SQL

Part 2: Set Operations and Aggregates

University of Waterloo
List of Slides

1 Complex Queries in SQL
2 Set Operations at Glance
3 Example: Union
4 Example: Set Difference
5 Multisets and Duplicates
6 Example
7 Bag Operations
8 Example
9 What About Nesting of Queries?
10 Naming Queries and Views
11 Example
12 FROM revisited
13 Example
14 Can’t we just use OR instead of UNION?
15 Summary on First-Order SQL
16 Aggregation
17 Operational Reading
18 Aggregation (cont).
19 Aggregation (cont.)
20 Example (count)
21 Example (sum)
22 HAVING clause
23 Example
24 Example (revisited.)
25 Summary
Complex Queries in SQL

- so far we can write only $\exists$, $\land$ queries
  $\Rightarrow$ the SELECT BLOCK queries
  $\Rightarrow$ not sufficient to cover all RC queries

- remaining connectives:
  1. $\lor$, $\neg$: are expressed using set operations
     $\Rightarrow$ easy to enforce safety requirements
  2. $\forall$: use negation and $\exists$
Set Operations at Glance

Answers to Select Blocks (and all SQL queries in general) are relations (sets of tuples).

⇒ we can apply set operations on them:

- set union: $Q_1 \text{ UNION } Q_2$
  ⇒ the set of tuples in $Q_1$ or $Q_2$.
  ⇒ used to express “or”.

- set difference: $Q_1 \text{ EXCEPT } Q_2$
  ⇒ the set of tuples in $Q_1$ but not in $Q_2$.
  ⇒ used to express “and not”.

- set intersection: $Q_1 \text{ INTERSECT } Q_2$
  ⇒ the set of tuples in both $Q_1$ and $Q_2$.
  ⇒ used to express “and” (redundant, rarely used).

$Q_1$ and $Q_2$ must have union-compatible signatures:
⇒ same number and types of attributes
Example: Union

List all publication ids for books or journals:

```
SQL> (select pubid from book)
  2  union
  3  (select pubid from journal);

PUBID
-----
ChSa98
JLP-3-98
```
Example: Set Difference

List all publication ids except those for articles:

```
SQL> (select pubid from publication)  
    2 minus  
    3 (select pubid from article);

PUBID
-------
ChSa98
DOOD97
JLP-3-98
```
Multisets and Duplicates

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

- this leads to additional (extra-logical) syntactic constructions:
  - a *duplicate elimination operator*
    - in the SELECT BLOCK: `SELECT DISTINCT ...`
  - **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
ChTo98
ChTo98a
ChTo98a
```

Note duplicate entries for publication id’s
⇒ for publications with $n$ authors we get $O(n^2)$ answers!
Bag Operations

- bag union **UNION ALL**
  \[\Rightarrow\] additive union: bag containing all in \(Q_1\) and \(Q_2\).

- bag difference **EXCEPT ALL**
  \[\Rightarrow\] subtractive difference (monus):
  \[\Rightarrow\] a bag all tuples in \(Q_1\) for which there is no "matching" tuple in \(Q_2\).

- bag intersection **INTERSECT ALL**
  \[\Rightarrow\] a bag of all tuples taking the maximal number either in \(Q_1\) or in \(Q_2\)
Example

For every book and article list all authors:

```
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).
What About Nesting of Queries?

Using the syntax (so far) we can use SELECT Blocks and Set operations inside (other) Set operations.

What is we need to use a Set Operation inside of a SELECT Block?

- use distributive laws
  \[ (A \lor B) \land C \equiv (A \land C) \lor (B \land C) \]
  \Rightarrow \text{often very cumbersome}

- nest set operation inside a select block.
  \Rightarrow \text{Views or extensions to the FROM clause.}
Naming Queries and Views

Idea: Queries denote relations. We provide a naming mechanism that allows us to assign names to (results of) queries.

⇒ can be used later in place of base relations.

- Syntax:

```
CREATE VIEW foo [ <opt-schema> ] AS
 ( <query-goes-here> )
```

- Views are permanently added to the schema
  ⇒ often used to define *External View* of the database
  ⇒ you must have a permission to create them
Example

List all publication titles for books or journals:

```sql
SQL> create view bookorjournal as
2     ( (select pubid from book)
3         union
4     (select pubid from journal)
5  );

SQL> select title
2     from publication, bookorjournal
3     where publication.pubid=bookorjournal.pubid;

<table>
<thead>
<tr>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logics for Databases and Information Systems</td>
</tr>
<tr>
<td>Journal of Logic Programming</td>
</tr>
</tbody>
</table>
```
FROM revisited

- using the view mechanism is often too cumbersome:
  ⇒ ad-hoc querying, program-generated queries:
    big overhead due to catalog access
    you must remember to discard (DROP) the views
  ⇒ you need a CREATE VIEW privilege

- SQL/92 allows us to inline queries in the FROM clause:

  FROM ..., ( <query-here> ) <id>,...

  ⇒ <id> stands for the result of <query-here>.
  ⇒ unlike for base relations, <id> is mandatory.

- in old SQL (SQL/89) views were the only option...
Example

List all publication titles for books or journals:

```
SQL> select title
2   from publication,
3       ( (select pubid from book)
4           union
5             (select pubid from journal) ) bj
6   where publication.pubid=bj.pubid;

<table>
<thead>
<tr>
<th>TITLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logics for Databases and Information Systems</td>
</tr>
<tr>
<td>Journal of Logic Programming</td>
</tr>
</tbody>
</table>
```
Can’t we just use OR instead of UNION?

• A common mistake:

  ⇒ use of OR in the WHERE clause instead of the UNION operator

• An incorrect solution:

  ```sql
  SELECT title
  FROM publication, book, journal
  WHERE publication.pubid=book.pubid
  OR publication.pubid=journal.pubid
  ```

• often works; but imagine there are no books...
Summary on First-Order SQL

- SQL introduced so far captures all of **Relational Calculus**
  - optionally with duplicate semantics
  - powerful (many queries can be expressed)
  - efficient (PTIME, LOGSPACE)

- Shortcomings:
  - some queries are hard to write (syntactic sugar)
  - no “counting” (aggregation)
  - no “path in graph” (recursion)
Aggregation

Standard and most useful extension of First-Order Queries.

- Aggregate (column) functions are introduced to
  - find number of tuples in a relation
  - add values of an attribute (over the whole relation)
  - find minimal/maximal values of an attribute

- Can apply to *groups* of tuples that with equal values in (some) attributes

- Generally, can **NOT** be written in Relational Calculus
Operational Reading

1. partition the input relation to groups with equal values of grouping attributes

2. on each of these partitions apply the aggregate function

3. collect the results and form the answer
Aggregation (cont).

Formal Definition [Klug]:

\[
\{ (x'_1, \ldots, x'_k, f_1, \ldots, f_l) : \\
\quad f_i := \text{agg}_i \{ (x_1, \ldots, x_k, y_1, \ldots, y_n) : \\
\quad \quad Q((x_1, \ldots, x_k, y_1, \ldots, y_n) \\
\quad \quad \quad \land x_1 = x'_1 \land \ldots \land x_k = x'_k \}
\land (\exists y_1, \ldots, y_n. Q)(x'_1, \ldots, x'_k) \}
\]

where

- \( x'_1, \ldots, x'_k \) are the **grouping** attributes.
- \( \text{agg}_i \) are the **aggregate functions**
  \[\Rightarrow\] e.g., count, sum, min, max, or avg.
- \( Q \) is the query on which aggregation is applied.
Aggregation (cont.)

The same in SQL syntax:

```sql
SELECT x1,...,xk, agg1,...,aggl
FROM Q
GROUP BY x1,...,xk
```

Restrictions:

- all attributes in the `SELECT` clause that are **NOT** in the scope of an aggregate function **MUST** appear in the `GROUP BY` clause.

- `aggi` are of the form `count(y), sum(y), min(y), max(y), or avg(y)` where `y` is an attribute of `Q` (usually not in the `GROUP BY` clause).
Example (count)

For each publication count the number of authors:

```sql
SQL> select publication, count(author)
2  from wrote
3  group by publication

<table>
<thead>
<tr>
<th>PUBLICAT</th>
<th>COUNT(AUTHOR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChSa98</td>
<td>2</td>
</tr>
<tr>
<td>ChTo98</td>
<td>2</td>
</tr>
<tr>
<td>ChTo98a</td>
<td>2</td>
</tr>
<tr>
<td>Tom97</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Example (sum)

For each author count the number of article pages:

```
SQL> select author,sum(endpage-startpage+1) as pages
   2  from wrote, article
   3  where publication=pubid
   4  group by author
```

<table>
<thead>
<tr>
<th>AUTHOR</th>
<th>PAGES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>87</td>
</tr>
<tr>
<td>2</td>
<td>68</td>
</tr>
</tbody>
</table>

...not quite correct: it doesn’t list 0 pages for author 3.
**HAVING clause**

- the `WHERE` clause can’t impose conditions on values of aggregates as the `WHERE` clause has to be used before `GROUP BY`

- SQL allows a `HAVING` clause instead
  \[\Rightarrow\] like `WHERE`, but for aggregates...

- The aggregate functions used in the `HAVING` clause may be different from those in the `SELECT` clause; the grouping, however, is common.
Example

List publications with exactly one author:

```
SQL> select publication, count(author)
  2  from wrote
  3  group by publication
  4  having count(author)=1

<table>
<thead>
<tr>
<th>PUBLICAT</th>
<th>COUNT(AUTHOR)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tom97</td>
<td>1</td>
</tr>
</tbody>
</table>
```

This query can be written without aggregation as well.
Example (revisited.)

For every author count the number of books and articles:

```
SQL> select name, count(aid)
  2  from author, (  
  3    ( select author  
  4      from wrote, book  
  5      where publication=pubid )  
  6    union all  
  7    ( select author  
  8      from wrote, article  
  9      where publication=pubid ) ) ba  
 10  where aid=author  
11  group by name,aid;
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>COUNT(AID)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toman, David</td>
<td>3</td>
</tr>
<tr>
<td>Chomicki, Jan</td>
<td>3</td>
</tr>
<tr>
<td>Saake, Gunter</td>
<td>1</td>
</tr>
</tbody>
</table>
Summary

• SQL covered so far:
  1. Simple SELECT BLOCK (naming of attributes, allowed expressions, etc.)
  2. Set operations
  3. Duplicates and Bag operations
  4. Formulation of complex queries, nesting of queries, and views
  5. Aggregation

  ... this covers ALL of SQL queries (i.e., they can be expressed in the syntax introduced so far, but it might be cumbersome)

  ⇒ (lots of) syntactic sugar coming next ...