CS5322 Database Security

Access Control vs. Inference

- Access controls ensure that all *direct* accesses to objects are authorized
- But they do not necessarily prevent *inference* of sensitive information

Motivating Example

- Suppose that
  - We have an NUH medical database that contains the medical records below
  - We allow a US researcher to study the data, but do not allow him to know the patient identities

Name	Birth Date	Gender	ZIP	Disease
Alice	1960/01/01	F	10000	flu
Bob	1965/02/02	M	20000	dyspepsia
Cathy	1970/03/03	F	30000	pneumonia
David	1975/04/04	M	40000	gastritis

Medical Records

Motivating Example

- A straightforward approach: restrict the US researcher's access to identifying information like names, IDs, etc.

~~Name~~	Birth Date	Gender	ZIP	Disease
~~Alice~~	1960/01/01	F	10000	flu
~~Bob~~	1965/02/02	M	20000	dyspepsia
~~Cathy~~	1970/03/03	F	30000	pneumonia
~~David~~	1975/04/04	M	40000	gastritis

Medical Records

Motivating Example

- A straightforward approach: restrict the US researcher's access to identifying information like names, IDs, etc.
- It looks good at the first glance
- But does it provide sufficient privacy protection?
- No

Birth Date	Gender	ZIP	Disease
1960/01/01	F	10000	flu
1965/02/02	M	20000	dyspepsia
1970/03/03	F	30000	pneumonia
1975/04/04	M	40000	gastritis

Medical Records

Motivating Example

- A straightforward approach: restrict the US researcher’s access to identifying information like names, IDs, etc.

match

Name	Birth Date	Gender	ZIP
Alice	1960/01/01	F	10000
Bob	1965/02/02	M	20000
Cathy	1970/03/03	F	30000
David	1975/04/04	M	40000

Voter Registration List

Birth Date	Gender	ZIP	Disease
1960/01/01	F	10000	flu
1965/02/02	M	20000	dyspepsia
1970/03/03	F	30000	pneumonia
1975/04/04	M	40000	gastritis

Medical Records

Privacy incident: the MGIC case

- Time: mid-1990s
- Publisher: Massachusetts Group Insurance Commission (MGIC)
- Data released: "anonymized" medical records
- Result: A PhD student at MIT was able to identify the medical record of the governor of Massachusetts

match

Name	Birth Date	Gender	ZIP
Alice	1960/01/01	F	10000
Bob	1965/02/02	M	20000
Cathy	1970/03/03	F	30000
David	1975/04/04	M	40000

Voter Registration List

Birth Date	Gender	ZIP	Disease
1960/01/01	F	10000	flu
1965/02/02	M	20000	dyspepsia
1970/03/03	F	30000	pneumonia
1975/04/04	M	40000	gastritis

Medical Records

Privacy incident: the MGIC case

- Research [Golle 06] shows that 63% of Americans can be uniquely identified by {date of birth, gender, zip code}

match

Name	Birth Date	Gender	ZIP
Alice	1960/01/01	F	10000
Bob	1965/02/02	M	20000
Cathy	1970/03/03	F	30000
David	1975/04/04	M	40000

Voter Registration List

Birth Date	Gender	ZIP	Disease
1960/01/01	F	10000	flu
1965/02/02	M	20000	dyspepsia
1970/03/03	F	30000	pneumonia
1975/04/04	M	40000	gastritis

Medical Records

Lesson Learned

- What went wrong?
- Intuition: Although the identifiers are removed from the data, some *quasi-identifiers* remain
- Can we solve the problem by removing quasi-identifiers? Unfortunately, no.

Name	Birth Date	Gender	ZIP
Alice	1960/01/01	F	10000
Bob	1965/02/02	M	20000
Cathy	1970/03/03	F	30000
David	1975/04/04	M	40000

Voter Registration List

Birth Date	Gender	ZIP	Disease
1960/01/01	F	10000	flu
1965/02/02	M	20000	dyspepsia
1970/03/03	F	30000	pneumonia
1975/04/04	M	40000	gastritis

Medical Records

Privacy incident: the AOL case

- In 2006, AOL released an "anonymized" log of their search engine to support research
- Example of the log:

User ID	Query	Date/Time	…
4417749	"Bitcoin price"	…	…
4417749	"1MDB Scandal"	…	…
4417749	"Clementi Mall opening hours"	…	…

- Each user only has an ID, i.e., no identifier or quasi-identifier is released
- However, the New York Time was able to identify a user from the log

Privacy incident: the AOL case

- What the New York Time did:
  - Find all log entries for AOL user 4417749
  - Multiple queries for businesses and services in Lilburn, GA (population 11K)
  - Several queries for Jarrett Arnold
  - Lilburn has 14 people with the last name Arnold
  - NYT contacts them, finds out AOL User 4417749 is Thelma Arnold

- The CTO of AOL resigned after the incident

Lesson Learned

- What went wrong?
- Intuition: Although all identifiers and quasi-identifiers are removed, the users’ behavior traces (i.e., their search keywords) reveal their identities
- The same problem occurred in another incident in 2006

Privacy incident: the Netflix case

- In 2006, the Netflix movie rental service released some movie ratings made by its users, for a competition with a 1M USD prize
- Example of data:

User ID	Movie	Rating	Date
123	Scary Movie 1	5	2006.07.01
123	Scary Movie 2	4	2006.07.08
123	Scary Movie 3	4	2006.07.15

- Each user only has an ID, i.e., no identifier or quasi-identifier is released
- However, two researchers from U. Texas were able to link some users to some online identities

Privacy incident: the Netflix case

User ID	Movie	Rating	Date
123	Scary Movie 1	5	2006.07.01
123	Scary Movie 2	4	2006.07.08
123	Scary Movie 3	4	2006.07.16

IMDB ID	Movie	Rating	Date
456	Scary Movie 1	5	2006.07.01
456	Scary Movie 2	4	2006.07.09
456	Scary Movie 3	4	2006.07.15

- What the researchers did:
  - Go to a movie review site IMDB, and get the ratings made by the IMDB users, as well as the dates
  - Match the an IMDB user to a Netflix user, if both users give the same ratings to the same movies on similar dates

Privacy incident: the Netflix case

User ID	Movie	Rating	Date
123	Scary Movie 1	5	2006.07.01
123	Scary Movie 2	4	2006.07.08
123	Scary Movie 3	4	2006.07.16

IMDB ID	Movie	Rating	Date
456	Scary Movie 1	5	2006.07.01
456	Scary Movie 2	4	2006.07.09
456	Scary Movie 3	4	2006.07.15

- In general, 99% of users can be identified with 8 ratings + dates
- Result: Netflix was sued; case settled out of court

Lessons learned

- Now we know that it is risky to publish detailed records of individual data, since
  - quasi-identifiers may reveal identities
  - behavior information may reveal identities, too
- What if we don't release detailed records, but only aggregate information?
- Answer: it could still fail to protect privacy

Privacy incident: the GWAS case

- The national institutes of health (NIH) used to publish aggregate information from genome wide association studies (GWAS)
- Typical setup:
  - Take the DNA of 1,000 individuals with a common disease (e.g., diabetes)
  - Check the DNAs for 100,000 DNA markers
  - For each marker, release its frequency in the 1,000 individuals

	Marker 1	Marker 2	Marker 3	…	Marker 100,000
Frequency	0.02	0.03	0.05	…	0.02

Privacy incident: the GWAS case

- Homer et al. [2008] demonstrate that it is possible to infer whether an individual is in the test population (i.e., the 1,000 individuals)

test population

	Marker 1	Marker 2	Marker 3	…	Marker 100,000
Frequency	0.02	0.03	0.05	…	0.02

Xiaokui's DNA

	Marker 1	Marker 2	Marker 3	…	Marker 100,000
Yes or No?	No	Yes	Yes	…	Yes

reference population

	Marker 1	Marker 2	Marker 3	…	Marker 100,000
Frequency	0.01	0.01	0.04	…	0.01

Privacy incident: the GWAS case

- Result: NIH removed public accesses to their GWAS results

test population

	Marker 1	Marker 2	Marker 3	…	Marker 100,000
Frequency	0.02	0.03	0.05	…	0.02

Xiaokui's DNA

	Marker 1	Marker 2	Marker 3	…	Marker 100,000
Yes or No?	No	Yes	Yes	…	Yes

reference population

	Marker 1	Marker 2	Marker 3	…	Marker 100,000
Frequency	0.01	0.01	0.04	…	0.01

Lessons learned

- Even aggregate information could endanger privacy, if the attacker could find the right background information to use
- In the GWAS case:
  - aggregation information: frequencies of DNA markers
  - background information: statistics from a reference population

Coming Next

- How we may alleviate inference by using *statistical databases*

Statistical Database: Motivation

- Databases that are intended to alleviate inference of sensitive information
- Idea:
  - Do not provide access to detailed records
  - Only provide statistics of records through SUM, MEAN, MEDIAN, COUNT, MAX, and MIN, etc.

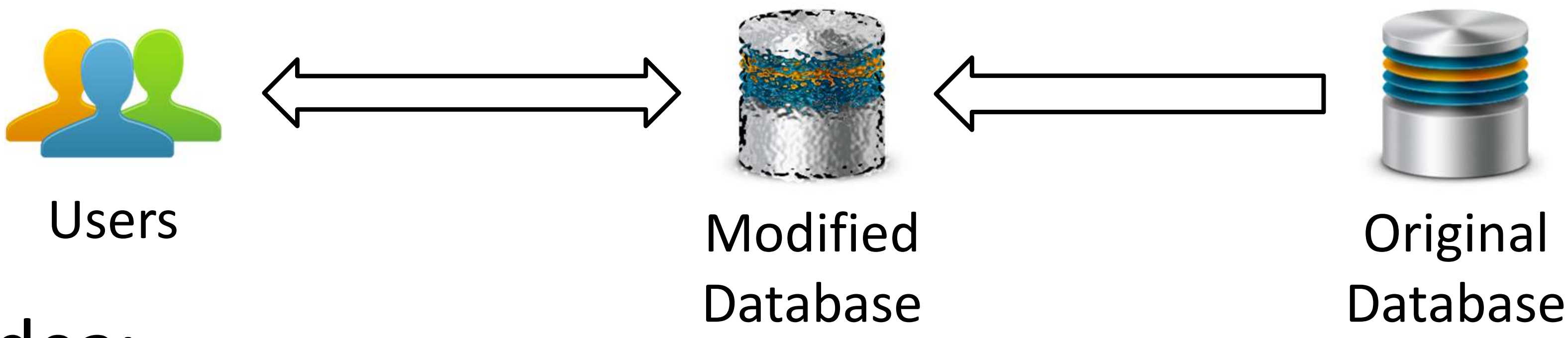
Statistical Database: Motivation

- "But we already know that statistics can be used for inference!"
- Yes. So a statistical database would apply some additional measure for inference control
  - Three canonical approaches:
    - Query auditing
    - Data perturbation
    - Output perturbation

Query Auditing

- Idea: Keep track of users' queries to a statistical database to see if the queries reveal sensitive information
- Two variants
  - Online auditing
    - Keep tack of queries in real time, and denies queries that are unsafe
  - Offline auditing
    - Keep a log of all queries, and run offline tests to check if any unsafe queries have been issued
- Online auditing is more difficult to do due to efficiency requirements

Data Perturbation



- Idea:
  - Modify the original data
  - Answer users' queries on the modified data instead of the original one
- Rationale:
  - As long as the modification is done properly, queries on the modified data would not reveal too much sensitive information
- Example:
  - Census data released by the US Census Bureau

Output Perturbation

- Idea:
  - Inject noise into each query answer to conceal sensitive information
- Example:
  - Query: “How many student get A+?”
  - Answer: [0, 5]
- Difference from query auditing:
  - Query auditing either denies a query or gives exact answers
  - Output perturbation only gives noisy answers

Coming Next

- Query auditing
- Data perturbation
- (We won't talk about output perturbation as it is less commonly used.)

Query Set Size Control

- The simplest form of query auditing
- Idea: impose requirements on the *selectivity* of each query
  - The selectivity of a query is defined as the number of tuples that satisfy the query predicate
  - Example: The query below has a selectivity of 4

```
SELECT   SUM(Grade)
FROM     Grades
WHERE    Program = 'CS'
```

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

Query Set Size Control

- Query set size control:
  - Each query's selectivity must be at least K
- Example: K = 2
  - Q1 is OK, but not Q2
- Rationale:
  - Queries with small selectivities (e.g., Q2) are likely to reveal sensitive information
- Question: Is this good enough?

```
Q1:   SELECT  SUM(Grade)
      FROM    Grades
      WHERE   Program = 'CS'

Q2:   SELECT  SUM(Grade)
      FROM    Grades
      WHERE   Program = 'CS'
              AND Gender = 'F'
```

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

selectivity of 1, so only Alice

Query Set Size Control

- Question: Is this good enough?
  - No

difference attack

- Example: K = 2
  - Q3 and Q4’s selectivities are 6 and 5, respectively
  - But Q3 – Q4 reveals Alice’s grade

```
Q3:    SELECT    SUM(Grade)
       FROM      Grades

Q4:    SELECT    SUM(Grade)
       FROM      Grades
       WHERE     Name <> 'Alice'
```

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

Query Set Size Control

- Observation:
  - Q3 and Q4 two queries reveal information because they *overlap* a lot
- What if we require that queries should not overlap too much?

```
Q3:     SELECT    SUM(Grade)
        FROM      Grades

Q4:     SELECT    SUM(Grade)
        FROM      Grades
        WHERE     Name <> ‘Alice’
```

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

Query Set Overlap Control

- Requirement: For any two queries Q and Q', their *query set* should overlap on at most *r* tuples
  - i.e., there should exist at most *r* tuples that satisfy the predicates in Q and Q' simultaneously
- Example: *r* = 1
  - If the user issues Q5 and then Q6, then Q6 would be denied, because their query sets overlap on 3 tuples

```
Q5:    SELECT   SUM(Grade)
       FROM     Grades
       WHERE    Program = 'CS'

Q6:    SELECT   SUM(Grade)
       FROM     Grades
       WHERE    Program = 'CS'
                AND Name <> 'Alice'
```

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

Query Set Overlap Control

- Requirement: For any two queries Q and Q', their *query set* should overlap on at most *r* tuples
  - i.e., there should exist at most *r* tuples that satisfy the predicates in Q and Q' simultaneously
- Example: *r* = 1
  - If the user issues Q7 and then Q8, then both queries would be answered, since their query sets overlap only on one tuple

```
Q7:    SELECT   SUM(Grade)
       FROM     Grades
       WHERE    Gender = 'F'

Q8:    SELECT   SUM(Grade)
       FROM     Grades
       WHERE    Program = 'CS'
```

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

Query Set Overlap Control

- Requirement: For any two queries Q and Q', their *query set* should overlap on at most *r* tuples
  - i.e., there should exist at most *r* tuples that satisfy the predicates in Q and Q' simultaneously
- Is this good enough?
- No; not even we control both query selectivity and query set overlap

```
Q7:     SELECT   SUM(Grade)
        FROM     Grades
        WHERE    Gender = 'F'

Q8:     SELECT   SUM(Grade)
        FROM     Grades
        WHERE    Program = 'CS'
```

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

Query Set Overlap Control

- Consider the four queries below
- We require that
  - Each query should have a selectivity at least K = 2
  - Any two queries should overlap on at most r = 1 tuple
- Selectivities: Q7: 2; Q8: 2; Q9: 2; Q10: 3
- Overlaps: Any two queries have overlap at most 1
- However, (Q7 + Q8 + Q9 – Q10)/3 reveals Alice's grade

Q7: SELECT SUM(Grade) FROM Grades
WHERE Gender = 'F' 2

Q8: SELECT SUM(Grade) FROM Grades
WHERE Program = 'CS' 2

Q9: SELECT SUM(Grade) FROM Grades
WHERE Age = 20 2

Q10: SELECT SUM(Grade) FROM Grades
WHERE Name = 'Bob' OR Name = 'Cathy' OR Name = 'Dave' 3

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Bob	21	M	CS	90
Cathy	21	F	IS	80
Dave	20	M	IS	70

Grades

Query Set Overlap Control

- What went wrong?
- Checking the overlaps of all 2-query combinations is not sufficient
- What if we check the overlaps of all k-query (k > 2) combinations?
- Information could still be leaked if the attacker exploits more query correlations than just overlaps
- Conclusion: query set overlap control does not really work

Q7: SELECT SUM(Grade) FROM Grades
WHERE Gender = 'F'

Q8: SELECT SUM(Grade) FROM Grades
WHERE Program = 'CS'

Q9: SELECT SUM(Grade) FROM Grades
WHERE Age = 20

Q10: SELECT SUM(Grade) FROM Grades
WHERE Name = 'Bob' OR Name = 'Cathy' OR Name = 'Dave'

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Bob	21	M	CS	90
Cathy	21	F	IS	80
Dave	20	M	IS	70

Grades

More Advanced Auditing

- Keep a log of all queries issued by a user
- Use an advanced algorithm to decide whether the queries can reveal any particular tuple
- But this can only be done if the queries are restricted to a certain type

More Advanced Auditing

- Example: If we only allow SUM queries
  - Each SUM query q_i can be modelled as a linear combination of the tuple values:
    $q_i = w_{i1}*x_1 + w_{i2}*x_2 + \ldots + w_{in}*x_n$, where
    - x_j denotes the j-th tuple value, and
    - $w_{ij} = 1$ if q_i covers the j-th tuple, otherwise $w_{ij} = 0$

```
Q1:    SELECT   SUM(Grade)
       FROM     Grades
       WHERE    Gender = 'F'
```

Answer = $[1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0] \cdot \begin{bmatrix} 80 \\ 90 \\ 90 \\ 100 \\ 90 \\ 90 \end{bmatrix}$

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

More Advanced Auditing

- If we have m queries, then they form a linear system $Q = WX$, where
  - the i-th element of Q is q_i,
  - the i-th row of W is $[w_{i1}, w_{i2}, \ldots, w_{in}]$,
  - and i-th element of X is x_i

Q1: SELECT SUM(Grade) FROM Grades
WHERE Gender = 'F'

Q2: SELECT SUM(Grade) FROM Grades
WHERE Program = 'CS'

$$\text{Answer} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 80 \\ 90 \\ 90 \\ 100 \\ 90 \\ 90 \end{bmatrix}$$

Name	Gender	Program	Grade
Alice	F	CS	80
Bob	M	CS	90
Cathy	F	IS	90
Daisy	F	IS	100
Eric	M	CS	90
Fred	M	CS	90

Grades

More Advanced Auditing

- If we have *m* queries, then they form a linear system $Q = WX$, where
  - the *i*-th element of Q is q_i,
  - the *i*-th row of W is $[w_{i1}, w_{i2}, \ldots, w_{in}]$,
  - and *i*-th element of X is x_i

Example: $Q = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 80 \\ 90 \\ 90 \\ 100 \\ 90 \\ 90 \end{bmatrix}$

- If the linear system can uniquely decide the value of a value x_k, then it means that the queries can reveal x_k
  - This can be check in a relatively efficient manner

More Advanced Auditing

- “OK! Let’s extend this approach to handle more general types of queries, e.g., SUM + MAX + MIN!”
- Well, it is not going to work…
- It is difficult to design an efficient auditing algorithm when non-linear queries (e.g., MAX) are involved
- Furthermore, denial of queries can actually leak information…

Inference from Denial of Queries

- Suppose that a user issues Q1, and then Q2
- For Q1, the database returns count = 3 and sum = 270
- Now consider Q2
- From the database's perspective:
  - "If I return 90 as the answer of Q2, the user would know that all three CS students' grades are 90!"
  - "I have to deny Q2."

```
Q1:     SELECT   COUNT(*), SUM(Grade)
        FROM     Grades
        WHERE    Program = 'CS'

Q2:     SELECT   MAX(Grade) FROM Grades
        WHERE    Program = 'CS'
```

Name	Age	Gender	Program	Grade
Alice	20	F	CS	90
Bob	21	M	CS	90
Cathy	21	F	CS	90
Daisy	20	M	IS	80

Grades

Inference from Denial of Queries

- The user sees that Q2 is denied
- From the user's perspective:
  - "The database denies Q2, only if Q1 and Q2 can jointly reveal the grade of some particular student…"
  - "But Q1 only tells me that there are 3 CS students, and their grade sum is 270…"
  - "Why would the max CS grade reveal sensitive information?"
  - "Oh! It only happens when the max CS grade is 90!"
- The user then learns Alice, Bob, and Cathy's grades based on the denial of Q2

```
Q1:     SELECT   COUNT(*), SUM(Grade)
        FROM     Grades
        WHERE     Program = 'CS'

Q2:     SELECT   MAX(Grade) FROM Grades
        WHERE    Program = 'CS'
```

Name	Age	Gender	Program	Grade
Alice	20	F	CS	90
Bob	21	M	CS	90
Cathy	21	F	CS	90
Daisy	20	M	IS	80

Grades

Inference from Denial of Queries

- How to avoid inference from the denial of queries?
- Make sure that the auditing algorithm is *data independent*
- That is, the algorithm should
  - Consider all possible tables
  - Deny queries as long as there is *one* possible table on which the queries would reveal information
- Example:
  - Even if Cathy's grade is 80 instead of 90, the database should still deny Q2
  - This prevents the user from learning anything from the denial of Q2
- However, designing algorithms like this is difficult

```
Q1:    SELECT   COUNT(*), SUM(Grade)
       FROM     Grades
       WHERE    Program = 'CS'

Q2:    SELECT   MAX(Grade) FROM Grades
       WHERE    Program = 'CS'
```

Name	Age	Gender	Program	Grade
Alice	20	F	CS	90
Bob	21	M	CS	90
Cathy	21	F	CS	90
Daisy	20	M	IS	80

Grades

Query Auditing: Summary

- We have discussed query auditing based on
  - Query set sizes
  - Query set overlaps
  - Linear systems
- All of them have limitations
- Strengthening them is not easy, especially because the denial of queries itself could reveal information

Query Auditing: Exercise

Name	Age	Gender	Program	Grade
Alice	20	F	CS	70
Bob	21	M	CS	80
Cathy	21	F	CS	90
Daisy	20	M	IS	100
Elsa	20	F	IS	90
Fion	21	F	IS	80
Gill	21	M	IS	70

Grades

- Assume that an adversary can issue queries in the form of

  SELECT SUM(Grade)
  FROM Grades
  WHERE …

- with the following constraints:
  - The query predicate can involve at most 2 attributes, and cannot involve Name
  - Each query set size is at least 3, and
  - Any two queries should overlap on at most 2 tuples
- Further assume that the adversary knows the Age, Gender, and Program of each student
- Demonstrate how the adversary may infer Alice's grade

Query Auditing: Exercise

- One possible answer:
  - Q1: SELECT SUM(Grade)
    FROM Grades
    WHERE Age = 20
  - Q2: SELECT SUM(Grade)
    FROM Grades
    WHERE Program = "CS"
  - Q3: SELECT SUM(Grade)
    FROM Grades
    WHERE (Age = 21 AND Program = "IS") OR (Age = 20 AND Program = "CS")
  - Q4: SELECT SUM(Grade)
    FROM Grades
    WHERE (Age <> 20) OR (Program <> "CS")
- (Q1+Q2+Q3-Q4)/3 reveals Alice's grade

Name	Age	Gender	Program	Grade
Alice	20	F	CS	70
Bob	21	M	CS	80
Cathy	21	F	CS	90
Daisy	20	M	IS	100
Elsa	20	F	IS	90
Fion	21	F	IS	80
Gill	21	M	IS	70

Grades

Query Auditing: Exercise

- Consider a table Grades(<u>Name</u>, Grade)
- Suppose that an adversary can issue queries in the following form:

  SELECT MEDIAN(Grade) FROM Grades

  WHERE [*some conditions on Name*]
- Examples
  - Median of {1, 3, 8, 10, 17} is 8
  - Median of {1, 8, 10, 17} is (8+10)/2 = 9
- Suppose that
  - We monitor the adversary's queries and deny a query whenever it reveals a specific student's exact grade
  - But we ignores the information leaked by the denial of queries
- Show an example in which an adversary can infer a student's exact grade based on the query denial

Query Auditing: Exercise

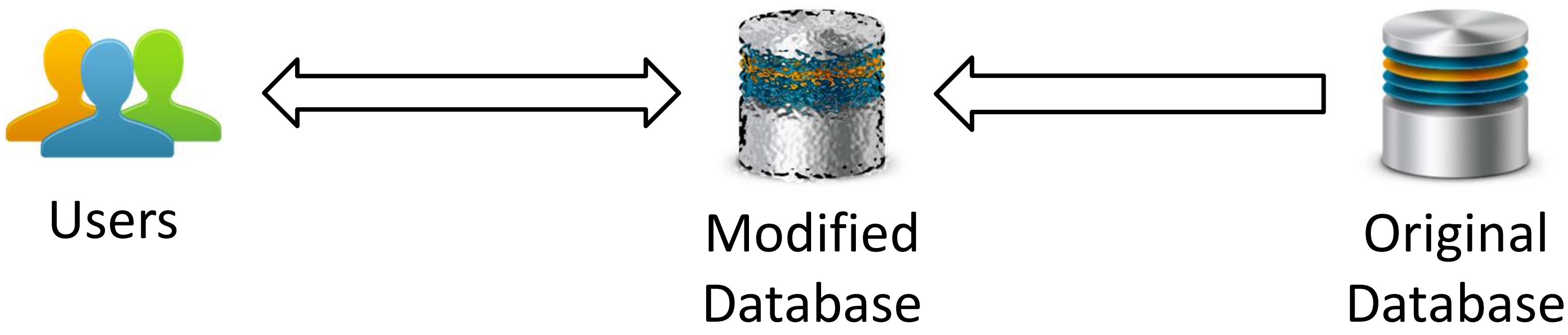
- One possible answer:
  - Three median queries:
    - Q1: {A, B}
    - Q2: {A, B, C}
    - Q3: {A, C}
  - Three answers:
    - Q1: 20
    - Q2: 20
    - Q3: denied
  - The adversary can learn that Student A's grade must be 20

Name	Grade
A	?
B	?
C	?

Coming Next

- Data perturbation

Data Perturbation



- Idea:
  - Modify the original data
  - Answer users' queries on the modified data instead of the original one
- We will discuss a few methods for data perturbation

Data Perturbation: Generalization

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- Idea: Modify the tuples to make them less distinguishable
- Observe that in the modified table
  - The first two tuples are indistinguishable based on Name, Age, Gender, and Program
  - The same goes for the third and fourth tuples
- This makes some query results less accurate
- Example 1: COUNT(*) WHERE Age = 20 AND Gender = ‘F’
  - Query result: [0, 2]

Data Perturbation: Generalization

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- Idea: Modify the tuples to make them less distinguishable
- Observe that in the modified table
  - The first two tuples are indistinguishable based on Name, Age, Gender, and Program
  - The same goes for the third and fourth tuples
- This makes some query results less accurate
- Example 2: SUM(Grade) WHERE Age = 20 AND Gender = ‘M’
  - Query result: [0, 150]

Data Perturbation: Generalization

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- Why would generalization work?
- Rationale:
  - Whatever queries a user issues, the best that the user can infer is the generalized table *T**
  - Even if the user learns the whole *T**, he still cannot pinpoint the grade of any student
- Example:
  - From *T**, the user cannot figure out which tuple belongs to Alice

Data Perturbation: Generalization

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- But how much generalization is needed?
- We need a way to measure the degree of protection

Data Perturbation: k-Anonymity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- A generalized table is *k-anonymous*, if each tuple is indistinguishable from at least k-1 other tuples on the non-sensitive attributes
- Example above: *T** is 2-anonymous
- Rationale:
  - If there are *k* indistinguishable tuples, then the adversary won't be able to uniquely link an individual to any tuple

Data Perturbation: k-Anonymity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- But does k-anonymity provide sufficient protection?
- No
- Why?
- Imagine that Alice's grade is 90 instead of 100

Data Perturbation: k-Anonymity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	90
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	90
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- *T** is still 2-anonymous
  - The adversary won't know whether Alice correspond to the first or second tuple
- But it does not matter…
- Both the first and second tuples have the same grade
- So the adversary knows that Alice's grade must be 90

Data Perturbation: k-Anonymity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	90
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	90
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- What went wrong?
  - k-anonymity ensures that an individual can be linked to at least *k* tuples via non-sensitive attributes
  - But it does not ensure anything about the sensitive attribute of those tuples
  - When those tuples happen to have homogeneous sensitive values, k-anonymity fails

Data Perturbation: k-Anonymity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	90
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	90
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- How could we improve k-anonymity?
- Idea: Generalize tuples into indistinguishable groups, such that
  - Each group has a *diverse* set of sensitive values
- This leads to the notion of l-diversity

Data Perturbation: *l*-diversity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- A generalized table is *l*-diverse, if each indistinguishable group of tuples has at least *l* *well-represented* sensitive values
  - The term "well-represented" is application dependent
- Example above:
  - If we define "well-represented" grades as grades that differ by at least 5 pair-wise, then *T** is 2-diverse
  - If we define "well-represented" grades as grades that differ by at least 20 pair-wise, then *T** is NOT 2-diverse

Data Perturbation: *l*-diversity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- Rationale:
  - If there are *l* well-represented sensitive values in each indistinguishable group, then the adversary won't be able to uniquely link an individual to any particular sensitive value

Data Perturbation: *l*-diversity

Name	Age	Gender	Program	Grade
Alice	20	F	CS	100
Cathy	21	F	CS	90
Bob	21	M	IS	80
Dave	20	M	IS	70

Original Table *T*

Name	Age	Gender	Program	Grade
*	20-21	F	CS	100
*	20-21	F	CS	90
*	20-21	M	IS	80
*	20-21	M	IS	70

Generalized Table *T**

- Is *l*-diversity "bulletproof"?
- No
  - Problem 1: It is not easy to correctly define what "*l* well-represented" sensitive values mean

Limitations of *l*-diversity

Name	Age	Gender	ZIP	Disease
Alice	30	F	100000	Breast Cancer
Bob	30	M	190000	HIV
Cathy	40	F	210000	Dyspepsia
Dave	40	M	280000	Pneumonia

Original Table *T*

Name	Age	Gender	ZIP	Disease
*	30	*	1*****	Breast Cancer
*	30	*	1*****	HIV
*	40	*	2*****	Dyspepsia
*	40	*	2*****	Pneumonia

Generalized Table *T**

- Example above:
  - *T** seems to be 2-diverse, since each indistinguishable group has at least two different diseases
  - However, if an adversary knows Bob's Age, Gender, and ZIP code, then he can easily infer Bob's disease, because
    - Bob can be linked to the first two tuples
    - The first two tuples' sensitive values are breast cancer and HIV
    - Bob is very unlikely to have breast cancer

Limitations of *l*-diversity

Name	Age	Gender	ZIP	Disease
Alice	30	F	100000	Breast Cancer
Bob	30	M	190000	HIV
Cathy	40	F	210000	Dyspepsia
Dave	40	M	280000	Pneumonia

Original Table *T*

Name	Age	Gender	ZIP	Disease
*	30	*	1*****	Breast Cancer
*	30	*	1*****	HIV
*	40	*	2*****	Dyspepsia
*	40	*	2*****	Pneumonia

Generalized Table *T**

- What went wrong?
  - The adversary was able to exclude some sensitive values from an indistinguishable group, based on a piece of background knowledge:
    - Men are unlikely to have breast cancer

Limitations of *l*-diversity

Name	Age	Gender	ZIP	Disease
Alice	30	F	100000	Breast Cancer
Bob	30	M	190000	HIV
Cathy	40	F	210000	Dyspepsia
Dave	40	M	280000	Pneumonia

Original Table *T*

→

Name	Age	Gender	ZIP	Disease
*	30	*	1*****	Breast Cancer
*	30	*	1*****	HIV
*	40	*	2*****	Dyspepsia
*	40	*	2*****	Pneumonia

Generalized Table *T**

- How can we deal with this?
  - Need to take into account the adversary's background knowledge
  - But it is difficult predict what background knowledge the adversary may have
- And this is not the only problem that *l*-diversity have…
- The algorithm used to generate *l*-diverse tables could leak information

l-diversity Algo

Name	Age	Gender	ZIP	Disease
Alice	20	F	100000	Flu
Betty	20	F	100000	Measles
Carl	50	M	800000	Dyspepsia
Dave	50	M	820000	HIV
Fred	50	M	820000	Pneumonia

Original Table T

Name	Age	Gender	ZIP	Disease
*	20	F	100000	Flu
*	20	F	100000	Measles
*	50	M	8*****	Dyspepsia
*	50	M	8*****	HIV
*	50	M	8*****	Pneumonia

Generalized Table T_1*

Name	Age	Gender	ZIP	Disease
*	20-50	*	******	Flu
*	20-50	*	******	Measles
*	20-50	*	******	Dyspepsia
*	50	M	820000	HIV
*	50	M	820000	Pneumonia

Generalized Table T_2*

- Consider the table T above
- We show two possible ways to generalize the table to satisfy 2-diversity
- Which one is better? T_1* or T_2*?
- T_1* is better since it preserves more information
- In general, when there are multiple ways to generate a l-diverse table, an algorithm would tend to choose one that preserves more information

l-diversity Algo

Name	Age	Gender	ZIP	Disease
Alice	20	F	100000	Flu
Betty	20	F	100000	Measles
Carl	50	M	800000	HIV
Dave	50	M	820000	HIV
Fred	50	M	820000	Pneumonia

Original Table T

Name	Age	Gender	ZIP	Disease
*	20	F	100000	Flu
*	20	F	100000	Measles
*	50	M	8*****	HIV
*	50	M	8*****	HIV
*	50	M	8*****	Pneumonia

Generalized Table T_1*

Name	Age	Gender	ZIP	Disease
*	20-50	*	******	Flu
*	20-50	*	******	Measles
*	20-50	*	******	HIV
*	50	M	820000	HIV
*	50	M	820000	Pneumonia

Generalized Table T_2*

- Now suppose Carl's disease is HIV instead of Measles
- Let's consider the two possible ways to generalize T
- T_1* is no longer 2-diverse; but T_2* still is
- So if we ask for a 2-diverse generalization, the algorithm would give us T_2*, even though it preserves less information

l-diversity Algo

Name	Age	Gender	ZIP	Disease
*	20	F	100000	Flu
*	20	F	100000	Measles
*	50	M	8*****	HIV
*	50	M	8*****	HIV
*	50	M	8*****	Pneumonia

Generalized Table T_1*

Name	Age	Gender	ZIP
Alice	20	F	100000
Betty	20	F	100000
Carl	50	M	800000
Dave	50	M	820000
Fred	50	M	820000

What the adversary knows

Name	Age	Gender	ZIP	Disease
*	20-50	*	******	Flu
*	20-50	*	******	Measles
*	20-50	*	******	HIV
*	50	M	820000	HIV
*	50	M	820000	Pneumonia

Generalized Table T_2*

- Suppose that we use T_2* to answer queries
- Consider an adversary who knows the Age, Gender, and ZIP code of each individual
- He sees T_2*, and starts thinking

l-diversity Algo

Name	Age	Gender	ZIP
Alice	20	F	100000
Betty	20	F	100000
Carl	50	M	800000
Dave	50	M	820000
Fred	50	M	820000

What the adversary knows

Name	Age	Gender	ZIP	Disease
*	20	F	100000	Flu
*	20	F	100000	Measles
*	50	M	8*****	HIV
*	50	M	8*****	HIV
*	50	M	8*****	Pneumonia

Generalized Table T_1*

Name	Age	Gender	ZIP	Disease
*	20-50	*	******	Flu
*	20-50	*	******	Measles
*	20-50	*	******	HIV
*	50	M	820000	HIV
*	50	M	820000	Pneumonia

Generalized Table T_2*

- "T_2* puts Alice, Betty, and Carl into one group, and Dave and Fred into another…"
- "This is really bad in terms of information preservation…"
- "Why don't they put Alice and Betty into one group, and Carl, Dave, and Fred into another? It would preserve more information…"
- "It must be the case that putting Carl, Dave, and Fred together would violate 2-diverslity!"
- "Carl must have HIV!"

l-diversity Algo

Name	Age	Gender	ZIP
Alice	20	F	100000
Betty	20	F	100000
Carl	50	M	800000
Dave	50	M	820000
Fred	50	M	820000

What the adversary knows

Name	Age	Gender	ZIP	Disease
*	20	F	100000	Flu
*	20	F	100000	Measles
*	50	M	8*****	HIV
*	50	M	8*****	HIV
*	50	M	8*****	Pneumonia

Generalized Table T_1*

Name	Age	Gender	ZIP	Disease
*	20-50	*	******	Flu
*	20-50	*	******	Measles
*	20-50	*	******	HIV
*	50	M	820000	HIV
*	50	M	820000	Pneumonia

Generalized Table T_2*

- What went wrong?
  - l-diversity ignores the fact that the adversary may know how the generalization algorithm works
  - Knowledge of the generalization algorithm could enable inference
- Kerckhoffs's principle (in cryptography):
  - A cryptosystem should be secure even if everything about the system, except the key, is public knowledge.

l-diversity Algo

Name	Age	Gender	ZIP	Disease
*	20	F	100000	Flu
*	20	F	100000	Measles
*	50	M	8*****	HIV
*	50	M	8*****	HIV
*	50	M	8*****	Pneumonia

Generalized Table T_1*

Name	Age	Gender	ZIP
Alice	20	F	100000
Betty	20	F	100000
Carl	50	M	800000
Dave	50	M	820000
Fred	50	M	820000

What the adversary knows

Name	Age	Gender	ZIP	Disease
*	20-50	*	******	Flu
*	20-50	*	******	Measles
*	20-50	*	******	HIV
*	50	M	820000	HIV
*	50	M	820000	Pneumonia

Generalized Table T_2*

- How to address this problem?
- Need to take into account that the adversary may know the generalization algorithm's details
- This makes the design of generalization algorithm quite challenging

Data Perturbation in Practice

- Despite the deficiencies of generalization, it is commonly used in practice
  - Often along with k-anonymity instead of l-diversity
- Reason: It is easy to understand
- Three other approaches used in practice
  - Data swapping
  - Synthetic data generation
  - Random perturbation