Introduction to Databases

Prof. Joseph G. Vella
Dept. of Computer Information Systems
JOSEPH.G.VELLA@UM.EDU.MT

1

Concurrency & Transaction Processing

Rationale & Challenges

2

Where had we started …

- **Definition:**
  - A **database** is a collection of structured data stored on persistent storage.
- **Environment and Contextual**
  - **Available** to a large number of end users.
    - **As in "Access my data when I need it …"**
      - **No delay, no access denial, no loss of data**
  - **Correctness**.
    - **As in "What's my balance?" gets the same value until it is changed.**
- **What's different from, say, programming?**
  - Data **persistence**;
  - Data **sharing**;
  - Data **independence**.

3

How is usage managed and controlled?

- **A Database Management System (DBMS)**
  - Is a computer-based application tool set for: defining, creating, manipulating, controlling, and managing databases.
- **DBMS Requirements:**
  End users, developers, administrators demand efficiency in terms of allocation of computational resources and storage space.
  - **Reliability**
  - **Openness** (e.g., in term of data connectivity)
  - **Scalability** (e.g., storage and throughput)

4

DBMS Functionality

- **DBMS also facilitate and insulates data access by system's end users (including "computer" naive *end users*).**
- **Three important functionalities include:**
  - ***Query Processing***
  - ***Transaction Management***
  - ***Storage management***
- **Two main areas of concern:**
  - ***Consistency*** (e.g. returns the same results under invariant states), and
  - ***Efficiency*** (e.g. computational time and space) of DBMS activities.
- **Some advantages:**
  - Execute a query in an efficient and opportunistic approach – **query optimisation;**
  - Iron out sharing inconsistencies between conflicting queries and transactions – **concurrency control;**
  - Undoing the effects of incomplete transactions under a system's instructions or in rectifying a system failure – **recovery.**
    - **Note: system here is most general: e.g., OS, DBMS, hard disk**

5

Transaction Processing

6

Transactions & Transaction Processing

- A *transaction* is a sequence of actions over a database that realise a logical operation.
  - For example, a double entry accounting package must execute the debit and the credit entries for the database to remain consistent;
    - In this case the transaction is made up of two SQL insert statements.
- The part of a DBMS that applies transaction to its database state is called the *Transaction Processing* module.
- Basic actions over a database are:
  - *read(X)* -- read item X from database and store into local var X; and
  - *write(X)* – write item X to database from local var X.
- Concurrency control is that part of the DBMS that *assures* that simultaneously executed transactions produce the same results as if they were executed in serially.
- The concurrency control mechanism must constantly seek to *balance* a high degree of concurrency against a satisfactory performance within a consistent state.

Joseph G Vella - Introduction to databases 7 Concurrency & Recovery

7

TP Principles - ACID

- Transaction's ACID principles for *short-lived* transactions are (over and above valid transactions):
  - Atomicity
    - The transaction executes completely or not at all,
  - Consistency
    - The transaction preserves the internal consistency of the database.
  - Isolation
    - The transaction executes as if it were running alone, with no other transactions.
  - Durability
    - The transaction's result will not be lost in the future.

Joseph G Vella - Introduction to databases 8 Concurrency & Recovery

8

Transaction Modeling

- **An explicit sequence of data manipulation commands over a database are abstracted as a single logical update (i.e a transaction).**
  - A transaction is **valid** if ***all*** its sub-parts (e.g. updates, deletes) have succeeded. A valid transaction can **commit,** otherwise we have to **rollback** the transaction.
  - It is possible to have ***read-only transactions*** apart from read-write transactions.
- **A number of basic TP management operations are required (to supplement READ & WRITE basic ops):**
  - BEGIN, END, Request a COMMIT, ROLLBACK;
  - UNDO and REDO specific log entries (e.g. READ & WRITE based).
- **Operationally, this complicates our scenario when allowing for data sharing! The DBMS typically insulates this from users (programmers and end users alike!).**

