SQL: Ordering Results, Duplicate Semantics and NULL Values
Fall 2017

School of Computer Science
University of Waterloo

Databases CS348
Ordering Results

- No particular ordering on the rows of a table can be assumed when queries are written. (This is important!)
- No particular ordering of rows of an intermediate result in the query can be assumed either.
- However, it is possible to order the final result of a query, using the `ORDER BY` clause at the end of the query.

General form:

```
ORDER BY e_1 [Dir_1], \ldots, e_k [Dir_k]
```

where \( \text{Dir}_i \) is either ASC or DESC.
List all authors in the database in ascending order of their name:

```
SQL> select distinct *  
2   from author  
3   order by name asc;
```

```
AID   NAME       
---   -----------
 2    Chomicki, Jan  
 3    Saake, Gunter  
 1    Toman, David
```

The `asc` keyword is optional, and is assumed by default. A `descending` order is obtained with the `desc` keyword. Minor sorts, minor minor sorts, etc., can be added.
Multisets and Duplicates

- SQL uses a **MULTISET/BAG** semantics rather than a **SET** semantics:
  - SQL tables are **multisets** of tuples
  - originally for efficiency reasons

- What does “allows duplicates” mean?

<table>
<thead>
<tr>
<th>part</th>
<th>bolt</th>
<th>bolt</th>
<th>bolt</th>
<th>nut</th>
<th>nut</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>part</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>bolt</td>
<td>3</td>
</tr>
<tr>
<td>nut</td>
<td>2</td>
</tr>
</tbody>
</table>
How does this impact Queries?

Example (Cheap Quantification–Projection)

<table>
<thead>
<tr>
<th>EMP</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Dept</td>
</tr>
<tr>
<td>Bob</td>
<td>CS</td>
</tr>
<tr>
<td>Sue</td>
<td>CS</td>
</tr>
<tr>
<td>Fred</td>
<td>PMath</td>
</tr>
<tr>
<td>Barb</td>
<td>Stats</td>
</tr>
<tr>
<td>Jim</td>
<td>Stats</td>
</tr>
</tbody>
</table>

$\{y | \exists x. EMP(x, y)\} \quad \overset{\rightarrow}{\Rightarrow} \quad \begin{array}{c|c|c}
\text{Dept} & \text{cnt} \\
\hline
\text{CS} & 2 \\
\text{PMath} & 1 \\
\text{Stats} & 2 \\
\end{array}$
Duplicates and Queries

... how do we define what an answer to a query is now?

Ideas

1. an finite valuation can appear $k$ times ($k > 0$) as an answer to $Q$
2. the number of duplicates is a function of the numbers of duplicates in subqueries

Definition (Duplicate Semantics (for Relational Calculus))

$\text{DB}, \theta\{k\} \models \varphi$ reads “finite valuation $\theta$ appears $k$ times in $\varphi$’s answer”

- $\text{DB}, \theta\{k\} \models R(x_1, \ldots, x_k)$ if $(\theta(x_1), \ldots, \theta(x_k)) \in R$ $k$ times
- $\text{DB}, \theta\{1\} \models x_i = x_j$ if $\theta(x_i) = \theta(x_j)$
- $\text{DB}, \theta\{m \cdot n\} \models \varphi \land \psi$ if $\text{DB}, \theta\{m\} \models \varphi$ and $\text{DB}, \theta\{n\} \models \psi$
- $\text{DB}, \theta\{m + n\} \models \varphi \lor \psi$ if $\text{DB}, \theta\{m\} \models \varphi$ and $\text{DB}, \theta\{n\} \models \psi$
- $\text{DB}, \theta\{\sum_{v \in D} n_v\} \models \exists x. \varphi$ if $\text{DB}, \theta[x := v]\{n_v\} \models \varphi$
- $\text{DB}, \theta\{\text{max}(0, m - n)\} \models \varphi \land \neg \psi$ if $\text{DB}, \theta\{m\} \models \varphi$ and $\text{DB}, \theta\{n\} \models \psi$
- $\text{DB}, \theta\{1\} \models \text{DISTINCT}(\varphi)$ if $\text{DB}, \theta\{m\} \models \varphi$

(University of Waterloo)
Allowing duplicates leads to additional syntax.

- a duplicate elimination operator
  \[ \Rightarrow \text{“SELECT DISTINCT } x\text{” v.s. “SELECT } x\text{” in SELECT-blocks} \]

- MULTISET (BAG) operators
  \[ \Rightarrow \text{equivalents of set operations} \]
  \[ \Rightarrow \text{but with multiset semantics.} \]
Example

SQL> select r1.publication
  2  from wrote r1, wrote r2
  3  where r1.publication=r2.publication
  4    and r1.author<>r2.author;

PUBLICAT
--------
ChSa98
ChSa98
ChTo98
ChTo98
ChTo98
ChTo98a
ChTo98a

⇒ for publications with \( n \) authors we get \( O(n^2) \) answers!
Bag Operations

- **bag union:** `UNION ALL`
  - additive union: bag containing all in $Q_1$ and $Q_2$.

