \text{MAX} ( \ S2.\text{age} ) \\
\text{FROM} \ \text{Sailors} \ S2 \\
\text{WHERE} \ S2.\text{rating} = 10 )
\]

OR

\[
\text{SELECT} \ S.\text{name} \\
\text{FROM} \ \text{Sailors} \ S \\
\text{WHERE} \ S.\text{age} > \text{ALL} ( \ \text{SELECT} \ S2.\text{age} \\
\text{FROM} \ \text{Sailors} \ S2 \\
\text{WHERE} \ S2.\text{rating} = 10 )
\]

Does ANY correspond to MIN or MAX?
Question: How can aggregates be expressed in relational algebra?

Answer: They cannot!
Groups

• Aggregate operations are applied to all (qualified) rows of a relation.

• What if ..
  – We want to apply aggregate operations to a number of groups in a relation
  – The number of groups is not known in advance (depends on the relation instance)
  – Example: Find the age of the youngest sailor for each rating level
Q: Find the age of the youngest sailor for each rating level

One possible solution:

SELECT MIN (S.age) 
FROM Sailors S 
WHERE S.rating = i

But

- Very tedious
- We do not know the levels in advance

It is obvious that we need an extension to SQL.
Extended SQL

Consists of:

- List of column names
- List of terms having the form: `aggregate_op (columns) AS new_name`

Must have a single value per group.
Q: Find the age of the youngest sailor for each rating level

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
GROUP BY S.rating
```
Q: Find the age of the youngest sailor who is eligible to vote (i.e. at least 18 years old) for each rating level with at least two such sailors.

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```
Q: Find the age of the youngest sailor who is eligible to vote (i.e. at least 18 years old) for each rating level with at least two such sailors.

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

<table>
<thead>
<tr>
<th>rating</th>
<th>minage</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>25.5</td>
</tr>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>

Important: The order of operations is significant!
Q: Find the age of the youngest sailor who is eligible to vote (i.e. at least 18 years old) for each rating level with at least two such sailors.

```
SELECT  S.rating, MIN (S.age) AS minage
FROM    Sailors S
WHERE   S.age >= 18
GROUP BY S.rating
HAVING  COUNT(*) > 1 AND EVERY ( S.age <= 60 )
```

<table>
<thead>
<tr>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>35.0</td>
</tr>
<tr>
<td>8</td>
<td>25.5</td>
</tr>
</tbody>
</table>

Important: The order of operations is significant!
What is the difference between:

```
SELECT S.rating, MIN(S.age) AS minage
FROM   Sailors S
WHERE  S.age >= 18 AND S.age <= 60
GROUP BY S.rating
HAVING COUNT(*) > 1
```

and

```
SELECT S.rating, MIN(S.age) AS minage
FROM   Sailors S
WHERE  S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1 AND EVERY ( S.age <= 60 )
```
Q: For each red boat, find the number of reservations for this boat.

SELECT B.bid, COUNT(*) AS reservationcount
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND B.color = 'red'
GROUP BY B.bid

Illegal! Only columns that appear in the GROUP BY clause can appear in the HAVING clause. Unless they appear as arguments to an aggregate operator in HAVING clause.
Q: Find the average age of sailors for each rating level that has at least two sailors.

Another solution:

\[
\text{SELECT} \quad \text{S.rating}, \quad \text{AVG}(\text{S.age}) \quad \text{AS} \quad \text{avgage} \\
\text{FROM} \quad \text{Sailors S} \\
\text{GROUP BY} \quad \text{S.rating} \\
\text{HAVING} \quad 1 < (\text{SELECT COUNT}(*) \\
\quad \text{FROM} \quad \text{Sailors S2} \\
\quad \text{WHERE} \quad \text{S.rating} = \text{S2.rating})
\]
Q: Find those ratings for which the average age of sailors is the minimum over all ratings.

```
SELECT S.rating
FROM   Sailors S
WHERE  AVG (S.age) = (
    SELECT MIN (AVG (S2.age))
    FROM   Sailors S2
    GROUP BY S2.rating )
```

The above statement is wrong and illegal!

```
SELECT Temp.rating, Temp.avgage
FROM   ( SELECT S.rating, AVG (S.age) AS avgage,
            FROM   Sailors S
            GROUP BY S.rating ) AS Temp
WHERE  Temp.avgage = ( SELECT MIN (Temp.avgage) FROM Temp )
```
Q: Find those ratings for which the average age of sailors is the minimum over all ratings.

What is the difference between this:

```
SELECT Temp.rating, Temp.avgage 
FROM   ( SELECT     S.rating, AVG (S.age) AS avgage, 
             FROM     Sailors S 
             GROUP BY S.rating) AS Temp 
WHERE   Temp.avgage = ( SELECT MIN (Temp.avgage) FROM Temp) 
GROUP BY Temp.rating
```

And this:

```
SELECT Temp.rating, MIN (Temp.avgage ) 
FROM   ( SELECT     S.rating, AVG (S.age) AS avgage, 
             FROM     Sailors S 
             GROUP BY S.rating) AS Temp
```
Question: How can grouping constructs be expressed in relational algebra?

Answer: They cannot!
Dealing with NULL

- Comparison with NULL using $>, <, \leq, \geq$ is unknown
- SQL provides:
  - IS NULL
  - IS NOT NULL
- With AND, OR, and NOT things are more complicated: True, False, Unknown
  - 3-values logic
Dealing with NULL

<table>
<thead>
<tr>
<th>AND</th>
<th>AND</th>
<th>AND</th>
<th>OR</th>
<th>OR</th>
<th>OR</th>
<th>NOT</th>
<th>NOT</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T/F</td>
<td>T/F</td>
<td>T</td>
<td>F</td>
<td>T/F</td>
<td>T</td>
<td>F</td>
<td>U</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>U</td>
<td>T/F</td>
<td>F</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>U</td>
<td>T</td>
<td>F</td>
<td>U</td>
<td>F</td>
<td>T</td>
<td>U</td>
</tr>
</tbody>
</table>

**OUTCOME**

- T: TRUE
- F: FALSE
- U: Unknown

- In evaluating SQL queries, any row that evaluates to False or Unknown is eliminated.
- Two rows are duplicates if the corresponding columns are either equal or both unknown.
- Arithmetic operations all return NULL if one of the arguments is NULL.
- Be very careful with aggregates.
Integrity Constraints Revisited

• Definition: describes conditions that every legal instance of a relation must satisfy

• Why?
  – Ensure application semantics
  – prevent inconsistencies

• Types:
  – Constraints over single table
  – Domain constraints
  – Constraints over multiple tables
IC Over Single Table

Format: CHECK conditional-expression

CREATE TABLE  Sailors
  ( sid INTEGER,
    sname CHAR(10),
    rating INTEGER,
    age  REAL,
    PRIMARY KEY  (sid),
    CHECK ( rating >= 1  AND rating <= 10 )
IC Over Single Table

**Format:** CHECK *conditional-expression*

CREATE TABLE Reserves
(sname CHAR(10),
bid INTEGER,
day DATE,
PRIMARY KEY (bid,day),
CONSTRAINT noInterlakeRes
CHECK (‘Interlake’ <>
(SELECT B.bname
FROM Boats B
WHERE B.bid=bid)))

When are IC checked?

Constraints can be names
Can use queries to express constraints.
IC: Domains

User can define a new domain
Adding more restrictions
Refers to a value in the domain
In case a new tuple is inserted and no value is specified for the attribute of that domain.

```
CREATE DOMAIN ratingval INTEGER DEFAULT 1
   CHECK ( VALUE >= 1 AND VALUE <= 10 )
```

Once the domain is defined, it can be used in a table definition. Example: `rating ratingval`
IC: Domains Vs Types

CREATE TYPE ratingtype AS INTEGER

• Creating a new type (not domain)
• This type is distinct from the base type “integer”.
• So a comparison of attribute of type ratingtype and integer will fail.
IC Over Several Tables

• **ASSERTIONS**: constraints not associated with anyone table.

CREATE TABLE Sailors
  ( sid INTEGER,
    sname CHAR(10),
    rating INTEGER,
    age REAL,
    PRIMARY KEY (sid),
    CHECK
      ( (SELECT COUNT (S.sid) FROM Sailors S)
        + (SELECT COUNT (B.bid) FROM Boats B) < 100 )

What do you think about this?

| Number of boats plus number of sailors is < 100 |

**Drawback:** If sailors is empty the constraint will hold even if we have more than 100 rows in boats!!
IC Over Several Tables

• **ASSERTIONS**: constraints not associated with anyone table.

Better way to do it:

```sql
CREATE ASSERTION smallClub
CHECK (( SELECT COUNT (S.sid) FROM Sailors S )
+ ( SELECT COUNT (B.bid) FROM Boats B )
< 100 )
```
Triggers

**Definition:** procedure that starts automatically if specified changes occur to the DB

- **Condition** (tests whether the triggers should run)
  - Can be a true/false statement or a query.
  - Empty result to a query is interpreted as false.

- **Event** (activates the trigger)

A database that has a set of associated triggers is called an **Active DATABASE**.
CREATE TRIGGER selcount AFTER INSERT ON Students REFERENCING NEW TABLE AS InsertedTuples
FOR EACH STATEMENT
    INSERT INTO StatisticsTable(ModifiedTable, ModificationType, Count)
    SELECT 'Students', 'Insert', COUNT *
    FROM InsertedTuples I
    WHERE 1.age < 18
Challenges in Triggers

• Can have a ripple effect (i.e. chain activations)
• What to do when a statement activates more than one trigger?
• Should we just use constraints?
SQL

Data Definition
- Defining the schema
- Integrity constraints
- Assertion

Query
- Basic query
- UNION, INTERSECT, EXCEPT
- Nested Queries
- Aggregate operators
- Triggers
Conclusions

• SQL was an important factor in the early acceptance of the relational model.

• Queries that can be expressed in RA can often be expressed more naturally in SQL.

• In practice, users need to be aware of how queries are optimized and evaluated for best results.