SQL: the Basics

David Toman

School of Computer Science
University of Waterloo

Introduction to Databases CS348

SQL

- Structured Query Language

  $\Rightarrow$ Developed in IBM Almaden (system R)
- Based on

  $\Rightarrow$ (Conjunctive) queries in Relational Calculus

  $\Rightarrow$ Set/Bag semantics and operations

  $\Rightarrow$ Aggregation
- **BAG SEMANTICS** (next time)
- A **committee** design

  $\Rightarrow$ choices often more "pragmatic" than "logical"

  $\Rightarrow$ several *standard* versions:

  SQL/89, **SQL/92** = SQL2, SQL3, ...

Starting Point: Range-restricted Queries

Definition (Range restricted queries)

$$
\begin{array}{lll}
Q & ::= & R(x_{i_1}, \ldots, x_{i_k}) \\
 & & \left.\begin{array}{l} Q \wedge Q \\ Q \wedge x_i = x_j \\ \exists x_i . Q \end{array}\right\} \quad x_i, x_j \in FV(Q) \\
 & & \left.\begin{array}{l} Q_1 \vee Q_2 \\ Q_1 \wedge \neg Q_2 \end{array}\right\} \quad FV(Q_1) = FV(Q_2)
\end{array}
$$

SQL (cont.)

Three major parts of the language:

1. DML (Data Manipulation Language)
   - $\Rightarrow$ Query language
   - $\Rightarrow$ Update language
2. DDL (Data Definition Language)
   - $\Rightarrow$ defines *schema* for relations
   - $\Rightarrow$ creates (modifies/destroys) database objects.
3. DCL (Data Control Language)
   - $\Rightarrow$ access control

Also: Embedded SQL (SQL/J) and ODBC (JDBC)

$\Rightarrow$ necessary for application development

Roadmap to SQL Queries

- the “select block”
  - simple select-from-where
  - subqueries in the “from” clause
  - grouping, aggregation, and having clauses
  - duplicates and “distinct”
  - subqueries in the “where” clause
  - ordering the output
- set operations
  - with duplicates
- naming queries and views

SQL Data Types

Values of attributes in SQL:

`integer`	integer (32 bit)
`smallint`	integer (16 bit)
`decimal(m,n)`	fixed decimal
`float`	IEEE float (32 bit)
`char(n)`	character string (length `n`)
`varchar(n)`	variable length string (at most `n`)
`date`	year/month/day
`time`	hh:mm:ss.ss

Sample Database Revisited

```
author(aid integer, name char(20))

wrote(author integer, publication char(8))

publication(pubid char(8), title char(70))

book(pubid char(8),
     publisher char(50), year integer)

journal(pubid char(8),
        volume integer, no integer, year integer)

proceedings(pubid char(8),
            year integer)

article(pubid char(8),  crossref char(8),
        startpage integer, endpage integer)
```

. . . SQL is **NOT** case sensitive.

The “SELECT Block”

Basic syntax:

```
SELECT DISTINCT <results>
FROM            <tables>
WHERE           <condition>
```

- Allows formulation of conjunctive ($\exists, \wedge$) queries
  $\Rightarrow$ a conjunction of `<tables>` and `<condition>`
  $\Rightarrow$ attributes not in `<result>` existentially quantified
  $\Rightarrow$ `<result>` specifies values in the resulting tuples

- many other *clauses* to follow later. . .

Example

List all authors in the database:

```
SQL> select distinct *
  2  from author;

AID NAME           URL
--- -------------- ----------------------
  1 Toman, David   http://db.uwaterloo.ca/~david
  2 Chomicki, Jan  http://cs.monmouth.edu/~chomicki
  3 Saake, Gunter
```

The `FROM` clause cannot be used on its own

$\Rightarrow$ the `"SELECT *"` notation

$\Rightarrow$ also reveals all attribute names

Naming Attributes

- **problem:** what if two relations use the same attribute name?