- **bag difference:** `EXCEPT ALL`
  - subtractive difference (monus):
  - a bag all tuples in $Q_1$ for which there is no “matching” tuple in $Q_2$.

- **bag intersection:** `INTERSECT ALL`
  - a bag of all tuples taking the maximal number common to $Q_1$ and $Q_2$.
Example

```
SQL> ( select author
       2   from wrote, book
       3   where publication=pubid )
  4 union all
  5 ( select author
  6   from wrote, article
  7   where publication=pubid );

      AUTHOR
       -------
          2
          3
          1
          2
          1
          2
          1
```
Summary

SQL covered so far:

1. Simple SELECT BLOCK
2. Set operations
3. Duplicates and Multiset operations
4. Formulation of complex queries, nesting of queries, and views
5. Aggregation

Note that duplicates in subqueries occurring in `where` clauses will not change the results computed by the top-level query, but that this is not true for subqueries in `with` or `from` clauses.
“Pure” SQL Equivalence, Revisited

Recall how nesting in the WHERE clause is syntactic sugar:

```
SELECT r.b
FROM r
WHERE r.a IN (
  SELECT b
  FROM s
) AS s
WHERE r.a = s.b
```

Rewriting does not generally hold if `DISTINCT` is removed.
What is a “null” value?

<table>
<thead>
<tr>
<th>Phone</th>
<th>Name</th>
<th>Office</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Joe</td>
<td>1234</td>
<td>3456</td>
</tr>
<tr>
<td></td>
<td>Sue</td>
<td>1235</td>
<td>?</td>
</tr>
</tbody>
</table>

- *Sue doesn’t have home phone* (value inapplicable)
- *Sue has home phone, but we don’t know her number* (value unknown)
Essentially *poor schema design*.

Better design:

<table>
<thead>
<tr>
<th>Office Phone</th>
<th>Home Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Office</td>
<td>Home</td>
</tr>
<tr>
<td>Joe</td>
<td>Joe</td>
</tr>
<tr>
<td>1234</td>
<td>3456</td>
</tr>
<tr>
<td>Sue</td>
<td></td>
</tr>
<tr>
<td>1235</td>
<td></td>
</tr>
</tbody>
</table>

Queries should behave *as if asked* over the above decomposition.

⇒ (relatively) easy to implement
**Value Unknown**

**Idea**

*Unknown values can be replaced by any domain value (that satisfies integrity constraints).*

⇒ *many possibilities (possible worlds)*

<table>
<thead>
<tr>
<th>Phone</th>
<th>Name</th>
<th>Office</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>1234</td>
<td></td>
<td>3456</td>
</tr>
<tr>
<td>Sue</td>
<td>1235</td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>

Phone

<table>
<thead>
<tr>
<th>Phone</th>
<th>Name</th>
<th>Office</th>
<th>Home</th>
</tr>
</thead>
<tbody>
<tr>
<td>Joe</td>
<td>1234</td>
<td></td>
<td>3456</td>
</tr>
<tr>
<td>Sue</td>
<td>1235</td>
<td></td>
<td>0000</td>
</tr>
</tbody>
</table>

→

```
\[ \begin{array}{c|c|c}
\text{Name} & \text{Office} & \text{Home} \\
\hline
\text{Joe} & 1234 & 3456 \\
\text{Sue} & 1235 & 0000 \\
\end{array} \]
```

...
Value Unknown and Queries

How do we answer queries?

Idea

Answers true in all possible worlds $W$ of an incomplete $D$.

Certain Answer

$$Q(D) = \bigcap_{W \text{ world of } D} Q(W)$$

⇒ answer common to all possible worlds.

Is this (computationally) feasible?

⇒ NO (NP-hard to undecidable except in trivial cases)

SQL’s solution: a (crude) approximation
What can we do with NULLs in SQL?

expressions
- general rule: a NULL as a parameter to an operation makes (should make) the result NULL
- \( 1 + \text{NULL} \rightarrow \text{NULL}, \ 'foo' \ | | \text{NULL} \rightarrow \text{NULL}, \text{etc.} \)

predicates/comparisons
- three-valued logic (crude approximation of “value unknown”)

set operations
- unique special value for duplicates

aggregate operations
- doesn’t “count” (i.e., “value inapplicable”)
Comparisons Revisited

