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 Name Address</td>
<td>c-id  c-name units</td>
</tr>
<tr>
<td>123 smith main str</td>
<td>15-413 s.e. 2</td>
</tr>
<tr>
<td>234 jones forbes ave</td>
<td>15-412 o.s. 2</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:

\[
\text{insert into student} \\
\text{select ssn, name, address} \\
\text{from foreign-student}
\]

DML - deletion etc

delete the record of ‘smith’

DML - deletion etc

delete the record of ‘smith’:

\[
\text{delete from student} \\
\text{where name}='\text{smith}'
\]

(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:

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

---

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>c-id</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>123</td>
</tr>
<tr>
<td>234</td>
</tr>
</tbody>
</table>
Inner join

<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>

<table>
<thead>
<tr>
<th>CLASS</th>
<th>c-id</th>
<th>c-name</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
</tr>
</tbody>
</table>

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

o.s.: gone!

Outer join

<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>

<table>
<thead>
<tr>
<th>CLASS</th>
<th>c-id</th>
<th>c-name</th>
<th>units</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>15-413</td>
<td>s.e.</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>15-412</td>
<td>o.s.</td>
<td>2</td>
</tr>
</tbody>
</table>

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

Outer join

```sql
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** t(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"

drop table student

delete from student

Data Definition Language

modify a table:
"alter table student drop address"

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)

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?
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 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
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.sql.*;
import java.util.*;

public class Create {
    static final String DBURL = "/dbclass/intro.cs.cmu.edu:1521/orcl"
    static final String DBUser = "your-oracle-username"
    static final String DBPassword = "your-oracle-password"
```
Walk-through Create.java

```java
static final String Password = "your-oracle-password";
static final String fileName = "PC.txt";
// file name for test data
public static void main(String[] args) {
    Connection con = null;
    try {
        // Load the Oracle Driver
        Class.forName("OracleDriver");
        // Get a Connection to the database
        con = DriverManager.getConnection("jdbc:sql:yourDB", User, Password);
        // Create a Statement object
        Statement stmt = con.createStatement();

        // Create a Table names pc
        String sqlTable = "CREATE TABLE PC (parent varchar(10), child varchar(10))";
        stmt.executeUpdate(sqlTable);
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        if (con != null) {
            // Close connection
            con.close();
        }
    }
}
```

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
    sqlStr = "INSERT INTO PC (parent, child) VALUES ('" + parent + ", " + child + ")";
    System.out.println("sqlStr = " + sqlStr);
    stmt.executeQuery(sqlStr);
    in.close();
    conn.close();
}
```

Overview

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

Walk-through showAll.java

- purpose: print all (parent, child) pairs
Walk-through showAll.java

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

Walk-through showAll.java

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

Conclusions

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