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).

DML

General form

```
select a1, a2, … an
from r1, r2, … rm
where P
[order by ….]
[group by …]
[having …]
```
Reminder: our Mini-U db

<table>
<thead>
<tr>
<th>STUDENT</th>
<th>CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ssn</td>
<td>c-id</td>
</tr>
<tr>
<td>123</td>
<td>15-413</td>
</tr>
<tr>
<td>234</td>
<td>15-412</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”

\[
\text{select ssn} \\
\text{from student} \\
\text{where name="smith"}
\]

DML - observation

General form

\[
\text{select } a_1, a_2, \ldots, a_n \\
\text{from } r_1, r_2, \ldots, r_m \\
\text{where } P
\]

equivalent rel. algebra query?

DML - observation

General form

\[
\text{select } a_1, a_2, \ldots, a_n \\
\text{from } r_1, r_2, \ldots, r_m \\
\text{where } P
\]

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

**General form**

```
select [distinct | all] a_1, a_2, ... a_n
from r_1, r_2, ... r_m
where P
```

\[
\pi_{a_1, a_2, ... a_n} (\sigma_P (r_1 \times r_2 \times ... \times r_m))
\]

select clause

```
select [distinct | all] name
from student
where address = "main"
```

where clause

```
find ssn(s) of all "smith"s on "main"
select ssn
from student
where address = "main" and
    name = "smith"
```

where clause

- boolean operators (and, or, not, ...)
- comparison operators (<, >, =, ...)
- and more...
What about strings?

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

```
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

```sql
select name
from student, takes
where student.ssn = takes.ssn and
takes.c-id = "15-415"
```

Overview - detailed - SQL

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

renaming - tuple variables

find names of people taking 15-415

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

2 reasons to rename:

- #1) shorthand
- #2) self-joins
#1) renaming - tuple variables

find names of people taking 15-415

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

#2) renaming - self-join

• self-joins: find Tom’s grandparent(s)

```sql
select gp.p-id
from PC as gp, PC
where gp.c-id= PC.p-id
and PC.c-id = "Tom"
```
#2) renaming - theta join

find course names with more units than 15-415

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

find course names with more units than 15-415

\[
\begin{align*}
\text{select} & \quad c1.c-name \\
\text{from} & \quad \text{class as c1, class as c2} \\
\text{where} & \quad c1.units > c2.units \\
\text{and} & \quad c2.c-id = "15-415"
\end{align*}
\]
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

\[
\begin{align*}
\text{TAKES} & \\
\hline
\text{SSN} & \text{c-id} & \text{grade} \\
123 & 15-413 & A \\
234 & 15-413 & B \\
\end{align*}
\]

set operations

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

\[
(\text{select ssn from takes where c-id="15-415"}) \text{ intersect } (\text{select ssn from takes where c-id="15-413"})
\]

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

\[
\begin{align*}
\text{select} & \quad * \\
\text{from} & \quad \text{student} \\
\text{where} & \quad \text{name}\text{ asc}
\end{align*}
\]

\text{asc} \text{ is the default}

Ordering

find student records, sorted in name order;

break ties by reverse ssn

\[
\begin{align*}
\text{select} & \quad * \\
\text{from} & \quad \text{student} \\
\text{order by} & \quad \text{name, ssn desc}
\end{align*}
\]
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

\[
\text{select } \frac{\text{sum grades}}{\text{count students}} \text{ from takes}
\]

- result: a single number
- Which other functions?

• A: \text{sum count min max (std)}
Aggregate functions
find total number of enrollments

\[
\text{select } \text{count}(\ast) \\
\text{from takes}
\]

<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 15-415

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

<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

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

<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

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

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

find total number of students in each course

select c-id, count(*)

from takes

group by c-id

order by c-id

Aggregate functions

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

select c-id, count(*) as pop

from takes

group by c-id

order by pop desc

Aggregate functions- ‘having’

find students with GPA > 3.0

select ???, avg(grade)

from takes

group by ???
Aggregate functions- ‘having’

find students with GPA > 3.0
 select ssn, avg(grade)
 from takes
 group by ssn

<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>16-413</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>avg(grade)</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>3</td>
</tr>
</tbody>
</table>

‘having’ <-> ‘where’ for groups

Aggregate functions- ‘having’

find students with GPA > 3.0
 select ssn, avg(grade)
 from takes
 group by ssn
 having avg(grade)>3.0

<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>16-413</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>avg(grade)</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>4</td>
</tr>
<tr>
<td>234</td>
<td>3</td>
</tr>
</tbody>
</table>

Overview - detailed - SQL

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

General form

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

Reminder: our Mini-U db

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c-id</th>
<th>c-name</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
</tr>
</tbody>
</table>

DML - nested subqueries

find names of students of 15-415

\[
\text{select name} \\
\text{from student} \\
\text{where } ...
\]

“ssn in the set of people that take 15-415”

DML - nested subqueries

find names of students of 15-415

\[
\text{select name} \\
\text{from student} \\
\text{where } ............ \\
\text{select ssn} \\
\text{from takes} \\
\text{where } c-id = "15-415"
DML - nested subqueries

find names of students of 15-415

```
select name
from student
where ssn in (
    select ssn
    from takes
    where c-id = "15-415"
)
```

DML - nested subqueries

- ‘in’ compares a value with a set of values
- ‘in’ can be combined other boolean ops
- it is redundant (but user friendly!):

