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>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
<td>c-id</td>
<td>SSN</td>
</tr>
<tr>
<td>123</td>
<td>15-413</td>
<td>123</td>
</tr>
<tr>
<td>234</td>
<td>15-413</td>
<td>234</td>
</tr>
<tr>
<td>Name</td>
<td>name</td>
<td>c-id</td>
</tr>
<tr>
<td>smith</td>
<td>smith</td>
<td>15-413</td>
</tr>
<tr>
<td>Address</td>
<td>address</td>
<td>name</td>
</tr>
<tr>
<td>main str</td>
<td>main</td>
<td>smith</td>
</tr>
<tr>
<td>str</td>
<td>str</td>
<td>name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>address</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:

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

DML - deletion etc

delete the record of 'smith'

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

(careful - it deletes ALL the 'smith's!)
DML - update etc

record the grade ‘A’ for ssn=123 and course 15-415

update takes
set grade="A"
where ssn="123" and c-id="15-415"

(will set to “A” ALL such records)

DML - view update

consider the db-takes view:
create view db-takes as
(select * from takes where c-id="15-415")

view updates are tricky - typically, we can only update views that have no joins, nor aggregates even so, consider changing a c-id to 15-222...

DML - joins

so far: ‘INNER’ joins, eg:

select ssn, c-name
from takes, class
where takes.c-id = class.c-id
DML - joins

Equivalently:

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

Joins

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

<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>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>s.e.</td>
</tr>
<tr>
<td>234</td>
<td>s.e.</td>
</tr>
</tbody>
</table>

o.s.: gone!

**Outer join**

<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>15-413</td>
<td>B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>s.e.</td>
</tr>
<tr>
<td>234</td>
<td>s.e.</td>
</tr>
<tr>
<td>null</td>
<td>o.s.</td>
</tr>
</tbody>
</table>

**Outer join**

```sql
SELECT ssn, c-name
FROM takes
RIGHT OUTER JOIN class ON takes.c-id = class.c-id
```

<table>
<thead>
<tr>
<th>SSN</th>
<th>c-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>s.e.</td>
</tr>
<tr>
<td>234</td>
<td>s.e.</td>
</tr>
<tr>
<td>null</td>
<td>o.s.</td>
</tr>
</tbody>
</table>
Outer join

- left outer join
- right outer join
- full outer join
- natural join

Null Values

- null \(\rightarrow\) 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

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

Data Definition Language

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),
check grade in ("A", "B", "C", "D", "F")

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)

Effect:
– expects that ssn to exist in ‘student’ table
– blocks ops that violate that - how??
  • insertion?
  • deletion/update?
Referential Integrity constraints

... foreign key ssn references student
  on delete cascade
  on update cascade,
...
• -> eliminate all student enrollments
• other options (set to null, to default etc)

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 zerograde 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: 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>`

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’)
EXEC SQL <emb. SQL stmt> END-EXEC

Q: why do we need embedded SQL??

Embedded SQL

SQL returns sets; host language expects a tuple - impedance mismatch!

solution: ‘cursor’, ie., a ‘pointer’ over the set of tuples.

example:

```c
main()
{
  ...
  EXEC SQL
      declare c cursor for
      select * from student
  END-EXEC
  ...
```
Embedded SQL - ctn’d

... EXEC SQL open c END-EXEC
...
while( !sqlerror ){
   EXEC SQL fetch c into :cssn, :cname, :cad END-EXEC
   fprintf( … , cssn, cname, cad);
}

Dynamic SQL

main() /* set all grades to user’s input */
... char *sqlcmd=" update takes set grade = ?";
EXEC SQL prepare dynsql from :sqlcmd ;
char inputgrade[5]="a";
EXEC SQL execute dynsql using :inputgrade;
... */ end main() */
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
Pictorially:

*andrew machine*

\[ \text{e.g.} \]

\[ \text{newcastle.db.cs.cmu.edu} \]

\[ \text{Linux box} \]

\[ \text{with postgres server} \]

---

Check python code

- Or follow instructions at
```python

# Author: Christos Faloutsos
# Program to illustrate process to create a file
# for a query, and create a write and query file
# and store it for queries. For the
# process to illustrate process to create a file

import sqlite3

test = 'test.db'

def create_table():
    conn = sqlite3.connect('test.db')
    cur = conn.cursor()
    cur.execute("CREATE TABLE IF NOT EXISTS test (name text, address text, state text, a city integer)"
```

```python

Faloutsos CMU SCS 15-415/615 852

```

```sql

print '--- csv2sql inserted ', fname
print '--- printing all tuples --- ' "cur = conn.cursor()
cur.execute('select * from test')
for row in cur:
    for elem in row:
        print elem, ', ',
    print ''
```

```sql

Faloutsos CMU SCS 15-415/615 853

```
Conclusions

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