

Introduction to Database Systems

2023-Fall

5. The Security and Integrity Constraints

Introduction

- The destruction of database is generally caused by the following factors:
 1. System failure
 2. Inconsistency caused by concurrent access
 3. Man-caused destruction (intentionally or accidentally)
 4. The data inputted is incorrect, the updating transaction didn't obey the rule of consistency preservation
- In above factors, 1 and 2 should be resolved by recovery mechanism of DBMS (Chapter 4); 3 belongs to database security; 4 belongs to integrity constraints

Introduction to DB Security

- ***Secrecy***: Users should not be able to see things they are not supposed to.
 - E.g., A student can't see other students' grades.
- ***Integrity***: Users should not be able to modify things they are not supposed to.
 - E.g., Only instructors can assign grades.
- ***Availability***: Users should be able to see and modify things they are allowed to.

Security of Database

- Protect databases not be accessed illegally.
 - *View and query rewriting*
 - *Access control*
 - General user
 - User with resource privilege
 - DBA
 - *Identification and authentication of users*
 - Password
 - Special articles, such as key, IC card, etc.
 - Personal features, such as fingerprint, signature, etc.

Access Controls

- A security policy specifies who is authorized to do what.
- A security mechanism allows us to enforce a chosen security policy.
- Two main mechanisms at the DBMS level:
 - Discretionary access control
 - Mandatory access control

Discretionary Access Control

- Based on the concept of access rights or ***privileges*** for objects (tables and views), and mechanisms for giving users privileges (and revoking privileges).
- Creator of a table or a view 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 privileges **ON** object **TO** users [**WITH GRANT OPTION**]

- The following privileges can be specified:
 - **SELECT**: Can read all columns (including those added later via **ALTER TABLE** command).
 - **INSERT(col-name)**: Can insert tuples with non-null or non-default values in this column.
 - **INSERT** means same right with respect to all columns.
 - **DELETE**: Can delete tuples.
 - **REFERENCES (col-name)**: Can define foreign keys (in other tables) that refer to this column.
- If a user has a privilege with the **GRANT OPTION**, can pass privilege on to other users (with or without passing on the **GRANT OPTION**).
- Only owner can execute **CREATE**, **ALTER**, and **DROP**.

GRANT and REVOKE of Privileges

- GRANT INSERT, SELECT ON Sailors TO Horatio
 - Horatio can query Sailors or insert tuples into it.
- GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION
 - Yuppy can delete tuples, and also authorize others to do so.
- GRANT UPDATE (*rating*) ON Sailors TO Dustin
 - Dustin can update (only) the *rating* field of Sailors tuples.
- GRANT SELECT ON ActiveSailors TO Guppy, Yuppy
 - This does NOT allow the 'uppies to query Sailors directly!
- **REVOKE**: When a privilege is revoked from X, it is also revoked from all users who got it *solely* from X.

GRANT/REVOKE on Views

- If the creator of a view loses the SELECT privilege on an underlying table, the view is dropped!
- If the creator of a view loses a privilege held with the grant option on an underlying table, (s)he loses the privilege on the view as well; so do users who were granted that privilege on the view!

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
 - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the *bid's* of boats that have been reserved.
- Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
- Together with GRANT/REVOKE commands, views are a very powerful access control tool.

Role-Based Authorization

- In SQL-92, privileges are actually assigned to authorization 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 organizations work.
 - Illustrates how standards often catch up with “de facto” standards embodied in popular systems.

Security to the Level of a Field!

- Can create a view that only returns one field of one tuple. (How?)
- Then grant access to that view accordingly.
- Allows for *arbitrary* granularity of control, *but*:
 - Clumsy to specify, though this can be hidden under a good UI
 - Performance is unacceptable if we need to define field-granularity access frequently. (Too many view creations and look-ups.)

Internet-Oriented Security

- **Key Issues: User authentication and trust.**
 - When DB must be accessed from a secure location, password-based schemes are usually adequate.
- For access over an external network, trust is hard to achieve.
 - If someone with Sam's credit card wants to buy from you, how can you be sure it is not someone who stole his card?
 - How can Sam be sure that the screen for entering his credit card information is indeed yours, and not some rogue site spoofing you (to steal such information)? How can he be sure that sensitive information is not "sniffed" while it is being sent over the network to you?
- *Encryption* is a technique used to address these issues.

Encryption

- “Masks” data for secure transmission or storage
 - $\text{Encrypt}(\text{data}, \text{encryption key}) = \text{encrypted data}$
 - $\text{Decrypt}(\text{encrypted data}, \text{decryption key}) = \text{original data}$
 - Without decryption key, the encrypted data is meaningless gibberish
- Symmetric Encryption:
 - Encryption key = decryption key; all authorized users know decryption key (a weakness).
 - DES, used since 1977, has 56-bit key; AES has 128-bit (optionally, 192-bit or 256-bit) key
- Public-Key Encryption: Each user has two keys:
 - User’s public encryption key: Known to all
 - Decryption key: Known only to this user
 - Used in RSA scheme (Turing Award!)

Authenticating Users

- Amazon can simply use password authentication, i.e., ask Sam to log into his Amazon account.
 - Done after SSL is used to establish a session key, so that the transmission of the password is secure!
 - Amazon is still at risk if Sam's card is stolen and his password is hacked. Business risk ...
- Digital Signatures:
 - Sam encrypts the order using his *private* key, then encrypts the result using Amazon's public key.
 - Amazon decrypts the msg with their private key, and then decrypts the result using Sam's public key, which yields the original order!
 - Exploits interchangeability of public/private keys for encryption/decryption
 - Now, no one can forge Sam's order, and Sam cannot claim that someone else forged the order.

Mandatory Access Control

- Based on system-wide policies that cannot be changed by individual users.
 - Each DB object is assigned a security class.
 - Each subject (user or user program) is assigned a clearance for a security class.
 - Rules based on security classes and clearances govern who can read/write which objects.
- Most commercial systems do not support mandatory access control. Versions of some DBMSs do support it; used for specialized (e.g., military) applications.

Why Mandatory Control?

- Discretionary control has some flaws, e.g., the *Trojan horse* problem:
- Dick creates Horsie and gives INSERT privileges to Justin (who doesn't know about this).
- Dick modifies the code of an application program used by Justin to additionally write some secret data to table Horsie.
- Now, Justin can see the secret info.
The modification of the code is beyond the DBMSs control, but it can try and prevent the use of the database as a channel for secret information.

Security of Database cont.

- ***Protect databases not be accessed illegally.***
 - Authorization
 - GRANT CONNECT TO JOHN IDENTIFIED BY xyzabc;
 - GRANT SELECT ON TABLE S TO U1 WITH GRANT OPTION;
 - Role
 - Data encryption
 - ***Audit trail***
 - AUDIT SELECT, INSERT, DELETE, UPDATE ON emp WHENEVER SUCCESSFUL;

Security of Statistical Database

- In many situation, the statistical data is public while the detailed individual data is secret.
- Public *statistical database*
- But some detailed individual data can be derived from public statistical data
 - How prevent this leak? --- not a easy thing

Security of Statistical Database

STATS

NAME	SEX	DEPENDENTS	OCCUPATION	SALARY
Wang	M	2	Programmer	120
Chang	F	2	Manager	240
Chen	F	0	Programmer	140
Li	F	2	Engineer	160
Liu	M	2	Clerk	110
Zhu	F	1	Teacher	80
Zhao	M	0	Professor	180
Sun	M	1	Teacher	110
Xu	F	2	Programmer	130
Ma	F	1	Programmer	150

Individual Tracker

- Suppose we know Wang is a male programmer, and salary in STATS is secret but other information is public, we can get wang's salary from public data.

- Q1: SELECT COUNT(*)
FROM STATS
WHERE SEX='M' AND OCCUPATION='programmer'

result = 1

- Q2: SELECT SUM(SALARY)
FROM STATS
WHERE SEX='M' AND OCCUPATION='programmer';

result = 120

Individual Tracker (c>b, b=2)

- Q3: SELECT COUNT(*)
FROM STATS;

result = 10

Q4: SELECT COUNT(*)
FROM STATS

WHERE NOT(SEX='M' AND OCCUPATION='programmer');

result = 9

Now we know only one male programmer, that must be Wang.

