Database Management Systems

Fall 2001

CMPUT 391: Database Recovery and Security

Dr. Osmar R. Zaïane

University of Alberta

Chapters 17 and 20 of Textbook

Course Content

- Introduction
- Database Design Theory
- Query Processing and Optimisation
- Concurrency Control
- **Data Base Recovery and Security**
- Object-Oriented Databases
- Inverted Index for IR
- XML
- Data Warehousing
- Data Mining
- Parallel and Distributed Databases
- Other Advanced Database Topics

Objectives of Lecture 5

Database recovery and security

- Review some causes of database failure and introduce the procedures to recover after such failures.
- Discuss the purpose of transaction log file and checkpointing during transaction logging.
- Review the kind of threats that can affect a database and discuss the scope of database security.

Database Recovery and Security

- Motivation and Assumptions
- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical Database Security

Recovery and the ACID properties

Atomicity: All actions in the transaction happen, or none happen.

Consistency: If each transaction is consistent, and the DB starts consistent, it ends up consistent.

Isolation: Execution of one transaction is isolated from that of other transactions.

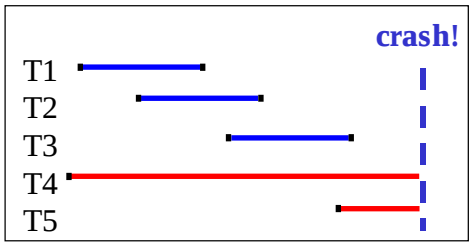
Durability: If a transaction commits, its effects persist.

- The **Recovery Manager** is responsible for ensuring two important properties of transactions: ***Atomicity*** and ***Durability***.
- Atomicity is guaranteed by undoing the actions of the transactions that did not commit.
- Durability is guaranteed by making sure that all actions of committed transactions survive crashes and failures.

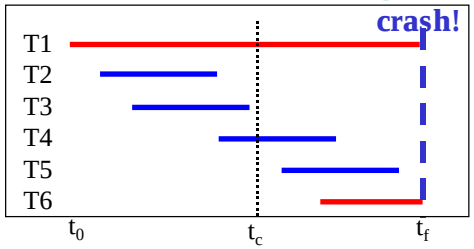
Motivation

- Atomicity:
  - Transactions may abort ("Rollback").
- Durability:
  - What if DBMS stops running? (Causes?)

❖ Example: Desired Behavior after system restarts:
- T1, T2 & T3 should be durable.
- T4 & T5 should be aborted (effects not seen).



Another Motivating Example



- Clearly T1 and T6 are not committed at time of crash➔**undo** T1 and T6 at restart.
- T2 and T3 made it to secondary storage before failure.
- What about T4 and T5? They committed, but did they make it from the volatile database buffer to the non-volatile storage?
- Since we do not know, Recovery manager may have to **redo** T2, T3, T4, and T5.

Assumptions

- Concurrency control is in effect.
  - Strict 2PL, in particular.
- Updates are happening "in place".
  - i.e. data is overwritten on (or deleted from) the disk.

- Is there a simple scheme to guarantee Atomicity & Durability?
- Remember the transaction log and the Aries Algorithm we introduced in Lecture 4?

Database Recovery and Security

- Motivation and Assumptions
- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical Database Security

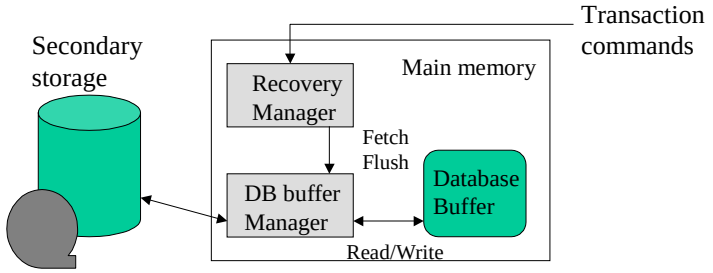
Types of Failures

- **System crashes**: due to hardware or software errors, resulting in loss of main memory
- **Media failures**: due to problems with disk head or unreadable media, resulting in loss of parts of secondary storage.
- **Application errors**: due to logical errors in the program which may cause transactions to fail.
- **Natural disasters**: physical loss of media (fire, flood, earthquakes, terrorism, etc.)
- **Sabotage**: intentional corruption or destruction of data
- **Carelessness**: unintentional destruction of data by user or operator.

