INFO20003 Database Systems

Dr Renata Borovica-Gajic*

Lecture 17
Transactions

*slides adopted
from David Eccles*

Week 9

- Why we need user-defined transactions

- Properties of transactions

- How to use transactions

- Concurrent access to data

- Locking and deadlocking

- Transaction recovery

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

What is a (database) transaction?

- A logical unit of work that must either be *entirely* completed or aborted (indivisible, atomic)

- DML statements are already atomic

- DBMS also allows for *user-defined* transactions

- These are a sequence of DML statements, such as:

  - a series of UPDATE statements to change values
  - a series of INSERT statements to add rows to tables
  - DELETE statements to remove rows

- A successful transaction changes the database from one consistent state to another

  - All data integrity constraints are satisfied

- Atomicity
  - A transaction is treated as a single, indivisible, logical unit of work. All operations in a transaction must be completed; if not, then the transaction is aborted

- Consistency
  - Constraints that hold before a transaction must also hold after it
  - multiple users accessing the same data see the same value

- Isolation
  - Changes made during execution of a transaction cannot be seen by other transactions until this one is completed

- Durability
  - When a transaction is complete, the changes made to the database are permanent, even if the system fails

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- Transactions solve TWO problems:

  1. users need the ability to define a unit of work

  Assignment Project Exam Help

  2. concurrent access to data by >1 user or program

  https://powcoder.com

  Add WeChat powcoder

- Single DML or DDL command (implicit transaction)
  - Changes are "all or none"
  - **Example**:
    - Update 700 records, but DBMS crashes after 200 records processed
    - Restart server -- you will find no changes to *any* records

Assignment Project Exam Help

- Multiple statements (user-defined transaction)

https://powcoder.com

START TRANSACTION;   (or, 'BEGIN')
  SQL statement;   Add WeChat powcoder
  SQL statement;
  SQL statement;

  …
COMMIT;   (commits the whole transaction)
  Or ROLLBACK (to undo everything)

- SQL keywords: **begin, commit, rollback**

- Each transaction consists of several SQL statements, embedded within a larger application program

- Transaction needs to be treated as an indivisible unit of work

- "Indivisible" means that either the whole job gets done, or none gets done: if an error occurs, we don't leave the database with the job half done, in an inconsistent state

In the case of an error:

- Any SQL statements already completed must be reversed

- Show an error message to the user

- When ready, the user can try the transaction again

- This is briefly annoying – but inconsistent data is disastrous

- Demonstrate Transactions
  - CRE_ACCOUNT TXN_ACCOUNT on LMS resources

```
 9     -- Transaction;
10 •   START TRANSACTION; -- An explicit start - but after any commit a NEW transaction begins
11
12     -- Statement 2
13 •   SELECT * FROM ACCOUNT;
14
15     -- (declare a temporary variable amount persistent for this session)
16 •   set @amount = 100;
17
18     -- Statement 3
19
20 •   UPDATE ACCOUNT set balance = balance - @amount where id =1;
21
22     -- Statement 4 confirm deduction from savings but amount deposited to credit
23 •   SELECT * FROM ACCOUNT;
24
25     --  Statement 5 deposit the amount into the credit account
26 •   UPDATE ACCOUNT set balance = balance + @amount where id = 2;
27
28     -- Statement 6 confirm all changes
29 •   SELECT * FROM ACCOUNT;
30
31     -- Statement 7 EXPLICIT COMMIT;
32 •   COMMIT;
33     |
34     -- ALL CHANGES PERMANENT CAN NOT BE UNDONE WITH ROLLBACK
```

- What happens if we have multiple users accessing the database at the same time? (this is reality)

- Concurrent execution of DML against a shared database

- Note that the sharing of data among multiple users is where much of the benefit of databases comes from – users communicate and collaborate via shared data

- But what could possibly go wrong?
  - lost updates
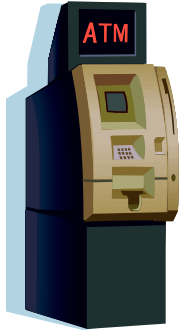 - uncommitted data
  - inconsistent retrievals

Alice

Read account balance
(balance = $1000)

Withdraw $100
(balance = $900)

Write balance
balance = $900

t1a   **t2a**   **t3a**

Time →

t1b   **t2b**   **t3b**

Bob
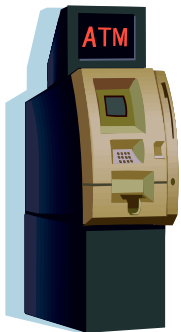
Read account balance
(balance = $1000)

Withdraw $800
(balance = $200)

Write balance
balance = $200

Balance should be $100

- Uncommitted data occurs when two transactions execute concurrently and the first is rolled back after the second has already accessed the uncommitted data

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Alice

Read balance
(balance = $200)

Withdraw $100
(balance = $100)

Write balance
balance = $100

t3

t5

t6

Time

t1

t2

t4

Read balance
(balance = $1000)

Withdraw $800
(balance = $200)

Rollback
balance = $1000

Bob
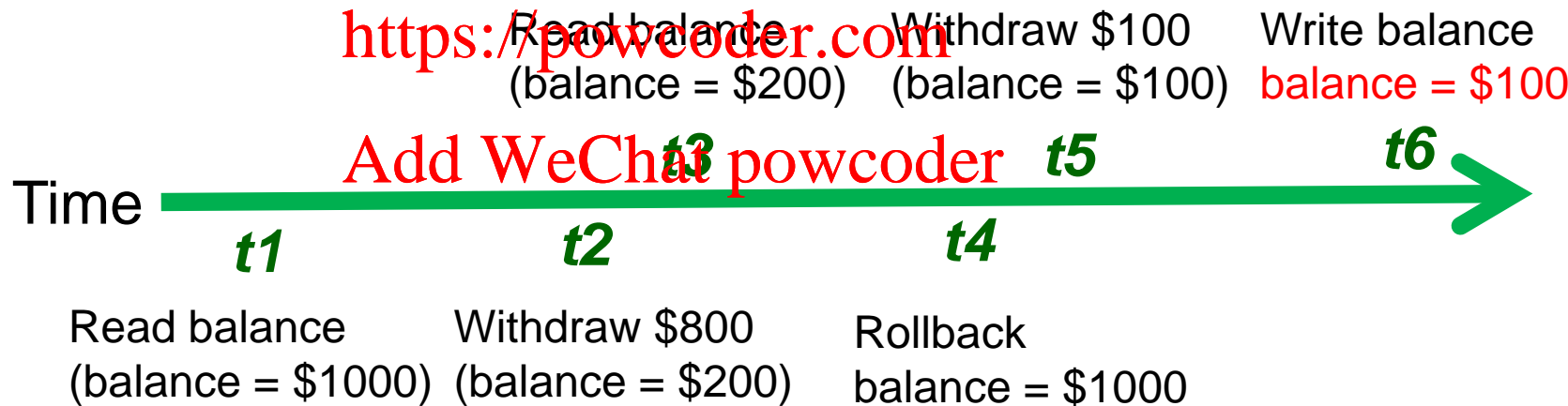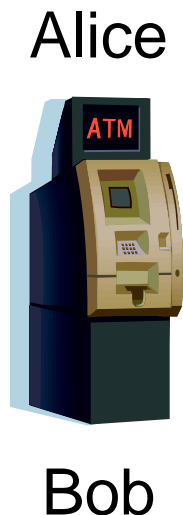
Balance should be $900

- Occurs when one transaction calculates some aggregate functions over a set of data, while other transactions are updating the data
  - Some data may be read after they are changed and some before they are changed, yielding *inconsistent* results

Alice	Bob

SELECT SUM(Salary)
    FROM Employee;
	UPDATE Employee
	        SET Salary = Salary * 1.01
	            WHERE EmpID = 33;

	UPDATE Employee
	        SET Salary = Salary * 1.01
	            WHERE EmpID = 44;

(finishes calculating sum)	COMMIT;

Time	Trans-action	Action	Value	T1 SUM	Comment
1	T1	Read Salary for EmpID 11	10,000	10,000	
2	T1	Read Salary for EmpID 22	20,000	30,000	
3	T2	Read Salary for EmpID 33	30,000		
4	T2	Salary = Salary * 1.01			
5	T2	Write Salary for EmpID 33	30,300		
6	T1	Read Salary for EmpID 33	30,300	60,300	*after* update
7	T1	Read Salary for EmpID 44	40,000	100,300	*before* update
8	T2	Read Salary for EmpID 44	40,000		
9	T2	Salary = Salary * 1.01			
10	T2	Write Salary for EmpID 44	40,400		
11	T2	COMMIT			
12	T1	Read Salary for EmpID 55	50,000	150,300	
13	T1	Read Salary for EmpID 66	60,000	210,300	

we want either
before $210,000
or
after $210,700
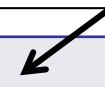
- Transactions ideally are "serializable"
  - Multiple, concurrent transactions *appear as if* they were executed one after another
  - Ensures that the concurrent execution of several transactions yields consistent results

Reality:
concurrent
execution

time

Appearance:
serial
execution

but true serial execution (i.e. no concurrency) is very expensive!

