Carnegie Mellon Univ.
Dept. of Computer Science
15-415 - Database Applications

Lecture#8 (cont’d): SQL, Part 2

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

DML - insertions etc

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

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

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’
```

(Don’t be careless - it deletes ALL the ‘smith’s!)
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>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>

<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>

Inner join

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

<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>

null o.s.

Outer join

<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>

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

null o.s.

<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>

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

null o.s.

<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>

null o.s.

```
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

- `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

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
\textit{drop table} student

\textit{delete from} student

modify a table:
\textit{alter table} student \textit{drop} address

\textit{alter table} student \textit{add} major char(10)

Data Definition Language

integrity constraints:
- \textbf{primary key}
- \textbf{foreign key}
- \textbf{check}(P)

create table \textit{takes}
\begin{align*}
\textit{ssn} & \text{char(9) not null,} \\
\textit{c-id} & \text{char(5) not null,} \\
\text{grade} & \text{integer,} \\
\textbf{primary key} & \text{(ssn, c-id),} \\
\textbf{check} & \text{grade in ("A", "B", "C", "D", "F")}
\end{align*}

Referential Integrity constraints

‘foreign keys’ - eg:
\textit{create table} \textit{takes(}
\begin{align*}
\textit{ssn} & \text{char(9) not null,} \\
\textit{c-id} & \text{char(5) not null,} \\
\text{grade} & \text{integer,} \\
\textbf{primary key} & \text{(ssn, c-id),} \\
\textbf{foreign key} & \text{ssn references student,} \\
\textbf{foreign key} & \text{c-id references class)
\end{align*}

Referential Integrity constraints

\begin{align*}
\textbf{foreign key} & \text{ssn references student,} \\
\textbf{foreign key} & \text{c-id references class)
\end{align*}

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 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.

e embedding:

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 into :cssn, :cname, :cad
    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
JDBC/ODBC
eg., sun4.andrew
Windows NT box;
With, say, ORACLE Server
Create.java  
Create.class

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";
  static final String OraDriver = "oracle.jdbc.driver.OracleDriver";
  static final String User = "your-andrew-id";
  static final String Passwd = "your-oracle-password";

  static final String fileName="PC.txt"; //file name for text data

  public static void main(String[] args) {
    try {
      // Load the Oracle Driver
      Class.forName(OraDriver);
      // Get a Connection to the database
      Connection 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 e) {
      // Handle exceptions
    }
  }
}
```

Walk-through Create.java

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

```java
while ((line = in.readLine()) != null) {
    // read in the names into 'parent' and 'child'
    // Execute a SQL - insert statement
    String sqlSt = "INSERT INTO PC (parent, child) VALUES ('" + parent + "', '" + child + ");"
    System.out.println("=== " + i++ + " ===> " + 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

```java
// after opening the connection …
String sqlSt = "SELECT * FROM PC";
```