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, ordering,
  - aggregate functions, nested subqueries
  - insertion, deletion, update
- other parts: DDL, authorization, triggers
- embedded SQL

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 - insertions etc

```
insert into student
values ("123", "smith", "main")
```

```
insert into student(ssn, name, address)
values ("123", "smith", "main")
```

DML - insertions etc

bulk insertion: how to insert, say, a table of ‘foreign-student’s, in bulk?

DML - insertions etc

bulk insertion:

```
insert into student
select ssn, name, address
from foreign-student
```

DML - deletion etc

delete the record of ‘smith’
DML - deletion etc

delete the record of ‘smith’:

```
delete from student
where name='smith'
```

(careful - it deletes ALL the ‘smith’!)
Joins

```sql
select [column list]
from table_name
[inner | {left | right | full} outer ] join
table_name
on qualification_list
where...
```

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>
<tr>
<td>15-413</td>
<td>s.e.</td>
</tr>
<tr>
<td>15-412</td>
<td>o.s.</td>
</tr>
</tbody>
</table>

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

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

Inner join

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

Outer join

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

o.s.: gone!
Outer join

```sql
select ssn, c-name
from takes
right outer join class
on takes.c-id = class.c-id
```

Null Values

- **null** -> unknown, or inapplicable, (or …)
- Complications:
  - 3-valued logic (true, false and unknown).
  - **null = null** : false!!

Overview - detailed - SQL

- **DML**
  - select, from, where, renaming, ordering,
  - aggregate functions, nested subqueries
  - insertion, deletion, update
- other parts: **DDL**, authorization, triggers
- embedded SQL
Data Definition Language

```sql
create table student
(ssn char(9) not null,
 name char(30),
 address char(50),
 primary key (ssn) )
```

Data Definition Language

```sql
create table r( A1 D1, …, An Dn,
 integrity-constraint1,
 …
 integrity-constraint-n)
```

Data Definition Language

Domains:
- `char(n), varchar(n)`
- `int, numeric(p,d), real, double precision`
- `float, smallint`
- `date, time`

Data Definition Language

delete a table: difference between
```
drop table student
```
```
delete from student
```
Data Definition Language

modify a table:

**alter table** student **drop** address

**alter table** student **add** major char(10)

---

Data Definition Language

**integrity constraints:**

- **primary key**
- **foreign key**
- **check(P)**

---

Data Definition Language

**create table** takes

(ssn **char(9)** not null,
 c-id **char(5)** not null,
 grade **char(1)**,
 **primary key** (ssn, c-id),

---

Referential Integrity constraints

‘foreign keys’ - eg:

**create table** takes(

  ssn **char(9)** not null,
  c-id **char(5)** not null,
  grade **integer**, 
  **primary key**(ssn, c-id),
  **foreign key** ssn **references** student,
  **foreign key** c-id **references** class)
Referential Integrity constraints

... foreign key ssn references student,
foreign key c-id references class)
Effect:
– expects that ssn to exist in ‘student’ table
– blocks ops that violate that - how??
  • insertion?
  • deletion/update?

Overview - detailed - SQL

• DML
  – select, from, where, renaming, ordering,
  – aggregate functions, nested subqueries
  – insertion, deletion, update
• other parts: DDL, authorization, triggers
• embedded SQL

Weapons for IC:

• assertions
  – create assertion <assertion-name> check
  <predicate>
• triggers (~ assertions with ‘teeth’)
  – on operation, if condition, then action
**Triggers - example**

```sql
define trigger zerochange on update takes
(if new takes.grade < 0
  then takes.grade = 0)
```

**Triggers - discussion**

- more complicated: “managers have higher salaries than their subordinates” - a trigger can automatically boost mgrs salaries
- triggers: tricky (infinite loops…)

**Overview - detailed - SQL**

- **DML**
  - select, from, where, renaming, ordering,
  - aggregate functions, nested subqueries
  - insertion, deletion, update
- other parts: DDL, **authorization**, triggers
- embedded SQL

**Authorization**

- **grant** `<priv.-list>` **on** `<table-name>` **to** `<user-list>`
- privileges for tuples: ??
- privileges for tables: ??
Authorization

- `grant <priv.-list> on <table-name> to <user-list>`
- privileges for tuples: read / insert / delete / update
- privileges for tables: create, drop, index

Authorization – cont’d

- variations:
  - `with grant option`
  - `revoke <priv.-list> on <t-name> from <user_ids>`
- Eg:
  - `grant read, update on takes to csSecy with grant option`

Overview - detailed - SQL

- DML
  - select, from, where, renaming, ordering,
  - aggregate functions, nested subqueries
  - insertion, deletion, update
- other parts: DDL/views, authorization, triggers
- embedded SQL

Views

`create view helpfulTable(ssn, gpa) as select ssn, avg(grade) from takes group by ssn`

find the ssn with the highest GPA - we can create a permanent, virtual table:
**Views**

- views are recorded in the schema, for ever (ie., until ‘drop view’)
- typically, they take little disk space, because they are computed on the fly
- (but: materialized views…)

**DML - view update**

consider the student_gpa view:

```sql
create view student_gpa as
    (select ssn, avg(grade) as gpa from takes)
```

view updates are tricky - typically, we can only update views that have no joins, nor aggregates

Example?

<table>
<thead>
<tr>
<th>ssn</th>
<th>c-id</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>15-415</td>
<td>4</td>
</tr>
<tr>
<td>123</td>
<td>15-412</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ssn</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>3.5</td>
</tr>
</tbody>
</table>

**Overview - detailed - SQL**

- DML
  - select, from, where, renaming, ordering,
  - aggregate functions, nested subqueries
  - insertion, deletion, update
- other parts: DDL, authorization, triggers
- **embedded SQL**; application development

**Embedded SQL**

from within a ‘host’ language (eg., ‘C’, ‘VB’)

```sql
EXEC SQL <emb. SQL stmt> END-EXEC
```

Q: why do we need embedded SQL??
Embedded SQL

from within a ‘host’ language (eg., ‘C’, ‘VB’)
EXEC SQL <emb. SQL stmt> END-EXEC

Q: why do we need embedded SQL??
A1: pretty formatting (eg., receipts, paychecks)
A2: plots
A3: forecasting; fancy math/stat computations

Overview - detailed - SQL

• DML
  – select, from, where, renaming, ordering,
  – aggregate functions, nested subqueries
  – insertion, deletion, update
• other parts: DDL, authorization, triggers
• embedded SQL; application development

Overview

• concepts of SQL programs
• walkthrough of embedded SQL example

Outline of an SQL application

• establish connection with db server
• authenticate (user/password)
• execute SQL statement(s)
• process results
• close connection
Check python code

- [link to csv2sql.py example]
- [link to instructions page]

```python
import sqlite3
import csv
fname = 'tst.csv'
dname = 'tst.db'

conn = sqlite3.connect(dname)
conn.execute('create table if not exists tst (name text, address text, state text, age integer);')
```

```python
print '---- csv2sql inserted', fname
for row in cur:
    print row
```

The code is available online at the provided links.
Conclusions

Outline of an SQL application:
• establish connection with db server
• authenticate (user/password)
• execute SQL statement(s) (using cursors)
• process results
• close connection