- To achieve efficient execution of transactions, the DBMS creates a *schedule* of read and write operations for concurrent transactions
- Interleaves the execution of operations, based on concurrency control algorithms such as *locking* or *timestamping*
- Several methods of achieving concurrency control
  - Locking ⟵ Main method used
  - Time stamping
  - Optimistic Concurrency Control } Alternatives

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

THE UNIVERSITY OF MELBOURNE

- Lock:
  - Guarantees exclusive use of a data item to a current transaction
    - T1 acquires a lock prior to data access; the lock is released when the transaction is complete
    - T2 does ~~not~~ access to item currently being used by T1
    - T2 has to wait until T1 releases the lock
  - Required to prevent another transaction from reading inconsistent data
- Lock manager
  - Responsible for assigning and policing the locks used by the transactions

- Question: at what granularity should we apply locks?

- Database-level lock
  - Entire *database* is locked
  - Good for batch processing but unsuitable for multi-user DBMSs
  - T1 and T2 can not access the same database concurrently even if they use different tables
  - Examples: SQLite, Access
- Table-level lock
  - Entire *table* is locked - as above but not quite as bad
  - T1 and T2 can access the same database concurrently as long as they use different tables
  - Can cause bottlenecks, even if transactions want to access different parts of the table and would not interfere with each other
  - Not suitable for highly multi-user DBMSs

- Page-level lock
  - An entire *disk page* is locked
  - Not commonly used now

- Row-level lock
  - Allows concurrent transactions to access *different rows* of the same table, even if the rows are located on the same page
  - Improves data availability, but with high overhead (each row has a lock that must be read and written to)
  - Currently the most popular approach (MySQL, Oracle)

- Field-level lock
  - Allows concurrent transactions to access the same row, as long as they access different attributes within that row
  - *Most flexible* lock but requires an extremely high level of overhead
  - Not commonly used

- Binary Locks
  - Has only two states: locked (1) or unlocked (0)
  - Eliminates "Lost Update" problem
    - the lock is not released until the statement is completed
  - Considered too restrictive to yield optimal concurrency,
    as it locks even for two READs (when no update is being done)

- The alternative is to allow both Shared and Exclusive locks
  - Often called Read and Write locks (discussed next)

- Exclusive lock
  - Access is reserved for the transaction that locked the object
  - Must be used when transaction intends to WRITE
  - Granted if and only if *no other locks* are held on the data item
  - In MySQL: "select … for update"

- Shared lock
  - Other transactions are also granted Read access
  - Issued when a transaction wants to READ data,
    and no Exclusive lock is held on that data item
    - Multiple transactions can each have a shared lock on the same data item if they are all just reading it
  - In MySQL: "select … for share"

- Condition that occurs when two transactions wait for each other to unlock data

  – T1 locks data item X, then wants Y

  – T2 locks data item Y, then wants X

  – Each waits to get a data item which the other transaction is already holding

  – Could wait forever if not dealt with

- Only happens with exclusive locks

- Deadlocks are dealt with by:

  – Prevention

  – Detection

  – (we won't go into details)

Transaction 1 — Supplier — **Holds** lock on Supplier — **Wants** lock on Part — **blocked** — **Transaction 2** — Part — **Holds** lock on Part — **Wants** lock on Supplier

- Two separate sessions
- In order:
1. Tx1 Update row 3 (Green)
2. Tx2 Update row 2 (White)
3. Tx3 Update row 2 (Green)
4. Tx4 Update row 3 (White)

- Note: Only the session which detects the deadlock rolls back the transaction. The Green session still holds locks on row 2 and 3

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- Timestamp
  - Assigns a global unique timestamp to each transaction
  - Each data item accessed by the transaction gets the timestamp
  - Thus for every data item, the DBMS knows which transaction performed the last read or write on it
  - When a transaction wants to read or write, the DBMS compares its timestamp with the timestamps already attached to the item and decides whether to allow access
- Optimistic
  - Based on the assumption that the majority of database operations *do not* conflict
  - Transaction is executed without restrictions or checking
  - Then when it is ready to commit, the DBMS checks whether any of the data it read has been altered – if so, rollback