Idea

Comparisons with a NULL value return UNKNOWN

Example

\[
\begin{align*}
1 &= 1 & \text{TRUE} \\
1 &= \text{NULL} & \text{UNKNOWN} \\
1 &= 2 & \text{FALSE}
\end{align*}
\]

Still short of proper logical behaviour:

\[x = 0 \lor x \neq 0\]

should be always true (no matter what \(x\) is, including NULL!), but...
Boolean operations have to handle \texttt{UNKNOWN} ⇒ \textit{extended truth tables for Boolean connectives}

\begin{center}
\begin{tabular}{c|cccc}
& T & U & F & \text{\ } \\
\hline
T & T & U & F & \text{\ } \\
U & U & U & F & \text{\ } \\
F & F & F & F & \text{\ } \\
\end{tabular}
\begin{tabular}{c|cccc}
& T & U & F & \text{\ } \\
\hline
T & T & T & T & \text{\ } \\
U & T & U & U & \text{\ } \\
F & T & U & F & \text{\ } \\
\end{tabular}
\begin{tabular}{c|cc}
& T & F & \text{\ } \\
\hline
T & F & \text{\ } \\
U & U & \text{\ } \\
F & T & \text{\ } \\
\end{tabular}
\end{center}

\ldots for tuples in which \( x \) is assigned the \texttt{NULL} value we get:

\[
x = 0 \lor x \neq 0 \rightarrow \texttt{UNKNOWN} \lor \texttt{UNKNOWN} \rightarrow \texttt{UNKNOWN}
\]

which is not the same as \texttt{TRUE}.
UNKNOWN in WHERE Clauses

How is this used in a WHERE clause?

- **Additional syntax** `IS TRUE, IS FALSE, and IS UNKNOWN`  
  \[\Rightarrow\] WHERE `<cond>` shorthand for WHERE `<cond>` IS TRUE

- **Special comparison** `IS NULL`

List all authors for which we don’t know a URL of their home page:

```
SQL> select aid, name
    2   from author
    3   where url IS NULL

   AID   NAME
  ----------  ------------
    3 Saake, Gunter
```
Counting NULLS

How do NULLs interact with counting (and aggregates in general)?

- \( \text{count}(\text{URL}) \) counts only non-NULL URL’s

\[ \Rightarrow \text{count}(*) \text{ counts “rows”} \]

```sql
db2 => select count(*) as RS, count(url) as US
db2 (cont.) => from author
```

<table>
<thead>
<tr>
<th>RS</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

1 record(s) selected.
Outer Join

Idea

allow “NULL-padded” answers that “fail to satisfy” a conjunct in a conjunction

- extension of syntax for the FROM clause
  \[ \text{\Rightarrow FROM } R \text{ <j-type> JOIN } S \text{ ON } C \]

  \[ \text{\Rightarrow the <j-type> is one of FULL, LEFT, RIGHT, or INNER} \]

- semantics (for \( R(x, y) \), \( S(y, z) \), and \( C = (r.y = s.y) \)).
  1. \( \{ (x, y, z) : R(x, y) \wedge S(y, z) \} \)
  2. \( \{ (x, y, \text{NULL}) : R(x, y) \wedge \neg(\exists z. S(y, z)) \} \) for LEFT and FULL
  3. \( \{ (\text{NULL}, y, z) : S(y, z) \wedge \neg(\exists x. R(x, y)) \} \) for RIGHT and FULL

  \[ \Rightarrow \text{syntactic sugar for UNION ALL} \]
Example

db2 => select aid, publication

db2 (cont.) => from author left join wrote

db2 (cont.) => on aid=author

<table>
<thead>
<tr>
<th>AID</th>
<th>PUBLICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ChTo98</td>
</tr>
<tr>
<td>1</td>
<td>ChTo98a</td>
</tr>
<tr>
<td>1</td>
<td>Tom97</td>
</tr>
<tr>
<td>2</td>
<td>ChTo98</td>
</tr>
<tr>
<td>2</td>
<td>ChTo98a</td>
</tr>
<tr>
<td>2</td>
<td>ChSa98</td>
</tr>
<tr>
<td>3</td>
<td>ChSa98</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
</tr>
</tbody>
</table>

8 record(s) selected.
For every author count the number of publications:

```
db2  => select aid, count(publication) as pubs
db2 (cont.) => from author left join wrote
db2 (cont.) => on aid=author
db2 (cont.) => group by aid
```

<table>
<thead>
<tr>
<th>AID</th>
<th>PUBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

4 record(s) selected.
### Summary

- **NULLs are necessary evil**
  - used to account for (small) irregularities in data
  - should be used sparingly

- **can be always avoided**
  - however some of the solutions may be inefficient

- **you can’t escape NULLs in practice**
  - easy fix for blunders in schema design
  - … also due to schema evolution, etc.