Distributed Database

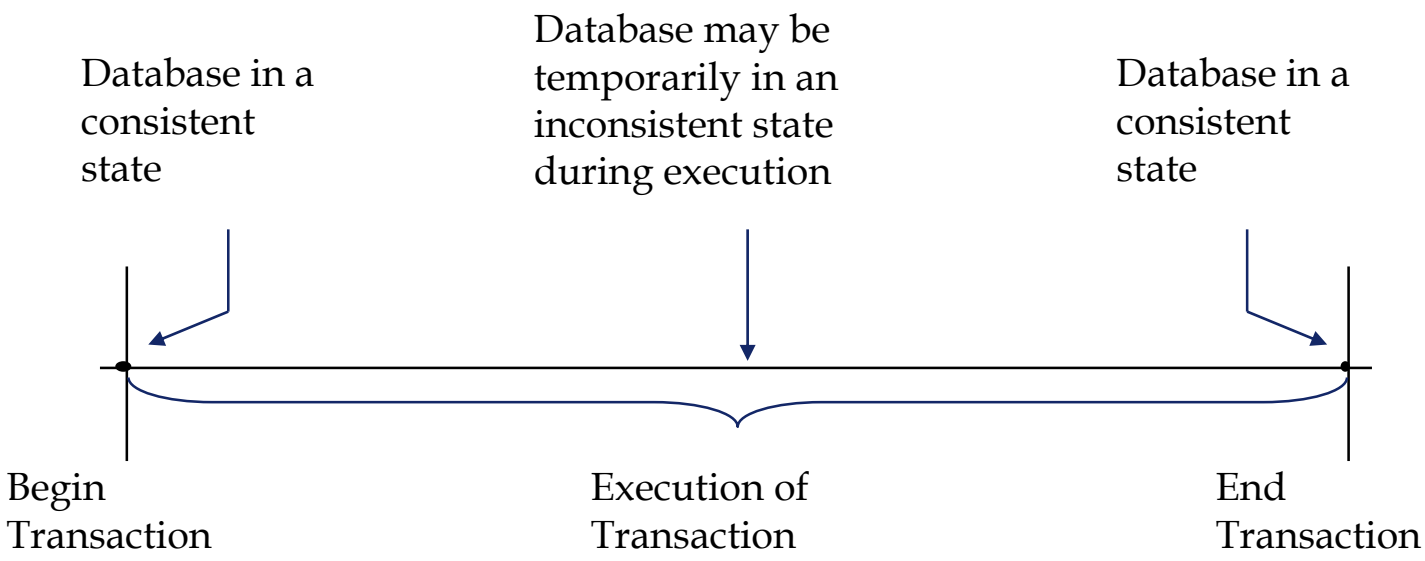
Munawar, PhD

Transaction Management and Architecture

Transaction

A transaction is a collection of actions that make consistent transformations of system states while preserving system consistency.

- concurrency transparency
- failure transparency



Transaction Example – A Simple SQL Query

Transaction BUDGET_UPDATE

begin

```
    EXEC SQL        UPDATE      PROJ
                    SET         BUDGET = BUDGET*1.1
                    WHERE       PNAME = "CAD/CAM"
```

end.

Example Database

Consider an airline reservation example with the relations:

FLIGHT(FNO, DATE, SRC, DEST, STSOLD, CAP)
CUST(CNAME, ADDR, BAL)
FC(FNO, DATE, CNAME,SPECIAL)

Example Transaction – SQL Version

```
Begin_transaction Reservation
begin
        input(flight_no, date, customer_name);
        EXEC SQL        UPDATE          FLIGHT
                        SET             STSOLD = STSOLD + 1
                        WHERE           FNO = flight_no AND DATE =
                date;
        EXEC SQL        INSERT
                        INTO            FC(FNO, DATE, CNAME,
                SPECIAL);
                        VALUES          (flight_no, date, customer_name,
                null);
        output("reservation completed")
end . {Reservation}
```

Termination of Transactions

```
Begin_transaction Reservation
begin
input(flight_no, date, customer_name);
EXEC SQL                    SELECT  STSOLD,CAP
                                    INTO            temp1,temp2
                                    FROM            FLIGHT
                                    WHERE           FNO = flight_no AND DATE =  date;
if temp1 = temp2 then
output("no free seats");
Abort
else
EXEC SQL                    UPDATE FLIGHT
                                                    SET             STSOLD = STSOLD + 1
                                                    WHERE           FNO = flight_no AND DATE =
date;
EXEC SQL                    INSERT
                                                    INTO            FC(FNO, DATE, CNAME,
SPECIAL);
                                                    VALUES          (flight_no, date, customer_name,
null);
                            Commit
                            output("reservation completed")
                            endif
end . {Reservation}
```

Example Transaction – Reads & Writes

```
Begin_transaction Reservation
begin
        input(flight_no, date, customer_name);
        temp ← Read(flight_no(date).stsold);
        if temp = flight(date).cap then
        begin
                output("no free seats");
                Abort
        end
        else begin
                Write(flight(date).stsold, temp + 1);
                Write(flight(date).cname, customer_name);
                Write(flight(date).special, null);
                Commit;
                output("reservation completed")
        end
end. {Reservation}
```

Characterization

- Read set (RS)
  - The set of data items that are read by a transaction
- Write set (WS)
  - The set of data items whose values are changed by this transaction
- Base set (BS)
  - RS & WS

Principles of Transactions

ATOMICITY

- all or nothing

CONSISTENCY

- no violation of integrity constraints

ISOLATION

- concurrent changes invisible $\Rightarrow$ serializable

DURABILITY

- committed updates persist

Atomicity

- Either all or none of the transaction's operations are performed.
- Atomicity requires that if a transaction is interrupted by a failure, its partial results must be undone.
- The activity of preserving the transaction's atomicity in presence of transaction aborts due to input errors, system overloads, or deadlocks is called transaction recovery.
- The activity of ensuring atomicity in the presence of system crashes is called crash recovery.

Consistency

- Internal consistency
  - A transaction which executes alone against a consistent database leaves it in a consistent state.
  - Transactions do not violate database integrity constraints.
- Transactions are correct programs

Consistency Degrees

- Degree 0
  - Transaction *T* does not overwrite dirty data of other transactions
  - Dirty data refers to data values that have been updated by a transaction prior to its commitment
- Degree 1
  - *T* does not overwrite dirty data of other transactions
  - *T* does not commit any writes before EOT

Isolation

- Serializability
  - If several transactions are executed concurrently, the results must be the same as if they were executed serially in some order.
- Incomplete results
  - An incomplete transaction cannot reveal its results to other transactions before its commitment.
  - Necessary to avoid cascading aborts.

Durability

- Once a transaction commits, the system must guarantee that the results of its operations will never be lost, in spite of subsequent failures.
- Database recovery

Thank You !

Munawar, PhD