- Allow us to restore the database to a previous consistent state
- If a transaction cannot be completed, it must be aborted and any changes rolled back
- To enable this, DBMS tracks *all* updates to data
- This *transaction log* contains:
  1. A record for the beginning of the transaction
  2. For each SQL statement
     - operation being performed (update, delete, insert)
     - objects affected by the transaction
     - "before" and "after" values for updated fields
     - pointers to previous and next transaction log entries
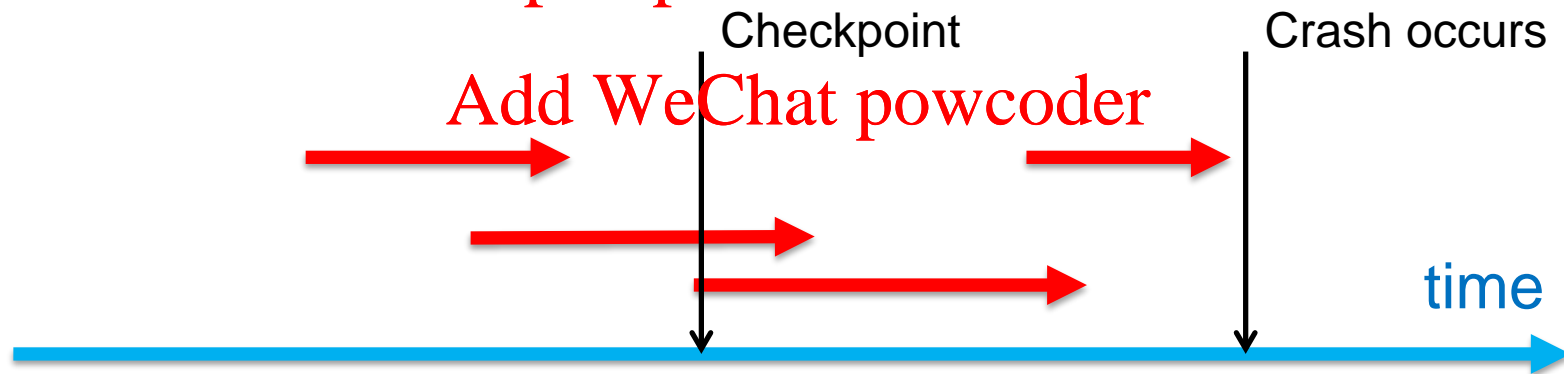 3. The ending (COMMIT) of the transaction

- Also provides the ability to restore a corrupted database
- If a system failure occurs, the DBMS will examine the log for all uncommitted or incomplete transactions and it will restore the database to a previous state

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

Checkpoint

Crash occurs

time

-25-

TRL ID	TRX NUM	PREV PTR	NEXT PTR	OPERATION	TABLE	ROW ID	ATTRIBUTE	BEFORE VALUE	AFTER VALUE
341	101	Null	352	START	****Start Transaction				
352	101	341	363	UPDATE	PRODUCT	54778-2T	PROD_QOH	45	43
363	101	352	365	UPDATE	CUSTOMER	10011	CUST_BALANCE	615.73	675.62
365	101	363	Null	COMMIT	**** End of Transaction				
397	106	Null	405	START	****Start Transaction				
405	106	397	415	INSERT	INVOICE	1009			1009,10016, ...
415	106	405	419	INSERT	LINE	1009,1			1009,1, 89-WRE-Q,1, ...
419	106	415	427	UPDATE	PRODUCT	89-WRE-Q	PROD_QOH	12	11
423				CHECKPOINT					
427	106	419	431	UPDATE	CUSTOMER	10016	CUST_BALANCE	0.00	277.55
431	106	427	457	INSERT	ACCT_TRANSACTION	10007			1007,18-JAN-2004, ...
457	106	431	Null	COMMIT	**** End of Transaction				
521	155	Null	525	START	****Start Transaction				
525	155	521	528	UPDATE	PRODUCT	2232/QWE	PROD_QOH	6	26
528	155	525	Null	COMMIT	**** End of Transaction				

* * * * * C *R*A* S* H * * * * *

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder

- Why do we need transactions?

- What is a transaction?

- ACID (properties of ACID)

- Locking levels & types including deadlock scenario

  – Exclusive and Shared Locks

- Concurrency

  – Being able to demonstrate concurrency

- Concurrency Issues

  – (Lost update, uncommitted changes, inconsistent retrieval)

- Logging

- Database administration

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder