Database Security

CS461/ECE422

Information Assurance

Fall 2009

Overview

- Database Model
- Access control models
- Inherent database integrity and availability

Reading Material

- Pfleeger & Pfleeger "Security in Computing" 6.3 – on reserve in library and on google books
- Griffiths and Wade, "An Authorization Mechanism for a Relational Database"
  - http://seclab.cs.uiuc.edu/docs/CS463/DatabaseSe

Motivation

- Databases are a common element in today's system architecture
- Hold important information
  - Target of attacks

Relational Model

- Information stored in relations or tables
  - Each row is a tuple of attributes
  - Manipulated by standard SQL language

Name	UID	College	GPA	Financial Aid
Alice	1232	Eng	4	0
Bob	3234	Eng	1.2	$5,000.00
Carol	4565	Bus	3.8	0
Dave	8988	Edu	2.1	0
Ellen	3234	ACES	3.1	$100.00
Alice	4534	LAS	2.9	$10,000.00

Combining tables

- Can use Join to create single set of tuples from multiple tables.

Name	UID	Major
Alice	1234	ECE
Bob	2345	NUC
Carol	3456	BA
Dave	4567	French

UID	Dorm
1234	LAR
2345	ISR
3456	FAR
4567	PAR

Name	Dorm	Major
Alice	LAR	ECE
Bob	ISR	NUC
Carol	FAR	BA
Dave	PAR	French

Making Queries

- Can select rows to create subtables
  - `Select Name, UID, Financial Aid from Students where College = 'Eng'`

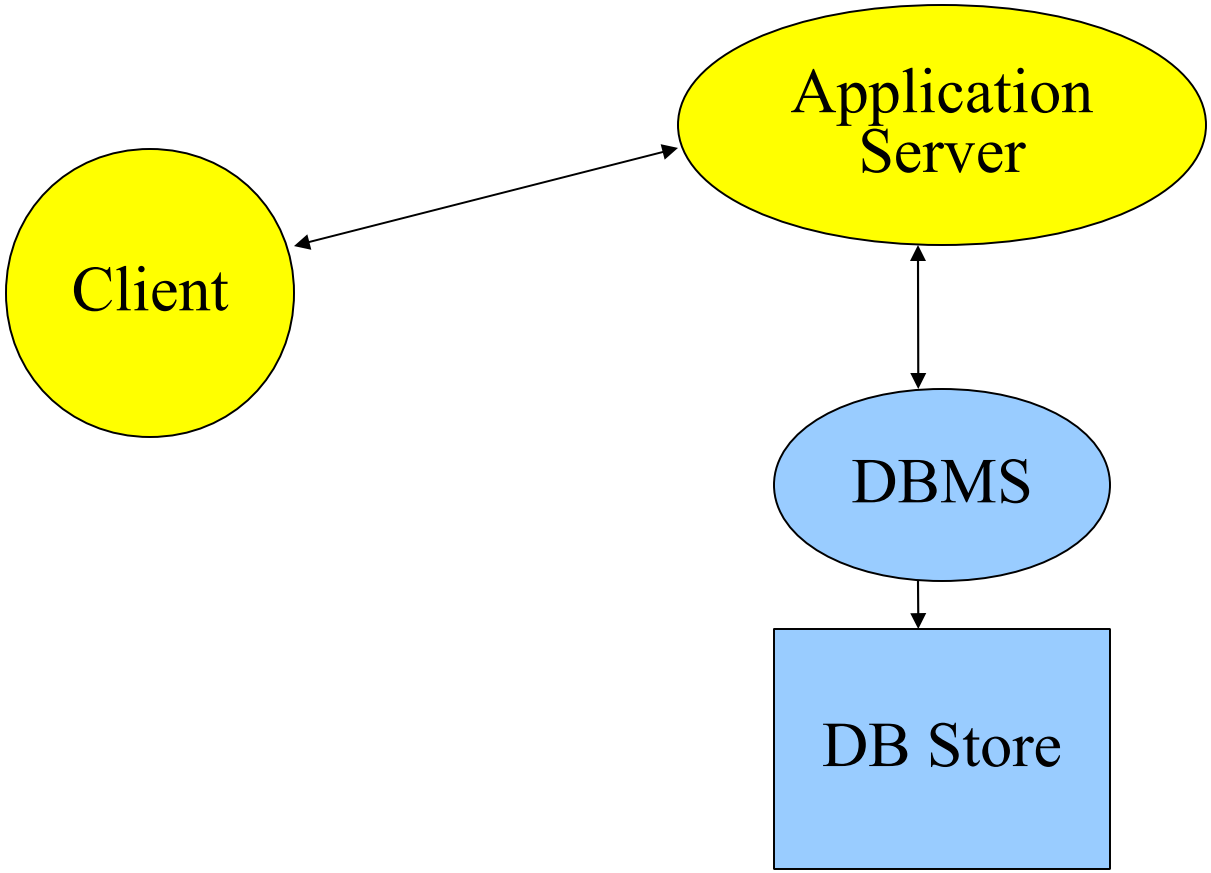
Name	UID	College	GPA	Financial Aid
Alice	1232	Eng	4	0
Bob	3234	Eng	1.2	$5,000.00
Carol	4565	Bus	3.8	0
Dave	8988	Edu	2.1	0
Ellen	3234	ACES	3.1	$100.00
Alice	4534	LAS	2.9	$10,000.00

