CS 4604: Introduction to Database Management Systems

SQL III

Virginia Tech CS 4604 Sprint 2021
Instructor: Yinlin Chen
Today’s Topics

• SQL Statements (Continue)
More on Set-Comparison Operators

• The comparison condition $v > \text{ALL } V$ returns TRUE if the value $v$ is greater than all the values in the multiset $V$.
  – If the nested query doesn’t return a value, it evaluates the condition as \text{TRUE}.

• The comparison condition $v > \text{ANY } V$ returns TRUE if the value $v$ is greater than at least one value in the multiset $V$.
  – If the nested query doesn’t return a value, it evaluates the whole condition as \text{FALSE}.
Queries with ALL/ANY

Q42: Find sailors whose rating is greater than that of some sailor called Popeye:

```
SELECT *
FROM Sailors S
WHERE S.rating > ANY
  (SELECT S2.rating
   FROM Sailors S2
   WHERE S2.sname='Popeye')
```
Queries with ALL/ANY

```
SELECT *
FROM   Sailors S
WHERE  S.rating > ALL
       (SELECT S2.rating
        FROM   Sailors S2)
```
Division

- Relational Division: “Find sailors who’ve reserved all boats.”
  
  Said differently: “sailors with no counterexample missing boats”

```
SELECT S.sname FROM Sailors S
WHERE NOT EXISTS
  (SELECT B.bid FROM Boats B
   WHERE NOT EXISTS
     (SELECT R.bid FROM Reserves R
      WHERE R.bid=B.bid
      AND R.sid=S.sid ))
```
What we have so far

• Joins
• Nested Queries
• ALL, ANY, MAX(), etc.
Example

• Find the sailor with the highest rating

<table>
<thead>
<tr>
<th>sid</th>
<th>sname</th>
<th>rating</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Popeye</td>
<td>10</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>OliveOyl</td>
<td>11</td>
<td>39</td>
</tr>
<tr>
<td>3</td>
<td>Garfield</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td>4</td>
<td>Bob</td>
<td>5</td>
<td>19</td>
</tr>
</tbody>
</table>
Example

- Find the sailor with the highest rating

\[
\text{SELECT} \quad \text{MAX}(S.\text{rating}) \\
\text{FROM} \quad \text{Sailors} \quad S;
\]

\[
\text{SELECT} \quad \text{MAX}(S.\text{rating}) \\
\text{FROM} \quad \text{Sailors} \quad S;
\]

\[
\text{VS}
\]

\[
\text{SELECT} \quad \text{S}.\ast, \quad \text{MAX}(S.\text{rating}) \\
\text{FROM} \quad \text{Sailors} \quad S;
\]

\[
\text{SELECT} \quad \text{S}.\ast, \quad \text{MAX}(S.\text{rating}) \\
\text{FROM} \quad \text{Sailors} \quad S;
\]

\[
\text{VS}
\]

\[
\text{SELECT} \quad \ast \\
\text{FROM} \quad \text{Sailors} \quad S \\
\text{WHERE} \quad S.\text{rating} \geq \text{ALL} \\
(\text{SELECT} \quad S2.\text{rating} \\
\text{FROM} \quad \text{Sailors} \quad S2)
\]

\[
\text{SELECT} \quad \ast \\
\text{FROM} \quad \text{Sailors} \quad S \\
\text{WHERE} \quad S.\text{rating} = \\
(\text{SELECT} \quad \text{MAX}(S2.\text{rating}) \\
\text{FROM} \quad \text{Sailors} \quad S2)
\]

\[
\text{SELECT} \quad \ast \\
\text{FROM} \quad \text{Sailors} \quad S \\
\text{ORDER BY} \quad \text{rating} \quad \text{DESC} \\
\text{LIMIT} \quad 1;
\]

\[
\text{SELECT} \quad \ast \\
\text{FROM} \quad \text{Sailors} \quad S \\
\text{ORDER BY} \quad \text{rating} \quad \text{DESC} \\
\text{LIMIT} \quad 1;
\]
Queries with Subqueries in SELECT/FROM

Q46: SELECT P.PRODNR, P.PRODNAME, 
    (SELECT SUM(QUANTITY) FROM PO_LINE POL 
    WHERE P.PRODNR = POL.PRODNR) AS TOTALORDERED 
FROM PRODUCT P

Q47: SELECT M.PRODNR, M.MINPRICE, M.MAXPRICE FROM 
    (SELECT PRODNR, MIN(PURCHASE_PRICE) AS MINPRICE, 
    MAX(PURCHASE_PRICE) AS MAXPRICE 
    FROM SUPPLIES GROUP BY PRODNR) AS M 
WHERE M.MAXPRICE - M.MINPRICE > 1
Set Semantics

- Set: a collection of distinct elements
- Standard ways of manipulating/combining sets
  - Union
  - Intersect
  - Except
- Treat tuples within a relation as elements of a set
Default: Set Semantics

- These are relations. They are not sets, since they have duplicates.

\[ R = \{A, A, A, A, B, B, C, D\} \]
\[ S = \{A, A, B, B, B, C, E\} \]

- **UNION**
  \[ \{A, B, C, D, E\} \]

- **INTERSECT**
  \[ \{A, B, C\} \]

- **EXCEPT**
  \[ \{D\} \]

- \[ A = \{10, 5, 25, 30, 45\} \]
- \[ B = \{15, 20, 10, 30, 50\} \]
- \[ A \cup B = \{5, 10, 15, 20, 25, 30, 45, 50\} \]
- \[ A \cap B = \{10, 30\} \]
- \[ A \setminus B = \{5, 25, 45\} \]
UNION vs UNION ALL

• The UNION operator is used to combine the result-set of two or more SELECT statements.
  – Each SELECT statement within UNION must have the same number of columns
  – The columns must also have similar data types
  – The columns in each SELECT statement must also be in the same order

• The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL
Example: UNION ALL

- Sid’s of sailors who reserved a red OR a green boat

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND
  (B.color = 'red' OR
   B.color = 'green')
```

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND
  B.color = 'red'

UNION ALL

SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND
  B.color = 'green'
```
**EXCEPT vs EXCEPT ALL**

- The EXCEPT operator returns distinct rows from the first (left) query that are not in the output of the second (right) query
  - The number of columns and their orders must be the same in the two queries
  - The data types of the respective columns must be **compatible**
- With ALL, a row that has $m$ duplicates in the left table and $n$ duplicates in the right table will appear $\max(m-n,0)$ times in the result set


**EXCEPT ALL:** \{A, A, D\}
Example: Except

- Find sailors who have **not** reserved a boat

SELECT S.sid
FROM   Sailors S

EXCEPT

SELECT S.sid
FROM   Sailors S, Reserves R
WHERE  S.sid=R.sid
INTERSECT vs INTERSECT ALL

- The INTERSECT operator returns any rows that are available in both result sets
  - The number of columns and their order in the SELECT clauses must be the same
  - The data types of the columns must be compatible
- With ALL, min of cardinalities


INTERSECT ALL: \{A, A, B, B, C\}
Example: Intersect

- Sid’s of sailors who reserved a red AND a green boat

```
SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND B.color = 'red'
INTERSECT

SELECT R.sid
FROM Boats B, Reserves R
WHERE R.bid = B.bid AND B.color = 'green'
```
Let’s Do Labs

• [https://github.com/VTCourses/CS4604_Labs](https://github.com/VTCourses/CS4604_Labs)
• Lab3: [3.more_queries](https://github.com/VTCourses/CS4604_Labs/3.more_queries)
SQL Views

- SQL views are part of the external data model
- A view is defined by a query over other relations (tables and/or views)
- A view is a virtual table that does not exist physically
- A view can be
  - Queried: the query processor replaces the view by its definition.
  - Used in other queries.
- Views allow for logical data independence which makes them a key component in the three-layer database architecture
Views: Named Queries

CREATE VIEW view_name
AS select_statement

CREATE VIEW Redcount

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

CREATE VIEW TOPSUPPLIERS
AS SELECT SUPNR, SUPNAME FROM SUPPLIER
WHERE SUPSTATUS > 50

CREATE VIEW TOPSUPPLIERS_SF
AS SELECT * FROM TOPSUPPLIERS
WHERE SUPCITY = 'San Francisco'
CREATE VIEW ORDEROVERVIEW
    (PRODNR, PRODNAME, TOTQUANTITY)
    AS
    SELECT P.PRODNR, P.PRODNAME, SUM(POL.QUANTITY)
    FROM PRODUCT AS P
    LEFT OUTER JOIN PO_LINE AS POL
    ON (P.PRODNR = POL.PRODNR)
    GROUP BY P.PRODNR
**SQL Views**

```
SELECT * FROM TOPSUPPLIERS_SF
SELECT * FROM redcount;
SELECT * FROM ORDEROVERVIEW WHERE PRODNAME LIKE '%CHARD%'
```

```
SELECT bname, scount
FROM Boats B,
 (SELECT B.bid, COUNT(*)
  FROM Boats B, Reserves R
  WHERE R.bid = B.bid AND B.color = 'red'
  GROUP BY B.bid) AS Reds(bid, scount)
WHERE Reds.bid = B.bid AND scount < 10
```

```
SELECT bname, scount
FROM Redcount R,
Boats B
WHERE R.bid = B.bid
AND scount < 10;
```
WITH Queries (Common Table Expressions)

- MySQL 8.0 finally support it

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

SELECT bname, scount
FROM Boats B, Reds
WHERE Reds.bid = B.bid
AND scount < 10
WITH Queries (Common Table Expressions)

- Can have many queries in WITH

WITH Reds (bid, scount) AS
  (SELECT B.bid, COUNT(*)
   FROM Boats B, Reserves R
   WHERE R.bid = B.bid AND
   B.color = 'red'
   GROUP BY B.bid),
  SELECT * FROM UnpopularReds;

UnpopularReds AS
  (SELECT bname, scount
   FROM Boats B, Reds
   WHERE Reds.bid=B.bid
   AND scount < 10)
ARGMAX GROUP BY

- The sailor with the highest rating per age

WITH maxratings(age, maxrating) AS
  (SELECT age, max(rating)
   FROM Sailors
   GROUP BY age)

SELECT S.*
FROM Sailors S, maxratings m
WHERE S.age = m.age
  AND S.rating = m.maxrating;
SQL Views

• Query modification: RDBMS modifies queries that query views into queries on the underlying base tables

• View **materialization**: a physical table is created when the view is first queried

• Unlike a table, a view cannot be updated unless it satisfies certain conditions
  — In this case, the view serves as a window through which updates are propagated to the underlying base table(s)
CREATE VIEW ORDEROVERVIEW(PRODNR, PRODNAME, TOTQUANTITY) AS SELECT P.PRODNR, P.PRODNAME, SUM(POL.QUANTITY) FROM PRODUCT AS P LEFT OUTER JOIN PO_LINE AS POL ON (P.PRODNR = POL.PRODNR) GROUP BY P.PRODNR

UPDATE VIEW ORDEROVERVIEW SET TOTQUANTITY=10 WHERE PRODNR= '0154'
SQL Views

• WITH CHECK option checks UPDATE and INSERT statements for conformity with the view definition

```
CREATE VIEW TOPSUPPLIERS
AS SELECT SUPNR, SUPNAME FROM SUPPLIER
WHERE SUPSTATUS > 50 WITH CHECK OPTION

UPDATE TOPSUPPLIERS
SET SUPSTATUS = 20
WHERE SUPNR = '32'

INSERT INTO TOPSUPPLIERS VALUES (12, 'new supplier');
```

```
UPDATE TOPSUPPLIERS
SET SUPSTATUS = 80
WHERE SUPNR = '32'

OK!

NOT OK!
```

```
Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given YoungStudents, but not Students or Enrolled, we can find students who are enrolled, but not the cid’s of the courses they are enrolled in.
Delete VIEW

- DROP VIEW TOPSUPPLIERS;
- Like a Symbolic Link: only the view definition is deleted
- delete from viewtest where sid = 11;  ?  NOT OK!
Null Values

- Field values are sometimes unknown
  - SQL provides a special value NULL for such situations.
  - Every data type can be NULL
- The presence of null complicates many issues. E.g.:
  - Selection predicates (WHERE)
  - Aggregation
- But NULLs comes naturally from Outer joins
NULL in the WHERE clause

• Consider a tuple where rating IS NULL.

INSERT INTO sailors VALUES
 (11, 'Jack Sparrow', NULL, 35);

SELECT * FROM sailors
WHERE rating > 8;

Is Jack Sparrow in the output?
NULL in Comparators

- Rule: \((x \text{ op \ NULL})\) evaluates to \ldots \ NULL!

  SELECT 100 = NULL;
  SELECT 100 < NULL;
  SELECT 100 >= NULL;

  SELECT * FROM sailors WHERE rating IS NULL;
  SELECT * FROM sailors WHERE rating IS NOT NULL;
NULL in Boolean Logic

Three-valued logic:

<table>
<thead>
<tr>
<th>NOT</th>
<th>T</th>
<th>F</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>T</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AND</th>
<th>T</th>
<th>F</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>N</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>T</th>
<th>F</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

General rule: NULL column values are ignored by aggregate functions
NULL and Aggregation

```sql
SELECT count(*) FROM sailors;
SELECT count(rating) FROM sailors;
SELECT sum(rating) FROM sailors;
SELECT avg(rating) FROM sailors;
```

General rule: NULL **column values** are ignored by aggregate functions
NULLs: Summary

- NULL op NULL is NULL
- WHERE NULL: do not send to output
- Boolean connectives: 3-valued logic
- Aggregates ignore NULL-valued inputs
SQL Privileges

- A privilege corresponds to the right to use certain SQL statements such as SELECT, INSERT, etc. on one or more database objects.

<table>
<thead>
<tr>
<th>Privilege</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT</td>
<td>Gives retrieval privilege</td>
</tr>
<tr>
<td>INSERT</td>
<td>Gives insert privilege</td>
</tr>
<tr>
<td>UPDATE</td>
<td>Gives update privilege</td>
</tr>
<tr>
<td>DELETE</td>
<td>Gives delete privilege</td>
</tr>
<tr>
<td>ALTER</td>
<td>Gives privilege to change the table definition</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>Gives the privilege to reference the table when specifying integrity constraints</td>
</tr>
<tr>
<td>ALL</td>
<td>Gives all privileges (DBMS specific)</td>
</tr>
</tbody>
</table>
**SQL Privileges**

**GRANT** SELECT, INSERT, UPDATE, DELETE **ON** SUPPLIER **TO** BBAESENS

**GRANT** SELECT (PRODNR, PRODNAME) **ON** PRODUCT **TO** PUBLIC

**REVOKE** DELETE **ON** SUPPLIER **FROM** BBAESENS

**GRANT** SELECT, INSERT, UPDATE, DELETE **ON** PRODUCT **TO** WLEMAHIEU **WITH** GRANT **OPTION**

**GRANT** REFERENCES **ON** SUPPLIER **TO** SVANDENBROUCKE
SQL Privileges

CREATE VIEW SUPPLIERS_NY
AS SELECT SUPNR, SUPNAME FROM SUPPLIERS
WHERE SUPCITY = 'New York'

GRANT SELECT ON SUPPLIERS_NY TO WLEMAHIEU
SQL for Metadata Management

• The catalog itself can also be implemented as a relational database
SQL for Metadata Management

Table(TableName, ...)

Key(Keyname, ...)

Primary-Key(PK-Keyname, PK-TableName, ...)
PK-Keyname is a foreign key referring to Keyname in Key
PK-TableName is a foreign key referring to TableName in Table

Foreign-Key(FK-Keyname, FK-TableName, FK-PK-Keyname, Update-rule, Delete-rule, ...)
FK-Keyname is a foreign key referring to Keyname in Key
FK-TableName is a foreign key referring to TableName in Table
FK-PK-Keyname is a foreign key referring to PK-Keyname in Primary-Key

Column(Columnname, C-TableName, Data type, Nulls, ...)
C-TableName is a foreign key referring to TableName in Table

Key-Column(KC-Keyname, KC-Columnname, KC-TableName, ...)
KC-Keyname is a foreign key referring to Keyname in Key
KC-Columnname is a foreign key referring to Columnname in Column
KC-TableName is a foreign key referring to C-TableName in Column
SQL for Metadata Management

SELECT * FROM INFORMATION_SCHEMA.COLUMNS
WHERE Table_Name = 'SUPPLIER' limit 5;

<table>
<thead>
<tr>
<th>TABLE_CATALOG (COLUMNS)</th>
<th>TABLE_SCHEMA (COLUMNS)</th>
<th>TABLE_NAME (COLUMNS)</th>
<th>COLUMN_NAME (COLUMNS)</th>
<th>ORDINAL_ (COLUMN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>def</td>
<td>pgdb___1606131858_5fbb0a926751c</td>
<td>supplier</td>
<td>SUPNR</td>
<td>1</td>
</tr>
<tr>
<td>def</td>
<td>pgdb___1606131858_5fbb0a926751c</td>
<td>supplier</td>
<td>SUPNAME</td>
<td>2</td>
</tr>
<tr>
<td>def</td>
<td>pgdb___1606131858_5fbb0a926751c</td>
<td>supplier</td>
<td>SUPADDRESS</td>
<td>3</td>
</tr>
<tr>
<td>def</td>
<td>pgdb___1606131858_5fbb0a926751c</td>
<td>supplier</td>
<td>SUPCITY</td>
<td>4</td>
</tr>
<tr>
<td>def</td>
<td>pgdb___1606131858_5fbb0a926751c</td>
<td>supplier</td>
<td>SUPSTATUS</td>
<td>5</td>
</tr>
</tbody>
</table>
Other SQL Functions

- DATEDIFF()
- ROUND(), Sum(), min(), max(), count()
- IFNULL()
- IF()
- ABS(), avg()
- MOD()
- Between…and
- CASE…WHEN

A lot more: https://www.w3schools.com/sql/sql_ref_mysql.asp
Triggers

- A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.
- For example, a trigger can be invoked when a row is inserted into a specified table or when certain table columns are being updated.
- Bad triggers: infinite loops...

Create trigger zero grade on update takes
(if new takes.grade < 0
then takes.grade = 0)
Assertions

- The assert statement is a useful shorthand for inserting debugging checks
- Verify one or more tables, one or more attributes
- It is in the SQL standard, most DBMS does not support it

```sql
assert condition [, message];
```

```sql
CREATE ASSERTION FewStudents CHECK ( (SELECT COUNT(*) FROM Students) <= (SELECT COUNT(*) FROM Courses) );
```

Can’t have more courses than students
Tips

• Life is not perfect, so does data
• Generate random data for testing
  – https://mockaroo.com/
• Try to construct data that could check for the following potential errors:
  – Incorrect output schema
  – Output may be missing rows from the correct answer (false negatives)
  – Output may contain incorrect rows (false positives)
  – Output may have the wrong number of duplicates.
  – Output may not be ordered properly.
Summary

• SQL views
• SQL privileges
• SQL Functions
Reading and Next Class

• SQL III: Ch 5

• Next: Storing Data and Indexes:
  – Ch 8.1, 8.2
  – Ch 9.1, 9.4
  – Ch 10.3 - 10.8