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>
<tr>
<td>Name</td>
<td>c-name</td>
</tr>
<tr>
<td>smith</td>
<td>s.e.</td>
</tr>
<tr>
<td>jones</td>
<td>s.e.</td>
</tr>
<tr>
<td>Address</td>
<td>units</td>
</tr>
<tr>
<td>main str</td>
<td>2</td>
</tr>
<tr>
<td>forbes ave</td>
<td>2</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

```sql
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’

```
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>C-id</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>123 smith</td>
<td>15-413</td>
</tr>
<tr>
<td>234 jones</td>
<td>15-412</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TAKES</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSN</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>123</td>
</tr>
<tr>
<td>234</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C-id</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-413</td>
</tr>
<tr>
<td>15-413</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
<tr>
<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>

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></td>
<td>null o.s.</td>
</tr>
</tbody>
</table>

Outer join

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

• left outer join
• right outer join
• full outer join
• natural join

Null Values

• \texttt{null} -> unknown, or inapplicable, (or …)
• Complications:
  – 3-valued logic (true, false and \textit{unknown}).
  – \texttt{null = null}: false!!

Overview - detailed - SQL

• DML
  – select, from, where, renaming, ordering,
  – aggregate functions, nested subqueries
  – insertion, deletion, update
• other parts: \texttt{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

```sql
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:
```sql
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

```sql
... 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

define trigger zero_grade 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 <table-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:

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);
}

Embedded SQL - ctn’d

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

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 Create.java
• walkthrough of showAll.java

Outline of an SQL application

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

andrew machine
dbclass.intro.cs.cmu.edu
eg., sun4.andrew

JDBC/ODBC

Create.java
Create.class

Windows NT box;
With, say, ORACLE
Server

Create.java

• Purpose: to load the parent-child table

legend:

→ interesting observation

→ very important point

Walk-through Create.java

```java
import java.io.*;
import java.util.*;
import java.sql.*;

public class Create {
    static final String DbURL =
        "jdbc:oracle:thin:@dbclass.intro.cs.cmu.edu:1521:dbintro";
    // Oracle server at cs.cmu
    static final String OraDriver = "oracle.jdbc.driver.OracleDriver";
    // Oracle driver
    static final String User = "your-andrew-id";
    static final String Passwd = "your-oracle-password";
```
Walk-through Create.java

static final String Passwd = "your-oracle-password";
static final String fileName="PC.txt";

public static void main(String[] args) {
    Connection con = null;
    try {
        // Load the Oracle Driver
        Class.forName(OraDriver);
        // Get a Connection to the database
        con = DriverManager.getConnection(DbURL, User, Passwd);
        // Create a Statement object
        Statement stmt = con.createStatement();
        // Create a table named as PC (varchar2(10), varchar2(10));
        String sqlSt = "CREATE TABLE PC (parent varchar2(10), child varchar2(10))";
        stmt.executeQuery(sqlSt);
    } catch (SQLException ex) {
        // Exception handling
    } finally {
        // Close the connection
        if (con != null) { con.close(); }
    }
}

rest of program:
• read input file
• insert one tuple at a time
• close connection
Walk-through Create.java

while ((line = in.readLine()) != null) {
    // read in the names into 'parent' and 'child'
    // Execute an SQL - insert statement
    sqlSt = "INSERT INTO PC (parent, child) VALUES ('" + parent + ", " + child + ");"
    System.out.println("===> " + sqlSt);
    stmt.executeQuery(sqlSt);
}
in.close();
con.commit();

Overview

• concepts of SQL programs
• walkthrough of Create.java
• walkthrough of showAll.java

Walk-through showAll.java

• purpose: print all (parent, child) pairs
// after opening the connection …
String sqlSt = "SELECT * FROM PC";

ResultSet rs = stmt.executeQuery(sqlSt);
while (rs.next()) {
    System.out.println(rs.getString(\"parent\") + "\", rs.getString(\"child\") );
}