CMU SCS

Carnegie Mellon Univ.
Dept. of Computer Science
15-415/615 - DB Applications

C. Faloutsos

Lecture#7 (cont'd): *Rel. model - SQL part3*

CMU SCS

General Overview - rel. model

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

CMU SCS

Overview - detailed - SQL

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

CMU SCS

Reminder: our Mini-U db

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

CLASS		
c-id	c-name	units
15-413	s.e.	2
15-412	o.s.	2

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

CMU SCS

DML - insertions etc

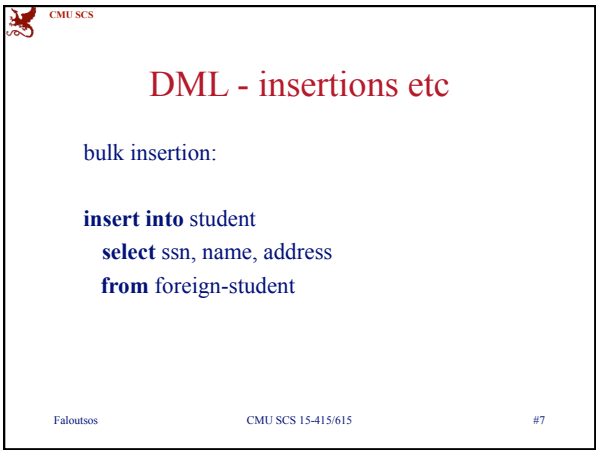
insert into student
values (“123”, “smith”, “main”)

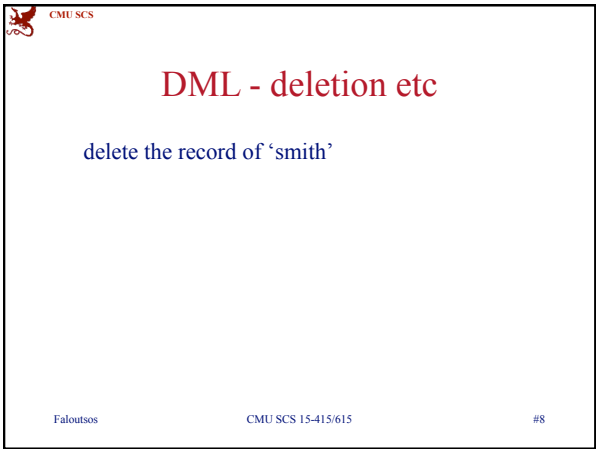
insert into student(ssn, name, address)
values (“123”, “smith”, “main”)

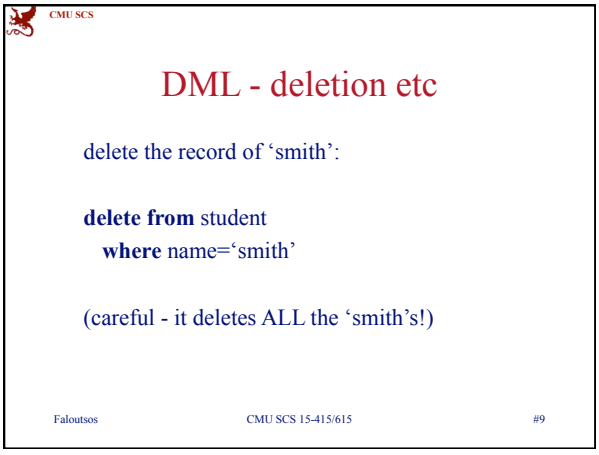
CMU SCS

DML - insertions etc

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


CMU SCS
DML - insertions etc
bulk insertion:
insert into student
select ssn, name, address
from foreign-student
Faloutsos
CMU SCS 15-415/615
#7


CMU SCS
DML - deletion etc
delete the record of 'smith'
Faloutsos
CMU SCS 15-415/615
#8


CMU SCS
DML - deletion etc
delete the record of 'smith':
delete from student
where name='smith'
(careful - it deletes ALL the 'smith's!)
Faloutsos
CMU SCS 15-415/615
#9

CMU SCS

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)

CMU SCS

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...

CMU SCS

DML - joins

so far: 'INNER' joins, eg:

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

CMU SCS

DML - joins

Equivalently:

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

CMU SCS

Joins

select [column list]
from *table_name*
[**inner** | {**left** | **right** | **full**} **outer**] **join**
table_name
on *qualification_list*
where…

CMU SCS

Reminder: our Mini-U db

STUDENT		
Ssn	Name	Address
123	smith	main str
234	jones	forbes ave

CLASS		
c-id	c-name	units
15-413	s.e.	2
15-412	o.s.	2

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

CMU SCS

Inner join

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

CLASS		
c-id	c-name	units
15-413	s.e.	2
15-412	o.s.	2

SSN	c-name
123	s.e
234	s.e

o.s.: gone!

CMU SCS

Outer join

TAKES		
SSN	c-id	grade
123	15-413	A
234	15-413	B

CLASS		
c-id	c-name	units
15-413	s.e.	2
15-412	o.s.	2

SSN	c-name
123	s.e
234	s.e.
null	o.s.

CMU SCS

Outer join

select ssn, c-name

from takes **right outer join** class **on** takes.c-id=class.c-id

SSN	c-name
123	s.e
234	s.e.
null	o.s.

CMU SCS

Outer join

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

CMU SCS

Null Values

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

CMU SCS

Overview - detailed - SQL

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

CMU SCS

Data Definition Language

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

CMU SCS

Data Definition Language

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

CMU SCS

Data Definition Language

Domains:

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

CMU SCS

Data Definition Language

delete a table: difference between

drop table student

delete from student

CMU SCS

Data Definition Language

modify a table:

alter table student **drop** address

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

CMU SCS

Data Definition Language

integrity constraints:

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

CMU SCS

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"))

CMU SCS

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)

CMU SCS

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?

CMU SCS

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)

CMU SCS

Overview - detailed - SQL

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

CMU SCS

Weapons for IC:

- assertions
  - **create assertion** <assertion-name> **check** <predicate>
- triggers (~ assertions with 'teeth')
  - on operation, if condition, then action

CMU SCS

Triggers - example

define trigger zerograde **on update** takes
(**if new** takes.grade < 0
then takes.grade = 0)

CMU SCS

Triggers - discussion

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

CMU SCS

Overview - detailed - SQL

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

CMU SCS

Authorization

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

CMU SCS

Authorization – cont'd

- variations:
  - **with grant option**
  - **revoke** <priv.-list> **on <t-name> from** <user_ids>

CMU SCS

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

CMU SCS

Embedded SQL

from within a 'host' language (eg., 'C', 'VB')
EXEC SQL <emb. SQL stmnt> END-EXEC

Q: why do we need embedded SQL??

CMU SCS

Embedded SQL

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

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

example:

CMU SCS

Embedded SQL

```
main(){
…
EXEC SQL
  declare c cursor for
  select * from student
END-EXEC
…
```

CMU SCS

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);
}
```


CMU SCS

Embedded SQL - ctn'd

```
…
EXEC SQL   close c END-EXEC
…
} /* end main() */
```


CMU SCS

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() */
```

CMU SCS

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**

CMU SCS

Overview

- concepts of SQL programs
- walkthrough of embedded SQL example

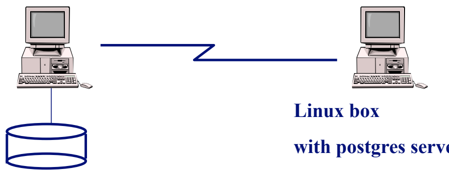
CMU SCS

Outline of an SQL application

- establish connection with db server
- authenticate (user/password)
- execute SQL statement(s)
- process results
- close connection

CMU SCS

Pictorially:



CMU SCS

```
csv2sql.py                                                          Page 1

#!/usr/bin/python

##################################################
# Author: christos faloutsos
# Date: Jan. 2012
# Purpose: Mainly wants to illustrate cursors
#          Specifically,
#             * expects a csv file, and
#             * loads it into a sqlite db file
#             And answers a few queries, for fun
##################################################


import sqlite3
import csv

fname='tst.csv'
dbname='tst.db'

# conn = sqlite3.connect(':memory:')
conn = sqlite3.connect(dbname)

conn.execute('create table if not exists tst (name text, address text, state text, s
alary integer)')

csvReader = csv.reader(open(fname), delimiter=',', quotechar='"')

print " --- csv2sql started on ", fname
for row in csvReader:
        conn.execute('insert into tst values' +
        '(?, ?, ?, ?)', row)

print " --- csv2sql inserted ", fname
print " "
print " --- printing all tuples --- "
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""

print " "
print " --- printing rich tuples (sal>2.5K), sorted  --- "
cur.execute('select * from tst where salary > 2500 order by salary desc ')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""

print " "
print " --- printing sum-salary per state ----"
cur.execute('select state, sum(salary) from tst  group by state')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""

conn.commit()
cur.close()
conn.close()
```


CMU SCS

Check python code

- http://www.cs.cmu.edu/~christos/courses/dbms.S13/PYTHON-examples/csv2sql.py
- Or follow instructions at
- http://www.cs.cmu.edu/~christos/courses/dbms.S13/PYTHON-examples/

CMU SCS

```
--
##################################################
# Author: christos faloutsos
# Date: Jan. 2012
# Purpose: Mainly wants to illustrate cursors
#          Specifically,
#             * expects a csv file, and
#             * loads it into a sqlite db file
#          And answers a few queries, for fun
##################################################

import sqlite3
import csv

fname='tst.csv'
dbname='tst.db'

# conn = sqlite3.connect(':memory:')
conn = sqlite3.connect(dbname)

conn.execute('create table if not exists tst (name text, address text, state text, s
alary integer)')
```


CMU SCS

```
print " --- csv2sql inserted ", fname
print " "
print " --- printing all tuples --- "
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""
```


CMU SCS

```
print " --- csv2sql inserted ", fname
print " "
print " --- printing all tuples --- "
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""
```

CMU SCS

```
print " --- csv2sql inserted ", fname
print " "
print " --- printing all tuples --- "
cur = conn.cursor()
cur.execute('select * from tst')
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""
```


CMU SCS

```
print " "
print " --- printing sum-salary per state ----"
cur.execute('select state, sum(salary) from tst  grou
for row in cur:
    for elem in row:
        print elem, "\t",
    print ""

conn.commit()
cur.close()
conn.close()
```


CMU SCS

Conclusions

Outline of an SQL application:

- establish connection with db server
- authenticate (user/password)
- execute SQL statement(s) (using **cursors**)
- process results
- close connection