General Overview - rel. model

- Formal query languages
  - rel algebra and calculi
- Commercial query languages
  - SQL
  - QBE, (QUEL)

Overview - detailed - SQL

- DML
  - select, from, where, renaming
  - set operations
  - ordering
  - aggregate functions
  - nested subqueries
- other parts: DDL, embedded SQL, auth etc
Relational Query Languages

• A major strength of the relational model: supports simple, powerful *querying* of data.
• Two sublanguages:
  • DDL – Data Definition Language
    – define and modify schema (at all 3 levels)
  • DML – Data Manipulation Language
    – Queries can be written intuitively.

Relational languages

• The DBMS is responsible for efficient evaluation.
  – Query optimizer: re-orders operations and generates query plan

The SQL Query Language

• The most widely used relational query language.
  – Major standard is SQL-1999 (=SQL3)
    • Introduced “Object-Relational” concepts
    • SQL 2003, SQL 2008 have small extensions
  – SQL92 is a basic subset
SQL (cont’d)

– PostgreSQL has some “unique” aspects (as do most systems).
– XML is the next challenge for SQL.

DML

General form

\[
\text{select } a_1, a_2, \ldots, a_n \newline
\text{from } r_1, r_2, \ldots, r_m \newline\text{where } P \newline\text{[order by ….]} \newline\text{[group by ….]} \newline\text{[having ….]} \]

Reminder: our Mini-U db

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>NAME</td>
</tr>
<tr>
<td>123</td>
<td>smith</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
</tr>
<tr>
<td>123</td>
</tr>
<tr>
<td>234</td>
</tr>
</tbody>
</table>
DML - eg:
find the ssn(s) of everybody called “smith”

```
select ssn
from student
where name="smith"
```

DML - observation

General form

```
select a1, a2, … an
from r1, r2, … rm
where P
```

Is this equivalent to a relational algebra query?

DML - observation

General form

```
select a1, a2, … an
from r1, r2, … rm
where P
```

\[
\pi_{a_1, a_2, \ldots, an}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))
\]
DML - observation

General form

\[
\pi_{a_1, a_2, \ldots, a_n}(\sigma_{P}(r_1 \times r_2 \times \ldots \times r_m))
\]

select clause

\[
\text{select}\ [\text{distinct} | \text{all}]\ \text{name}
\text{from student}
\text{where address="main"}
\]

where clause

\[
\text{find ssn(s) of all "smith"s on "main"}
\text{select ssn}
\text{from student}
\text{where address="main" and}
\text{name = "smith"}
\]
where clause

- boolean operators \( \text{and or not} \ldots \)
- comparison operators \( <, >, =, \ldots \)
- and more…

What about strings?

find student ssns who live on “main” (st or str or street - ie., “main st” or “main str” …)

```
select ssn
from student
where address like “main%”
%
_: variable-length don’t care
_: single-character don’t care
```
from clause

find names of people taking 15-415

select name
from student, takes
where ???

from clause

find names of people taking 15-415

select name
from student, takes
where student.ssn = takes.ssn and
takes.c-id = "15-415"
renaming - tuple variables

find names of people taking 15-415

select name
from ourVeryOwnStudent, studentTakingClasses
where ourVeryOwnStudent.ssn = studentTakingClasses.ssn
and studentTakingClasses.c-id = "15-415"

renaming - tuple variables

find names of people taking 15-415

select name
from ourVeryOwnStudent as S,
studentTakingClasses as T
where S.ssn = T.ssn
and T.c-id = "15-415"

renaming - self-join

self-joins: find Tom's grandparent(s)
renaming - self-join

find grandparents of “Tom” (PC(p-id, c-id))

```sql
select gp.p-id
from PC as gp, PC
where gp.c-id = PC.p-id
and PC.c-id = "Tom"
```

renaming - theta join

find course names with more units than 15-415

```sql
select c1.c-name
from class as c1, class as c2
where c1.units > c2.units
and c2.c-id = "15-415"
```
find course names with more units than 15-415

```sql
select c1.name
from class as c1, class as c2
where c1.units > c2.units
and c2.c-id = "15-415"

\{ t | \exists c1 \in \text{CLASS} \exists c2 \in \text{CLASS} ( \\
    c1[c - id] = 15 - 415 \land \\
    c2[units] > c1[units] \land \\
    t[c - name] = c2[c - name] ) \}
```

find course names with more units than 15-415

```sql
select c2.name
from class as c1, class as c2
where c2.units > c1.units
and c1.c-id = "15-415"

\{ t | \exists c1 \in \text{CLASS} \exists c2 \in \text{CLASS} ( \\
    c1[c - id] = 15 - 415 \land \\
    c2[units] > c1[units] \land \\
    t[c - name] = c2[c - name] ) \}
```