General Failures

- Transaction failures
  - Transaction aborts (unilaterally or due to deadlock)
  - 3% of transaction abort abnormally on average
- System failures
  - Failure of processor, main memory, power supply…
  - Main memory contents are lost but nor secondary storage
- Media failures
  - Failure of secondary storage
  - Head crash or controller failure

Recovery Manager



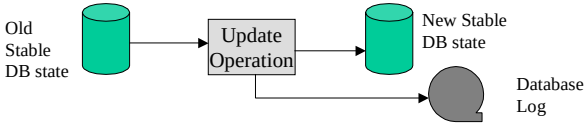
- Volatile storage: main memory (DB buffer)
- Stable storage: disks and other media. Resilient to failure and loses its content only in the presence of media failure or intentional attacks (Database and logs)

Database Recovery and Security

- Motivation and Assumptions
- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical Database Security

Action Logging

- Every update action of a transaction must also write a log entry to an append-only file.
- No record needs to be appended in the log if the action merely reads the database
- Why do we need a log? The log is consulted by the system to achieve both atomicity and durability.
- The log is generally stored on a different mass storage device than the database.



Database Log

- The log contains records (or entries) that are appended.
- For recovery, the log is read backwards
- An entry in the log contains:
  - A transaction identifier
  - Type of operation
  - Objects accessed to perform action
  - Old value of object (before image)
  - New value of object (after image)
  - …
- Log entries (also called update records) are used to undo changes in case of aborts: using transacion ID replace object value in database with before image

Other Entries in the Log

- Log may also contain action such as *begin-transaction*, *commit-transaction*, *abort-transaction*. (Why?)
- If a transaction T is aborted, then rollback ➔ scan log backwards and when T's update records are encountered, the before image is written to DB undoing the change. However, How long do we have to scan backwards looking for T's entries in the log?
- We thus need the ***begin-transaction*** entry indicating where to stop scanning for a transaction actions.
- In a rollback due to a crash, the system needs to distinguish between the transactions that completed and the still active transactions➔ commit and abort entries.

Problems With Log

- Does the commit request guarantee durability?
- If a crash occurs after the transaction has made changes and requested the commit, but before the commit record is written in log, the transaction will be aborted by the recovery procedure and the changes are not durable.
- What are the different scenarios?

Making Changes in DB

- Transaction T made changes to x. There is use of DB buffer. There is a crash while operations are performed.
- **Scenario1**: Neither operations made it to secondary storage➔No problem. T is aborted.
- **Scenario2**: x is updated on disk but crash occurred before log is updated➔No way to rollback since we don't have before image➔inconsistent state.
- **Scenario3**: The update made it to the log but the changes to x on disk didn't make it➔T aborted and before image is used to overwrite old value➔no problem.
- What do we learn from these scenarios?

Write-Ahead Log

- The update record must always be appended to the log before the database is updated. The log is referred to as a ***write-ahead log***.
- The Write-Ahead Logging Protocol:
  - ① Must force the log record for an update *before* the corresponding data page gets to disk.
  - ② Must write all log records for a transaction *before commit*.
    - #1 guarantees Atomicity.
    - #2 guarantees Durability.
- Exactly how is logging (and recovery!) done? We'll study the ARIES algorithms.

Checkpointing

- In case of a crash, the recovery procedure needs to identify transaction that are still active in order to abort them. This is done by scanning the log backwards.
- How far do we have to scan the log before stopping and guaranteeing that there is not transaction still active?
- To avoid a complete backward scan of the log during recovery, we must include a mechanism to specify where to stop.
- New entry in the log called ***Checkpoint.***

Chechpoints

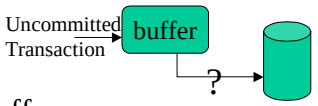
- The system periodically appends a checkpoint record to the log that lists the current active transactions.
- The recovery process must scan backward at least as far as the most recent checkpoint.
- If T is named in the checkpoint, then T was still active during crash➔ continue scan backward until *begin-transaction* T.

Database Recovery and Security

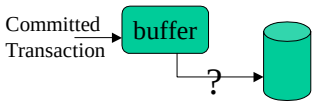
- Motivation and Assumptions
- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical Database Security

Recovery Manager and Buffer Manager Interaction

- Can a Buffer Manager decide to write some of the buffer pages being accessed by a transaction into stable storage or does it wait for recovery Manager to instruct it?
  - **Steal** / **no-steal** decision
  - No-steal means RM pins pages in buffer



- Does the RM force the BM to write certain buffer pages in stable database at the end of a transaction's execution?
  - **Force** / **no-force** decision



Possible Execution Strategies

- **Steal / No-force**

BM may have written some of the updated pages into disk. RM writes a commit

- Steal / force

BM may have written some of the updated pages into disk. RM issues a *flush* and writes a commit

- No-steal / no-force

None of the updated pages have been written. RM writes a commit and sends unpin to BM for all pinned pages.

- No-steal / force

None of the updated pages have been written. RM issues a *flush* and writes a commit

- Force every write to disk?
  - Poor response time.
  - But provides durability.
- Steal buffer-pool frames from uncommitted transaction?
  - If not, poor throughput.
  - If so, how can we ensure atomicity?

Recovering From a Crash

- There are 3 phases in the *Aries* recovery algorithm:
  - *Analysis*: Scan the log forward (from the most recent *checkpoint*) to identify all transactions that were active, and all dirty pages in the buffer pool at the time of the crash.
  - *Redo*: Redoes all updates to dirty pages in the buffer pool, as needed, to ensure that all logged updates are in fact carried out and written to disk.
  - *Undo*: The writes of all transactions that were active at the crash are undone (by restoring the *before value* of the update, which is in the log record for the update), working backwards in the log. (Some care must be taken to handle the case of a crash occurring during the recovery process!)

Database Recovery and Security

- Motivation and Assumptions
- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical Database Security

Introduction to DB Security

Preventing unauthorized access to databases

- 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.

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

Database Recovery and Security

- Motivation and Assumptions
- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical Database Security

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 but just the view!
- 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, he or she 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.

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.

Typical Security Classes

- Objects (e.g., tables, views, tuples)
- Subjects (e.g., users, user programs)
- Security classes:
  - Top secret (TS), secret (S), confidential (C), unclassified (U): TS > S> C > U
- Each object and subject is assigned a class.
  - Subject S can ***read*** object O only if class(S) >= class(O) (no reads in higher security)
  - Subject S can ***write*** object O only if class(S) <= class(O) (no writes in lower security)

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 modifes 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.

Does it Work?

- Idea is to ensure that information can never flow from a higher to a lower security level.
- E.g., If Dick has security class C, Justin has class S, and the secret table has class S:
  - Dick's table, Horsie, has Dick's clearance, C.
  - Justin's application has his clearance, S.
  - So, the program cannot write into table Horsie.
- The mandatory access control rules are applied in addition to any discretionary controls that are in effect.

Multilevel Relations

bid	bname	color	class
101	Salsa	Red	S
102	Pinto	Brown	C

- Users with S and TS clearance will see both rows; a user with C will only see the 2nd row; a user with U will see no rows.
- If user with C tries to insert <101,Pasta,Blue,C>:
  - Allowing insertion violates key constraint
  - Disallowing insertion tells user that there is another object with key 101 that has a class > C!
  - Problem resolved by treating class field as part of key.

Database Recovery and Security

- Motivation and Assumptions
- Types of Failures and Recovery Manager
- Transaction Logging and Checkpointing
- Recovery Strategies
- Introduction to Database Security
- Discretionary and Mandatory Access Control
- Statistical Database Security

Statistical Database Security

- **Statistical databases** are used to produce statistics on various populations.
  - individual information is considered confidential.
  - users may allow to access statistical information on the population, (i. e., applying statistic functions to a population of tuples).
- **Techniques** for protecting privacy of individual information in statistical DBs are beyond the scope of this course. Problems and possible solutions are illustrated by examples:
- Person(name, ssn, income, address, city, sex, last_ degree)
- Suppose we are allowed to retrieve only the statistical information over this relation by using **SUM, AVG, MIN, MAX, COUNT**,etc. (i.e. allow only aggregate queries. e.g., average age, rather than Joe's age).

Example 1

The following two queries are valid:

Q1: *find the average income of women who have Ph. D.. and live in Calgary, Alberta.*

```
select AVG( income)
from Person
where last_ degree = "Ph. D.."
and sex = "F"
and city = "Calgary"
```

Q2: *find the total number of women who have Ph. D.. and live in Calgary, Alberta.*

```
select COUNT(*)
from Person
where last_ degree = "ph. D."
and sex = "F"
and city = "Calgary"
```

If we know Mary Black has Ph. D. and lives in Calgary and we want to know her income, we may use the above two queries.

• When query Q1 returns one, the result of Q2 is the income of Mary.
• Otherwise we may issue a number of subsequent queries using MAX and MIN, we may easily know the close range of Mary's income.

Example 2

Using the following two queries, one can easily get the income of Mary.

```
select SUM( income)
from Person
where last_ degree = "Ph. D.."
and sex = "F"
and city = "Calgary"
and name <> "Mary"
```

```
select SUM( income)
from Person
where last_ degree = "Ph. D.."
and sex = "F"
and city = "Calgary"
```

The first query returns the sum of all incomes but Mary's, while the second query returns the sum of all incomes. The difference is Mary's income.

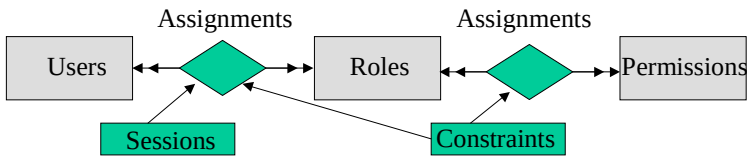
Restrictions

A number of restrictions can be used to reduce the possibility of deducing individual information from statistical queries:

- No statistical queries are permitted whenever the number of tuples in the population specified by the selection condition falls below some threshold.
- Restricting the number of tuples in the intersection of subsequent query results.
- Another technique is to prohibit sequences of queries that refer repeatedly to the same population of tuples.

Role-Based Access Control

- There are other access control mechanisms that complement the discretionary and mandatory mechanisms.
- Users are given roles and roles are assigned permissions. Objects have access permissions with regard to some roles.



Summary

- Recovery Manager guarantees Atomicity & Durability.
- Use Write-Ahead Logging to allow STEAL/NO-FORCE without sacrificing correctness.
- Checkpointing: A quick way to limit the amount of log to scan on recovery.
- Recovery works in 3 phases:
  - Analysis: Scan the log forward to identify all active transactions, and all dirty pages.
  - Redo: all updates to dirty pages in the buffer pool.
  - Undo: The writes of all transactions that were active at the crash.

Summary Cont.

- Three main security objectives: secrecy, integrity, availability.
- DB admin is responsible for overall security.
  - Designs security policy, maintains an audit trail, or history of users' accesses to DB.
- Two main approaches to DBMS security: discretionary and mandatory access control.
  - Discretionary control based on notion of privileges.
  - Mandatory control based on notion of security classes.
- Statistical DBs try to protect individual data by supporting only aggregate queries, but often, individual information can be inferred.