A Database Transaction : an Example

- **"Ensure that the employee's salary is upped by 25% and each salesperson commission is augmented by Euro 250."**
- **A generic SQL command sequence :**

```
START transaction;
UPDATE emp
      SET sal = sal * 1.25;
Remark IMPLICIT mechanism - ON error ROLLBACK
UPDATE emp
      SET comm = comm + 250 WHERE comm IS NOT NULL;
Remark IMPLICIT mechanism - ON error ROLLBACK
COMMIT;
END transaction;
```

TP & a sequence of SQL DML statements

- Here is a sequence of generic SQL statements that are entered consecutively. We will assume that a START TRANSACTION statement at the beginning of each transaction is implicit.

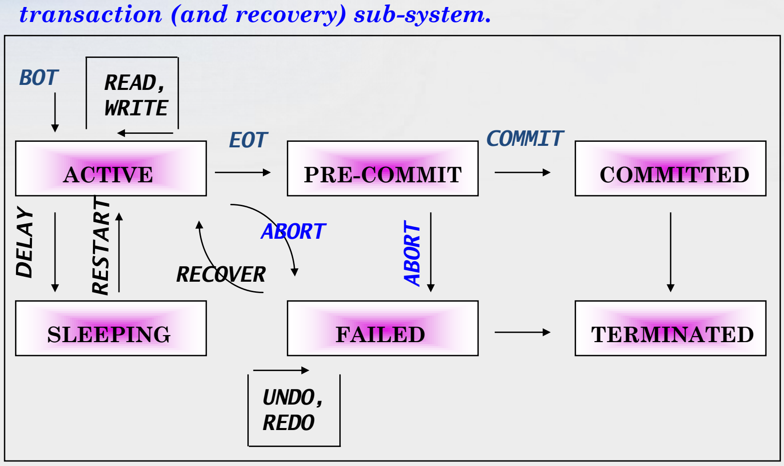
%	SQL Construct	Explanation
1	`INSERT ...`	Ineffective as regards persistence.
2	`DELETE ...`	*ditto*
3	`SAVEPOINT sp1`	Define savepoint sp1.
4	`INSERT ...`	Ineffective as regards persistence.
5	`SAVEPOINT sp2`	Define savepoint sp2.
6	`DELETE ...`	Ineffective as regards persistence.
7	`ROLLBACK WORK TO SAVEPOINT SP2`	All DML since *sp2* are un-done (i.e. 6).
8	`UPDATE ...`	Ineffective as regards persistence.
9	`ROLLBACK WORK TO SAVEPOINT SP1`	All DML since *sp1* are un-done (i.e. 4 & 8)
10	`DELETE ...`	Ineffective as regards persistence.
11	`UPDATE ...`	Ineffective as regards persistence.
12	`COMMIT WORK`	All DML since the start become persistent (i.e. 1,2,10 & 11).
13	`INSERT ...`	Ineffective as regards persistence.
14	`quit`	All pending DML since the start are un-done (i.e. 13).

Joseph G Vella - Introduction to databases 11 Concurrency & Recovery

11

TP State Transition Model

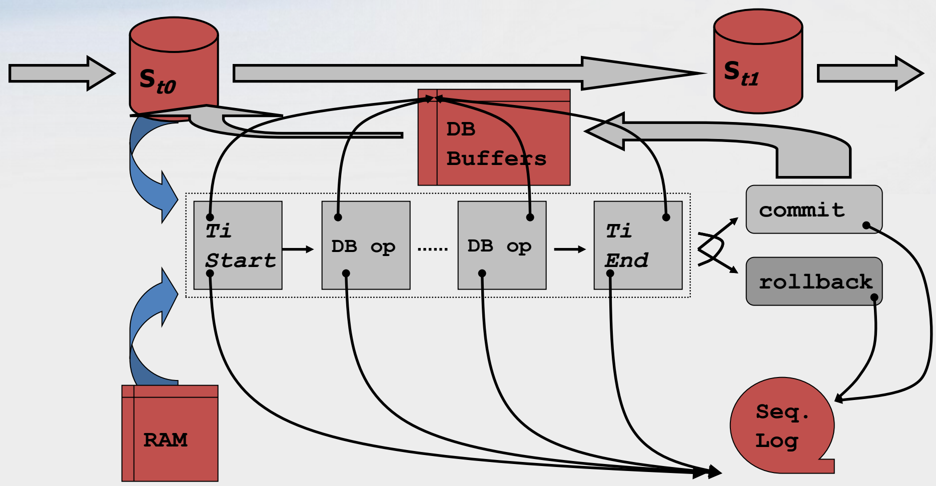
A state transition diagram for a transaction (and recovery) sub-system.