Name	UID	Financial Aid
Alice	1232	0
Bob	3234	$5,000.00

Database Advantages

- Years and years of technology improvements
  - Data integrity and consistency
  - Decent performance in face of integrity and consistency requirements
- Common well understood model
  - Shared access
  - Controlled access

Access Control in System Design



Access Control in the SQL Model

- Don't have to have a single owner of all data
  - Can create new table
  - Use “Grant” to give others privileges on table
- Can create views to have finer granularity with access control
- Can delegate privilege granting authority to others

SQL **`grant`** Syntax

```
grant privilege_list on resource
  to user_list;
```

- Privileges include **`select`**, **`insert`**, *etc.*
- Resource may be a table, a database, a function, *etc.*
- User list may be individual users, or may be a user group

Example Application

- Alice owns a database table of company employees:

```
name varchar(50),
ssn int,
salary int,
email varchar(50)
```

- Some information (ssn, salary) should be confidential, others can be viewed by any employee.

Simple Access Control Rules

- Suppose Bob needs access to the whole table (but doesn't need to make changes):

```
grant select on employee to bob;
```

- Suppose Carol is another employee, who should only access public information:

```
grant select(name,email) on employee to
  carol;
```

  - not implemented in PostgreSQL (see next two slides for how to work around this)
  - not implemented for **`select`** in Oracle
  - implemented in MySQL

Creating Views

- Careful with definitions!
  - A subset of the database to which a user has access, or:
  - A virtual table created as a "shortcut" query of other tables
- View syntax:

```
create view view_name as
  query_definition;
```

- Querying views is nearly identical to querying regular tables

View-Based Access Control

- Alternative method to grant Carol access to `name` and `email` columns:

```
create view employee_public as
  select name,email from
  employee;
grant select on employee_public
  to carol;
```

Row-Level Access Control

- Suppose we also allow employees to view their own ssn, salary:

```
create view employee_Carol as
  select * from employee
  where name='Carol';
grant select on employee_Carol to carol;
```

- And we allow them to update their e-mail addresses:

```
grant update(email) on employee_Carol to
  carol;
```

  - (Or create yet another new view…)

Delegating Policy Authority

```
grant privilege_list on resource to
  user_list with grant option;
```

- Allows other users to grant privileges, including "with grant option" privileges
- "Copy right" from Access Control lecture (slide 21)
- Can grant subset privileges too
  - Alice: `grant select on table1 to bob with grant option;`
  - Bob: `grant select(column1) on table1 to carol with grant option;`

SQL **`revoke`** Syntax

```
revoke privilege_list on resource
  from user_list;
```

- What happens when a user is granted access from two different sources, and one is revoked?
- What happens when a “with grant option” privilege is revoked?

Revoke Example 1

- Alice gives Read, Update, Insert privileges to Bob for table X
- Carol gives Read, Update privileges to Bob for table X
- Alice revokes Read, Update, Insert privileges from Bob for table X
- What privileges should Bob now have on table X?

Revoke Example 2

- Alice gives Read, Update, Insert privileges to Bob for table X with Grant option
- Bob gives Read, Update privileges to Carol for table X
- Alice revokes all privileges from Bob for table X
- What privileges should Bob have on table X?
- What privileges should Carol have on table X?

Revoke Example 3

- Alice gives Read, Update, Insert privileges to Bob for table X with Grant option
- Bob gives Read, Update privileges to Carol for table X with Grant option
- Carol gives Read, Update privileges to Bob for table X
- Alice revokes all privileges from Bob for table X
- What privileges do Bob and Carol have now?

Disadvantages to SQL Model

- Too many views to create
  - Tedious for many users, each with their own view
  - View redefinitions that change the view schema require dropping the view, redefining, then reissuing privileges
  - Fine-grained policies each require their own view—and no obvious way to see that the views come from the same table
- Other techniques being developed but not yet widely deployed

Data Consistency

- Data is consistent if
  - It never changes OR
    - Only one person ever changes things at a time AND
      - The system never crashes during a change OR
      - All related changes can be made at once (atomic)
- Can loosen these restrictions with transactions and two-phase commits

ACID Transactions

- Atomic – All changes in the transaction have occurred or none of them occur
- Consistent – Transaction does not leave database in half-finished state
- Isolation – Other participants do not see transaction changes until transaction is completed
- Durability – Committed changes will survive system failure

Student Residence Database

Dorm	Room	Student 1	Student 2
LAR	10	Alice	Eve
FAR	13	Mary	Carol
ISR	123	Nancy	

One Student Moving

- Actions
  1. Remove student from old dorm list
  2. Add to new dorm list
- Alice wants to move
  - System crashes after step 1
- Alice is now homeless?

Two Students Moving

- A space opens up in dorm (ISR)
  - Both Alice and Carol want to move there
  - Independently talk with clerks to make the move
- Actions
  - Remove student from old dorm list
  - Add to new dorm list
- Only one space, so only Alice or Carol can make the move
  - Don't want to leave either without a dorm

Student moving

Dorm	Room	Student 1	Student 2
LAR	10	Alice	Eve
FAR	13	Mary	Carol
ISR	123	Nancy	

Two Phase Update or Commit

- First phase
  - Check the commit-flag (lock)
  - Gather information to perform the transaction
    - Set read lock as you go
    - Store values persistently: shadow values or log
  - Turn on the commit flag (lock)
- Second phase
  - Copy logged or shadow values to real values
  - Turn off the commit flag (lock)
- IBM example
  - http://publib.boulder.ibm.com/infocenter/db2luw/

Example Transaction

Shadow old row

Create new row

Set Commit Flag

Shadow old row

Create new row

Set Commit Flag

OK

Already Set

Replace old row with new

Give up and drop any resources

Students Moving

- Alice and Carol check commit-flag
  - Both create shadow versions of their old room row and the new room row
- Alice sets commit-flag first
  - Transaction checks that the target room is still available
  - Makes real copies same has her shadow
  - Clears commit-flag
- Carol fails to get commit flag

What does this mean for security?

- Integrity controls in DB design for protection against accidents.
  - Also means that malevolent user cannot game the entry of data into the DB for his/her benefit
- In general a secure programming or secure development feature.

Key Points

- Database technology has inherently developed good integrity mechanisms
- Access control models are available but not perfect