4.1. Transaction processing

In this lecture we look at...
[Section notes PDF 86Kb]

4.1.01. Distributed Databases

- Transactions
- Unpredictable failure
  - Commit and rollback
- Stored procedures
- Brief PL overview
  - Cursors

4.1.02. Transactions

- Real world database actions
- Rarely single step
- Flight reservation example
  - Add passenger details to roster
  - Charge passenger credit card
  - Update seats available
  - Order extra vegetarian meal

4.1.04. Desirable properties of transactions

ACID test

- Atomicity
  - transaction as smallest unit of processing
  - transactions complete entirely or not at all
    - consequences of partial completion in flight example
- Consistency
  - complete execution preserves database constrained state/integrity
  - e.g. Should a transaction create an entity with a foreign key then the reference entity must exist (see 4 constraints)

4.1.05. ACID test continued

- Isolation
  - not interfered with by any other concurrent transactions
- Durable (permanency)
  - commited changes persist in the database, not vulernable to failure

4.1.06. Commit

- Notion of Commit (durability)
- Transaction failures
    - From flight reservation example
        - Add passenger details to roster
        - Charge passenger credit card
        - Update seats available: No seats remaining
        - Order extra vegetarian meal
- Rollback

4.1.07. PL/SQL overview

- Language format
    - Declarations
    - Execution
    - Exceptions
    - Handling I/O
    - Functions
    - Cursors

4.1.08. PL/SQL

- Blocks broken into three parts
    - Declaration
        - Variables declared and initialised
    - Execution
        - Variables manipulated/actioned
    - Exception
        - Error raised and handled during exec
-

```
DECLARE
    ---declarations
BEGIN
    ---statements
EXCEPTION
    ---handlers
END ;
```

4.1.09. Declaration

- DECLARE
    - age NUMBER;
    - name VARCHAR(20);
    - surname employee.fname%TYPE;
    - addr student.termAddress%TYPE;

4.1.10. Execution

- BEGIN (not in order)
  - /* sql_statements */
    - UPDATE employee SET salary = salary+1;
  - /* conditionals */
    - IF (age < 0) THEN
      - age: = 0;
    - ELSE
      - age: = age + 1;
    - END IF;
  - /* transaction processing */
    - COMMIT; ROLLBACK;
  - /* loops */ /* cursors */
- [END;] (if no exception handling)

4.1.11. Exception passing

- Beginnings of PL I/O
- CREATE TABLE temp (logmessage varchar(80));
  - Can create transfer/bridge relation outside
- Within block (e.g. within exception handler)
  - WHEN invalid_age THEN
    - INSERT INTO temp VALUES(‘Cannot have negative ages’);
  - END;
  - SELECT * FROM temp;
    - To review error messages

4.1.12. Exception handling

- DECLARE
  - invalid_age exception;
- BEGIN
  - IF (age < 0) THEN
    - RAISE invalid_age
  - END IF;
- EXCEPTION
  - WHEN invalid_age THEN
    - INSERT INTO temp VALUES(‘Cannot have negative ages’);
  - END;

4.1.13. Cursors

- Cursors
  - Tuple by tuple processing of relations
  - Three phases (two)
    - Declare
    - Use

- Exception (as per normal raise)

4.1.14. Impact

- PL blocks coherently change database state
- No runtime I/O
- Difficult to debug
- SQL tested independently

4.1.15. PL Cursors

- DECLARE
- name_attr EMPLOYEE.NAME%TYPE;
- ssn_attr EMPLOYEE.SSN%TYPE;
- /* cursor declaration */
- CURSOR myEmployeeCursor IS
  - SELECT NAME,SSN FROM EMPLOYEE
  - WHERE DNO=1
  - FOR UPDATE;
- emp_tuple myEmployeeCursor%ROWTYPE;

4.1.16. Cursors execution

- BEGIN
- /* open cursor */
- OPEN myEmployeeCursor;
- /* can pull a tuple attributes into variables */
- FETCH myEmployeeCursor INTO name_attr,ssn_attr;
- /* or pull tuple into tuple variable */
- FETCH myEmployeeCursor INTO emp_tuple;
- CLOSE myEmployeeCursor;

- [LOOP…END LOOP example on handout]

4.1.17. Concurrency Introduction

- Concurrent transactions
- Distributed databases (DDB)
- Fragmentation
- Desirable transaction properties
- Concurrency control techniques
  - Locking
  - Timestamps

4.1.18. Notation

- Language
  - PL too complex/long-winded
- Simplified database model
  - Database as collection of named items
  - Granularity, or size of data item
  - Disk block based, each block X
- Basic transaction language (BTL)
  - read_item(X);
  - write_item(X);
  - Basic algebra, X=X+N;

4.1.19. Transaction processing

- DBMS Multiuser system
  - Multiple terminals/clients
    - Single processor, client side execution
  - Single centralised database
    - Multiprocessor, server
    - Resolving many transactions simultaneously
- Concurrency issue
  - Coverage by previous courses (e.g. COMS12100)
  - PL/SQL scripts (Transactions) as processes
- Interleaved execution

4.1.20. Transactions

- Two transactions, T_1 and T_2
- Overlapping read-sets and write-sets
- Interleaved execution
- Concurrency control required
- PL/SQL example
  - Commit; and rollback;

4.1.21. Concurrency issues

- Three potential problems
  - Lost update
  - Dirty read
  - Incorrect summary
- All exemplified using BTL
  - Transaction diagrams to make clearer
  - C-like syntax for familiarity
  - Many possible examples of each problem

4.1.22. Lost update

T_1	T_2
read_item(X);	
X=X-N;	

write_item(X); read_item(Y);	read_item(X); X=X+M;	• T_1 X update
Y=Y+N; write_item(Y);	write_item(X); overwritten	

4.1.23. Dirty read (or Temporary update)

T_1	T_2

read_item(X);
X=X-N;
write_item(X);	

	read_item(X);
	X=X+M;
	write_item(X);

<T_1 fails> <T_1 rollback>	

read_item(X);
X=X+N;
write_item(X);	

- T_2 reads temporary incorrect value of X

4.1.24. Incorrect summary

T_1	T_2

	sum=0;
	read_item(A)
	sum=sum+A;

read_item(X);
X=X-N;
write_item(X);	

	read_item(X);
	sum=sum+X;
	read_item(Y);
	sum=sum+Y;

read_item(Y);
Y=Y-N;
write_item(Y);	

- T_2 sums after X-N and before Y-N

4.1.25. Serializability

- Schedule S is a collection of transactions (T_i)
- Serial schedule S_1
  - Transactions executed one after the other
  - Performed in a serial order
  - No interleaving

    - Commit or abort of active transaction (T_i) triggers execution of the next (T_{i+1})
    - If transactions are independent
        - all serial schedules are correct

4.1.26. Serializability

- Serial schedules/histories
    - No concurrency
    - Unfair timeslicing
- Non-serial schedule S_2 of n transactions
    - Serializable if
- equivalent to some serial schedule of the same n transactions
    - correct
- n! serial schedules, more non-serial

4.1.27. Distribution

- DDB, collection of
    - multiple logically interrelated databases
    - distributed over a computer network
    - DDBMS
- Multiprocessor environments
    - Shared memory
    - Shared disk
    - Shared nothing

4.1.28. Advantages

- Distribution transparency
    - Multiple transparency levels
    - Network
    - Location/dept autonomy
    - Naming
    - Replication
    - Fragmentation
- Reliability and availability
- Performance, data localisation
- Expansion

4.1.29. Fragmentation

- Breaking the database into
    - logical units
    - for distribution (DDB design)
- Global directory to keep track/abstract
- Fragmentation schema/allocation schema
    - Relational

    - Horizontal
        - Derived (referential), complete (by union)
    - Vertical
    - Hybrid

4.1.30. Concurrency control in DDBs

- Multiple copies
- Failure of individual sites (hosts/servers)
- Failure of network/links
- Transaction processing
    - Distributed commit
    - Deadlock
- Primary/coordinator site - voting

4.1.31. Distributed commit

- Coordinator elected
- Coordinator prepares
    - writes log to disk, open sockets, sends out queries
- Process
    - Coordinator sends ‘Ready-commit’ message
    - Peers send back ‘Ready-OK’
    - Coordinator sends ‘Commit’ message
    - Peers send back ‘Commit-OK’ message

4.1.32. Query processing

- Data transfer costs of query processing
    - Local bias
    - High remote access cost
    - Vast data quantities to build intermediate relations
- Decomposition
    - Subqueries resolved locally

4.1.33. Concurrency control

- Must avoid 3+ problems
    - Lost update, dirty read, incorrect summary
    - Deadlock/livelock - dining example
- Data item granularity
- Solutions
    - Protocols, validation
    - Locking
    - Timestamps

4.1.34. Definition of terms

- Binary (two-state) locks
- locked, unlocked associated with item X
- Mutual exclusion
- Four requirements
    - Must lock before access
    - Must unlock after all access
    - No relocking of already locked
    - No unlocking of already unlocked

4.1.35. Definition

- Multiple mode locking
- Read/write locks
- aka. shared/exclusive locks
- Less restrictive (CREW)
- read_lock(X), write_lock(X), unlock(X)
    - e.g. acquire read/write_lock
    - not reading or writing the lock state

4.1.36. Rules of Multimode locks

- Must hold read/write_lock to read
- Must hold write_lock to write
- Must unlock after all access
- Cannot upgrade/downgrade locks
    - Cannot request new lock while holding one
- Upgrading permissable (read lock to write)
    - if currently holding sole read access
- Downgrading permissable (write lock to read)
    - if currently holding write lock