Joseph G Vella - Introduction to databases 12 Concurrency & Recovery

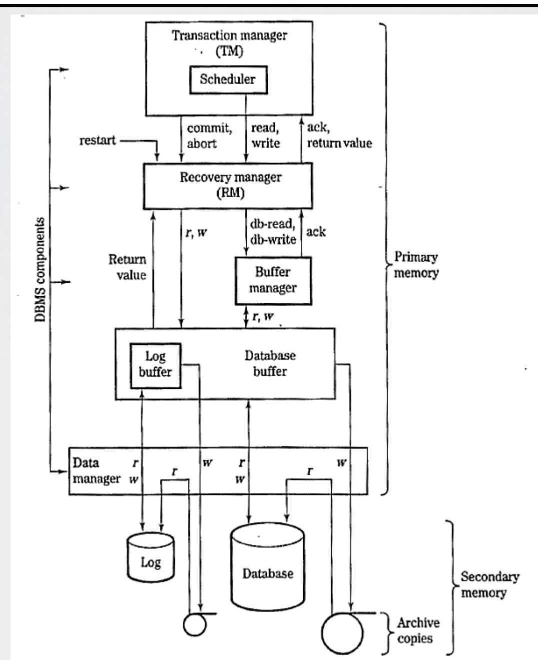
12

Transaction Modeling and Database States



Joseph G Vella - Introduction to databases 13 Concurrency & Recovery The end!

13



Organization of the primary/secondary memory interface around a recovery manager.

Joseph G Vella - Introduction to databases

14

SQL Transaction Management

- SQL provides constructs for the management of transactions:
  - Starting a transaction
    - `START TRANSACTION #in some dialects!`
  - Set a transaction sharing level (per session or transaction)
    - ```
      SET TRANSACTION ISOLATION LEVEL
      { READ {UNCOMMITTED | COMMITTED } |
        REPEATABLE READ |
        SNAPSHOT |
        SERIALIZABLE };
      ```
  - Committing a transaction (i.e. Requesting a Commit)
    - `COMMIT`
    - To make permanent the changes made by statements issued and the beginning of a transaction.
  - Save points
    - `SAVEPOINT sp4`
    - To establish a point back to which you may roll.
  - Rolling back transactions
    - `ROLLBACK TO SAVEPOINT sp4`
    - To undo all changes since the beginning of a transaction or some savepoint.

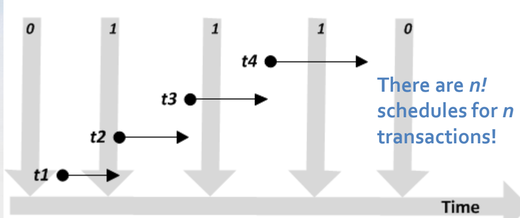
Joseph G Vella - Introduction to databases 15 Concurrency & Recovery

15

Serial *vs.* Interleaving Execution

Serial

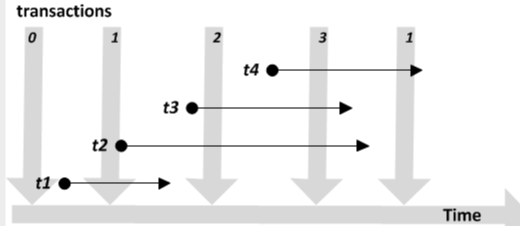
Number of concurrent transactions is 1 at most



There are *n!* schedules for *n* transactions!

Interleaving

Number of concurrent transactions



There are *(sum(tnj)!/n!)* schedules for *n* transactions (with *tnj* representing the number of R/W ops of transaction *tn*!
i.e., much much more!?

Joseph G Vella - Introduction to databases 16 Concurrency & Recovery

16
      ```

Unrestricted access to a common database state: *dirty read*

- **An unrestricted sharing over data is problematic:**

Time	Database state	Process & Trans. 1 Transaction	Local memory	Process and Trans. 2 Transaction	Local memory	Comments
			X=0		Y=0	
1	A=10, & B=20	read(A,X)	X=10			Read db rec A to var X.
2	A=10, & B=20	X:=X+100	X=110			
3	A=10, & B=20	write(A,X)	X=110			Write var X to db A.
4	A=110, & B=20			read(A,Y)	Y=110	T2 read from T1, and T1 is not yet closed.
5	A=110, & B=20			write(B,Y)	Y=110	
6	A=110, & B=110	rollback	x=10			
7	A=10, & B=110					At this point B is incorrect

17

Unrestricted access to a common database state: *lost update*

- **Concurrency problem - Lost operations**
  - This problem arises due to the following sequence of events;
  -

Time	*Tran. T1*	*Tran. T2*
t1	Read(X)	
t2		Read(X)
t3		X:=(X+1)
t4		Write(X)
t5	X:=(X+1)	
t6	Write(X)	

18

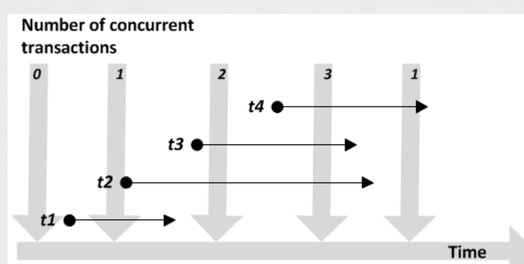
Unrestricted access to a common database state: *inconsistent read*

- **Concurrency problem -- Inconsistencies**
  - This is due to the presence of an integrity constraint. The problem arises each time a transaction accesses or modifies a transitional state of the database characterised by the fact that an integrity constraint is nor verified.
  - *I.C. A = B;*

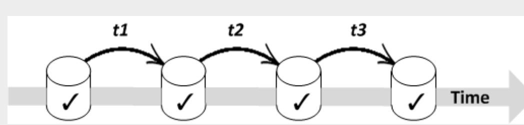
Time	*Tran. T1*	*Tran. T2*	
t1	Read(A)		
t2	A:=A+1		
t3	Write(A)		
t4		Read(A)	
t5		*Print(A)*	*ic broken A<>B*
t6		Read(B)	
t7		*Print(B)*	*ic broken A<>B*
t8	Read(B)		
t9	B:=B+1		
t10	Write(B)		

Sharing and Consistency: Aims

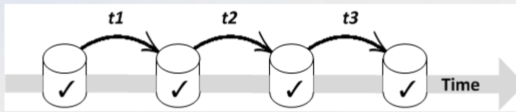
- **Ideally the DBMS must support:**
  - High level of sharing
    - **That is, the more concurrent users are accessing the database the better.**

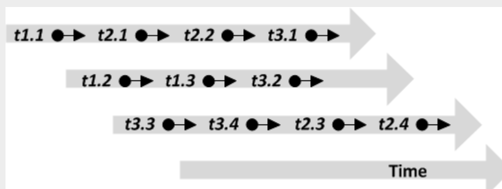


  - Consistency enforcement
    - **But only at the end of each transaction consistency is maintained.**



Is (there any) Serial and its Interleaved transaction resulting in the same database end state?





If yes, then that interleaving as the same effect as the serial!

21

TP & Concurrent Control: Two Phase Locking

22

Two Phase Locking (2PL)

- **Is there an algorithm that can ensure concurrent transactions over a database have the same affect as a (defined) sequence of serial transactions?**
  - Remember the extreme points:
    - **Serial means correct but lowest sharing;**
    - **Interleaved means better utilization through higher access but possible sharing violations (if unrestricted).**
- **Yes for centralised data servers and "short" transactions we use the *two phase locking* algorithm.**

Joseph G Vella - Introduction to databases 23 Concurrency & Recovery

23

Locking

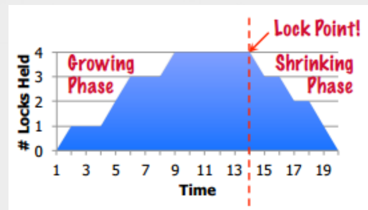
- ***Locks* are variables linked with instances and holds the state of the instances acceptable access operations. Most DBMS have a *lock manager*.**
- **Types of locks include:**
  - ***binary locks***
    - **an item can either be LOCKED or UNLOCKED.**
      - ideal for *mutual exclusion*.
    - **typical operations of a binary lock are:**
      - lock(X);
      - unlock(X).
        - » **THESE OPERATIONS MUST BE INDIVISIBLE.**
    - **for the transaction model a lock operation in invoked when:**
      - lock(X) before every read(X) or write(X) of Ti;
      - unlock(X) after all r/w s are completed in Ti;
      - no issues of lock(X) if it already holds a lock on X;
      - no issues of unlock(X) if it already holds X.
- **Locking on its own does not solve concurrent access violation – unless we don't apply serial transaction processing (i.e. lock the whole database for each transaction in turn).**
- **Heuristic: *where there are locks there is deadlocks!?***

Joseph G Vella - Introduction to databases 24 Concurrency & Recovery

24

2PL

- It is the 2pl scheduler job to manage the locks by controlling when transactions get and free locks.
- 2pl protocols are characterised by diving locking operations into *two* phases: the first being the locks hoarding (*growing*) phase followed by a *shrinking* (releasing) of locks phase.
  - During the growing phase ***no locks are released***.
  - During the shrinking phase ***no locks are requested***.



- **Example:**
  - `T1:rlk(Y),r(Y),ulk(Y),wlk(X),r(X),...,w(X),ulk(X).`
  - T1's schedule does not follow the 2pl protocol because
    - **ulk(Y) precedes wlk(X)!**
  - `T2:rlk(Y),r(Y),wlk(X),ulk(Y),r(X),...,w(X),ulk(X).`
  - T2's schedule does follow a 2pl protocol!

2PL protocols can help with serializability

- There is an *important* theorem that states:
  - ***If all transactions in a schedule adhere to the 2PL protocol then the schedule is serialisable.***
    - **no need to test for serialisability of schedules;**
    - **the protocol enforces serialisability;**
    - **but limits the level of concurrency in a schedule**
      - Why?

2PL variants

- **Other types of 2pl (the previous one is called basic 2pl):**
  - ***conservative 2pl***
    - **Hoard all locks at the beginning of a transaction.**
      - If any *lock* (on a read or write of the transaction) cannot be accessed it waits until all required locks are available.
    - **This protocol is of course deadlock-free!**
  - ***strict 2pl***
    - **Release all of a transaction's locks on commit.**
    - **This protocol is *not* deadlock-free!?**

- **The scheduler needs a strategy for *detecting deadlocks* (i.e. no transaction is indefinitely blocked).**
  - A simple mechanism is by a ***time-out span***. If a transaction has been waiting *more* than the set time-out for a lock it assumes there is a deadlock and aborts (usually, the transaction longest in waiting).

27

SQL Isolation vs Data Sharing Issues

28

SQL Isolation vs Data Sharing Issues

Set a transaction sharing level (per session or transaction)

```
SET TRANSACTION ISOLATION
LEVEL
{ READ {UNCOMMITTED |
COMMITTED } |
  REPEATABLE READ |
  SNAPSHOT |
  SERIALIZABLE };
```

Concurrency: High → Low

Consistency: Low → High

Isolation Level	Phenomena Allowed
Read uncommitted	Dirty Read Non Repeatable Read Phantoms
Read committed	Non Repeatable Read Phantoms
Repeatable read	Phantoms
Serializable	-

Very Important:
This is how the SQL Standard see management of TP; Each vendor adopts with its own mechanism. This leads to some interpretation issues.

Joseph G Vella - Introduction to databases 29 Concurrency & Recovery

29

Database Recovery

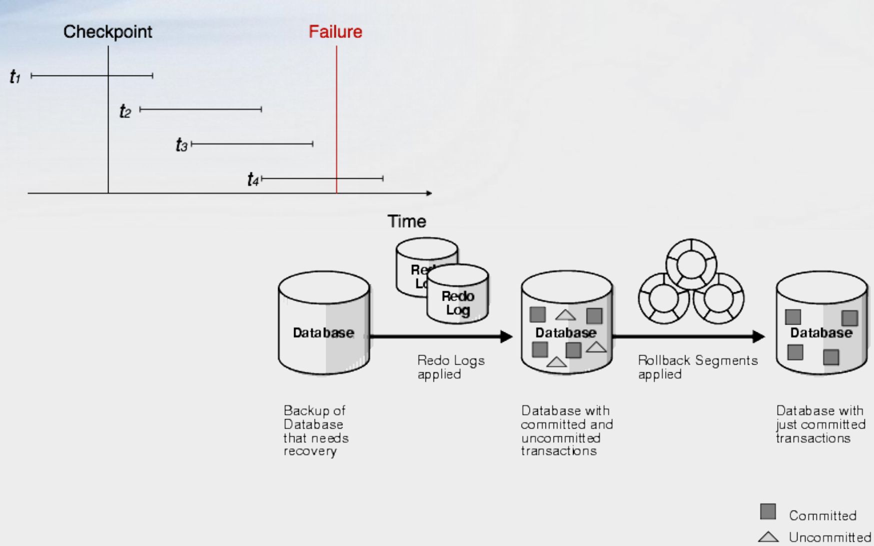
30

Database Recovery

- **DBMSs must *ensure* that a transaction either succeeds or else, on failure, the database is kept in a consistent state (i.e. this process is called recovery).**
- **What are the types of *failures* the DBMSs' transaction sub-system must recover from:**
  - computer hardware, software or network related failures (external to the DBMSs activities)
    - **can affect main and secondary storage structures held by the DBMS;**
  - the DBMSs sub-system that handles transaction processing can have an operational failure
    - **divide by zero during the processing of a transaction;**
    - **illogical sequence of coding in the transaction or system;**
  - the transaction cannot continue due to conflicts with the present data
    - **cannot deduct money from an account that is closed!;**
  - concurrency enforcement problems;
  - disk, network, etc, failure during a DBMS activity;
  - accidental / intentional factors (earthquakes, viruses, deletion of database entities by the system administrator).

31

We need to restore database state as at end of *t3*



32

Recovery Process

- **Recovery entails:**
  - *detection* of a problem; and
  - *co-ordinated action* to eliminate the problem and its effects.
- **How to recover?**
  - Clearly the errors just mentioned require different *strategies* and *tactics*.
    But the worst are those that are associated with a *loss of data*.
  - Most recovery algorithms have these two components:
    - **precautions taken during normal transaction processing to provide for a high level of state recovery;**
    - **actions taken after a failure to ensure the database recovery is graceful and effective.**
  - By building a matrix of storage types vs failure types one can distinctly visualise the variety and extent of recovery procedures: