Authorization, Security, and Privacy

5DV119 — Introduction to Database Management

Umeå University
Department of Computing Science

Stephen J. Hegner
hegner@cs.umu.se
http://www.cs.umu.se/~hegner

Basic Notions

- Access to large databases is generally selective:
  - Privileges are local to each user or *rôle*.
  - The process of defining and granting these privileges is called *authorization*.
  - Authorization is a positive action, designed to grant specific users or rôles specific privileges.
- Large databases must also be protected from those who try to obtain information which they are not intended to have.
  - Intruders may attempt to gain access to the system from the outside.
  - Insiders may attempt to circumvent the authorization mechanism and gain access to information which they are not allowed to have.
  - Authorized users may attempt to extract unintended information from databases via techniques such as statistical tracking.
  - Measures taken to to control such access fall under the general heading of *security*, which is generally a negative or preventive measure.

Users and Rôles

- Rather than assign privileges directly to individual users, a more contemporary approach is to assign privileges to rôles.

Rôles: A *rôle* is a classification of users who are to be granted the same access privilege.

Examples: Supervisor, travel secretary.

- *Rôle-based* methods for authorization are becoming more widely used, particularly in large organizations in which many people will have the same right to a given part of a database.
- In these slides, when the term *user* is employed, it should be understood that the user may in fact be a rôle.
- Rôles were called *NPD*s (*named protection domains*) in early work [Baldwin 1990].

Authorization

- There are two general flavors of authorization:

  Discretionary authorization: Individuals (or rôles) are given certain access privileges on data objects, as well as privileges to propagate (*grant*) such privileges to others.

  Mandatory authorization: In this mode, each data object has a certain fixed classification, as does each user or rôle.

  - Only users (or rôles) with a qualifying classification may access a given data object.

Discretionary Access Control

Authority: An *authority* is a statement that a certain user or rôle has the right to perform a given action on the database.

Grant: The action of assigning authority is called *granting*.

Revoke: The action of relinquishing authority which has previously been granted is called *revocation*.

Basic rules governing granting and revocation:

- A user/rôle U has privilege P if and only if some other user/rôle U' with the authority to grant privilege P has in fact granted it to U.
- Only a user/rôle U with privilege P and the authority to grant P to others may in fact grant P to another user/rôle U'.
- A user/rôle U' may revoke a privilege P from user or rôle U if and only if U' had earlier granted that privilege to U.
  - However, U retains privilege P as long as at least one other user has granted that privilege to U, regardless of how many others have revoked it.
- The database administrator (DBA) grants initial privileges, to avoid a chicken-and-egg problem.

Authorization and SQL

- The general syntax for the assignment of a privilege is as follows:

```
GRANT <list of privileges>
ON    <list of database objects>
TO    <list of users>
[WITH GRANT OPTION];
```

  Notation: `[foo]` = 0 or 1 occurrences of `foo`.

- The allowed privileges are:
  - `SELECT`
  - `INSERT`
  - `DELETE`
  - `UPDATE`
  - `REFERENCES` = *references in integrity constraints (all integrity constraints, including by not limited to foreign keys).*

Basic Examples of Authorization in SQL

- The following gives users Smith and Jones the right to issue read-only (*i.e.*, `SELECT`) queries on the tables `Employee` and `Department`.

```
GRANT SELECT
ON    Employee, Department
TO    Smith, Jones;
```

- The following gives users Smith and Jones not only the `SELECT` privilege on the table, but also the right to pass this privilege along to other users.

```
GRANT SELECT
ON    Employee, Department
TO    Smith, Jones
WITH GRANT OPTION;
```

- Both assume that the issuer of the commands has the right to grant the specified privileges.
  - Otherwise, they fail.

Basic Examples of Authorization in SQL — 2

- The following gives users Smith and Jones the right to issue both `SELECT` queries and `UPDATE` commands on the `Employee` table.

```
GRANT SELECT, UPDATE
ON    Employee
TO    Smith, Jones;
```

  - Note that `UPDATE` has a specific semantics in SQL — namely to change the values in fields of a tuple.
  - It does not include the right to insert new tuples or to delete existing ones.

Basic Examples of Authorization in SQL — 3

- The following statement grants all forms of access except `REFERENCES`.

```
GRANT SELECT, UPDATE, INSERT, DELETE
ON    Employee
TO    Smith, Jones;
```

- In principle, it is possible to grant modification privileges without view privileges, but this would be problematic in terms of usage.

```
GRANT UPDATE, INSERT, DELETE
ON    Employee
TO    Smith, Jones;
```

- If `Smith` and `Jones` did not already have read privileges, they would be able to write data which they would not be allowed to read again.

Authorization within Views in SQL

- To grant privileges on only part of a relation or relations, a view must first be created.

```
CREATE VIEW Poor_Names_Only AS
       SELECT LastName, FirstName, MiddleInit
       FROM Employee
       WHERE (Salary < 20000);

GRANT SELECT
ON Poor_Names_Only
TO Smith;
```

Authorization within Views in SQL — 2

- It is even possible to grant privileges which are valid only at certain times:

```
CREATE VIEW Poor_Names_9_to_5 AS
       SELECT LastName, FirstName, MiddleInit
       FROM Employee
       WHERE (Salary < 20000)
       AND (Current_Time >= '09:00:00')
       AND (Current_Time <= '17:00:00');

GRANT SELECT
ON Poor_Names_9_to_5
TO Smith;
```

The REVOKE Directive of SQL

- The complement of GRANT is REVOKE.
- The general syntax is as follows:

```
REVOKE [GRANT OPTION FOR ] <list of privileges>
ON                         <list of database objects>
FROM                       <list of users>
RESTRICT | CASCADE;
```

  Notation: A | B = A or B.

- Here GRANT OPTION FOR is not just a noise phrase.
  - If specified, it indicates the revocation is just for the privilege to grant the privilege(s), not for the privilege itself.
  - If not specified, the command is to revoke the privilege(s) itself/themselves.
- A privilege or grant option for a privilege may only be revoked by a rôle which has granted that privilege or option in the first place.

Examples of REVOKE

- The following statement revokes the privilege of Smith to execute select operations on the relation Employee, and also revokes (in cascading fashion) any such privileges which Smith alone has granted.

```
REVOKE SELECT
ON     Employee
FROM   Smith
CASCADE;
```

- The following is similar, except that it does nothing if it would be required that the privilege be revoked from some other user in cascading fashion.

```
REVOKE SELECT
ON     Employee
FROM   Smith
RESTRICT;
```

Multiple GRANTs and REVOKE

Example: Suppose that both Washington and Lincoln issue identical GRANT commands of the following form.

```
GRANT SELECT
ON    Employee, Department
TO    Smith;
```

- Now suppose that Washington issues the following REVOKE.

```
REVOKE SELECT
ON     Employee
FROM   Smith
RESTRICT;
```

- In this case, although the command "succeeds", Smith retains the privilege because it was also granted by Lincoln.
- On the other hand, if Lincoln subsequently issues the same REVOKE command, Smith will lose the privilege.

Multiple GRANTs and REVOKE with CASCADE

- First, suppose that Washington grants a right to Lincoln:

```
GRANT SELECT ON Employee, Department TO Lincoln
WITH  GRANT OPTION;
```

- Now suppose that Lincoln passes this right on to Smith:

```
GRANT SELECT ON Employee, Department TO Smith;
```

- If Washington now executes the following statement, Smith as well as Lincoln will lose the associated privileges.

```
REVOKE SELECT ON Employee FROM Lincoln CASCADE;
```

- If CASCADE is replaced by RESTRICT, the directive will fail and both SMITH and Lincoln will retain the privilege.
- It is not clear how this failure is to be reported, since SQL does not have a standard status-return mechanism.

REVOKE with CASCADE — Further Issues

- When a privilege is revoked with the CASCADE option, any objects which require that privilege are also revoked.

Example: Suppose that Smith is granted read privileges on the Employee relation.

- Smith then creates a (read-only) view consisting of employees in the research department.
- If the privilege of reading the Employee relation is subsequently revoked from Smith with the CASCADE option, the view itself is dropped.
- This process is necessary to avoid *abandonment* — the existence of an object with no access.
- This suggests that CASCADE should be used with great care.

Authorization in PostgreSQL

- Privileges may be granted to any other user, but these privileges are useful only if that user is allowed to connect to the database on which the privileges were granted.
- If a user is allowed to connect to a database, then that user always has the privilege of creating new relations and using them.
- A user is always the owner of a relation created from that user account, regardless of the ownership of the actual database.
- Thus, if access is granted at all to a database, then the privilege of creating and owning new relations by those with access is irrevocable. even by the DBA.
- If you allow a user to connect to your database, then that user will be able to create and control relations within your database.
  - You may not even be able to read them!
  - The creator must grant privileges to you!

But... this applies only to access directly via PostgreSQL.

- More useful access control may be achieved via applications using ODBC or PHP.

Mandatory Access Control

- Mandatory access control is used in situations in which users (or rôles) may be assigned *security classes*.

Assumptions and notation:

- The security classes form a total order.
  Example: `Top_Secret > Secret > Confidential > Unclassified`
- Each user or rôle is assigned a security class.
  - Write $\mathrm{Clearance}\langle U \rangle$ to denote the clearance of U.
- Each data object is also assigned a security class.
  - Write $\mathrm{Classification}\langle P \rangle$ to denote the classification of P.

Simple security property: User or rôle U has read access to object P iff $\mathrm{Clearance}\langle U \rangle \geq \mathrm{Classification}\langle P \rangle$.

- No *read-up*.

Star property: User or rôle U has write access to object P iff $\mathrm{Clearance}\langle U \rangle \leq \mathrm{Classification}\langle P \rangle$.

- No *write-down*.

Analysis of the Star Property

- The intent of the star property is to prevent information from being passed down from a higher classification to a lower one.

Problem: Under the star property, a rôle may write data which it is not allowed to read.

Strong star property: Some sources stipulate the *strong star property*: $\mathrm{Clearance}\langle U \rangle = \mathrm{Classification}\langle P \rangle$

Question: Is this better?

Question: Is either star property realistic in practice?

Answer: Probably not without some modification.

- It should be possible to trust people with higher classifications not to carelessly write this information into documents or databases at lower classifications.
  - Thus, $\mathrm{Clearance}\langle U \rangle \geq \mathrm{Classification}\langle P \rangle$ seems more reasonable.
  - A review process can catch inadvertent errors.

Authority of the Database Administrator

- The *database administrator (DBA)* is the database equivalent of a system administrator.
- Typically, the DBA has sole authority in the following areas of authorization:
  - Create new accounts, and delete existing ones.
  - Assign security levels to accounts.
  - Assign initial authorization levels.
- Some of these responsibilities may be delegated in the management of very large systems, but only in very controlled ways.

Security

- There are at least three key security issues.

1. Prevent attacks from outside intruders.
   - The problem of *SQL injection* and its prevention will be examined in these slides.
2. Prevent unauthorized access from insiders.
   - A key technique is to maintain detailed logs.
3. Take care not to grant privileges unintentionally.
   - This aspect of security is called *privacy*, particularly when the data include confidential information about people (such as medical records).
   - This problem is particularly relevant in the area of statistical databases.
   - The problem of *statistical tracking* will be examined in these slides.

SQL Injection

- One of the most common ways to obtain unauthorized access to a database is via *SQL injection*.
- This problem occurs when parameters to an SQL query are included by pasting in the text received from the user.

Example: Prompt the user for an SSN, and then provide all information about the associated employee which is in the Employee relation.

- The proper way to implement this query in ODBC is to use argument parameters:

```
SELECT * FROM Employee WHERE SSN=?
```

Question: Why not be clever and do something like this instead?

query_left ← "SELECT * FROM Employee WHERE SSN='"
query_right ← "'"
query_total ← query_left1 · user_input · query_right

- So if the user types 999887777, then

query_total ← "SELECT * FROM Employee WHERE SSN='999887777'

SQL Injection — 2

Question: Why not be clever and do something like this instead?

query_left ← `"SELECT * FROM Employee WHERE SSN='"`
query_right ← `"'"`
query_total ← query_left1 · user_input · query_right

Answer: What if the user types `' OR 1=1 --` ?

- The query becomes:

query_total ← `"SELECT * FROM Employee WHERE SSN='' OR 1=1 --'`

- This query returns all tuples in the Employee relation!

Answer: What if the user types `'; DROP TABLE Employee --` ?

- The query becomes:

query_total ← `"SELECT * FROM Employee WHERE SSN=''; DROP TABLE Employee --'`

- This query should drop the entire Employee relation!
- Fortunately, most current ODBC implementations will only execute the first query in a sequence, or flag an error.

Preventing SQL Injection

- The best protection against SQL injection is to use parameters in ODBC queries, and not to use string concatenation.

- Concealing error messages from end users also helps, because such messages can give insight into the nature of the database schema.

- A sample Python program which illustrates SQL injection is available on the course Web site.

Privacy-Preserving Data Publishing

- It is common to grant *summary* access to large databases, without permitting detailed access.

Example query for a company database: Provide the average salary of all employees in the research department.

- The idea is to provide information about the general state of things, without revealing detailed, confidential information about individuals.
- Some databases, particularly those maintained by government agencies, are explicitly stated to be maintained for purposes of summary information only, with details about individuals held "strictly confidential".
- The buzzphrase is *privacy-preserving data publishing*.

Question: Can such privacy be maintained, and if so, how?

Anonymization

- A basic step in privacy-preserving data publishing is *anonymization*.
- This means that the data have been transformed so that the identity of individuals may no longer be determined via queries.

Problem: Anonymization is a matter of degree.

- That which seems to be sufficient typically is not.

Example: Suppose that in a large statistical database, the personal ID numbers of individuals are replaced with distinct, random values.

Question: Does such a measure suffice to protect the anonymity of each individual?

Answer: Individual trackers may be able to harvest information about individuals even when such measures are taken.

Individual Trackers

- An *individual tracker* is a query or sequence of queries designed to extract information about an individual in a statistical database.
- The following example is from D. E. Denning and P. J. Denning, Data Security, ACM Computing Surveys, Vol. 11, No. 3, 1979, pp. 227-249.

Context: An apparently anonymized medical database which allows only statistical queries.

Query 1: How many patients have these characteristics?

Male	Age 45-50	Married
Two children	Harvard law degree	Bank vice president

- Suppose that the questioner knows that Jones has these characteristics and the query returns a count of one.

Query 2: How many patients have these characteristics?

Male	Age 45-50	Married
Two children	Harvard law degree	Bank vice president
		Took drugs for depression

- The combined answers to these two statistical queries uses *background knowledge* to tell whether Jones took drugs for depression.

Minimum Query-Set Control

- A candidate solution to the problem of individual trackers is *minimum query-set control*.

Minimum query-set control: Fix a number $0 \leq q \leq 100$.

- Every query must retrieve at least q% of the records and no more than $(100 - q)\%$.
- Choose q so that both $q\%$ and $(100 - q)\%$ of the records is a large set.

- This eliminates the the tracking method illustrated in the example on the previous slide.

Problem: Even with such controls, security may be compromised.

- The trick is to use a *statistical tracker* rather than an individual tracker.

Statistical Tracking

Context: Assume that the Company database has been anonymized so that the values of the SSN, LName, FName, MInit, Address, BDate, and Super_SSN attributes have been anonymized.

- This includes corresponding attributes, such as ESSN of the Works_On and Dependent relations, and Mgr_SSN of the Department relation.

Query: Find the salary of Joyce English.

Known: Joyce is the only female who works on the ProductY project.

- The following statistical query is no longer allowed (for $q > 1$ and $q < r$, with r the total number of employees), since it returns only one tuple.

```
SELECT AVG(Salary)
FROM   Employee JOIN Works_On ON (SSN=ESSN)
                JOIN Project ON (PNO=PNumber)
WHERE  (PName='ProductY') AND (SEX='F');
```

- While both SSN and ESSN are anonymized, equality is values is preserved in the anonymization, so SSN=ESSN still provides correct matches.

General Trackers

- To overcome the limitations imposed by minimum query-set control, begin by identifying a general tracker.

General tracker (of degree q, $0 \leq q \leq 100$): The idea is that such a tracker must return at least q% of the possible tuples, and at most (100-q)%,

- with the additional condition that retrieving one more or one less tuple will not violate this condition.

- It satisfies minimum query-set control with a little room to spare.

More precisely: Suppose that the total number of tuples of the form which a query Q can possibly return return is n_Q.

- If Q retrieves tuples of a single relation, then n_Q is the total number of tuples in the relation.
- In general, think of creating a view first and then computing n_Q for the relation of that view.
- If n is the number of tuples which Q actually returns, then:

$$(n-1)/n_Q \geq q/100 \qquad (n+1)/n_Q \leq (100-q)/100$$

General Trackers — Counters

Example: Suppose that the query T below is a general tracker.

```
SELECT SUM(Salary)
FROM   Employee JOIN Department ON (DNO=DNumber)
WHERE (DName='Administration') ;
```

Counter: The *counter* T_0 of T counts the number of tuples returned.

```
SELECT Count(*), SUM(Salary)
FROM   Employee JOIN Department ON (DNO=DNumber)
WHERE (DName='Administration') ;
```

- This query is identical to T save that it also counts the number of employees in the sum.

General Trackers — Classifiers

The classifier: This query Q_0 determines whether Joyce is in the result set of T_0.

```
SELECT Count(*), SUM(Salary)
FROM  Employee JOIN Department ON (DNO=DNumber)
WHERE (DName='Administration') OR
      (SSN IN (SELECT E.SSN
               FROM Employee E JOIN Works_On ON (E.SSN=ESSN)
                               JOIN Project ON (PNO=PNumber)
               WHERE (PName='ProductY') AND (Sex='F'))) ;
```

- If the count returned by T_0 is one larger than that returned by Q_0, then Joyce does not work in the `Administration` department.
- In that case, it is easy to compute the salary of Joyce as the difference in total salaries of the two queries.

General Trackers — Complementary Counters / Classifiers

- If the count returned by Q_0 is the same as the count returned by T_0, then use the complementary counter.

Complementary counter: The *complementary counter* T_1 of T counts the number of tuples not returned by T_0.

```
SELECT Count(*), SUM(Salary)
FROM   Employee JOIN Department ON (DNO=DNumber)
WHERE (DName<>'Administration') ;
```

The complementary classifier: Q_1

```
SELECT Count(*), SUM(Salary)
FROM  Employee JOIN Department ON (DNO=DNumber)
WHERE (DName<>'Administration') OR
      (SSN IN (SELECT E.SSN
               FROM Employee E JOIN Works_On ON (E.SSN=ESSN)
                               JOIN Project ON (PNO=PNumber)
               WHERE (PName='ProductY') AND (Sex='F'))) ;
```

- The salary of Joyce may be obtained as the difference of the salary sum returned by T_1 and that returned by Q_1.

Quasi-Identifiers

Anon_ID	Sex	BirthDate	PostCode	Illness	Duration	Treatment

- Consider the medical-record relation shown above.
- Anon_ID is an *anonymized ID*; *e.g.*, an encrypted personnummer.
  - It may or may not be included, depending upon whether or not full association between illness instances is to be retained.

Quasi-identifier A *quasi-identifier* (or *quasi-ID*) is a set of attributes from which the ID can be recovered or nearly recovered (*i.e.*, up to a small set of possibilities) in a large percentage of cases.

Example: $\{\text{Sex}, \text{BirthDate}, \text{PostCode}\}$ is a likely candidate for a quasi-ID in the above case.

- In contrast to the example of statistical tracking, a quasi-ID is not customized for a given individual; it should work for most individuals.

Question: How can privacy be preserved in the presence of quasi-identifiers?

Classification of Attributes

Anon_ID	Sex	BirthDate	PostCode	Illness	Duration	Treatment

- Attributes may be classified into two groups.

Identifier attributes: Used to identify the individual

- Typically part of a quasi-ID.
- Not viewed as highly sensitive information.

Example: $\{\text{Sex}, \text{BirthDate}, \text{PostCode}\}$ are likely candidates for identifier attributes in the above case.

Sensitive attributes: Attributes containing information about individuals which must be protected

Example: $\{\text{Illness}, \text{Duration}, \text{Treatment}\}$ are likely candidates for sensitive attributes in the above case.

Aggregation Techniques for Privacy Preservation

Anon_ID	Sex	BirthDate	PostCode	Illness	Duration	Treatment

Suppression (total aggregation): In *suppression*, an entire attribute is removed (or all entries replaced with the same value).

- Often but not necessarily an identifier attribute.

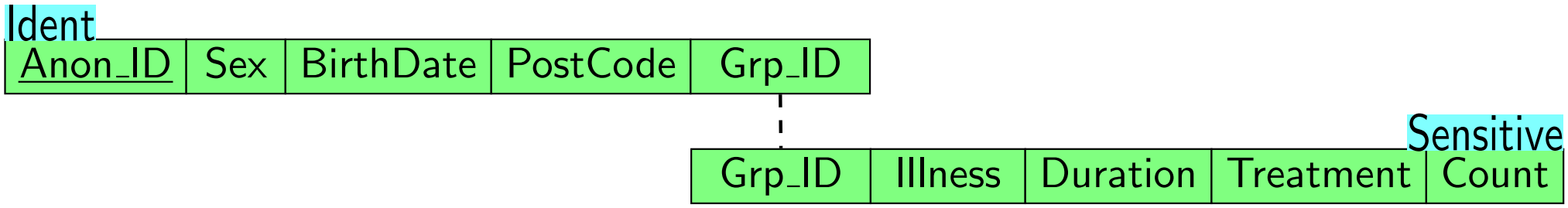
Example: Replace Sex (or BirthDate, or both) with the same value `unspecified` in all tuples.

Generalization (partial aggregation): Replace values with an equivalence class.

Example: Replace BirthDate with just the year of birth, or replace it with its five-year block (*e.g.*, 1960-1964, 1965-1969, *etc*).

Assessment: Aggregation is effective in reducing loss of privacy, but it also collapses attribute values which may be useful in statistical analysis.

Anatomization Techniques for Privacy Preservation



Anatomization: Decompose the relation into two parts, one containing the identifier attributes and the other containing the sensitive attributes.

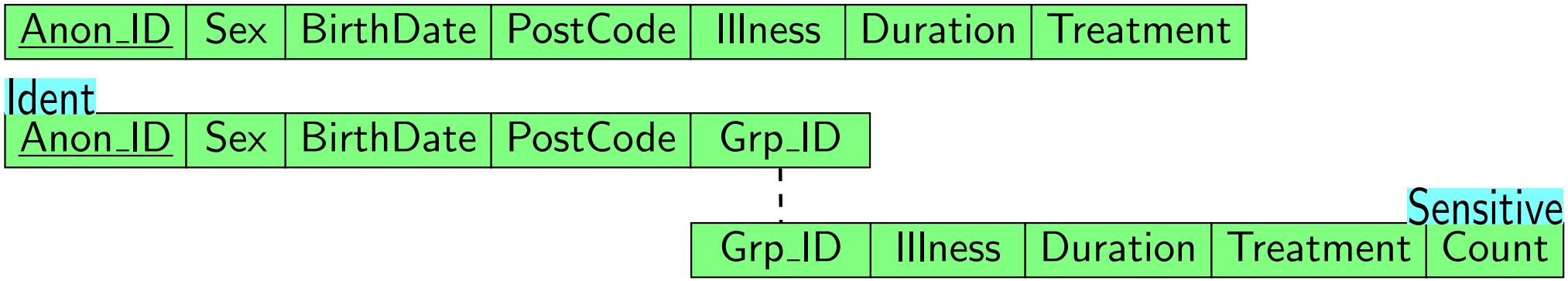
- Also called *bucketization*.

Grouping: The tuples in the Ident relation are grouped, with each group having a unique *group ID*.

- Tuples in the same group have the same group ID.

- Group ID is also an attribute of the Sensitive relation.
  - If some tuple in the Ident relation matches the data in a row of the Sensitive relation, the associated group ID is included in that row.
  - A count of the number of matches for a given group ID may also be included.
- The join of these two relations will provide too many associations, thus enhancing privacy.

Comparison of Aggregation and Anatomization



Aggregation: In aggregation, it is the values themselves which are made less specific to enhance privacy.

- The table is not broken, so there is no loss of association between what is left of the identifier and what is left of the sensitive values after aggregation.

Anatomization: In anatomization, values are preserved; it is the association between identifiers and values which is made less specific in order to preserve privacy.

Assessment: Which of these two approaches is most appropriate depends upon what the application needs in terms of data.

- They mangle the data in different ways.

Other Ways of Preserving Privacy

Grouping and Breaking: Aggregation and anatomization fall under the general heading of *grouping and breaking*.

- Other approaches to preserving privacy include:

Adding noise:

- Introduce "noise" into the result of a query, so that numerical answers are not exact.
- This must be done in certain ways, so that the noise cannot be filtered out by integrating the results of a large number of queries.

Random samples:

- Instead of providing a database with all individuals, include only a random sample.
- This technique is useful for very large statistical-only databases, such as census databases.
- Users do not know who is in the sample.

For More Information on Privacy-Preserving Data Publishing

- A good survey which is easy to find:
  - Benjamin C. M. Fung, Ke Wang, Rui Chen, and Philip S. Yu, Privacy-Preserving Data Publishing: A Survey of Recent Developments, *ACM Computing Surveys*, Vol 42, No. 4, Article 14, June 2010.

- A comprehensive survey, but not publicly available:
  - Raymond Chi-Wing Wong and Ada Wai-Chee Fu, Privacy-Preserving Data Publishing: An Overview, *Synthesis Lectures on Data Management*, Morgan and Claypool, 2010.