Overview - detailed - SQL

- DML
  - select, from, where
  - set operations
  - ordering
  - aggregate functions
  - nested subqueries
- other parts: DDL, embedded SQL, auth etc
set operations

find ssn of people taking both 15-415 and 15-413

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-413</td>
<td>A</td>
</tr>
<tr>
<td>234</td>
<td>16-413</td>
<td>B</td>
</tr>
</tbody>
</table>

select ssn from takes where c-id="15-415" and c-id="15-413"

other ops: union, except
Overview - detailed - SQL

- DML
  - `select`, `from`, `where`
  - set operations
  - ordering
  - aggregate functions
  - nested subqueries
- other parts: DDL, embedded SQL, auth etc

Ordering
find student records, sorted in name order
```sql
select *
from student
order by name asc
```

asc is the default
Ordering

find student records, sorted in name order; break ties by reverse ssn

\[
\text{select * from student order by name, ssn desc}
\]

Overview - detailed - SQL

- DML
  - select, from, where
  - set operations
  - ordering
  - aggregate functions
  - nested subqueries
- other parts: DDL, embedded SQL, auth etc

Aggregate functions

find avg grade, across all students

\[
\text{select ?? from takes}
\]
Aggregate functions

find avg grade, across all students

select avg(grade)
from takes

• result: a single number
• Which other functions?

### Table

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>16-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>3</td>
</tr>
</tbody>
</table>

---

Aggregate functions

• A: sum count min max (std)

---

Aggregate functions

find total number of enrollments

select count(*)
from takes

### Table

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>16-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>3</td>
</tr>
</tbody>
</table>
Aggregate functions

find total number of students in 15-415

\[
\text{select count(*) from takes where c-id="15-415"}
\]

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>3</td>
</tr>
</tbody>
</table>

Aggregate functions

find total number of students in each course

\[
\text{select count(*) from takes where ???}
\]

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>3</td>
</tr>
</tbody>
</table>

Aggregate functions

find total number of students in each course

\[
\text{select c-id, count(*) from takes group by c-id}
\]

<table>
<thead>
<tr>
<th>c-id</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-413</td>
<td>2</td>
</tr>
</tbody>
</table>
Aggregate functions

find total number of students in each course

\[
\text{select c-id, count(*) from takes group by c-id order by c-id}
\]

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>16-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>16-413</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c-id</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-413</td>
<td>2</td>
</tr>
</tbody>
</table>

Aggregate functions

find total number of students in each course, and sort by count, decreasing

\[
\text{select c-id, count(*) as pop from takes group by c-id order by pop desc}
\]

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>16-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>16-413</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c-id</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-413</td>
<td>2</td>
</tr>
</tbody>
</table>

Aggregate functions- ‘having’

find students with GPA > 3.0

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>16-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>16-413</td>
<td>3</td>
</tr>
</tbody>
</table>
Aggregate functions - ‘having’

find students with GPA > 3.0

```
select ???, avg(grade)
from takes
group by ???
```

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>3</td>
</tr>
</tbody>
</table>

‘having’ <-> ‘where’ for groups
Aggregate functions- ‘having’

find students and GPA, for students with > 5 courses

\[\text{select ssn, avg(grade)}\]
\[\text{from takes} \]
\[\text{group by ssn} \]
\[\text{having count(*) > 5}\]

<table>
<thead>
<tr>
<th>SSN</th>
<th>l-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-413</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>3</td>
</tr>
</tbody>
</table>

Overview - detailed - SQL

• DML
  – select, from, where
  – set operations
  – ordering
  – aggregate functions
  – nested subqueries
• other parts: DDL, embedded SQL, auth etc