A Few DBMS Security Issues
Issue #1: Availability

Two goals that often conflict:

- Making authorized access easy
- Making unauthorized access hard

Two categories of access controls:

---

DAC Features of SQL (1 / 4)

Views are a very basic form of DAC:

- Gives users access to necessary information
- Completely hides origins of values
- Is a form of ‘security by obscurity’
Options to the CREATE USER command:

Form: CREATE USER <username> [ <option(s)> ];

Typical options include:

Providing privileges with the GRANT command:

Form: GRANT <privilege>
    [ ON <object> ]
    TO <user>
    [ WITH GRANT OPTION ];

Example(s):
DAC Features of SQL (4 / 4)

What can be GRANTed may be REVOKEd:

Form: REVOKE <privilege>
[ ON <object> ]
FROM <user>;

Example(s):

Mandatory Access Controls (1 / 3)

Idea: The DBMS has default security procedures that must be followed.
Example: The Bell–LaPadula Model (1974)

Security classes are applied to two groups:

Bell-Lapadula enforces two restrictions on security classes (class) assigned to a subject (S) and an object (O):
Issue #2: Confidentiality

To help maintain confidentiality, we can require:

A Special Case: Statistical DBMS Security

Restriction: Users may ask aggregate queries only

Example(s):

Example(s):
Issue #3: Integrity

Idea: Be able to recover DBs after accident or disaster

Some Standard Oracle Security Features

These are available by default in recent versions of Oracle:

- User authentication
- User privileges and roles
- Virtual Private DBs (via query modification)
- Classification of fields
- Network data encryption (via PL/SQL's DBMS_CRYPTO)
- Digital certificate authentication
- Database auditing
A portion of the roster of teams registered for the 2009 ACM North Central Programming Contest at Lincoln, NE:

```
<table>
<thead>
<tr>
<th>Team</th>
<th>State</th>
<th>University</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kansas State U</td>
<td>United States</td>
<td>Kansas State University</td>
</tr>
<tr>
<td>Mount Marty College</td>
<td>United States</td>
<td>Mount Marty College Lancers</td>
</tr>
<tr>
<td>Nebraska Wesleyan U</td>
<td>United States</td>
<td>Epik High</td>
</tr>
<tr>
<td>South Dakota State U</td>
<td>United States</td>
<td>2+2</td>
</tr>
<tr>
<td>South Dakota State U</td>
<td>United States</td>
<td>Never Gonna Let You Down</td>
</tr>
<tr>
<td>South West Minnesota State U</td>
<td>United States</td>
<td>Mustang 1</td>
</tr>
<tr>
<td>South West Minnesota State U</td>
<td>United States</td>
<td>Mustang 2</td>
</tr>
<tr>
<td>University of Nebraska - Lincoln</td>
<td>United States</td>
<td>'; DROP TABLE TEAMS;</td>
</tr>
<tr>
<td>University of Nebraska - Lincoln</td>
<td>United States</td>
<td>Audrey II</td>
</tr>
<tr>
<td>University of Nebraska - Lincoln</td>
<td>United States</td>
<td>Estrogen Attack</td>
</tr>
<tr>
<td>University of Nebraska - Lincoln</td>
<td>United States</td>
<td>Incendiary Pigs</td>
</tr>
<tr>
<td>University of Nebraska - Lincoln</td>
<td>United States</td>
<td>Phelpsiann 4E</td>
</tr>
<tr>
<td>University of Nebraska - Lincoln</td>
<td>United States</td>
<td>Smiley Faces :)</td>
</tr>
<tr>
<td>University of Nebraska - Lincoln</td>
<td>United States</td>
<td>ThreadDeath</td>
</tr>
<tr>
<td>University of Nebraska - Omaha</td>
<td>United States</td>
<td>Team Name</td>
</tr>
</tbody>
</table>
```

A Common DBMS Attack: SQL Injection (2 / 5)

The attack:

A user tries to add (inject) SQL into an incomplete query, in hopes of getting the DBMS to reveal additional information.
Example(s):

Consider this dynamically–constructed SQL query:

But what if the user types this input?
Preventing Injection Attacks: