ISYS2120: Data & Information Management

Week 7A: Database Security

Alan Fekete

Based on slides by Rohm, Khushi, Fekete and from textbooks

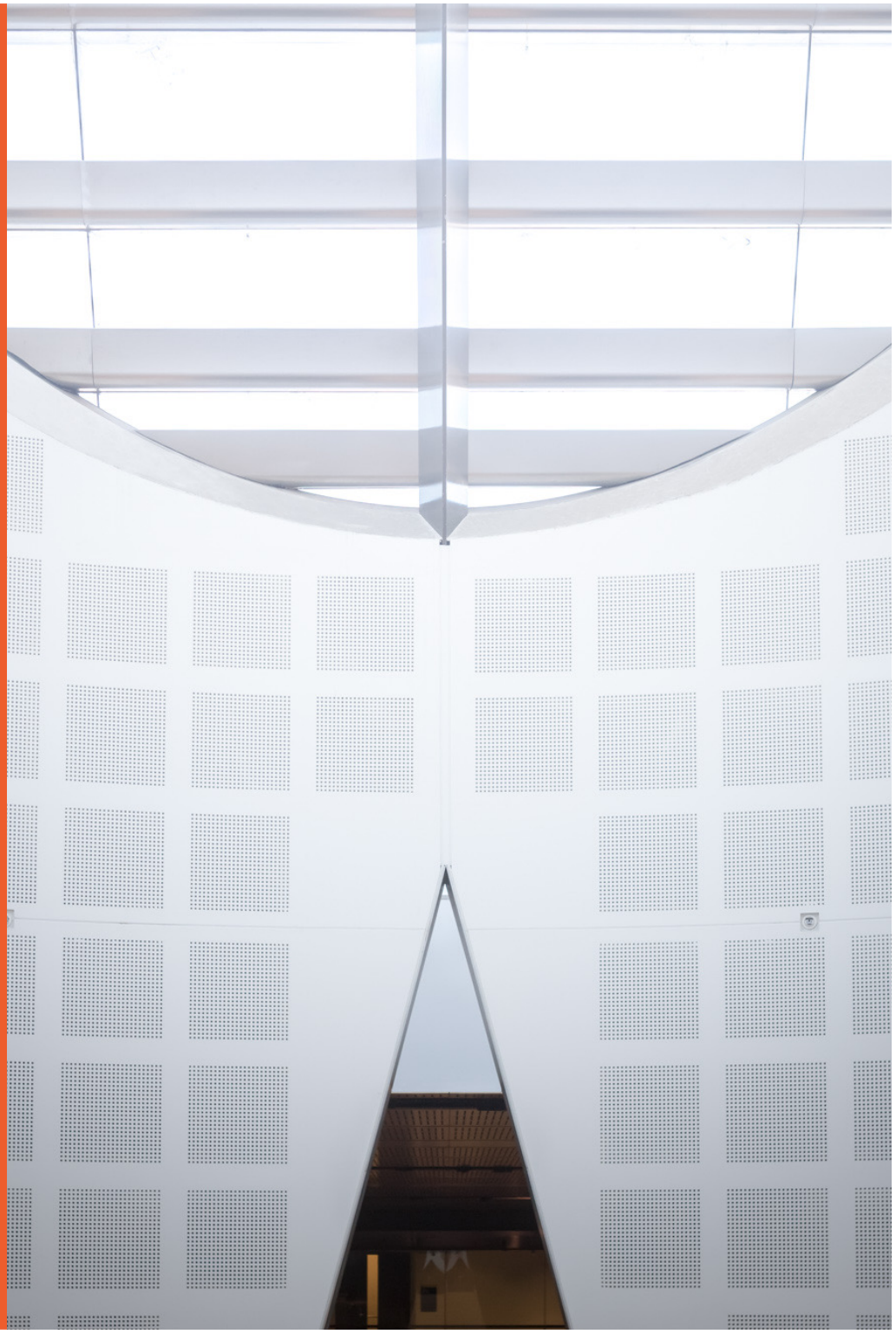
Cf. Kifer/Bernstein/Lewis – Chapter 3.2-3.3

Ramakrishnan/Gehrke – Chapter 5.7-5.9;

Silberschatz/Korth/Sudarshan – 4.2, 4.4, 5.3

Ullman/Widom – Chapter 7

THE UNIVERSITY OF SYDNEY



Introduction

- Many examples of data breach or other security failure
  - Eg ANU data breach (staff, student records)
  - See https://en.wikipedia.org/wiki/List_of_data_breaches
- Lots of damage
  - Financial loss
  - Reputation loss
  - Legal penalties

Database Security

- Databases usually contain commercially valuable information
  - Possibly also personal information
- There are many adversaries who could benefit from accessing information, or altering it
- The organisation that owns the data should be willing to spend money to protect the data against inappropriate uses

Database Security Goals

- System and Organisation need to set things up so as to guarantee:
  - **Confidentiality Properties**: Users should not be able to see things they are not supposed to.
    - E.g., A student can't see other students' grades.
  - **Integrity Properties**: Users should not be able to modify things they are not supposed to.
    - E.g., Only instructors can assign grades.
  - **Availability Properties**: Users should be able to see and modify things they are allowed to.

Policy and Mechanism

- A security policy indicates who *should be allowed* to do which actions, and who should not be allowed
  - This is set by senior management, based on legal obligations, business value, risk, etc
- A security mechanism is how the system controls who *is allowed* to do which actions
  - This is controlled by DBA, possibly with detailed adjustment by data owner
- We expect that the security mechanism will enforce the security policy in place
  - This correspondence needs to be checked!

Database Access Control

- **Access control (for authorization)**
  - A mechanism provided by dbms for controlling which users perform various operations on various parts of the database
  - It is called “discretionary” access control, because the decision about what is permitted or not, is set by appropriate people as the system operates, and can be adjusted as the situation changes
    - Contrast to mandatory access control, where the decisions are built in to the system, and unchangeable
- In SQL, based on the concept of access rights or **privileges** for objects (tables), and commands for giving users privileges (and revoking privileges).
  - Creator of a table automatically gets all privileges on it.
    - DMBS keeps track of who subsequently gains and loses privileges, and ensures that only requests from users who have the necessary privileges (at the time the request is issued) are allowed.

GRANT command

GRANT privilegelist ON *object* TO *userlist* [WITH GRANT OPTION]

- The following **privileges** can be mentioned in privilegelist:
  - **SELECT**: Allows to read all columns (including those added later via ALTER TABLE command).
  - **INSERT:** Allows to insert extra tuples.
  - **DELETE**: Allows to delete tuples.
  - **UPDATE (col-name)**: allows to modify the values in this column in any tuples of the table
    - **UPDATE** without col0name means modify the values in any columns
  - **REFERENCES (col-name)**: Can define foreign keys (in other tables) that refer to this column.
- The object can be a table
  - Or a view (discussed later)
- If a user has a privilege with the GRANT OPTION, they can themselves pass privilege on to other users (with or without passing on the GRANT OPTION).
- Only owner can execute CREATE, ALTER, and DROP.

Access Control in SQL - Examples

- Examples:

```
GRANT SELECT ON Enrolled TO jason
```

  - Jason is allowed to select (read) any tuples in the **`Enrolled`** table
  - Individual columns cannot be specified for SELECT access (SQL standard) – all columns of **`Enrolled`** can be read (including any added later via ALTER TABLE command).
  - SELECT access control to individual columns can be simulated through views (discussed later)

- `GRANT UPDATE(grade) ON Enrolled TO roehm`
  - Only the `grade` column can be updated by user 'roehm'

- `GRANT DELETE ON Students TO jon WITH GRANT OPTION`
  - Jon is allowed to delete tuples that are in Students table, and also Lon can authorize others to do so by executing the appropriate GRANT command himself.

Revoke of Privileges

```
REVOKE privilege_list
    ON table
  FROM user_list
```

- When a privilege is revoked from X, it is also revoked from all users who got it *solely* from X.
- But if a user has this privilege via several routes, it is still there until all granters have revoked it

Authorization Mode REFERENCES

- Foreign key constraint could be exploited to
  - *prevent some kinds of modification*: eg can prevent deletion of rows in some other table

```
CREATE TABLE DontDismissMe (
        id INTEGER,
        FOREIGN KEY (id) REFERENCES Student
                         ON DELETE NO ACTION )
```

  - *reveal information*: successful insertion into **DontDissmissMe** reveals that a row with particular value exists in **Student**
  - Example:

```
INSERT INTO DontDismissMe VALUES (11111111);
```

- Existence of REFERENCES privilege allows to limit these powers to appropriate users
  - Most uses will not have REFERENCES privilege and so cannot use these tricks

```
GRANT REFERENCES ON Student TO flexsis
```

Role-based Authorization

- In SQL-92, privileges are actually assigned to *authorisation ids*, which can denote a single user or a group of users.
- In SQL:1999 (and in many current systems), privileges are assigned to **roles**.
  - Roles can then be granted to users and to other roles.
  - Reflects how real organisations work.
  - Much more flexible and less error-prone, especially on large schemas

=> use role-based authorization whenever possible

- Example:

```
CREATE ROLE manager
GRANT select,insert ON students TO manager
GRANT manager TO shari

REVOKE manager FROM shari
```

Fine-grained access control

- The Grant statement works on a whole table
  - Or a column, for GRANT UPDATE(column)
- Policy often calls for access to be limited to certain rows only, in a table
- The decision about whether a row should be accessed by someone, may depend on the contents of the some fields in the row
  - Eg John should be able to modify salaries, but only for tuples with Dept = ‘Accounts’
  - Eg Each student should be able to see their own grades, but not grades of other students
- SQL DBMS allows one to achieve this by defining a VIEW with only the relevant data, and then GRANT access to that VIEW

Summary access

- Similarly, policy may call for certain users to be able to see aggregates and summaries, without seeing all the separate data items that combine for that summary
  - Eg Jane is allowed to know the level of average salary in each faculty, without knowing salaries of individual staff

Relational Views

- A **view** is a *virtual* relation, but we store a *definition*, rather than a set of tuples.
  - When the view is used within an SQL statement, the result is as if the contents of the view were computed at that time, from the current values in each of the tables mentioned in the view definition
- This provides a mechanism to release access to certain data, without allowing access to all of a genuine table
  - Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Users can operate on the view as if it were a stored table
    - DBMS calculates the value whenever it is needed
- Syntax:

```
CREATE VIEW name AS <query expression>
```

  - where
    - <*query expression*> is any legal query expression (can even combine multiple relations)

View Examples

- A view on the students showing their age.

```
CREATE VIEW ageStudents AS
      SELECT sid, name, extract(year from sysdate) -
                        extract(year from birthdate) AS age
        FROM Student
```

- A view on the female students enrolled in 2020sem2

```
CREATE  VIEW FemaleStudents (name, grade)
          AS SELECT S.name, E.grade
               FROM Student S, Enrolled E
              WHERE S.sid = E.sid  AND
                    S.gender = 'F' AND
                    E.semester = '2020sem2'
```

Updating Views

- Question: Since views look like tables to users, can they be updated?
- Answer: Yes – a view update changes the underlying base table to produce the requested change to the view

```
CREATE VIEW   CsReg (StudId, CrsCode, Semester) AS
SELECT        T.StudId, T. CrsCode, T.Semester
FROM          Transcript T
WHERE     T.CrsCode LIKE 'CS%'  AND  T.Semester='S2020'
```

Updating Views - Problem 1

INSERT INTO CsReg (*StudId, CrsCode, Semester*)
VALUES (1111, 'CSE305', 'S2020')

- **Question**: What value should be placed in attributes of underlying table that have been projected out (e.g., *Grade*)?
- **Answer**: NULL (assuming null allowed in the missing attribute) or DEFAULT

Updating Views - Problem 2

INSERT INTO CsReg (*StudId, CrsCode, Semester*)
VALUES (1111, 'ECO105', 'S2020')

- **Problem**: New tuple will not be visible in view
- **Solution**: Create View WITH CHECK OPTION ensures the that the new rows satisfy the view-defining condition.

Updating Views - Problem 3

- Update to a view: maybe there is not a unique way to change the base table(s) that results in the desired modification of the view Example:

```
CREATE VIEW   ProfDept (PrName, DeName)  AS
SELECT    P.Name, D.Name
FROM      Professor P, Department D
WHERE     P.DeptId = D.DeptId
```

- **Updatable Views:** SQL-92 allows updates only on simple views (defined by SELECT-FROM-WHERE on a single relation, without aggregates or distinct
  - SQL:1999 allows more, but many implementations don't support the extra cases

Updating Views - Problem 3 (cont'd)

- Eg Try to perform

  DELETE FROM ProfDept WHERE PrName='Smith' AND DeName='CS'

  - Updates on such a view are difficult or impossible to translate to actions on the base tables, and hence are disallowed.
- Tuple <Smith, CS> can be deleted from ProfDept by:
  - Deleting row for Smith from Professor (but this is inappropriate if he is still at the University)
  - Deleting row for CS from Department (not what is intended)
  - Updating row for Smith in Professor by setting *DeptId* to null (seems like a good idea, but how would the computer know?)

Updating Views – More problems

Suppose we have:

```
CREATE VIEW AvgSalary (DeptId, Avg_Sal )
AS
    SELECT  E.DeptId, AVG(E.Salary)
    FROM    Employee E
    GROUP BY E.DeptId
```

then how do we handle:

```
UPDATE AvgSalary
    SET Avg_Sal = 1.1 * Avg_Sal
```

Which tuples should we change, by how much? There are many ways to achieve the required overall increase

View Benefits: Security

- *Need-to-know*: Users not granted access to base tables. Instead they are granted access to the view of the database appropriate to their needs.
  - *External schema* is composed of views.
  - View allows owner to provide others with SELECT access to a subset of columns and a subset of rows

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
- Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
  - Granting a privilege on a view does not imply changing any privileges on the underlying relations.
  - If creator of base tables revokes SELECT right from view creator, view is no longer useful
- Together with GRANT/REVOKE commands, views are a very powerful access control tool.

Example: GRANTs and VIEWs

- User A: CREATE TABLE Student (sid INT, …);
  GRANT SELECT ON Student TO B WITH GRANT OPTION;
  ** note: without GRANT OPTION, B cannot pass SELECT privilege on its view on to C */*

- User B: CREATE VIEW LimitedInfoStud AS
  SELECT sid, name FROM A.Student;
  GRANT SELECT ON MyStud TO C;
- C is allowed to observe sid and name, but not other information such as address of students
- User C: SELECT * FROM B.LimitedInfoStud; -- works
  SELECT * FROM A.Student; -- does not work

- Question: If User A does REVOKE SELECT ON Student FROM B;
  -- what happens now?

Example: GRANTs and VIEWs

- User A: CREATE TABLE Student (sid INT, …);
  GRANT SELECT ON Student TO B WITH GRANT OPTION;
  ** note: without GRANT OPTION, B cannot pass SELECT privilege on its view on to C */*

- User B: CREATE VIEW CSStud AS
  SELECT * FROM A.Student
  WHERE Degree = 'Bachelor of CS';
  GRANT SELECT ON CSStud TO C;
- C is allowed to observe information about students in this degree, but not about other students
- User C: SELECT * FROM B.CSStud; -- works
  SELECT * FROM A.Student; -- does not work

Data Minimalism

- The best protection against unauthorized access to data in your database is to consider very carefully what you store in the first place!
- A database should only store information that is absolutely necessary for the operation of your application.
- Some data is even not allowed to be stored
  - For example: Sensitive authentication data such as the security code of a credit card
    - Cf. https://www.pcisecuritystandards.org/documents/pa-dss_v2.pdf
  - In Australia, the Tax File Number or the Medicare numbers is specifically protected from being used outside government
  - Any personal health information

Data Privacy

- Some information is specifically protected and requires specific standards and auditing procedures
  - especially for governmental organisations or large businesses
- In Australia, the Privacy Act 1988 (the Privacy Act) governs the protection rules regarding personal information
  - Personal information: information where an individual is reasonably identifiable, i.e. information that identifies/could identify an individual
  - regulates e.g. what and how to collect, disclosure rules, requirement to ensure information quality, when to delete
- Australian Privacy Principles (APP)
  - https://www.oaic.gov.au/agencies-and-organisations/app-guidelines
  - http://www.privacy.gov.au/
- Other jurisdictions can also be relevant when storing personal information about their residents

User identity (authentication)

- How does the system know which user's privilege to check, when a command is issued that requests access to data?
- Every request to dbms comes from some identified user
  - They connected to the dbms
  - Identity was determined then (eg they had to loigin with a password)
  - Every command they issue comes on a connection and carries the connection id
- In many systems, the dbms has its own set of user identities/passwords
  - Different from those of the operating system etc
- Some dbms may use Single Sign On, to avoid separate identity management

Powerful users

- A DBA has a lot of privilege, over objects of all the ordinary users
  - This is necessary for managing things, fixing errors etc
- So DBA identity must be carefully protected
  - Warning: never leave default password (as provided when system initialised) for dba account etc; see https://www.zdnet.com/article/pgminer-botnet-attacks-weakly-secured-postgresql-databases/

Whose privileges when code is run?

- Many databases are accessed indirectly
  - End-user does not write and submit SQL, but rather runs a program that (team of) coders have written to perform useful activity
  - Eg a student changes their address through MyUni
- The program can do lots of checking of whether access is appropriate, before sending SQL to dbms
  - Also the program can filter or summarise data, so user does not see everything the program gets from the dbms
- The program may run with its own appropriate level of privilege (or that or the coders), rather than from the end-user who is the source of request
  - Indeed, the end-user may not have a dbms account at all
- Often, the program has quite a lot of privilege, but this is risky if there are mistakes in the code

SQL Injection Attacks

- Web-applications often construct a SQL-statement from separate strings, which may include some provided by the end-user (eg name, address)
  - If a web-application does not thoroughly check the user's input, in general every database on every operating system is vulnerable.
- Do not assume that the end-user types a simple string; they may maliciously include text that affects the SQL, such as a semicolon and then another command
  - This allows user to provide SQL that will be run with web-app's privileges
- Humour: See https://xkcd.com/327/

SQL Injection Attacks

Code example is not examinable!

- Example: Consider the following SQL query in PHP

```
$result=$conn->query('SELECT * FROM users
                WHERE
username="'.$_REQUEST['username'].'"');
```

  - The query selects all rows from the users table where the username is equal to the one put in the query string.
  - Problem:
    quotes in $_REQUEST['username'] not escaped & the string not validated
  - Consider what would happen if we supply:

```
" OR 1 OR username = "
```

    (a double-quote, followed by a textual " OR 1 OR username = " followed by another double-quote)….
  - Also, another line of SQL code can be added by adding a quote and a semicolon to the end followed by another command

- Many more problems possible…