SQL: Ordering Results, Duplicate Semantics and NULL Values

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 $Dir_i$ is either ASC or DESC; ASC is the default.
Example

List all authors in the database in ascending order of their name:

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

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

Again, 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 has always had a **MULTISET/BAG** semantics rather than a **SET** semantics:
  - \(\Rightarrow\) SQL tables are **multisets** of tuples
  - \(\Rightarrow\) originally for efficiency reasons

- What does “**allows duplicates**” mean?

```
part
bolt
bolt
bolt
nut
nut
```

\(\iff\)

```
part  cnt
bolt  3
nut   2
```
How does this impact Queries?

Example (Cheap Quantification–Projection)

<table>
<thead>
<tr>
<th>EMP</th>
<th>Dept</th>
</tr>
</thead>
<tbody>
<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>

\[ \{y \mid \exists x. \text{EMP}(x,y) \} \]

\[ \text{Dept cnt} \]

<table>
<thead>
<tr>
<th>Dept</th>
<th>cnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>CS</td>
<td>2</td>
</tr>
<tr>
<td>PMath</td>
<td>1</td>
</tr>
<tr>
<td>Stats</td>
<td>2</td>
</tr>
</tbody>
</table>
Range-restricted Queries for Multisets

Definition (Range restricted formulas with \textsc{Distinct})

A formula (condition) $\varphi$ is \textit{range restricted} when, for $\varphi_i$ that are also range restricted, $\varphi$ has the form

$$R(x_{i_1}, \ldots, x_{i_k}),$$

$$\varphi_1 \land \varphi_2,$$

$$\varphi_1 \land (x_i = x_j) \quad (\{x_i, x_j\} \cap \text{FV}(\varphi_1) \neq \emptyset),$$

$$\exists x_i. \varphi_1 \quad (x_i \in \text{FV}(\varphi_1)),$$

$$\varphi_1 \lor \varphi_2 \quad (\text{FV}(\varphi_1) = \text{FV}(\varphi_2)),$$

$$\varphi_1 \land \neg \varphi_2 \quad (\text{FV}(\varphi_2) = \text{FV}(\varphi_1)),$$

or

$\textsc{Distinct}(\varphi).$
Duplicates and Queries

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

Ideas

1. A finite valuation can appear \( k \) times \((k > 0)\) as a query answer.
2. The number of duplicates is a function of the numbers of duplicates in formulas.
3. \( \text{DB}, \theta, k \models \varphi \) reads “finite valuation \( \theta \) appears \( k \) times in \( \varphi \)’s answer”.

Definition (Multiset Semantics for the Relational Calculus)

\[
\begin{align*}
\text{DB}, \theta, 0 \models \varphi & \quad \text{if} \quad \text{DB}, \theta \not\models \varphi \\
\text{DB}, \theta, k \models R(x_1, \ldots, x_k) & \quad \text{if} \quad (\theta(x_1), \ldots, \theta(x_k)) \in R \quad k \text{ times} \\
\text{DB}, \theta, m \cdot n \models \varphi \land \psi & \quad \text{if} \quad \text{DB}, \theta, m \models \varphi \text{ and } \text{DB}, \theta, n \models \psi \\
\text{DB}, \theta, m \models \varphi \land (x_i = x_j) & \quad \text{if} \quad \text{DB}, \theta, m \models \varphi \text{ and } \theta(x_i) = \theta(x_j) \\
\text{DB}, \theta, \sum_{v \in D} n_v \models \exists x. \varphi & \quad \text{if} \quad \text{DB}, \theta[x := v], n_v \models \varphi \\
\text{DB}, \theta, m + n \models \varphi \lor \psi & \quad \text{if} \quad \text{DB}, \theta, m \models \varphi \text{ and } \text{DB}, \theta, n \models \psi \\
\text{DB}, \theta, m - n \models \varphi \land \lnot \psi & \quad \text{if} \quad \text{DB}, \theta, m \models \varphi, \text{DB}, \theta, n \models \psi \text{ and } m > n \\
\text{DB}, \theta, 1 \models \text{DISTINCT}(\varphi) & \quad \text{if} \quad \text{DB}, \theta, m \models \varphi
\end{align*}
\]
Allowing duplicates leads to additional syntax.

- A duplicate elimination operator
  ⇒ “SELECT DISTINCT x” v.s. “SELECT x” in SELECT-blocks

- MULTISET (BAG) operators
  ⇒ equivalents of set operations
  ⇒ 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
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
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 );

<table>
<thead>
<tr>
<th>AUTHOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

...a fragment of a more meaningful query (coming later).
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.
Recall how nesting in the WHERE clause is syntactic sugar:

```sql
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)
Value Inapplicable

- Essentially *poor schema design*.

- Better design:

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

<table>
<thead>
<tr>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Joe</td>
</tr>
<tr>
<td>Sue</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>Joe</td>
</tr>
<tr>
<td>Sue</td>
</tr>
</tbody>
</table>

(University of Waterloo)
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**
- \( 1 + \text{NULL} \rightarrow \text{NULL}, \ '\text{foo}' \mid | \text{NULL} \rightarrow \text{NULL}, \text{etc.} \)

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

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 = 1</td>
<td>TRUE</td>
</tr>
<tr>
<td>1 = NULL</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>1 = 2</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Still short of *proper logical* behaviour:

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

should be always `true` (no matter what \( x \) is, including `NULL`!), but...
**Idea**

*Boolean operations have to handle* **UNKNOWN**

⇒ *extended truth tables for Boolean connectives*

\[
\begin{array}{c|cccc}
\wedge & T & U & F \\
T & T & U & F \\
U & U & U & F \\
F & F & F & F \\
\end{array}
\quad
\begin{array}{c|cccc}
\vee & T & U & F \\
T & T & T & T \\
U & T & U & U \\
F & T & U & F \\
\end{array}
\quad
\begin{array}{c|c}
\neg & \\
T & F \\
U & U \\
F & T \\
\end{array}
\]

...for tuples in which \( x \) is assigned the **NULL** value we get:

\[ x = 0 \vee x \neq 0 \rightarrow \text{UNKNOWN} \vee \text{UNKNOWN} \rightarrow \text{UNKNOWN} \]

which is not the same as **TRUE**.
UNKNOWN in WHERE Clauses

How is this used in a WHERE clause?

- Additional syntax IS TRUE, IS FALSE, and IS UNKNOWN
  \[ \Rightarrow \text{WHERE } <\text{cond}> \text{ shorthand for WHERE } <\text{cond}> \text{ IS TRUE} \]

- Special comparison IS NULL

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

```sql
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)?

- \texttt{count(URL)} counts only non-NULL URL’s
  \[\Rightarrow \texttt{count(*)} \text{ counts “rows”}\]

```sql
db2 => select count(*) as RS, count(url) as US
     db2 => 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
  \[
  \text{FROM } R \ <j\text{-type}> \ JOIN \ S \ ON \ C
  \Rightarrow \text{the } <j\text{-type}> \text{ is one of FULL, LEFT, RIGHT, or INNER}
  \]

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

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

db2 => select aid, publication
db2 => from author left join wrote
db2 => 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.
Counting with OJ

For every author count the number of publications:

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