SQL

Part 1: The Basics

University of Waterloo

List of Slides

1
2 SQL
3 SQL (cont.)
4 Roadmap to SQL Queries
5 SQL Data Types
6 Sample Database Revisited
7 The “SELECT Block”
8 Example
9 Naming Attributes
10 Example
11 Naming Attributes (cont.)
12 Example
13 The **"FROM"** Clause (summary)
14 The **"SELECT"** Clause
15 Standard Expressions
16 Example
17 Naming the Results
18 Example
19 The **"WHERE"** Clause
20 Example(s)
21 Boolean Connectives
22 Example
23 Semantics
24 Declarative
25 Operational
26 Summary

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**

- A **committee** design

  $\Rightarrow$ choices often more "pragmatic" than "logical"
  $\Rightarrow$ several *standard* versions:
  SQL/89, **SQL/92** = SQL2, SQL3, . . .

- creeping featuralism

SQL (cont.)

Three major parts of the language:

1. DML (Data Manipulation Language)

   ⇒ Query language
   ⇒ Update language

2. DDL (Data Definition Language)

   ⇒ defines *schema* for relations
   ⇒ creates (modifies/destroys) database objects.

3. DCL (Data Control Language)

   ⇒ access control

Also: Embedded SQL and ODBC
⇒ 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

`decimal(m,n)` : **`m`** number of digits, **`n`** number of digits after `.`;

Sample Database Revisited

```
author(aid integer,              /* author id */
       name char(20))

wrote(author integer,            /* author id */
      publication char(8))       /* publ. id  */

publication(pubid char(8),       /* publ. id  */
            title char(70))

book(pubid char(8),              /* publ. id  */
     publisher char(50),
     year integer)

journal(pubid char(8),           /* publ. id  */
        volume integer, no integer,
        year integer)

proceedings(pubid char(8),       /* publ. id  */
            year integer)

article(pubid char(8),           /* publ. id  */
        crossref char(8),        /* publ. id  */
        startpage integer,
        endpage integer)
```

The "SELECT Block"

Basic syntax:

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

- Allows formulation of $\exists, \wedge$-queries

  $\Rightarrow$ a conjunction of **<tables>** and **<condition>**

  $\Rightarrow$ attributes not in **<result>** existentially quantified

  $\Rightarrow$ **<result>** specifies values in the resulting tuples

Example

List all authors in the database:

```
SQL> select *
  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? E.g.:

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

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

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

  $\Rightarrow$ we prefix the ambiguous attributes by name of the appropriate relation, e.g.,
  `publication.pubid` (first "p")
  `book.pubid` (second "p")

Example

List titles of all books:

```
SQL> select 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
  several times in the **`FROM`** clause?
- example: list publications with at least two authors

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

  $\Rightarrow$ problem: y_1 and y_2 ae both called **`pubid`**
  $\Rightarrow$ . . . and they both appear only in a **`wrote`** relation

- solution: corelation names in the **`FROM`** clause
  $\Rightarrow$ e.g., **`FROM wrote r1, wrote r2`** makes

$$y_1 = \mathbf{r1.author} \qquad y_2 = \mathbf{r2.author}$$

  $\Rightarrow$ **`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;

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

The `"FROM"` Clause (summary)

Syntax:

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

where:

- 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 (arguments) 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:

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

- Allows us to
  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

Besides listing attribute names in the **`SELECT`** clause we can combine the values in retrieved tuples using **built-in** functions:

- on numeric types:

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

- on strings:

  $||$ (concatenation), **`substr`**, . . .

- constants (of appropriate types)
  $\Rightarrow$ **`"SELECT 1"`** is a valid query in SQL/92

- UDF (user defined functions)

Note: All attributes used in expressions **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

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

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

Optionally we can **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

Adds additional **conditions** on tuples that qualify for the answer. Syntax:

`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 in the **`WHERE`** clause can be combined using **boolean connectives**:

- **`AND`** conjunction
- **`OR`** disjunction
- **`NOT`** negation

with the usual precedence and associativity.

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
```

Semantics

```
SELECT e1 AS i1, ... , ek AS ik
FROM   R1 r1, ... , Rn rn
WHERE  C
```

Declarative

$$\{(e_1, \ldots, e_k) : \exists y_1, \ldots, y_m . R_1(\ldots) \wedge \ldots \wedge R_n(\ldots) \wedge C\}$$

$\Rightarrow$ give me all $e_1, \ldots, e_k$ such that
there are values for $y_1, \ldots, y_m$ that
make all R_i and C true in the database.

$\Rightarrow$ arguments of R_i among x_i's and y_i's.

$\Rightarrow$ arguments of e_j's and C bound in R_i's.

Operational

$$\pi_{e_1,\ldots,e_k}\sigma_C(R_1 \times \ldots \times R_n)$$

$\Rightarrow$ for every tuple $t_1 \in R_1$ and . . . and every tuple $t_n \in R_n$

let $t := t_1 \cdot \ldots \cdot t_n$ in

if $C(t)$ then

output $(e_1(t), \ldots, e_k(t))$;

done

Summary

- simple SELECT block accounts for many queries

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

- additional features

  $\Rightarrow$ assigning alternative names to relations

  $\Rightarrow$ expressions and naming in the output

  $\Rightarrow$ built-in atomic predicates and boolean connectives

- well defined semantics (declarative and operational)