  $\Rightarrow$ `publication(pubid, ...)`
  $\Rightarrow$ `book(pubid, ...)`

- ...and we want to get, e.g., titles of all books

$$\exists p, x, y.\texttt{publication}(p, n) \wedge \texttt{book}(p, x, y)$$

$\Rightarrow$ we prefix the ambiguous attributes names by the name of the appropriate relation:

  - `publication.pubid` (first "p")
  - `book.pubid` (second "p")

Example

List titles of all books:

```
SQL> select distinct title
  2  from publication, book
  3  where publication.pubid=book.pubid;

TITLE
----------------------------------------
Logics for Databases and Information Systems
```

Naming Attributes (cont.)

- what if we need to use the **same** table
  to be used several times in the `FROM` clause?
- list publications with at least two authors

$$\exists y_1, y_2.\texttt{wrote}(y_1, x) \wedge \texttt{wrote}(y_2, x) \wedge y_1 \neq y_2$$

  $\Rightarrow$ problem: y_1 and y_2 ae both called `pubid`
  $\Rightarrow$ ... and they both appear in the `wrote` relation
- solution: corelation names in the `FROM` clause
  $\Rightarrow$ e.g., `FROM wrote r1, wrote r2` makes

$$y_1 = \texttt{r1.author}$$
$$y_2 = \texttt{r2.author}$$

  $\Rightarrow$ `r1` and `r2` are **corelation** names

Example

List all publications with at least two authors:

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

PUBLICATION
-----------
ChSa98
ChTo98
ChTo98a
```

The "FROM" Clause (summary)

Syntax:

$$\texttt{FROM}\ \ R_1[\ n_1], \ldots, R_k[\ n_k]$$

- R_i are relation (table) names
- n_i are distinct identifiers
- the clause represents a **conjunction** $R_1 \wedge \ldots \wedge R_k$
  $\Rightarrow$ all variables of R_i's are *distinct*
  $\Rightarrow$ we use (co)relation names to resolve ambiguities
- can NOT appear alone
  $\Rightarrow$ only as a part of the *select block*

The "SELECT" Clause

Syntax:

$$\texttt{SELECT DISTINCT } e_1\texttt{[AS } i_1\texttt{]}, \ldots, e_k\texttt{[AS } i_k\texttt{]}$$

1. eliminate superfluous attributes from answers ($\exists$)
2. form **expressions**:

   $\Rightarrow$ built-in functions applied to values of attributes
3. give names to attributes in the answer

Standard Expressions

we can create values in the answer tuples using **built-in** functions:

- on numeric types:

  $+, -, *, /, \ldots$ (usual arithmetic)
- on strings:

  $||$ (concatenation), `substr`, ...
- constants (of appropriate types)

  `"SELECT 1"` is a valid query in SQL/92
- UDF (user defined functions)

Note: all attribute names **MUST** be "present" in the `FROM` clause.

Example

For every article list the number of pages:

```
SQL> select pubid, endpage-startpage+1
  2  from article;

PUBID     ENDPAGE-STARTPAGE+1
----- --------------
ChTo98                     40
ChTo98a                    28
Tom97                      19
```

Naming the Results

Results of queries $\Longleftrightarrow$ Tables

What are the names of attributes in the result of a `SELECT` clause?

- A single attribute: inherits the name
- An expression: implementation dependent

we can—and should—**explicitly** name the resulting attributes:

$\Rightarrow$ "`<expr> AS <id>`" where `<id>` is the new name

Example

and name the resulting attributes `id,numberofpages`:

```
SQL> select pubid as id,
  2         endpage-startpage+1 as numberofpages
  3  from article;

ID        NUMBEROFPAGES
----- ----------
ChTo98              40
ChTo98a             28
Tom97               19
```

The "WHERE" Clause

Additional **conditions** on tuples that qualify for the answer.

WHERE *C*

- standard atomic conditions:
  1. equality: =, != (on all types)
  2. order: <, <=, >, >=, <> (on numeric and string types)
- conditions may involve *expressions*

  $\Rightarrow$ similar conditions as in the SELECT clause

Example(s)

Find all journals printed since 1997:

```
SQL> select * from journal where year>=1997;

PUBID          VOLUME          NO        YEAR
-------- ---------- ---------- ----------
JLP-3-98           35          3        1998
```

Find all articles with more than 20 pages:

```
SQL> select * from article
  2  where endpage-startpage>20;

PUBID    CROSSREF  STARTPAGE    ENDPAGE
-------- --------- --------- ----------
ChTo98   ChSa98           31         70
ChTo98a  JLP-3-98        263        290
```

Boolean Connectives

Atomic conditions can be combined using **boolean connectives**:

- `AND` (conjunction)
- `OR` (disjunction)
- `NOT` (negation)

Example

List all publications with at least two authors:

```
SQL> select distinct r1.publication
  2  from wrote r1, wrote r2
  3  where r1.publication=r2.publication
  4    and not r1.author=r2.author;

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

Summary

- simple SELECT block accounts for many queries

  $\Rightarrow$ all in $\exists, \wedge$ fragment of relational calculus

- additional features
  - alternative names for relations
  - expressions and naming in the output
  - built-in atomic predicates and boolean connectives

- well defined semantics (declarative and operational)

Complex Queries in SQL

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

- remaining connectives:
  1. $\vee, \neg$: are expressed using **set operations**
     $\Rightarrow$ easy to enforce *range-restriction requirements*
  2. $\forall$: rewrite using negation and $\exists$
     $\Rightarrow$ the same for $\rightarrow$, $\leftrightarrow$, etc.

Set Operations at Glance

Answers to *Select Blocks* are **relations** (sets of tuples)

$\Rightarrow$ we can apply **set operations** on them:

- set union: Q_1 `UNION` Q_2
  $\Rightarrow$ the set of tuples in Q_1 or in Q_2.
  $\Rightarrow$ used to express "or".
- set difference: Q_1 `EXCEPT` Q_2
  $\Rightarrow$ the set of tuples in Q_1 but not in Q_2.
  $\Rightarrow$ used to express "and not".
- set intersection: Q_1 `INTERSECT` Q_2
  $\Rightarrow$ the set of tuples in both Q_1 and Q_2.
  $\Rightarrow$ used to express "and" (redundant, rarely used).

Q_1 and Q_2 must have **union-compatible** signatures:

$\Rightarrow$ 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  except
  3  (select pubid from article);

PUBID
-----
ChSa98
DOOD97
JLP-3-98
```

What About Nesting of Queries?

We can use *SELECT Blocks* (and other *Set operations*)
as arguments of *Set operations*.

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

- we can use **distributive laws**
  $\Rightarrow (A \vee B) \wedge C \equiv (A \wedge C) \vee (B \wedge C)$
  $\Rightarrow$ often **very** cumbersome
- nest set operation inside a select block.
  $\Rightarrow$ 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.

$\Rightarrow$ 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

  $\Rightarrow$ often used to define *External Schema* of the database

  $\Rightarrow$ you must have **permissions** to create them

Example

List all publication titles for books or journals:

```
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;

TITLE
------------------------------
Logics for Databases and Information Systems
Journal of Logic Programming
```

The FROM clause 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 the 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;

TITLE
------------------------------
Logics for Databases and Information Systems
Journal of Logic Programming
```

Can't we just use `OR` instead of `UNION`?

- A **common** mistake:

  $\Rightarrow$ use of `OR` in the `WHERE` clause instead of the `UNION` operator

- An incorrect solution:

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

- often works; but imagine there are no `book`s...

Summary on First-Order SQL

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

- Shortcomings:
  $\Rightarrow$ some queries are hard to write (syntactic sugar)
  $\Rightarrow$ no *"counting"* (aggregation)
  $\Rightarrow$ no *"path in graph"* (recursion)