Individual Tracker (c>b, b=2)

- Q5: SELECT SUM(SALARY)
FROM STATS;

result = 1420

Q4: SELECT SUM(SALARY)
FROM STATS

WHERE NOT(SEX='M' AND OCCUPATION='programmer');

result = 1300

Wang's salary = Q5 - Q6 = 120

Individual Tracker ($b < c < n - b$, $b = 2$, n is 10)

- Q7: SELECT COUNT(*)
FROM STATS
WHERE SEX = 'M';

result = 4

Q8: SELECT COUNT(*)
FROM STATS
WHERE SEX='M' AND NOT(OCCUPATION='programmer');

result = 3

Now we know only one male programmer, that must be Wang.

Individual Tracker ($b < c < n - b$, $b = 2$, n is 10)

- Q9: SELECT SUM(SALARY)
FROM STATS
WHERE SEX = 'M';

result = 520

Q10: SELECT SUM(SALARY)
FROM STATS
WHERE SEX='M' AND NOT(OCCUPATION='programmer');

result = 400

Wang's salary = Q9 – Q10 = 120

General Tracker

- ***Individual tracker***

- Suppose predicate $p=p_1$ and p_2 , $SET(p)$ is set of tuples which fulfill p , then
 - $SET(p) = SET(p_1 \text{ and } p_2) = SET(p_1) - SET(p_1 \text{ and not } p_2)$

- ***General tracker***

- It is a predicate T which fulfill:
 - $2b \leq |SET(T)| \leq (n-2b), b < n/4$
- Suppose a tuple R can be limited uniquely by predicate p , that is $SET(p) = \{R\}$, then
 - $SET(p) = SET(p \text{ or } T) \bar{\cup} SET(p \text{ or not } T) - SET(T) - SET(\text{not } T)$
- $\bar{\cup}$ means ***union all*** (without eliminating repeated tuples).

Integrity Constraints

- An IC describes conditions that every *legal instance* of a relation must satisfy.
 - Inserts/deletes/updates that violate IC's are disallowed.
 - Can be used to ensure application semantics (e.g., *sid* is a key), or prevent inconsistencies (e.g., *sname* has to be a string, *age* must be < 200)

Types of Integrity Constraints

- ***Static constraints***: constraints to database state
 - Inherent constraints (data model), such as **1NF**
 - Implicit constraints : implied in data schema, indicated by DDL generally. Such as domain constraints, primary key constraints, foreign key constraints.
 - ***Domain constraints***: Field values must be of right type. Always enforced.
 - Explicit constraints or general constraints
- ***Dynamic constraints***: constraints while database transferring from one state to another. Can be combined with trigger.

An Example of Foreign Keys

- E.g. *sid* is a foreign key referring to **Students**:
 - Students(*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: decimals)
 - Enrolled(*sid*: string, *cid*: string, *grade*: string)

Enrolled

<i>sid</i>	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

<i>sid</i>	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Database Modification

- If α is foreign key in r_2 which references to K_1 in r_1 , the following tests must be made in order to preserve the following referential integrity constraint:

$$\Pi_{\alpha}(r_2) \subseteq \Pi_{K_1}(r_1)$$

- **Insert.** If a tuple t_2 is inserted into r_2 , the system must ensure that there is a tuple t_1 in r_1 such that $t_1[K_1] = t_2[\alpha]$. That is

$$t_2[\alpha] \in \Pi_{K_1}(r_1)$$

- **Delete.** If a tuple, t_1 is deleted from r_1 , the system must compute the set of tuples in r_2 that reference t_1 :

$$\sigma_{\alpha = t_1[K_1]}(r_2)$$

If this set is not empty, either the delete command is *rejected as an error*, or the tuples that reference t_1 must themselves be deleted (*cascading deletions* are possible).

Database Modification (Cont.)

- **Update.** There are two cases:
 - If a tuple t_2 is updated in relation r_2 and the update modifies values for foreign key α , then a test similar to the insert case is made. Let t_2' denote the new value of tuple t_2 . The system must ensure that

$$t_2'[\alpha] \in \Pi_{K_1}(r_1)$$

- If a tuple t_1 is updated in r_1 , and the update modifies values for the primary key (K_1), then a test similar to the delete case is made. The system must compute

$$\sigma_{\alpha = t_1[K_1]}(r_2)$$

using the old value of t_1 (the value before the update is applied). If this set is not empty, the update may be **rejected as an error**, or the **update may be cascaded** to the tuples in the set, or the tuples in the set may be **deleted**.

Types of Integrity Constraints

- Static constraints: constraints to database state
 - Inherent constraints (data model), such as 1NF
 - Implicit constraints : implied in data schema, indicated by DDL generally. Such as domain constraints, primary key constraints, foreign key constraints.
 - *Domain constraints*: Field values must be of right type. Always enforced.
 - ***Explicit constraints or general constraints***
- Dynamic constraints: constraints while database transferring from one state to another. Can be combined with trigger.

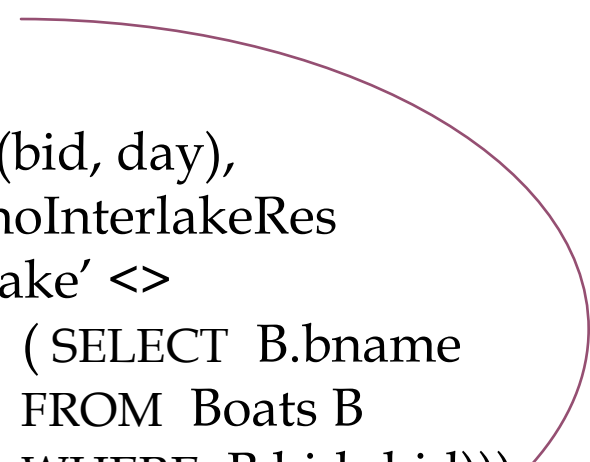
Definition of Integrity Constraints

- *Indicated with **procedure***
 - Let application programs responsible for the checking of integrity constraints.
- *Indicated with **ASSERTION***
 - Defined with *assertion specification language*, and checked by DBMS automatically
 - ASSERT balanceCons ON account: balance>=0;
- Indicated with **CHECK** clause in base table definition, and checked by DBMS automatically

General Constraints

- Useful when more general ICs than keys are involved.
 - Can use queries to express constraint.
 - Constraints can be named.

```
CREATE TABLE Reserves
( sname CHAR(10),
  bid INTEGER,
  day DATE,
  PRIMARY KEY (bid, day),
  CONSTRAINT noInterlakeRes
  CHECK ('Interlake' <>
        ( SELECT B.bname
          FROM Boats B
          WHERE B.bid=bid)))
```



```
CREATE TABLE Sailors
( sid INTEGER,
  sname CHAR(10),
  rating INTEGER,
  age REAL,
  PRIMARY KEY (sid),
  CHECK ( rating >= 1
        AND rating <= 10))
```

General Constraints

- Use alter table clause to **add** CHECK constraints:
 alter table [Users]
 add constraint [CK_Users_Balance] check (Balance > 0)
- Use alter table clause to **drop** CHECK constraints:
 alter table Teachers
 drop constraint ck_age

Constraints Over Multiple Relations

```
CREATE TABLE Sailors
( sid INTEGER,
  sname CHAR(10),
  rating INTEGER,
  age REAL,
  PRIMARY KEY (sid),
```

CHECK

```
( (SELECT COUNT (S.sid) FROM Sailors S)
  +(SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

*Number of boats
plus number of
sailors is < 100*

- Awkward and wrong!
- If Sailors is empty, the number of Boats tuples can be anything!

Assertion

- **ASSERTION** is the right solution; not associated with either table.

```
CREATE ASSERTION smallClub  
CHECK  
( (SELECT COUNT (S.sid) FROM Sailors S)  
+(SELECT COUNT (B.bid) FROM Boats B) < 100 )
```

Different Constraint Types

Type	Where Declared	When activated	Guaranteed to hold?
Attribute CHECK	with attribute	on insertion or update	not if contains subquery
Tuple CHECK	relation schema	insertion or update to relation	not if contains subquery
Assertion	database schema	on change to any relation mentioned	Yes

Triggers

- Trigger: procedure that starts automatically if specified changes occur to the DBMS
- Three parts:
 - **E**vent (activates the trigger)
 - **C**ondition (tests whether the triggers should run)
 - **A**ction (what happens if the trigger runs)

Triggers: Example (SQL:1999)

```
CREATE TRIGGER youngSailorUpdate
  AFTER INSERT ON SAILORS
  REFERENCING NEW TABLE NewSailors
  FOR EACH STATEMENT
  INSERT
    INTO YoungSailors(sid, name, age, rating)
    SELECT sid, name, age, rating
    FROM NewSailors N
    WHERE N.age <= 18
```