```
select name
from student
where c-id = "15-415"
and student.ssn = takes.ssn
```

DML - nested subqueries

find names of students of 15-415

```
select name
from student
where ssn in (
    select ssn
    from takes
    where c-id = "15-415"
)
```

DML - nested subqueries

- ‘in’ compares a value with a set of values
- ‘in’ can be combined other boolean ops
- it is redundant (but user friendly!):

```
select name
from student
where c-id = "15-415"
and student.ssn = takes.ssn
```
DML - nested subqueries

- ‘in’ compares a value with a set of values
- other operators like ‘in’ ?

DML - nested subqueries

find student record with highest ssn

```sql
select *
from student
where ssn is greater than every other ssn
```

DML - nested subqueries

find student record with highest ssn

```sql
select *
from student
where ssn greater than every
select ssn from student
```

DML - nested subqueries

find student record with highest ssn

```sql
select *
from student
where ssn > all (select ssn from student)
```

almost correct
DML - nested subqueries

find student record with highest ssn

```
select *
from student
where ssn >= all (
    select ssn from student)
```

Puzzle

S1

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
</tr>
</tbody>
</table>

S2

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>smith</td>
<td>main str</td>
</tr>
<tr>
<td>234</td>
<td>jones</td>
<td>forbes ave</td>
</tr>
</tbody>
</table>

S1 x S2

S1.ssn > S2.ssn

gives all but the smallest ssn - aha!
DML - nested subqueries

find student record with highest ssn - without nested subqueries?

```sql
select S1.ssn, S1.name, S1.address
from student as S1, student as S2
where S1.ssn < S2.ssn
```
gives all but the highest - therefore….

DML - nested subqueries

find student record with highest ssn - without nested subqueries?

```sql
(select * from student) except
(select S1.ssn, S1.name, S1.address
from student as S1, student as S2
where S1.ssn < S2.ssn)
```

DML - nested subqueries

(\(\text{select * from student}\) except
(\(\text{select S1.ssn, S1.name, S1.address}
\text{from student as S1, student as S2}
\text{where S1.ssn < S2.ssn}\)))

select *
from student
where ssn  \(\geq\) all (\(\text{select ssn from student}\))
DML - nested subqueries

Drill: Even more readable than
\[
\begin{align*}
\text{select} & \ast \text{ from student} \\
\text{where} & \text{ ssn} \geq \text{ all (select ssn from student)} \\
\text{select} & \ast \text{ from student} \\
\text{where} & \text{ ssn in} \\
(\text{select max(ssn) from student})
\end{align*}
\]

DML - nested subqueries

Drill: find the ssn of the student with the highest GPA

\[
\begin{array}{|c|c|c|c|}
\hline
\text{STUDENT} & \text{CLASS} \\
\text{Ssn} & \text{Name} & \text{Address} & \text{c-id} & \text{c-name} & \text{units} \\
\hline
123 & smith & main str & 15-413 & s.e. & 2 \\
234 & jones & forbes ave & 15-412 & o.s. & 2 \\
\hline
\end{array}
\]

DML - nested subqueries

Drill: find the ssn and GPA of the student with the highest GPA

\[
\begin{align*}
\text{select ssn, } & \text{avg(grade) from takes} \\
\text{group by ssn} \quad \text{having avg(grade) .......} \\
& \text{greater than every other GPA on file}
\end{align*}
\]
Drill: find the ssn and GPA of the student with the highest GPA

```sql
select ssn, avg(grade) from takes
  group by ssn
having avg(grade) >= all
  ( select avg(grade) 
      from student 
      group by ssn )
```

• `in` and `>= all` compares a value with a set of values
• other operators like these?

• `<all>()`, `<=>all()` ...
• `<>all` is identical to `not in`
• `>some()`, `>= some()` ...
• `= some()` is identical to `in`
• `exists`
DML - nested subqueries

Drill for ‘exists’: find all courses that nobody enrolled in

```sql
select c-id from class
where not exists
  (select * from takes
   where class.c-id = takes.c-id)
```

DML - derived relations

find the ssn with the highest GPA

```sql
select ssn, avg(grade) from takes
  group by ssn
having avg(grade) >= all
  (select avg(grade) from takes
   group by ssn)
```

DML - derived relations

find the ssn with the highest GPA

Query would be easier, if we had a table like:
```
helpfulTable (ssn, gpa):
```

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>3.5</td>
</tr>
<tr>
<td>678</td>
<td>3.3</td>
</tr>
</tbody>
</table>

then what?

```sql
select ssn, gpa
from helpfulTable
where gpa in (select max(gpa) from helpfulTable)
```
DML - derived relations

find the ssn with the highest GPA
Query for helpfulTable (ssn, gpa)?

select ssn, avg(grade)
from takes
group by ssn

DML - derived relations

find the ssn with the highest GPA
Query for helpfulTable (ssn, gpa)?

select ssn, gpa
from (select ssn, avg(grade)
from takes
group by ssn)
as helpfulTable(ssn, gpa)
where gpa = (select max(gpa)
from helpfulTable)

DML - derived relations

find the ssn with the highest GPA
helpfulTable(ssn, gpa)
select ssn, gpa
from helpfulTable
where gpa = (select max(gpa)
from helpfulTable)
Overview - detailed - SQL

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

Next lecture:

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