

CS143: SQL Query (2)

Book Chapters

(5th) Chapter 3.5-8, 3.11, 4.7-8

(6th) Chapter 3.6, 3.7, 3.9, 4.1, 5.4-5

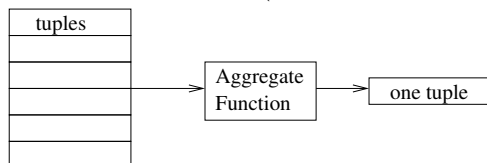
(7th) Chapter 3.6, 3.7, 3.9, 4.1, 5.4-5

Things to Learn

- Aggregate function
- Window function
- Case function
- ORDER BY and FETCH FIRST
- SQL data modifications
- Null and three-valued logic
- Outer join
- Bag semantics
- SQL expressive power

Aggregates

- The operators so far check the condition “tuple-by-tuple”
- They never “summarize” multiple tuples into one.
For example, 'SUM', 'AVG' of GPA is not possible.
- Aggregate function (aggregate diagram)



- **Query 1:** Find the average GPA

- Common aggregate functions: SUM, AVG, COUNT, MIN, MAX on single attribute or COUNT(*).

Problems of Duplicates

- **Query 2:** The number of students taking CS classes

- **Query 3:** The average GPA of the students taking CS classes

GROUP BY clause

- Sometimes, we want to get separate statistics for each group of tuples

Example:

Age	AVG(GPA)
17	3.7
19	2.1
20	3.1

But AVG() takes average over *all* tuples.

- **Query 4:** Find the average GPA for each age group

Q: Is the following query meaningful?

```
SELECT sid, age, AVG(GPA)
FROM Student
GROUP BY age
```

- SELECT can have only attributes that have a single value in each group or *aggregates*

- **Query 5:** Find the number of classes each student is taking

Q: What about the students who take no classes?

Comments: We will learn about outer join that can address this issue later.

HAVING clause

- **Query 6:** Find students who take two or more classes

- Conditions on aggregates should appear in the HAVING clause.

Q: Can we rewrite the query without HAVING clause?

- In general, we can rewrite a query not to have a HAVING clause.

Window Function

- **Query 7:** Per each result row, return a student's name, their GPA, and the overall GPA average
 - Q: Will this work?

```
SELECT name, GPA, AVG(GPA) FROM Student
```

- *Window function:*

- Syntax: `FTN() OVER()`
 - * Append `OVER()` to convert an aggregate function to a window function
- Introduced in SQL 2003
- Aggregate function merges all input tuples into a *single* output tuple
- Window function generates *one output tuple per each input tuple*, but the function is computed over all input tuples
- PARTITION BY:
 - **Query 8:** Per each result row, return a student's name, their GPA, and the average GPA within the student's age group

- `OVER(PARTITION BY attr)`
- With PARTITION BY, window function is applied only within the same partition

Case Function

- Limited support of if-then-else
 - Return different values depending on conditions
- Syntax: CASE


```

      WHEN <condition> THEN <expr>
      WHEN <contidion> THEN <expr>
      ELSE <expr>
      
```

END
- Can be used anywhere a column name can be referenced
 - SELECT, WHERE, GROUP BY, ...
- **Query 9:** Average GPA of the child vs adult group

- Q: What if we want to show “child” and “adult” as part of the output?
- Q: What if we want to return two columns, “childGPA” and “adultGPA”?

ORDER BY clause

- Sometimes we may want to display tuples in a certain order. For example order all students by their GPA
- ```
SELECT sid, GPA
FROM Student
ORDER BY GPA DESC, sid ASC
```

  - All students and GPAs, in the descending order of their GPAs and the ascending order of sids. Default is ASC if omitted.
  - Does not change SQL semantics. Just makes the display easier to look at and understand

## FETCH FIRST clause

- **Query 10:** Top-3 students ordered by GPA
  - Sometimes, we just want a few rows from the result. Is there a way to limit result size?
- SQL 2008 Syntax: 

```
[OFFSET <offset> ROWS] FETCH FIRST <count> ROWS ONLY
```

  - From the result, skip first *offset* rows and return the subsequent *count* rows

- Unfortunately, this was standardized only in SQL 2008. Many systems use their own syntax, including MySQL.
- Variations:
  - MySQL: `LIMIT <count> OFFSET <offset>`
  - Oracle used to use `rownum`, DB2 used to use `SELECT TOP`, but they both support `FETCH FIRST` now
  - MS SQL server requires `ORDER BY` clause *and* `OFFSET` to use `FETCH FIRST`

## General SQL SELECT statement

- SELECT attributes, aggregates  
 FROM relations  
 WHERE conditions  
 GROUP BY attributes  
 HAVING conditions on aggregates  
 ORDER BY attributes, aggregates  
 FETCH FIRST n ROWS ONLY
- Evaluation order: FROM → WHERE → GROUP BY → HAVING → ORDER BY → FETCH FIRST → SELECT

## Data Modification in SQL (INSERT/DELETE/UPDATE)

- **Insertion:** INSERT INTO *Relation* *Tuples*

- Q: Insert tuple (301, CS, 201, 01) to Enroll?
- Q: Populate Honors table with students of GPA > 3.7?

- **Deletion:** DELETE FROM *R* WHERE *Condition*

- Q: Delete all students who are not taking classes

- **Update:** Update *R*

SET  $A1 = V1, A2 = V2, \dots, An = Vn$   
WHERE *Condition*

- Q: Increase all CS course numbers by 100

## More Advanced SQL

We now go over a bit more esoteric yet important details of SQL

### NULL and Three-valued logic

- **Arithmetic operators and comparison**

**Q:** `SELECT name`  
`FROM Student`  
`WHERE GPA * 100/4 > 90`  
What should we do if GPA is NULL?

– **Q:** What should be the value for `GPA * 100/4`?

– Rule: Arithmetic operators with NULL input returns NULL

– **Q:** What should be `NULL > 90`?

– Rule: Arithmetic comparison with NULL value return **Unknown**

- \* SQL is **Three-valued logic**: True, False, Unknown

- \* SQL returns only **True** tuples

- \* `GPA * 100/4 > 90` does not return a tuple if GPA is NULL

- **Three-valued logic**

– **Q:** `GPA > 3.7 AND age > 18`. What if GPA is NULL and age < 18?

– **Q:** `GPA > 3.7 OR age > 18`. What if GPA is NULL and age < 18?



- Truth table
  - \* AND:  $U \text{ AND } T = U$ ,  $U \text{ AND } F = F$ ,  $U \text{ AND } U = U$
  - \* OR:  $U \text{ OR } T = T$ ,  $U \text{ OR } F = U$ ,  $U \text{ OR } U = U$
- NOT Unknwon = Unknown. It's not known
- **SQL returns only True tuples**

- **Aggregates**

|      |    |      |                                  |
|------|----|------|----------------------------------|
| – Q: | ID | GPA  | SELECT AVG(GPA)                  |
|      | 1  | 3.0  | FROM Student                     |
|      | 2  | 3.6  | What should be the result?       |
|      | 3  | 2.4  | What about COUNT(*)? COUNT(GPA)? |
|      | 4  | NULL |                                  |

- Rule: Aggregates are computed ignoring NULL value, except COUNT(\*).
- \* Too much information is lost otherwise.
- \* COUNT(\*) considers a NULL tuple as a valid tuple
- \* When the input to an aggregate is empty, COUNT returns 0; all others return NULL.

- **Set operators** ( $\cup, \cap, -$ )

- Q: What should be  $\{2.4, 3.0, \text{NULL}\} \cup \{3.6, \text{NULL}\}$ ?

- Rule: NULL is treated like other values in set operators

- **Checking NULL**

- IS NULL or IS NOT NULL to check if the value is null.

- **COALESCE() function**

- Return first non-NULL value in the list
- Example: COALESCE(phone, email, addr)

## OUTER join

- Q: How many classes does each student take?
  - Q: What about student 208, Esther? What should we print? What is the problem?
  - Q: Anyway to preserve dangling tuples?
- OUTER JOIN operator in FROM clause:
  - R LEFT OUTER JOIN S ON R.A = S.A
    - \* Keep all dangling tuples from R by padding S attributes with NULL.
  - R RIGHT OUTER JOIN S ON R.A = S.A
    - \* keep all dangling tuples from S by padding R attributes with NULL
  - R FULL OUTER JOIN S ON R.A = S.A
    - \* keep all dangling tuples both from R and S with appropriate padding
- Q: How to rewrite the above query to include Esther?

## SQL and bag semantics

- What is a bag (multiset)?
  - A set with duplicate elements
  - Order does not matter
  - **Example:**  $\{a, a, b, c\} = \{a, c, b, a\} \neq \{a, b, c\}$
- SQL and bag semantics
  - Default SQL statements are based on bag semantics
    - \* We already learned the bag semantics
    - \* **Except set operators** (UNION, INTERSECT, EXCEPT), which use set semantics
  - We can enforce set semantics by using DISTINCT keyword
- Bag semantics for set operators
  - UNION ALL, INTERSECT ALL, EXCEPT ALL
    - \* MySQL supports only UNION ALL
  - **Q:**  $\{a, a, b\} \cup \{a, b, c\}$ ?
  - **Q:**  $\{a, a, a, b, c\} \cap \{a, a, b\}$ ?
  - **Q:**  $\{a, a, b, b\} - \{a, b, b, c\}$ ?
- What rules still hold for Bag?
  - **Q:** Under bag semantics,  $R \cup S = S \cup R$ ?  $R \cap S = S \cap R$ ?  
 $R \cap (S \cup T) = (R \cap S) \cup (R \cap T)$ ?
    - \* Under bag semantics, some rules still hold, some do not
    - \* Consider,  $R = \{a\}, S = \{a\}, T = \{a\}$  to check the distributive rule.

## Expressive power of SQL

- **Example:** All ancestors

| child | parent |
|-------|--------|
| Susan | John   |
| John  | James  |
| James | Elaine |
| ...   | ...    |

- **Q:** Can we find all ancestors of Susan using SQL?

- **Example:** All reachable destination

| city 1 | city 2 |
|--------|--------|
| A      | B      |
| B      | D      |
| A      | C      |
| E      | F      |
| G      | H      |
| ...    | ...    |

- **Q:** Find all cities reachable from A?

- **Comments:** SQL92 does not support “recursion” and thus cannot compute the *transitive closure*.

- Recursion is supported in SQL1999.

- WITH RECURSIVE R(A1, A2) AS ...

```
WITH RECURSIVE Ancestor(child, ancestor) AS (
 (SELECT child, parent AS ancestor FROM Parent)
 UNION
 (SELECT P.child, A.ancestor
 FROM Parent P, Ancestor A
 WHERE P.parent = A.child))
SELECT * FROM Ancestor
```

- MySQL introduced support for recursive common table expression in v8.0