

Database Security

Security Tasks

- Authentication: verifying the id of a user
- Authorization: checking the access privileges
- Auditing: looking for violations (in the past)

Data Security

Dorothy Denning, 1982:

- “Data Security is the science and study of methods of protecting data (...) from unauthorized disclosure and modification”
- Data Security = Confidentiality + Integrity

How to attack an IT System?

- Misuse of Authority
- Inference and Aggregation
- Masking
- Bypass Access Control
- Browsing
- Trojans
- Hidden Channels

Security in Databases

● Privacy (Confidentiality)

- Do not allow access to private information
- But support statistic analyses over private information
- Avoid inference attacks (e.g., there is only one 42y man)
- Q does not leak info for secret S if: **$P(S | Q) = P(S)$**

```
SELECT count(*)  
FROM Patient  
WHERE age=42  
and sex='M'  
and diagnostic='schizophrenia'
```



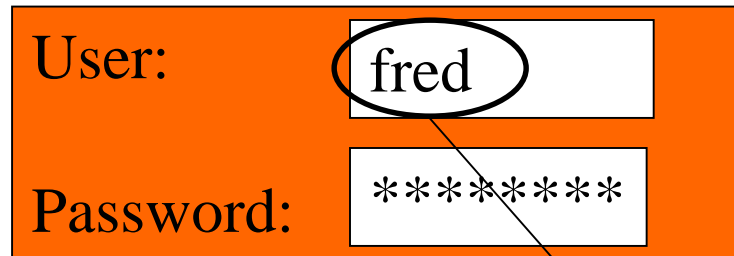
```
SELECT name  
FROM Patient  
WHERE age=42  
and sex='M'  
and diagnostic='schizophrenia'
```



SQL Injection

Problem: Naïve implementation of GUIs:

Login:



A diagram of a login form with an orange background. It contains two input fields: 'User:' with the value 'fred' and 'Password:' with the value '*****'. Both values are circled in black. A thin black line connects the circle around 'fred' to the circle around 'fred' in the SQL query below.

Search Box in Application (SQL Interface in Anwendung):



A diagram of a search box with an orange background. It contains a single input field with the value 'Dr. Lee', which is circled in black. A thin black line connects this circle to the circle around 'Dr. Lee' in the SQL query below.

```
SELECT...FROM...WHERE doctor='Dr. Lee' and patientID='fred'
```

SQL Injection

Another example of poor application implementation:

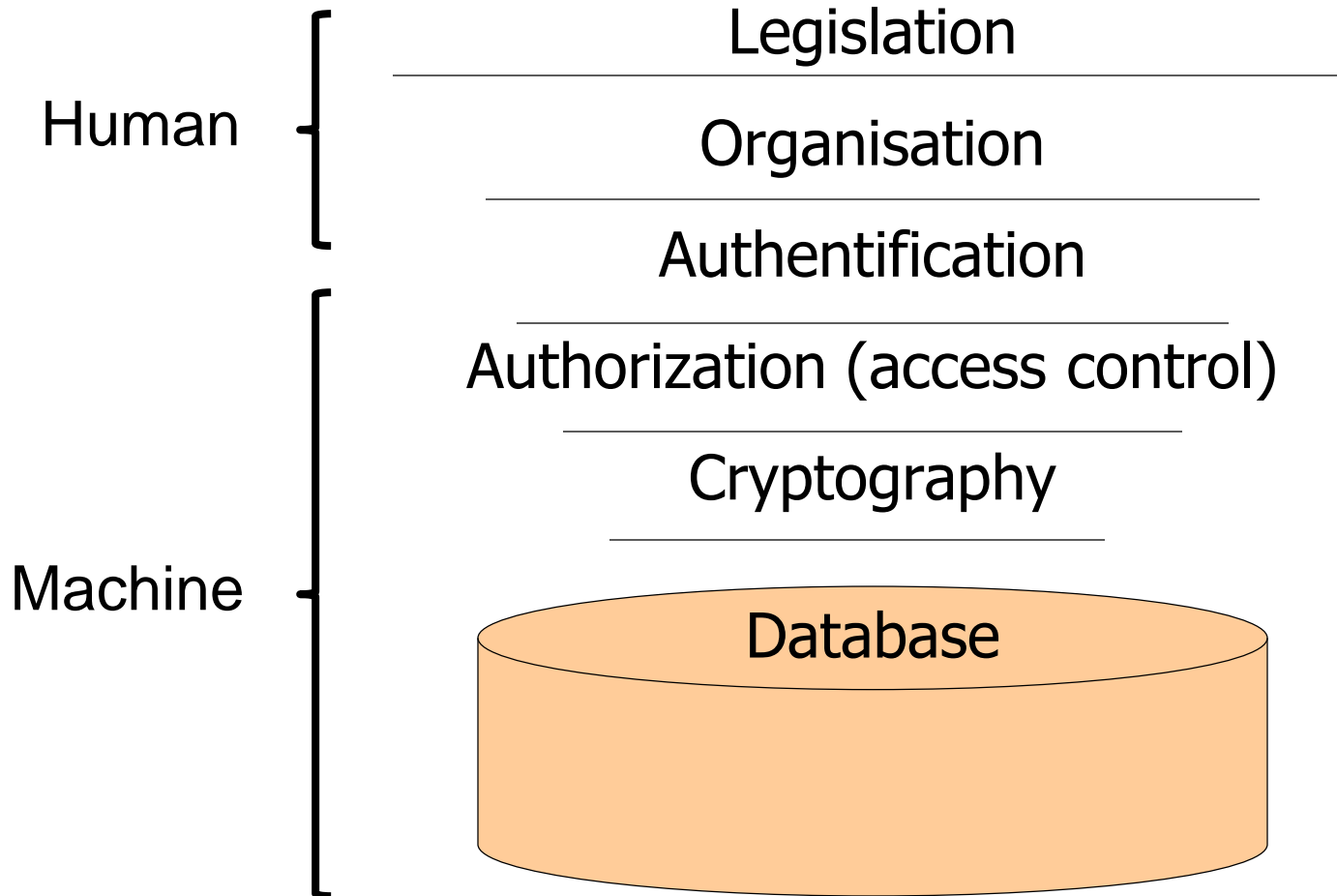
Search:

Dr. Lee'; DROP TABLE Patients; --

SQL Injection: Summary

- The DBMS does the right thing.
 - Why does SQL injection work so often?
- Quick & Dirty Reply:
 - Bad Programmers
 - (easy fix if you are careful in your Java code)
- Sad Truth:
 - Security is implemented in the app and not in the DBMS.
 - (similar discussion as for integrity constraints)
 - Question: How many users does your project DB manage?
 - (Probably, no project has more than one user!)

Layers of Security



Problem: When human meets machine.

- Authentication and sometimes authorization

Discretionary Access Control

Access rules (o, s, t, p, f) :

- $o \in O$, set of *objects* (e.g., table, tuple, attribute)
- $s \in S$, set of *subjects* (e.g., user, processes, apps)
- $t \in T$, set of *access rights* (e.g., read, write, delete)
- p a predicate (e.g., *Level* = „FP“)
- f a **Boolean value** specifying whether s may *grant* the privilege (o, t, p) to another subject s' .

Discretionary Access Control

	Unix	SQL
S	User, groups	User
O	Files, directories	DB, tables, tuples, attributes, views
T	Read, write, execute	CRUD
P	Not supported	Supported via views
F	Fixed (owner, root)	Grant option

Discretionary Access Control

Implementation:

- Access matrix
- Views
- „Query Modification“

Disadvantages:

- Creator of data needs to manage authorizations
- Managing authorizations is cumbersome and error-prone
- Exercise: Find all functional dependencies in the access matrix of Unix! Normalize the access matrix of unix!

Access Control in SQL

Example:

grant select

on Professor

to eickler;

o=Professor, s=eickler, t=SELECT
implicit: p=true, f=false

grant update (Legi, Lecture, PersNr)

on tests

to eickler;

o= Π (tests), s=eickler, t=UPDATE
implicit: p=true, f=false

Access Control in SQL

Other privileges:

- Delete
- insert
- create references (inference attack)
- **grant** option (f = true)

Revoke privileges

```
revoke update (Legi, Lecture, PersNr)  
on tests  
from eickler cascade;
```

Views

Implementation of predicates (p in the (o,s,t,p,f) model)

```
create view FirstSemesterStudents as  
  select *  
  from Student  
  where Semester = 1;  
grant select  
  on FirstSemesterStudents  
  to tutor;
```

Protecting personal records by aggregation

```
create view StrictProf (Nr, AvgGrade) as  
  select Nr, avg(Grade)  
  from tests  
  group by Nr;
```

Group Access Rights

```
CREATE VIEW StudentGrades AS  
SELECT * FROM tests t  
WHERE EXISTS (SELECT * FROM Student  
WHERE Legi = t.Legi AND Name = USER)
```

```
GRANT SELECT ON StudentGrades  
TO <StudentGroup>
```

- Give students right to access their own test results.
 - specific to Oracle
 - magic „Name = USER“ predicate not standardized

Auditing

- Components of Audits

- Logging: keep a record of all (interesting) activities
- Analyze the logs
- (Ideally query logs with DBMS – in practice not poss.)

audit session by system

whenever not successful;

audit insert, delete, update on Professor;

Summary: Security in SQL

Cons:

- coarse granularity: objects are tables or views – not tuples
 - Example: auction system; read your own bids, but not bids of others
- responsibility to manage privileges with creator of table
 - Not necessarily the creator of the data
- difficult to deal with DBMS access violations in app layer
 - (Analogous to “Integrity Constraints”)

Implications

- Access control is implemented at the app tier
- App access DB as “super-user”
- **Security features of DB are not used**

Goal: Increase managability of privileges.

Kinds of Privileges

- explicit / implicit privileges
- positive / negative privileges
- strong / weak privileges

Authorization Algo:

Input: (o, s, t) (May Subject s access Object o using Operation t ?)

Output: Boolean (grant access / refuse access)

if (exists explicit or implicit **strong privilege** (o, s, t))

then return true

else if (exists explicit or implicit **strong negative privilege** $(o, s, \neg t)$)

then return false

else if (exists explicit or implicit **weak privilege** (o, s, t))

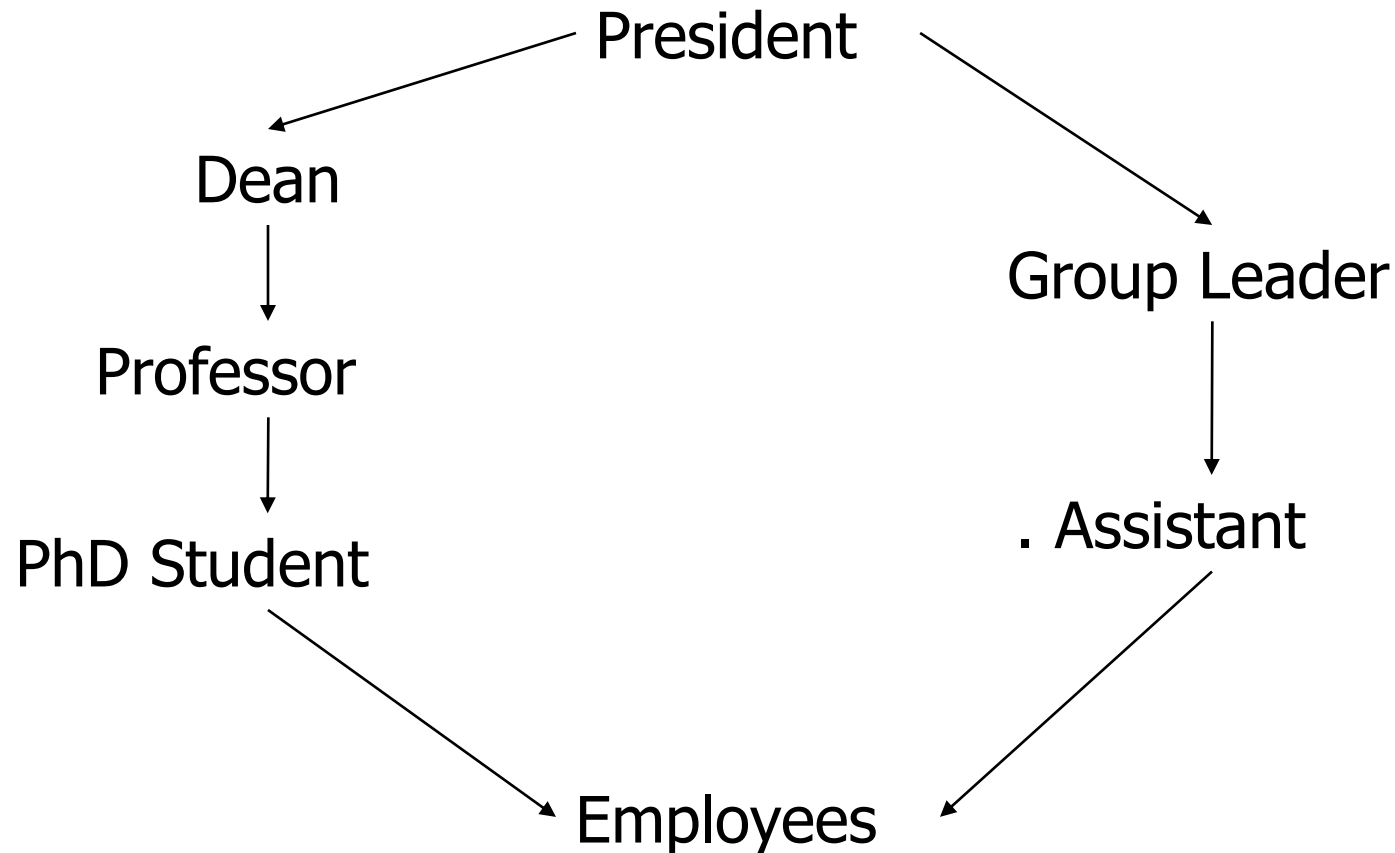
then return true

else if (exists explicit or implicit **weak negative privilege** $(o, s, \neg t)$)

then return false

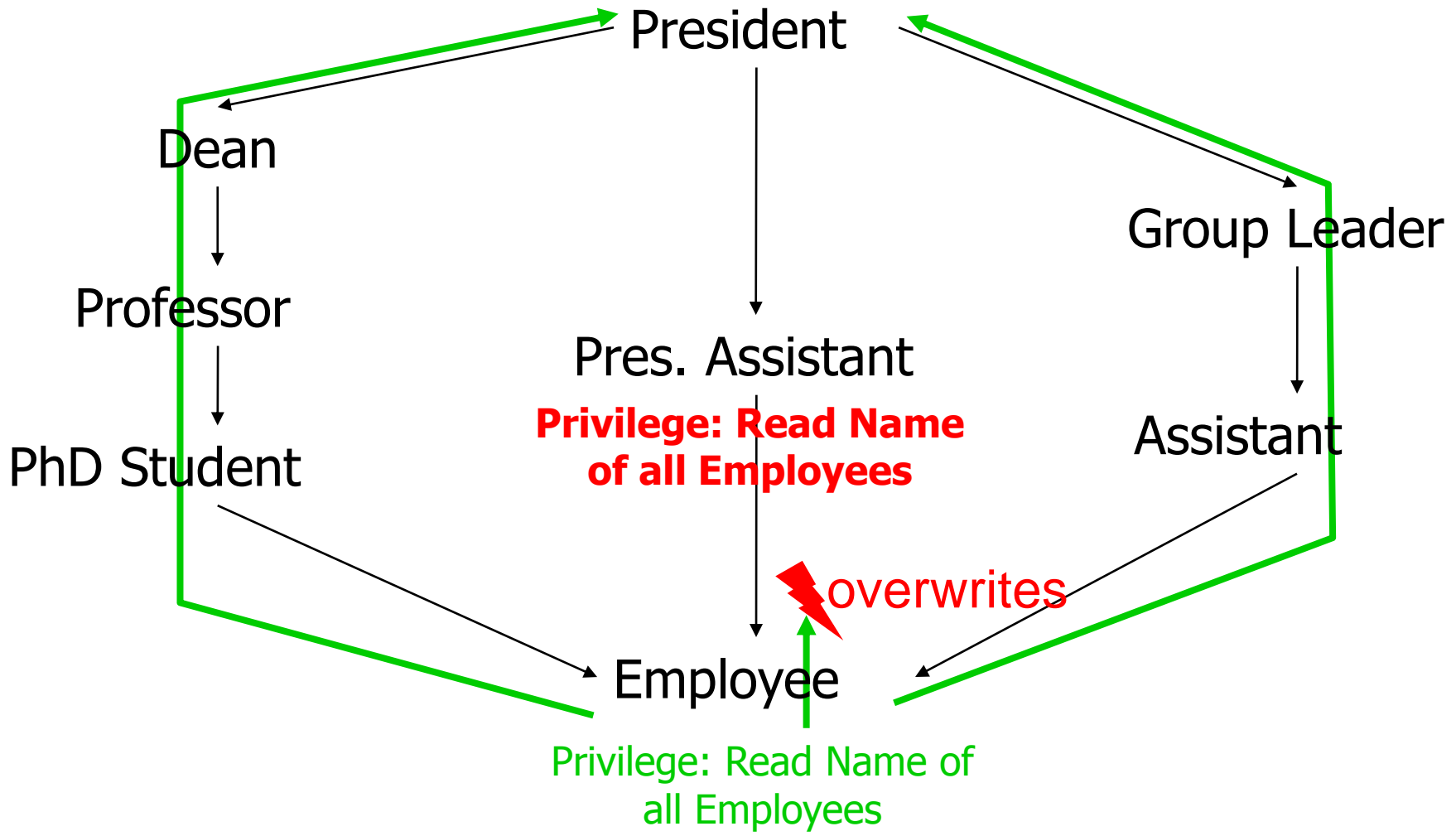
return false // default: reject

Implicit Authorization: Subject Hierarchy



- **explicit positive** privilege at one level
⇒ **implicit positive** privileges on all **higher** levels
- **explicit negative** privilege at one level
⇒ **implicit negative** privilege on all **lower** levels

Strong and Weak Privileges



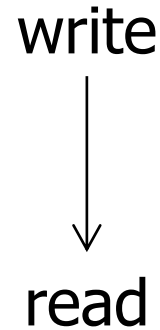
strong pos. privilege

weak pos. Privilege

strong neg. privilege

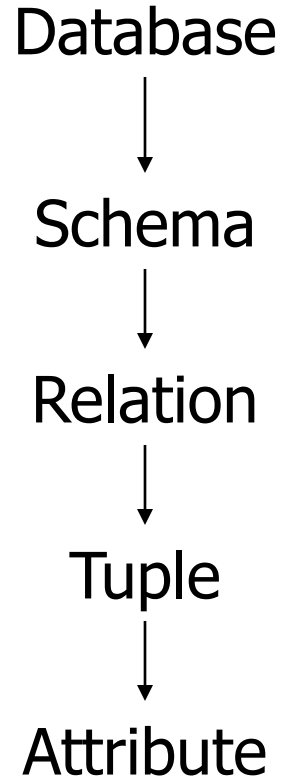
weak neg. privilege

Implicit Privileges: Operation Hierarchy



- **explicit positive** privilege at one level
 - ⇒ **implicit positive** privileges on all **lower** levels
- **explicit negative** privilege at one level
 - ⇒ **implicit negative** privilege on all **higher** levels

Implicit Privileges: Object Hierarchy



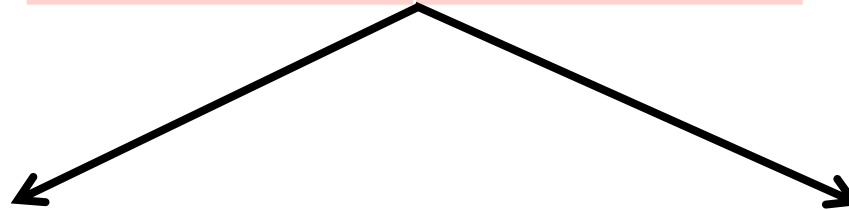
- Implications depend on operations!

Implicit Authorizations & Normalization

Subject	Object
Prof. X	Tuple A.1
Prof. X	Tuple A.2
Prof. X	Tuple A.3
...	...
Prof. X	Table A

Implicit Authorizations & Normalization

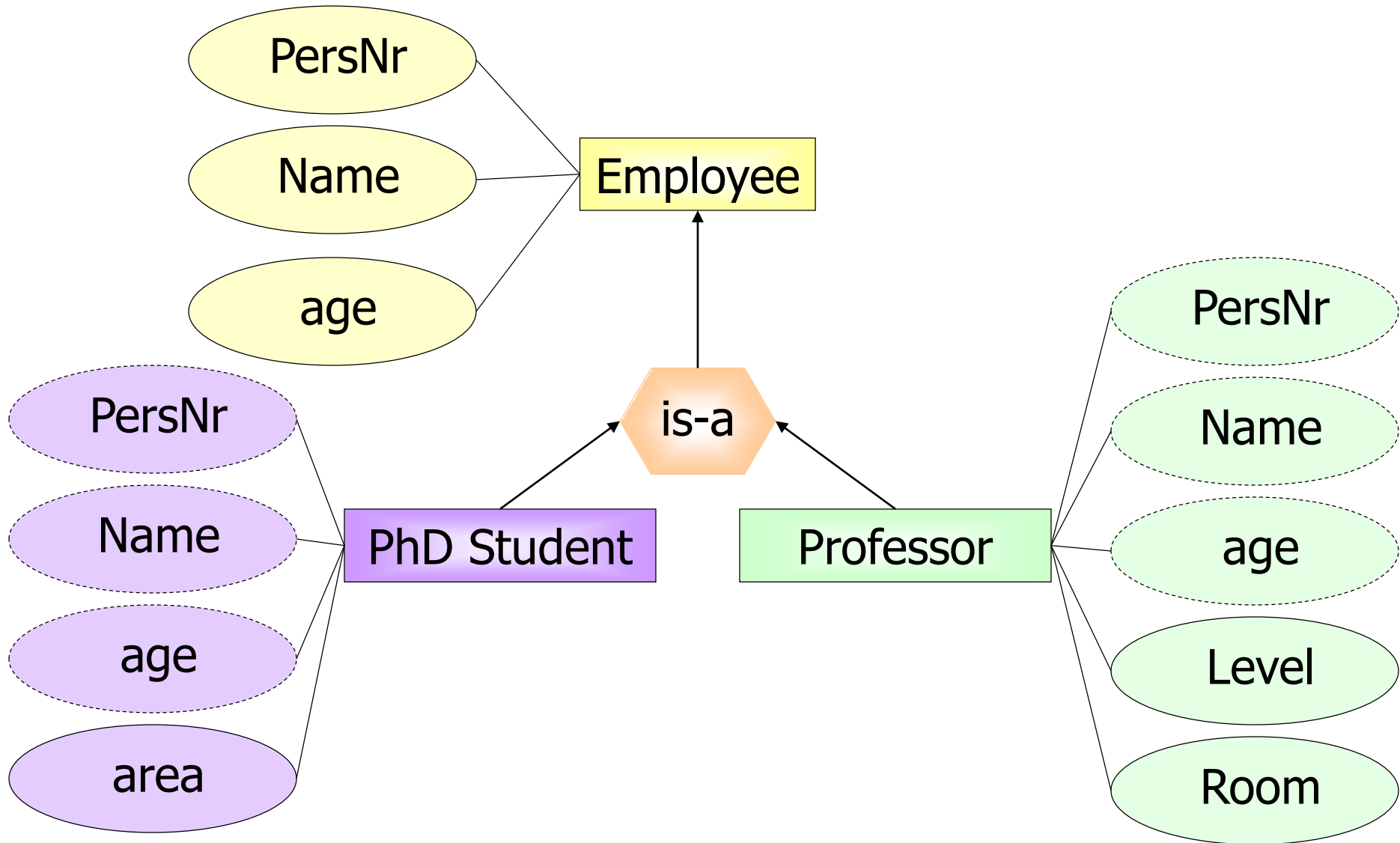
Subject	Object
Prof. X	Tuple A.1
Prof. X	Tuple A.2
Prof. X	Tuple A.3
...	...
Prof. X	Table A



Tuple	Relation
Tuple A.1	Table A
Tuple A.2	Table A
Tuple A.3	Table A
...	...

Subject	Object
Prof. X	Table A

Implicit Privileges: Type Hierarchy



Implicit Privileges: Type Hierarchy

User Groups:

- Group leaders may read the name of all employees
- PhD students may read the name of all professors

Queries:

- Read the name of all PhD students
- Read the name of all professors

Implicit Privileges: Type Hierarchy

Rules:

- Privilege on A.x implies Privilege on B.x if B is subtype of A
- Privilege on Class A implies Privilege on A.x if x is inherited from supertype
- Privilege on Class A does NOT imply Right on B.x if B is subtype of A and x is defined in B.

Mandatory Access Control

- Classification of „importance“ of subjects and objects
- *clear(s)*, for Subject *s*
- *class(o)*, for Object *o*
- Main idea: control *data flow* from low to high levels
 - *s* may read *o* iff $class(o) \leq clear(s)$
 - Class of *o* depends on importance of creator *s*
 $clear(s) \leq class(o)$
- Used in military: data flows bottom-up
 - (control flows from top-down)

Multi-level Databases

Goal: User should not know what he/she cannot see!

„s“ < „ts“; Id is key of the SecretAgent relation

SecretAgent						
TC	Id	IC	Name	NC	Skills	SC
ts	007	s	Bond, James	s	meucheln	ts
ts	008	ts	Clouseau	ts	spitzeln	ts

View of a user who is classified as „s“

SecretAgent						
TC	Id	IC	Name	NC	Skills	SC
s	007	s	Bond, James	s	-	s

Problems:

- „s“ user inserts tuple with Id „008“
- „s“ user modifies the Skills of „007“

Multi-level Relations

Multilevel-Relation \mathcal{R} with schema

$$\mathcal{R} = \{A_1, C_1, A_2, C_2, \dots, A_n, C_n, TC\}$$

Instances \mathcal{R}_C with tuples of the form

$$[a_1, c_1, a_2, c_2, \dots, a_n, c_n, tc]$$

- Visibility of tuples in \mathcal{R}_C : $c_i \geq c_\kappa$ for all attr. i ; key κ
- Visibility of attributes in \mathcal{R}_C : a_i is visible if $clear(s) \geq c_i$
- $tc \geq c_i$ (classification of tuple; not used here)

Integrity Constraints

κ is the (logical) key of Multi-level Relation \mathcal{R}

Entity-Integrity. \mathcal{R} is entity consistent iff, for all instances \mathcal{R}_c of \mathcal{R} and all $r \in \mathcal{R}_c$:

1. $A_i \in \kappa \Rightarrow r.a_i \neq \text{Null}$
2. $A_i, A_j \in \kappa \Rightarrow r.c_i = r.c_j$
3. $A_i \notin \kappa \Rightarrow r.c_i \geq r.c_{\kappa}$ (c_{κ} is class of the key)

- All visible tuples of R can be identified by their key at all levels.
- It is possible to hide some (non-key) attributes of a tuple.

Integrity Constraints

κ is the (logical) key of Multi-level Relation \mathcal{R}

Null-Integrity. \mathcal{R} is Null consistent iff, for all instances \mathcal{R}_c of \mathcal{R} :

1. $\forall r \in \mathcal{R}_c : r.a_j = \text{Null} \Rightarrow r.c_j = r.c_\kappa$
 2. \mathcal{R}_c has no subsumptions. That is, there are no two tuples r, s so that for all attributes A_j :
 - $r.a_j = s.a_j$ and $r.c_j = s.c_j$ OR
 - $r.a_j \neq \text{Null}$ and $s.a_j = \text{Null}$
- (See next slides for subsumption violations.)

Subsumption-free Relations

a) R_{ts}

SecretAgent						
TC	Id	IC	Name	NC	Skill	SC
s	007	s	Bond, James	s	-	s

b) Correct update of R_{ts} by user classified as „ts“

SecretAgent						
TC	Id	IC	Name	NC	Skill	SC
ts	007	s	Bond, James	s	meucheln	ts

c) Incorrect update of R_{ts} by user classified as „ts“

SecretAgent						
TC	Id	IC	Name	NC	Skill	SC
s	007	s	Bond, James	s	-	s
ts	007	s	Bond, James	s	meucheln	ts

Integrity Constraints

Inter-instance Integrity:

for $c' < c$, $R_{c'}$ can be computed from R_c

$$R_{c'} = f(R_c, c')$$

Filter function f can be defined as follows:

1. foreach $r \in R_c$, $r.c_k \leq c'$ create $s \in R_{c'}$ such that

$$s.a_i = \begin{cases} r.a_i & \text{if } r.c_i \leq c' \\ \text{Null} & \text{otherwise} \end{cases}$$

$$s.c_i = \begin{cases} r.c_i & \text{if } r.c_i \leq c' \\ r.c_k & \text{otherwise} \end{cases}$$

2. $R_{c'}$ does not contain any other tuples.
3. Eliminate subsumed tuples.

Integrity Constraints

Poly-instantiation Integrity. For all instances R_C and all attributes A_j the following functional dependency holds:

$$\{\kappa, C_{\kappa}, C_j\} \rightarrow A_j$$

- κ is the visible key of each instance
- If all integrity constraints are met, then a multi-value relation can be implemented as a set of (regular) relations on top of a (regular) relational database system.