SQL

Part 1: The Basics

University of Waterloo
List of Slides

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

- Structured Query Language
  - Developed in IBM Almaden (system R)
- Based on
  - (Conjunctive) queries in Relational Calculus
  - Set/Bag semantics and operations
  - Aggregation
- **BAG SEMANTICS**
- A **committee** design
  - choices often more “pragmatic” than “logical”
  - several *standard* versions:
    - SQL/89, **SQL/92** = SQL2, SQL3, . . .
- creeping featuralism
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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>integer (32 bit)</td>
</tr>
<tr>
<td>smallint</td>
<td>integer (16 bit)</td>
</tr>
<tr>
<td>decimal(m,n)</td>
<td>fixed decimal</td>
</tr>
<tr>
<td>float</td>
<td>IEEE float (32 bit)</td>
</tr>
<tr>
<td>char(n)</td>
<td>character string (length n)</td>
</tr>
<tr>
<td>varchar(n)</td>
<td>variable length string (at most n)</td>
</tr>
<tr>
<td>date</td>
<td>year/month/day</td>
</tr>
<tr>
<td>time</td>
<td>hh:mm:ss.ss</td>
</tr>
</tbody>
</table>

\[ \text{decimal}(m,n) : m \text{ number of digits, } n \text{ number of digits after .}; \]
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)
The “SELECT Block”

Basic syntax:

\[
\text{SELECT } \langle \text{results} \rangle \\
\text{FROM } \langle \text{tables} \rangle \\
\text{WHERE } \langle \text{condition} \rangle \\
\]

- Allows formulation of $\exists, \land$-queries
  - a conjunction of $\langle \text{tables} \rangle$ and $\langle \text{condition} \rangle$
  - attributes not in $\langle \text{result} \rangle$ existentially quantified
  - $\langle \text{result} \rangle$ specifies values in the resulting tuples
Example

List all authors in the database:

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

<table>
<thead>
<tr>
<th>AID</th>
<th>NAME</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Toman, David</td>
<td><a href="http://db.uwaterloo.ca/~david">http://db.uwaterloo.ca/~david</a></td>
</tr>
<tr>
<td>2</td>
<td>Chomicki, Jan</td>
<td><a href="http://cs.monmouth.edu/~chomicki">http://cs.monmouth.edu/~chomicki</a></td>
</tr>
<tr>
<td>3</td>
<td>Saake, Gunter</td>
<td></td>
</tr>
</tbody>
</table>

The **FROM** clause cannot be used on its own

⇒ the **"SELECT *" notation**
⇒ also reveals all attribute names
Naming Attributes

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

  \[
  \Rightarrow \text{publication}(\text{pubid}, \ldots )
  \]
  \[
  \Rightarrow \text{book}(\text{pubid}, \ldots )
  \]

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

  \[
  \{ n : \exists p, x, y. \text{publication}(p, n) \land \text{book}(p, x, y) \}
  \]

  \Rightarrow \text{we prefix the ambiguous attributes by name of the appropriate relation, e.g.,}
  \[
  \text{publication.pubid} \text{ (first "p")}
  \]
  \[
  \text{book.pubid} \text{ (second "p")}
  \]
Example

List titles of all books:

```sql
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

\[
\{ x : \exists y_1, y_2. \text{wrote}(y_1, x) \land \text{wrote}(y_2, x) \land y_1 \neq y_2 \} 
\]

⇒ problem: \( y_1 \) and \( y_2 \) are both called \text{pubid}
⇒ ... and they both appear only in a \text{wrote} relation

- solution: corelation names in the **FROM** clause

⇒ e.g., \texttt{FROM wrote r1, wrote r2} makes

\[
y_1 = r1.\text{author} \quad y_2 = r2.\text{author}
\]
⇒ \textit{r1} and \textit{r2} are \textit{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:

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

where:

- \( R_i \) are relation (table) names
- \( n_i \) are (distinct) identifiers
- the clause represents a conjunction \( R_1 \land \ldots \land R_k \)
  \[\Rightarrow\] 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:

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

- Allows us to
  1. eliminate superfluous attributes from answers ($\exists$)
  2. form expressions:
     \[ \Rightarrow \text{ built-in functions applied to values of attributes} \]
  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 \text{ (usual arithmetic)}

- on strings:
  
  \(\|\) \text{ (concatenation)}, \text{ `substr`, \ldots}

- constants (of appropriate types)
  
  \Rightarrow \text{"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;
```

<table>
<thead>
<tr>
<th>PUBID</th>
<th>ENDPAGE-STARTPAGE+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChTo98</td>
<td>40</td>
</tr>
<tr>
<td>ChTo98a</td>
<td>28</td>
</tr>
<tr>
<td>Tom97</td>
<td>19</td>
</tr>
</tbody>
</table>
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:

⇒ "<expr> AS <id>" where <id> is the new name
Example

and name the resulting attributes \texttt{id, numberofpages}:

\begin{verbatim}
SQL> select pubid as id,  
  2    endpage-startpage+1 as numberofpages  
  3  from article;
\end{verbatim}

\begin{tabular}{l|l}
\textbf{ID} & \textbf{NUMBEROFPAGES} \\
\hline
ChTo98 & 40 \\
ChTo98a & 28 \\
Tom97 & 19 \\
\end{tabular}
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*  
  ⇒ similar conditions as in the **SELECT** clause
Example(s)

Find all journals printed since 1997:

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

<table>
<thead>
<tr>
<th>PUBID</th>
<th>VOLUME</th>
<th>NO</th>
<th>YEAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>JLP-3-98</td>
<td>35</td>
<td>3</td>
<td>1998</td>
</tr>
</tbody>
</table>
```

Find all articles with more than 20 pages:

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

<table>
<thead>
<tr>
<th>PUBID</th>
<th>CROSSREF</th>
<th>STARTPAGE</th>
<th>ENDPAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChTo98</td>
<td>ChSa98</td>
<td>31</td>
<td>70</td>
</tr>
<tr>
<td>ChTo98a</td>
<td>JLP-3-98</td>
<td>263</td>
<td>290</td>
</tr>
</tbody>
</table>
```
Boolean Connectives

Atomic conditions in the \texttt{WHERE} clause can be combined using \textbf{boolean connectives}:

- \textbf{AND} conjunction
- \textbf{OR} disjunction
- \textbf{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) \land \ldots \land R_n(\ldots) \land C\}\}

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.

arguments of $R_i$ among $x_i$’s and $y_i$’s.
arguments of $e_j$’s and $C$ bound in $R_i$’s.
Operational

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

⇒ for every tuple \( t_1 \in R_1 \) and \( \ldots \) 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
  ⇒ all in $\exists, \land$ fragment of relational calculus

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

- well defined semantics (declarative and operational)