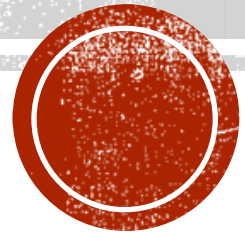
DATABASE SECURITY



SICUREZZA DOCUMENTALE

A.A. 2017-2018

FILESYSTEM E AUTORIZZAZIONI

- Abbiamo visto come il file system consente di associare "privilegi" (permessi) per gli oggetti (file) che gestisce
  - Lettura, scrittura, esecuzione
- In particolare, abbiamo visto come sia possibile stabilire a quali utenti assegnari privilegi
  - Utenti, gruppo, altri

DATABASE E AUTORIZZAZIONI

- Allo stesso modo e' possibile assegnare privilegi agli utenti di una base di dati
  - Faremo riferimento al linguaggio SQL
- SQL identifica un set piu' dettagliato di privilegi sugli oggeti (relazioni) rispetto al file system
- Si tratta di 9 privilegi in tutto, alcuni dei quali consentono di imporre restrizioni perfino alla colonna di una relazione

DATABASE E AUTORIZZAZIONI

- Alcune tipologie di privilegi:
  1. SELECT = consente di effettuare delle query sulla relazione.
  2. INSERT = consente di inserire tuple.
     - Si puo' applicare anche ad un solo attributo.
  3. DELETE = consente di cancellare tuple.
  4. UPDATE = consente di fare l'update di tuple.
     - Si puo' applicare anche ad un solo attributo.

ESEMPIO PRIVILEGI

```
INSERT INTO Beers(name)
  SELECT beer FROM Sells
  WHERE NOT EXISTS
        (SELECT * FROM Beers
         WHERE name = beer);
```

- Abbiamo bisogno di privilegi per la SELECT su Sells e Beers, e INSERT su Beers o Beers.name.

DATABASE E OGGETTI

- Gli oggetti su cui si possno assegnare priviledi includono stored tables e views (viste).
- Altri privilegi sono il diritto di creare oggetti di un certo tipo e.g., triggers
- Le Viste (Views) sono un importante strumento per il controllo degli accessi

ESEMPIO: LE VIEWS COME STRUMENTI DI ACCESS CONTROL

- Potremmo non voler concedere privilegi per la SELECT su Emps(name, addr, salary).
- E' piu' sicuro consentire la SELECT su una vista:

```
CREATE VIEW SafeEmps AS
  SELECT name, addr FROM Emps;
```

- Le query su SafeEmps non richiedono privilegi di SELECT su Emps, ma bensi su SafeEmps.

AUTHORIZATION ID'S

- Un utente viene riferito tramite un *authorization ID*, tipicamente il login name.
- C'e' un authorization ID PUBLIC.
  - Consentire privilegi a PUBLIC equivale a concedere privilegi a qualsiasi authorization ID.

GRANTING PRIVILEGES

- Ognuno ha tutti i privilegi sugli oggetti (es. Relazioni) create.
- Si possono garantire i privilegi ad altri utenti (authorization ID's), incluso PUBLIC.
- Si possono garantire privilegi con `WITH GRANT OPTION`, che consente a chi riceve un privilegio di passarlo ad altri.

LO STATEMENT GRANT

- **Per garantire privilegi, si usa la sintassi:**

  GRANT <list of privileges>
  ON <relation or other object>
  TO <list of authorization ID's>;

- Se si vuole consentire il trasferimenti di privilegi ad altri si aggiunge:

  WITH GRANT OPTION

ESEMPIO: STATEMENT GRANT

- **Supponiamo di essere i proprietari della tabella `Sells`, possiamo allora scrivere:**

```
GRANT SELECT, UPDATE(price)
ON Sells
TO sally;
```

  - Sally ha il diritto di eseguire query su `Sells` e pou' effettuare l'update del campo price.

ESEMPIO: STATEMENT GRANT OPTION

- **Supponiamo ora di aggiungere:**

```
GRANT UPDATE ON Sells TO sally
WITH GRANT OPTION;
```

- **Ora, Sally non solo e' in grado di aggiornare qualsiasi attributo di Sells, ma può concedere ad altri il privilegio** `UPDATE ON` **su** `Sells`.
  - Inoltre, si può concedere privilegi più specifici come `UPDATE (price) ON Sells`.

REVOCA PRIVILEGI

- La sintassi per revocare privilegi e':

```
REVOKE <list of privileges>
ON <relation or other object>
FROM <list of authorization ID's>;
```

- La concessione di questi privilegi non può più essere utilizzata da questi utenti per giustificare il loro uso del privilegio
  - Ma possono ancora avere il privilegio, perché hanno ottenuto in modo indipendente……

REVOCA PRIVILEGI, OPZIONI

- Si deve aggiungere allo statement `REVOKE`:
  1. `CASCADE`. Ora, ogni privilegio stabilito da chi sta revocando non sara' piu' valido, non importa la catena che ha garantito il privilegio
  2. `RESTRICT`. Se il privilegio e' stato concesso ad altri, `REVOKE` genera un warning che qualcosa deve essere fatto per tracciare il privilegio.

DIAGRAMMA DEI PRIVILEGI

- I privilegi concessi, possono essere rappresentati tramite un diagramma a grafo:
  - Nodi = user/privilege/grant option?/is owner?
  - `UPDATE ON` R, `UPDATE(a)` on R, and `UPDATE(b)` ON R sono rappresentati come nodi differenti.
  - `SELECT ON` R and `SELECT ON` R `WITH GRANT OPTION` rappresentano nodi differenti
- Un arco *X* -> *Y* significa che il nodo *X* e' stato usato per garantire *Y*.

DIAGRAMMA DEI PRIVILEGI: NODI

- Si usa *AP* per il nodo che rappresenta authorization ID *A* avente il privilegio *P*.
  - *P* * = privilegio *P* con grant option.
  - *P* ** = la sorgente del privilegio *P*.
    - *A* e' il proprietario dell'oggetto sul quale P e' un privilegio.
    - ** implica grant option

DIAGRAMMA DEI PRIVILEGI: ARCHI

- Quando *A* concede il privilegio *P* a *B*, si traccia un arco da *AP* * o *AP* ** a *BP*.
  - oppure a *BP* * se il grant include grant option
  - **Se *A* concende un subprivilegio *Q* di *P*** [`es.UPDATE(a)` **on R when *P* is** `UPDATE ON` **R**] **l'arco va invece a *BQ* o *BQ* *.**

DIAGRAMMA DEI PRIVILEGI: ARCHI

- **Fondamentale**: un user *C* ha il privilegio *Q* qualora vi sia un path *XP* ** a *CQ*, *CQ* *, o *CQ* **, e *P* e' un superprivilegio di *Q*.
  - Si noti che P potrebbe essre Q, e X potrebbe essere C.

DIAGRAMMA DEI PRIVILEGI: ARCHI

- **Fondamentale**: un user *C* ha il privilegio *Q* qualora vi sia un path *XP* ** a *CQ*, *CQ* *, o *CQ* **, e *P* e' un superprivilegio di *Q*.
  - Si noti che P potrebbe essre Q, e X potrebbe essere C.
- Se *A* revoca *P* da *B* con l'opzione `CASCADE`, si cancella l'arco da *AP* a *BP*.

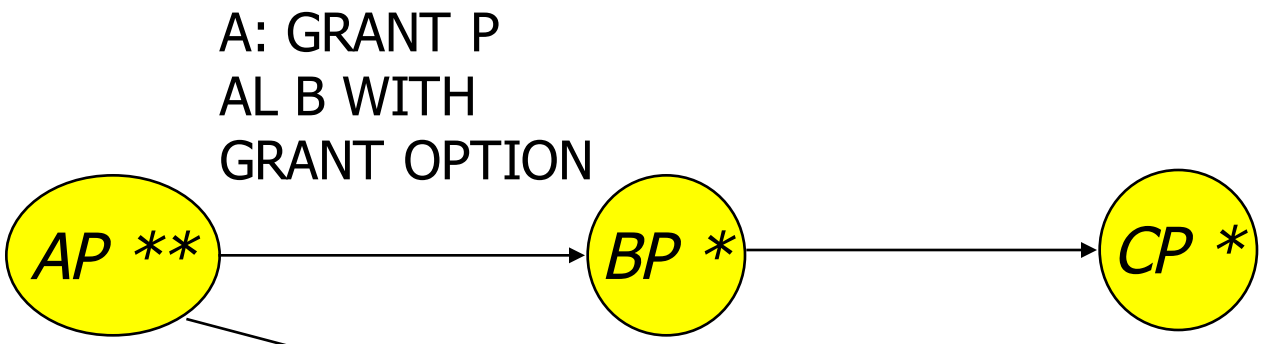
DIAGRAMMA DEI PRIVILEGI: ARCHI

- **Fondamentale**: invece se *A* usa `RESTRICT`, ed esiste un arco da *BP* verso un altro nodo, allora la revoke viene respinta e il grafo non viene cambiato.

DIAGRAMMA DEI PRIVILEGI: ARCHI

- Bisogna controllare che ogni nodo ha un path da qualche nodo **, che rappresenta la proprieta' (di un privilegio)
- Ciascun node senza uno di questi path rappresent un privilegio revocato e viene cancellato dal diagramma.

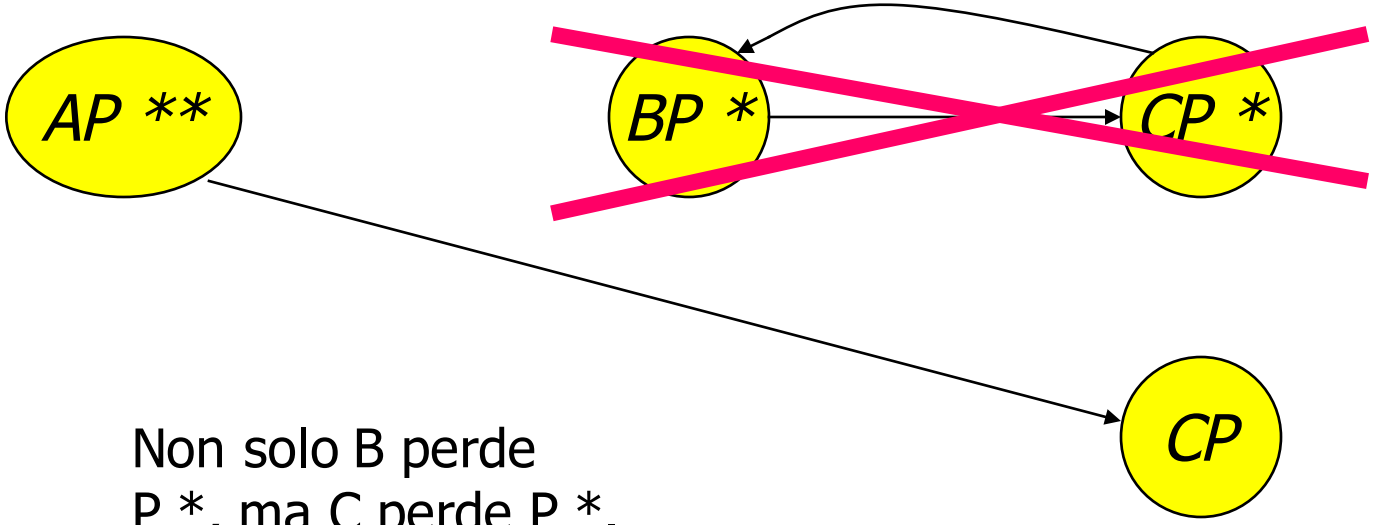
DIAGRAMMA DEI PRIVILEGI: ESEMPIO



A possiede l'oggetto
su cui P è un privilegio

DIAGRAMMA DEI PRIVILEGI: ESEMPIO

A esegue
REVOKE P FROM B CASCADE;



C aveva passato P
to B, quindi entrambi
i nodi vengono cancellati.

Non solo B perde
P *, ma C perde P *.
Si eliminano inoltre BP * e CP *.

Tuttavia, C ha ancora P senza grant option
a causa della concessione diretta.

RIEPILOGO

```
GRANT privileges
    ON object
    TO users
    [WITH GRANT OPTIONS]
```

privileges = SELECT |
INSERT(column-name) |
UPDATE(column-name) |
DELETE |
REFERENCES(column-name)

object = table | attribute

ESEMPIO

```
GRANT INSERT, DELETE ON Customers
TO Yuppy WITH GRANT OPTIONS
```

Queries concesse a Yuppy:

```
INSERT INTO Customers(cid, name, address)
    VALUES(32940, 'Joe Blow', 'Seattle')

DELETE Customers
    WHERE LastPurchaseDate < 1995
```

Queries negate a Yuppy:

```
SELECT Customer.address
FROM Customer
WHERE name = 'Joe Blow'
```

ESEMPIO

GRANT SELECT ON Customers TO **Michael**

Ora **Michael** puo' usare SELECT, ma non INSERT oDELETE

ESEMPIO

```
GRANT SELECT ON Customers
    TO Michael WITH GRANT OPTIONS
```

Michael puo' utilizzare, ad esempio::

GRANT SELECT ON Customers TO **Yuppi**

Ora **Yuppi** puo' fare la SELECT su Customers

ESEMPIO

```
GRANT UPDATE (price)  ON Product TO Leah
```

Leah puo' fare update, ma solo su Product.price, e non su Product.name

ESEMPIO

Customer(cid, name, address, balance)
Orders(oid, cid, amount) cid= foreign key

Bill ha diritti di INSERT/UPDATE su Orders.
MA NON PUO'' FARE INSERT !

GRANT REFERENCES (cid) ON Customer TO **Bill**

Ora **Bill** puo' fare INSERTdi tuple in Orders

VIEWS E SECURITY

David e' il proprietario

Fred non ' autorizzato a vedere questa colonna

Customers:

Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

David scrive

```
CREATE VIEW PublicCustomers
        SELECT Name, Address
        FROM Customers
GRANT SELECT ON PublicCustomers TO Fred
```

VIEWS E SECURITY

Customers:

Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

John puo'
solo vedere le righe
dove balances<0

David scrive

```
CREATE VIEW BadCreditCustomers
        SELECT *
        FROM Customers
        WHERE Balance < 0
GRANT SELECT ON BadCreditCustomers TO John
```

VIEWS E SECURITY

- Ogni utente dovrebbe vedere solo i suoi record

David scrive

Name	Address	Balance
Mary	Huston	450.99
Sue	Seattle	-240
Joan	Seattle	333.25
Ann	Portland	-520

```
CREATE VIEW CustomerMary
        SELECT *  FROM Customers
        WHERE name = 'Mary'
GRANT SELECT
ON CustomerMary TO Mary
```

```
CREATE VIEW CustomerSue
        SELECT *  FROM Customers
        WHERE name = 'Sue'
GRANT SELECT
ON CustomerSue TO Sue
```

C'e' bisogno di un *row-level* access control

E se ci fossero 10.000 utenti?

. . .

REVOCATION

```
REVOKE   [GRANT OPTION FOR]   privileges
                ON object FROM users  {  RESTRICT  |  CASCADE   }
```

L'amministratore scrive:

```
REVOKE SELECT ON Customers  FROM David CASCADE
```

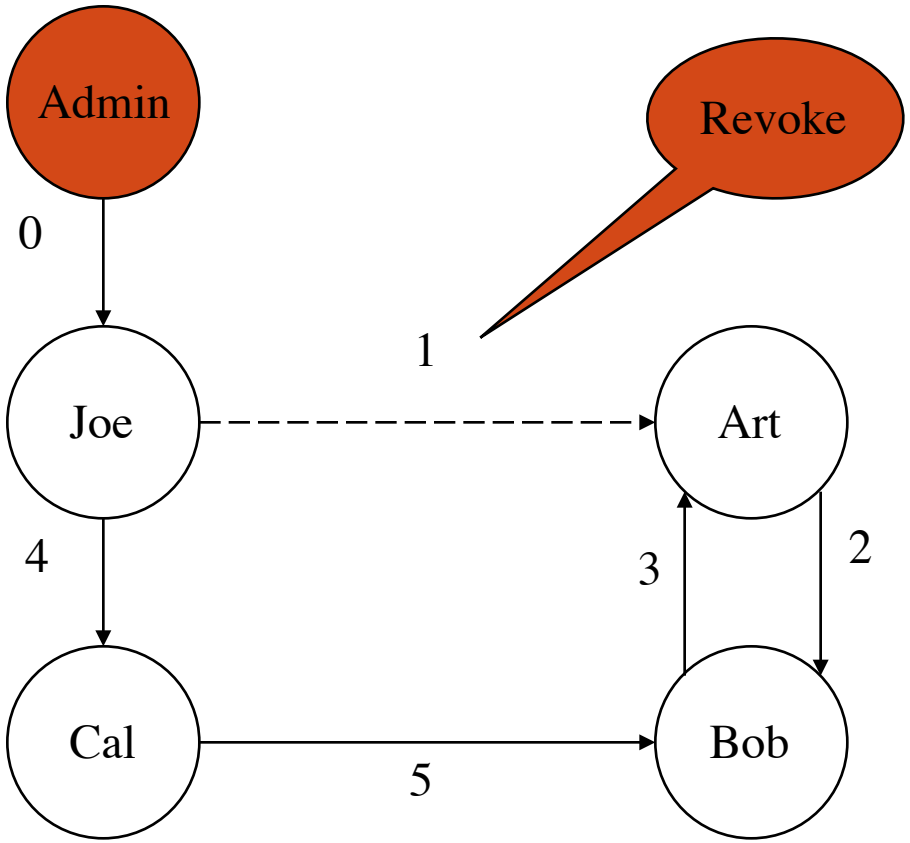
John perde il privilegio SELECT su BadCreditCustomers

REVOCATION

Joe: GRANT [….] TO Art …
Art: GRANT [….] TO Bob …
Bob: GRANT [….] TO Art …
Joe: GRANT [….] TO Cal …
Cal: GRANT [….] TO Bob …
Joe: REVOKE [….] FROM Art CASCADE

Stesso privilegio,
stesso oggetto,
GRANT OPTION

REVOCATION



Ognuno mantiene il suo privilegio

SOMMARIO DELLA SICUREZZA IN SQL

Limitazioni:

- Non c'e' row level access control
- Chi crea la tabella possiede i dati!

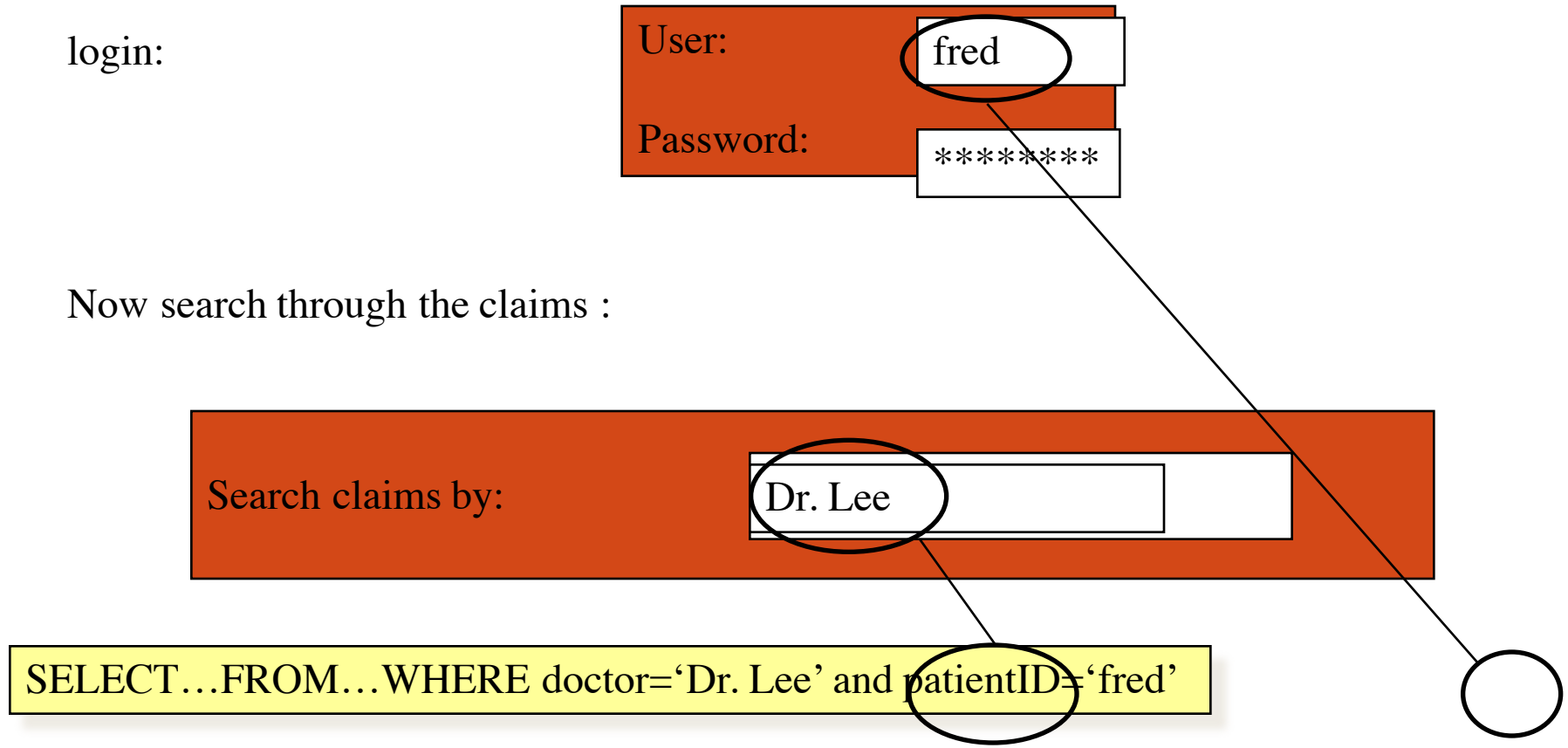
DUE ATTACCHI FAMOSI

- SQL injection
- Sweeney's example

[Chris Anley, *Advanced SQL Injection In SQL*]

SQL INJECTION

Your health insurance company lets you see the claims online:



SQL INJECTION

Now try this:

Search claims by: Dr. Lee' OR patientID = 'suciu'; --

…..WHERE doctor='Dr. Lee' OR patientID='suciu'; --' and patientID='fred'

Better:

Search claims by: Dr. Lee' OR 1 = 1; --

SQL INJECTION

When you're done, do this:

Search claims by: Dr. Lee'; DROP TABLE Patients; --

LATANYA SWEENEY'S FINDING

- In Massachusetts, ilGroup Insurance Commission (GIC) e' responsabile dell'acquisto delle assicurazioni sulla vita degli impiegati nel settore della sanita'
- GIC deve pubblicare i dati:

GIC(**zip, dob, sex**, diagnosis, procedure, ...)

LATANYA SWEENEY'S FINDING

- Sweeney ha pagato $20 e comprato la lista degli aventi diritti al voto di Cambridge Massachusetts:

GIC(**zip, dob, sex**, diagnosis, procedure, ...)
VOTER(name, party, ..., **zip, dob, sex**)

LATANYA SWEENEY'S FINDING

zip, dob, sex

- William Weld (ex governatore) vive a Cambridge, percio' e' un VOTER
- 6 persone in VOTER condividono il suo **dob**
- solo 3 sono uomini (same **sex**)
- Weld era l'unico ad avere quel codice **zip**
- Sweeney e' riuscita a venire a conoscenza dei record medici di Weld's !

DUE NUOVE TECNICHE

- K-anonymity, information leakage
- Row-level access control

[Samarati&Sweeney'98, Meyerson&Williams'04]

INFORMATION LEAKAGE:K-ANONYMITY

Definition: ogni tupla e' uguale a k-1 altre

Anonimizzazione: attraverso cancellazione e generalizzazione

First	Last	Age	Race	Disease
*	Stone	30-50	Afr-Am	Flue
John	R*	20-40	*	Measels
*	Stone	30-50	Afr-am	Pain
John	R*	20-40	*	Fever

[Miklau&S'04, Miklau&Dalvi&S'05, Yang&Li'04]

INFORMATION LEAKAGE: QUERY-VIEW SECURITY

Have data: TABLE Employee(name, dept, phone)

Secret Query	View(s)	Disclosure ?
S(name)	V(name,phone)	total
S(name,phone)	V1(name,dept) V2(dept,phone)	big
S(name)	V(dept)	tiny
S(name) where dept='HR'	V(name) where dept='RD'	none

FINE-GRAINED ACCESS CONTROL

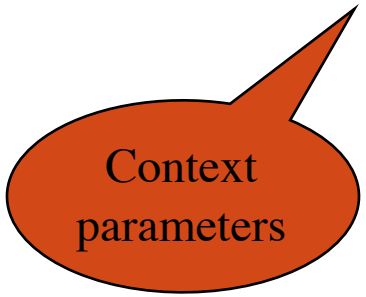
Control access at the tuple level.

- Policy specification languages
- Implementation

POLICY SPECIFICATION LANGUAGE

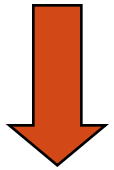
No standard, but usually based on parameterized views.

```
CREATE AUTHORIZATION VIEW PatientsForDoctors AS
   SELECT Patient.*
   FROM Patient, Doctor
   WHERE Patient.doctorID = Doctor.ID
        and     Doctor.login = %currentUser
```



IMPLEMENTATION

```
SELECT Patient.name, Patient.age
FROM Patient
WHERE Patient.disease = 'flu'
```



```
SELECT Patient.name, Patient.age
FROM Patient, Doctor
WHERE Patient.disease = 'flu'
      and Patient.doctorID = Doctor.ID
      and Patient.login = %currentUser
```

e.g. Oracle

TWO SEMANTICS

- The Truman Model = filter semantics
  - transform reality
  - ACCEPT all queries
  - REWRITE queries
  - Sometimes misleading results

```
SELECT count(*)
FROM Patients
WHERE disease='flu'
```

- The non-Truman model = deny semantics
  - reject queries
  - ACCEPT or REJECT queries
  - Execute query UNCHANGED
  - May define multiple security views for a user

[Rizvi'04]

SUMMARY ON INFORMATION DISCLOSURE

- The theoretical research:
  - Exciting new connections between databases and information theory, probability theory, cryptography

- The applications:
  - many years away

[Abadi&Warinschi'05]

SUMMARY OF FINE GRAINED ACCESS CONTROL

- Trend in industry: label-based security
- Killer app: application hosting
  - Independent franchises share a single table at headquarters (e.g., Holiday Inn)
  - Application runs under requester's label, cannot see other labels
  - Headquarters runs Read queries over them
- Oracle's Virtual Private Database

[Rosenthal&Winslett'2004]