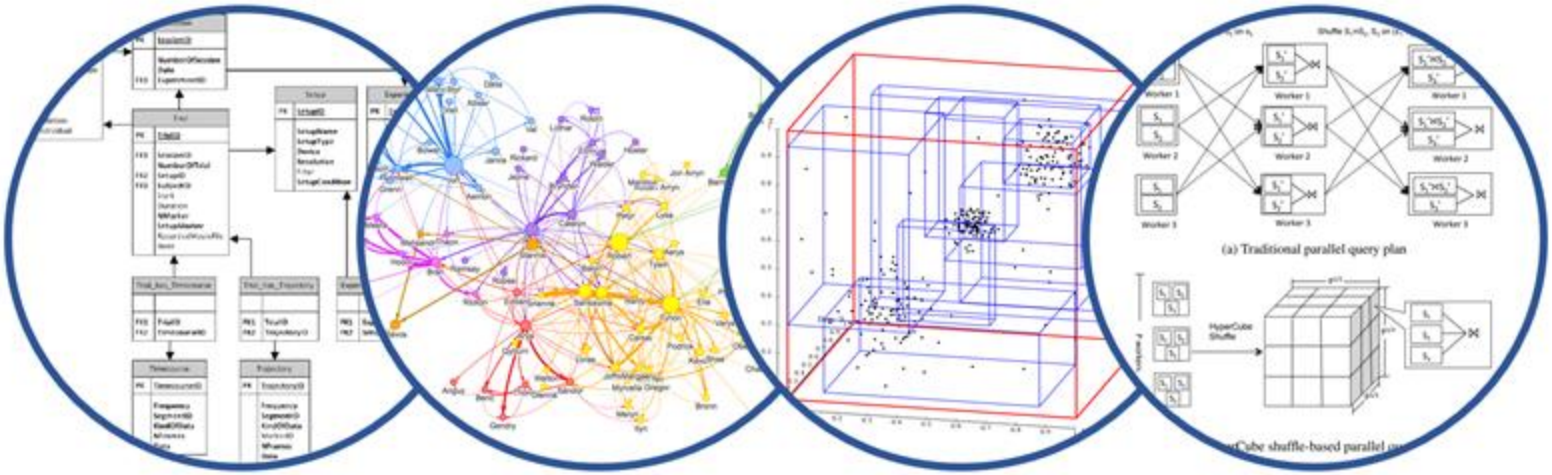


Introduction to Data Management

Application Data Management

Amal Jacob, Guest Lecturer

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Announcements

- HW5 is due on Friday
- HW6 M1 will be released end of this week
  - HW6 is substantial - don’t procrastinate
  - **No late days for M1**
- Go to section!
  - Connect to Azure + setup, demo, useful tips

Agenda

- **Access Control**
- Passwords
- Data Privacy

DBMS controls

- Block unauthorized access
  - Tiered-access hierarchy
- Usually a built-in access control

DBMS controls

- Block unauthorized access
  - Tiered-access hierarchy
- Usually a built-in access control
- HW3/HW6 autograder runs student submissions on staff tables in Azure
  - How do we protect from student INSERTs, DELETEs, etc. on read-only tables?

DBMS controls

Example: On Azure…

- Create a user

```
CREATE USER <username>
WITH PASSWORD = <password>
```

```
CREATE USER autograder
WITH PASSWORD = 'pass123'
```

DBMS controls

Example: On Azure…

- Create a user

```
CREATE USER <username>
WITH PASSWORD = <password>
```

```
CREATE USER autograder
WITH PASSWORD = 'pass123'
```

- Set permissions

```
GRANT <permissions>
ON <table>
TO <user/role>
```

```
GRANT SELECT
ON MySecureTable
TO autograder
```

DBMS controls

Example: On Azure…

- Create a user

```
CREATE USER <username>
WITH PASSWORD = <password>
```

```
CREATE USER autograder
WITH PASSWORD = 'pass123'
```

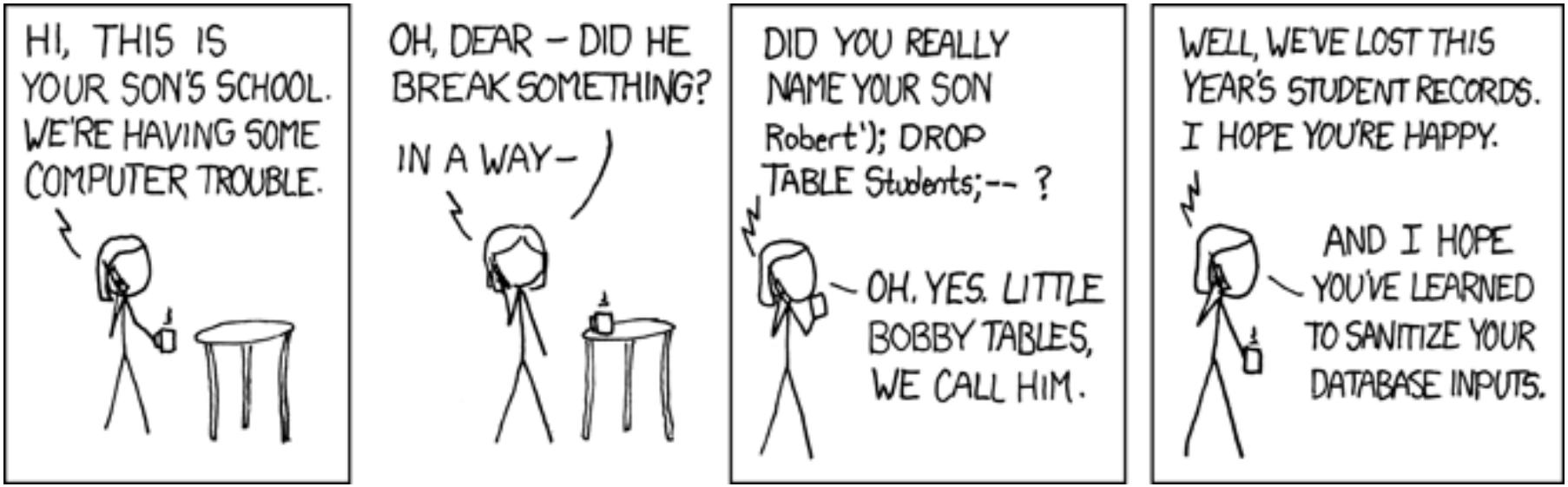
- Set permissions

```
GRANT <permissions>
ON <table>
TO <user/role>
```

```
GRANT SELECT
ON MySecureTable
TO autograder
```

If you login to *autograder*, you won't be able to UPDATE, DELETE, etc.!

SQL Injection



SQL Injection

- **SQL Injection**: Application input acts as code
- In applications, SQL queries are strings
  - Partly consists of user input
- Malicious user can trick DBMS into thinking their input is part of SQL code

SQL Injection

- Example: School admin database

```
INSERT INTO Students (id, 'name');
```

SQL Injection

- Example: School admin database

```
INSERT INTO Students (id, 'name');
```

- Malicious user input (in application):

```
id = 123
name = "Robert');   DROP TABLE Students;--"
```

SQL Injection

- Example: School admin database

```
INSERT INTO Students (id, 'name');
```

- Malicious user input (in application):

```
id = 123
name = "Robert');   DROP TABLE Students;--"
```

- Send application's query string to database:

```
INSERT INTO Students (123, 'Robert'); DROP TABLE Students;-- ');
```

SQL Injection

- Example: School admin database

```
INSERT INTO Students (id, 'name');
```

- Malicious user input (in application):

```
id = 123
name = "Robert');   DROP TABLE Students;--"
```

- Send application's query string to database:

```
INSERT INTO Students (123, 'Robert'); DROP TABLE Students;-- ');
```

```
INSERT INTO Students (123, 'Robert'); DROP TABLE Students;-- ');
```

SQL Injection

- Other types of SQL injections
  - Union attack, retrieve another table's data

```
SELECT username, email
FROM Users
WHERE id =
    'user1' UNION SELECT username, password FROM AdminUsers --';
```

SQL Injection

- Other types of SQL injections
  - Union attack, retrieve another table's data

```
SELECT username, email
FROM Users
WHERE id =
    'user1' UNION SELECT username, password FROM AdminUsers --';
```

  - Tautology attack, force TRUE condition to bypass filters

```
SELECT *
FROM Users
WHERE username = 'user1' OR 1 = 1 --' AND password = 'pass'
```

SQL Injection

- Consistently one of the top web-based attacks
  - In 2012, Yahoo! exposed ~450k emails/passwords
  - In 2011, Sony potentially exposed PII from 1M+ users
  - ~23% of all web vulnerabilities in 2023

SQL Injection

- Consistently one of the top web-based attacks
  - In 2012, Yahoo! exposed ~450k emails/passwords
  - In 2011, Sony potentially exposed PII from 1M+ users
  - ~23% of all web vulnerabilities in 2023

- Considered a “solved” problem
  - **Parameterize queries with prepared statements**

Java: JDBC

- Java API library to access DBMS
- PreparedStatement: prevent SQL injection attacks

Java: JDBC

- Java API library to access DBMS
- PreparedStatement: prevent SQL injection attacks

```
Statement withoutPlaceholder = con.createStatement();
withoutPlaceholder.execute("INSERT INTO Students VALUES('" + userInput + "')");

PreparedStatement withPlaceholder =
                 con.prepareStatement("INSERT INTO Student VALUES(?)");
withPlaceholder.setString(1, userInput);
withPlaceholder.execute();
```

Agenda

- Access Control
- **Passwords**
- Data Privacy

Storing Passwords

- Passwords are special
  - High potential for additional security compromises
  - Only operation that should be done is equality comparison

Storing Passwords

- Naive solution?

Storing Passwords

- Naive solution?

Username	Password
bobtheninja246	password
xDragonSlayerx	asdf
annabelle2001	password
lamamaster123	ilovefish
theSQLexpert234	j62ld12446
seahawksrule12	j62ld12446

Storing Passwords

NEVER store passwords in plaintext!!

Username	Password
bobtheninja246	password
xDragonSlayerx	asdf
annabelle2001	password
lamamaster123	ilovefish
theSQLexpert234	j62ld12446
seahawksrule12	j62ld12446

Hashing

- Hash(input) → hash value
  - Input → “scrambled” output

Hashing

- Hash(input) → hash value
  - Input → “scrambled” output
- Hashing is <u>deterministic</u>
  - Same input → same output

Hashing

- Hash(input) → hash value
  - Input → “scrambled” output
- Hashing is <u>deterministic</u>
  - Same input → same output
- Hashing (should be) <u>noninvertible</u>
  - For secure hash functions, computationally infeasible to derive input from the hash value

Storing Hashed Passwords

- Store hash instead

Username	Password
bobtheninja246	password
xDragonSlayerx	asdf
annabelle2001	password
lamamaster123	ilovefish
theSQLexpert234	j62ld12446
seahawksrule12	j62ld12446

Username	HashedPassword
bobtheninja246	3da541…
xDragonSlayerx	bfd361…
annabelle2001	3da541…
lamamaster123	5baa61…
theSQLexpert234	ca8612…
seahawksrule12	ca8612…

Validating Hashed Passwords

- Store hash instead
  - Validate any given password by hashing it and comparing with stored hash

Username	HashedPassword
bobtheninja246	3da541…
xDragonSlayerx	bfd361…
annabelle2001	3da541…
lamamaster123	5baa61…
theSQLexpert234	ca8612…
seahawksrule12	ca8612…

Validating Hashed Passwords

- Store hash instead
  - Validate any given password by hashing it and comparing with stored hash

- lamamaster123 logs in with "ilovefish"

Username	HashedPassword
bobtheninja246	3da541…
xDragonSlayerx	bfd361…
annabelle2001	3da541…
lamamaster123	5baa61…
theSQLexpert234	ca8612…
seahawksrule12	ca8612…

Validating Hashed Passwords

- Store hash instead
  - Validate any given password by hashing it and comparing with stored hash

- lamamaster123 logs in with “ilovefish”
- hash(“ilovefish”) → 5baa61…

Username	HashedPassword
bobtheninja246	3da541…
xDragonSlayerx	bfd361…
annabelle2001	3da541…
lamamaster123	5baa61…
theSQLexpert234	ca8612…
seahawksrule12	ca8612…

Validating Hashed Passwords

- Store hash instead
  - Validate any given password by hashing it and comparing with stored hash

- lamamaster123 logs in with "ilovefish"
- hash("ilovefish") → 5baa61…
- 5baa61… == 5baa61…
  - Accept login

Username	HashedPassword
bobtheninja246	3da541…
xDragonSlayerx	bfd361…
annabelle2001	3da541…
lamamaster123	5baa61…
theSQLexpert234	ca8612…
seahawksrule12	ca8612…

Storing Passwords

- But… there are always users with bad passwords

Username	Password
bobtheninja246	**password**
xDragonSlayerx	**asdf**
annabelle2001	**password**
lamamaster123	ilovefish
theSQLexpert234	**j62ld12446**
seahawksrule12	**j62ld12446**

Username	HashedPassword
bobtheninja246	**3da541…**
xDragonSlayerx	**bfd361…**
annabelle2001	**3da541…**
lamamaster123	5baa61…
theSQLexpert234	**ca8612…**
seahawksrule12	**ca8612…**

Storing Passwords

- But… there are always users with bad passwords
  - Easy to search for common password hashes!

Username	Password
bobtheninja246	**password**
xDragonSlayerx	**asdf**
annabelle2001	**password**
lamamaster123	ilovefish
theSQLexpert234	**j62ld12446**
seahawksrule12	**j62ld12446**

Username	HashedPassword
bobtheninja246	**3da541…**
xDragonSlayerx	**bfd361…**
annabelle2001	**3da541…**
lamamaster123	5baa61…
theSQLexpert234	**ca8612…**
seahawksrule12	**ca8612…**

Storing Passwords

- But… there are always users with bad passwords
  - Easy to search for common password hashes!
  - Easy to spot shared or reused passwords!

Username	Password
bobtheninja246	**password**
xDragonSlayerx	**asdf**
annabelle2001	**password**
lamamaster123	ilovefish
theSQLexpert234	**j62ld12446**
seahawksrule12	**j62ld12446**

Username	HashedPassword
bobtheninja246	**3da541…**
xDragonSlayerx	**bfd361…**
annabelle2001	**3da541…**
lamamaster123	5baa61…
theSQLexpert234	**ca8612…**
seahawksrule12	**ca8612…**

Salting

- **Salting** adds randomness to password

Salting

- **Salting** adds randomness to password

- Generate random salt, pass it to hash function

```
salt = getRandomSalt();
saltedPasswordHash = hash(password, salt)
```

Salting

- **Salting** adds randomness to password

- Generate random salt, pass it to hash function

```
salt = getRandomSalt();
saltedPasswordHash = hash(password, salt)
```

- Store salt and salted hash in database

Storing Salted + Hashed Passwords

Username	Password
bobtheninja246	**password**
xDragonSlayerx	**asdf**
annabelle2001	password
lamamaster123	ilovefish
theSQLexpert234	j62ld12446
seahawksrule12	j62ld12446

Username	Salt	HashedPassword
bobtheninja246	17	**7a4959…**
xDragonSlayerx	m9	**59438a…**
annabelle2001	23	4c812e…
lamamaster123	q7	3e0e04…
theSQLexpert234	k3	dcfea6…
seahawksrule12	ji	e840fc…

- Harder for attackers to search for common hashes
  - Unique salts → unique hashes for the same password

Storing Salted + Hashed Passwords

Username	Password
bobtheninja246	**password**
xDragonSlayerx	asdf
annabelle2001	**password**
lamamaster123	ilovefish
theSQLexpert234	**j62ld12446**
seahawksrule12	**j62ld12446**

Username	Salt	HashedPassword
bobtheninja246	17	**7a4959…**
xDragonSlayerx	m9	59438a…
annabelle2001	23	**4c812e…**
lamamaster123	q7	3e0e04…
theSQLexpert234	k3	**dcfea6…**
seahawksrule12	ji	**e840fc…**

- Harder to spot pairs of users ***sharing passwords***
  - Unique salts → unique hashes within dataset

Storing Salted + Hashed Passwords

Username	Password
bobtheninja246	**password**
xDragonSlayerx	asdf
annabelle2001	**password**
lamamaster123	ilovefish
theSQLexpert234	**j62ld12446**
seahawksrule12	**j62ld12446**

Username	Salt	HashedPassword
bobtheninja246	17	**7a4959…**
xDragonSlayerx	m9	59438a…
annabelle2001	23	**4c812e…**
lamamaster123	q7	3e0e04…
theSQLexpert234	k3	**dcfea6…**
seahawksrule12	ji	**e840fc…**

- Harder to spot pairs of users ***sharing passwords***
  - Unique salts → unique hashes within dataset
- Harder to spot users ***reusing passwords*** from other stolen datasets
  - Unique salts → unique hashes across datasets

Validating Hashed Passwords

- Validate by applying the stored salt before hashing

Username	Salt	Hashed Password
…	…	…
lamamaster123	q7	3e0e04…
…	…	…

- lamamaster123 logs in with “ilovefish”

Validating Hashed Passwords

- Validate by applying the stored salt before hashing

Username	Salt	Hashed Password
…	…	…
lamamaster123	q7	3e0e04…
…	…	…

- lamamaster123 logs in with “ilovefish”
- **salt = *getSaltFromDB*(‘lamamaster123’)**

Validating Hashed Passwords

- Validate by applying the stored salt before hashing

Username	Salt	Hashed Password
…	…	…
lamamaster123	q7	3e0e04…
…	…	…

- lamamaster123 logs in with “ilovefish”
- **salt = *getSaltFromDB*(‘lamamaster123’)**
- hash(“ilovefish”, **salt**) → 3e0e04…

Validating Hashed Passwords

- Validate by applying the stored salt before hashing

Username	Salt	Hashed Password
…	…	…
lamamaster123	q7	3e0e04…
…	…	…

- lamamaster123 logs in with “ilovefish”
- **salt = *getSaltFromDB*(‘lamamaster123’)**
- hash(“ilovefish”, **salt**) → 3e0e04…
- 3e0e04… == 3e0e04…
  - Accept login

Agenda

- Access Control
- Passwords
- **Data Privacy**

Existing Laws and Regulations

- HIPAA (Health Information Portability and Accountability Act), 1996
  - Mandatory for healthcare and health insurance institutions
  - Privacy rule - patient rights, PHI, use/disclosure
  - Security rule - standards for safeguards

Existing Laws and Regulations

- HIPAA (Health Information Portability and Accountability Act), 1996
  - Mandatory for healthcare and health insurance institutions
  - Privacy rule - patient rights, PHI, use/disclosure
  - Security rule - standards for safeguards

- GDPR (General Data Protection Regulation), 2018
  - Corporate disclosure and limits on user data storage
  - User data rights over PII

Existing Laws and Regulations

- FERPA (Family Education Rights and Privacy Act)
- Mandatory for education institutions
  - Requires written consent to disclose academic info, with certain exceptions (court orders, school officials, etc.)

Existing Laws and Regulations

- FERPA (Family Education Rights and Privacy Act)
- Mandatory for education institutions
  - Requires written consent to disclose academic info, with certain exceptions (court orders, school officials, etc.)

- Allows institutions to disclose “directory information” without consent (institution policies can be stronger)
  - Name
  - Email
  - Photographs
  - Phone Number

Implicit Disclosure

- If users can derive sensitive information like grades, it violates FERPA

Implicit Disclosure

- If users can derive sensitive information like grades, it violates FERPA

FERPA Deidentification

1) ID to anonymous ID mapping should be secret

2) Aggregate data (minimum n-size)
   - **Suppression** → Don't provide data ☹
     - Necessary for very small groups
   - **Rounding** → Bucket data or introduce noise ☺
     - More people means you can be more specific

Implicit Disclosure

- “Hey, can you give me the directory information for students with a GPA of 3.5?”

Reveals sensitive information by context

```
SELECT D.*
  FROM Directory AS D, Grades AS G
 WHERE D.id = G.id AND
       G.gpa = 3.5
```

Implicit Disclosure

- "Hey, can you give me the directory information for students with a GPA of 3.5?"

Reveals sensitive information by context

```
SELECT D.*
  FROM Directory AS D, Grades AS G
 WHERE D.id = G.id AND
       G.gpa = 3.5
```

- Database admins/designers should prevent these sorts of queries from being possible!

Anonymity

- Common practice for making a dataset private: remove Personal Identifiable Information (PII)

Anonymity

- Common practice for making a dataset private: remove Personal Identifiable Information (PII)

- But by linking data from distinct datasets one can reveal private information

Anonymity

- Common practice for making a dataset private: remove Personal Identifiable Information (PII)
- But by linking data from distinct datasets one can reveal private information
- In her PhD thesis* (2001) Latanya Sweeney described a famous example

* https://dspace.mit.edu/handle/1721.1/8589

Implicit Disclosure

Re-identification of Mass. Governor William Weld

- **Public voter data**
  - Name
  - ZIP code
  - Sex
  - Birth date
  - …
- **Anonymous insurance data** (released under assurance of anonymity from Gov. Weld)
  - ZIP code
  - Sex
  - Birth date
  - Prescription
  - Diagnosis
  - …

Latanya Sweeney's Finding

- Massachusetts: GIC* is responsible for health insurance of state employees; public data

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

*Group Insurance Commission

Latanya Sweeney's Finding

- Massachusetts: GIC* is responsible for health insurance of state employees; public data
- Sweeney paid $20 and bought voter registration list for Cambridge, MA

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

```
VOTER(name, party, ...,
     zip, dob, sex)
```

*Group Insurance Commission

Latanya Sweeney's Finding

- Massachusetts: GIC* is responsible for health insurance of state employees; public data
- Sweeney paid $20 and bought voter registration list for Cambridge, MA
- William Weld** lived in Cambridge: in VOTER

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

```
VOTER(name, party, ...,
     zip, dob, sex)
```

*Group Insurance Commission
**former governor

Latanya Sweeney's Finding

- Massachusetts: GIC* is responsible for health insurance of state employees; public data
- Sweeney paid $20 and bought voter registration list for Cambridge, MA
- William Weld** lived in Cambridge: in VOTER
- 6 people had same **dob**

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

```
VOTER(name, party, ...,
     zip, dob, sex)
```

*Group Insurance Commission
**former governor

Latanya Sweeney's Finding

- Massachusetts: GIC* is responsible for health insurance of state employees; public data
- Sweeney paid $20 and bought voter registration list for Cambridge, MA
- William Weld** lived in Cambridge: in VOTER
- 6 people had same **dob**
- 3 had also **sex**='M'

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

```
VOTER(name, party, ...,
     zip, dob, sex)
```

*Group Insurance Commission
**former governor

Latanya Sweeney's Finding

- Massachusetts: GIC* is responsible for health insurance of state employees; public data
- Sweeney paid $20 and bought voter registration list for Cambridge, MA
- William Weld** lived in Cambridge: in VOTER
- 6 people had same **dob**
- 3 had also **sex**='M'
- Weld only one in that **zip**

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

```
VOTER(name, party, ...,
     zip, dob, sex)
```

*Group Insurance Commission
**former governor

Latanya Sweeney's Finding

Cambridge, MA Voter Data ($20)

Name	ZIP	Sex	Bday
…	…	…	…
W. Weld	12345	M	Feb 30
…	…	…	…

$\bowtie$

Anon. Insurance Data for Researchers

ZIP	Sex	Bday	MedInfo
…	…	…	…
12345	M	Feb 30	Afluenza
…	…	…	…

Latanya Sweeney's Finding

Cambridge, MA Voter Data ($20)

Name	ZIP	Sex	Bday
…	…	…	…
W. Weld	12345	M	Feb 30
…	…	…	…

$\bowtie$

Anon. Insurance Data for Researchers

ZIP	Sex	Bday	MedInfo
…	…	…	…
12345	M	Feb 30	Afluenza
…	…	…	…

6 matches on ZIP
3 matches on Sex
1 match on Bday

Name	…	MedInfo
…	…	…
W. Weld	…	Afluenza
…	…	…

Sweeney learned Weld's medical records !

Latanya Sweeney's Finding

Cambridge, MA Voter Data ($20)

Name	ZIP	Sex	Bday
…	…	…	…
W. Weld	12345	M	Feb 30
…	…	…	…

⋈

Anon. Insurance Data for Researchers

ZIP	Sex	Bday	MedInfo
…	…	…	…
12[illegible]	M	Feb 30	Afluenza
…	…	…	…

Legal in 1997
Deemed private since 2003

6 matches on ZIP
3 matches on Sex
1 match on Bday

Name	…	MedInfo
…	…	…
W. Weld	…	Afluenza
…	…	…

Sweeney learned Weld's medical records !

Sensitive Information

- Personal Identifiable Information (PII)
  - Names
  - Student ID
  - Social security number
  - License number
- Protected data (for legal and/or ethical reasons)
  - Academic records (FERPA)
  - Protected Health Information (HIPAA)
  - User Web Data (GDPR)
- Passwords

Takeaways

- Always parameterize input into prepared statements to prevent SQL injection
  - JDBC PreparedStatements for HW6
- Always hash + salt passwords before storing them in a database
  - You will implement this in HW6
- Be careful about what information can be inferred from your datasets
  - Always protect sensitive data!