

Transazioni

Introduction

- **Single-User System:**
 - At most one user at a time can use the system
- **Multuser System:**
 - Many users can access the system concurrently
- **Concurrency**
 - **Interleaved processing:**
 - Concurrent execution of processes is interleaved in a single CPU
 - **Parallel processing:**
 - Processes are concurrently executed in multiple CPUs

Introduction

- A Transaction:
 - Logical unit of database processing that includes one or more access operations (read - retrieval, write - insert or update, delete)
- A transaction (set of operations) may be stand-alone specified in a high level language like SQL submitted interactively, or may be embedded within a program
- An application program may contain several transactions separated by the Begin and End transaction boundaries

24

Controllo di affidabilità

- **Nell'esecuzione di una transazione, il DBMS è responsabile che:**
 - Tutte le operazioni della transazione siano completate (ed il loro effetto memorizzato) con successo
 - Nessuna delle operazioni della transazione abbia effetto sulla base dati
- **Il DBMS non deve consentire che solo una parte delle operazioni di una transazione abbia affetto sulla base dati**
 - Questo potrebbe succedere se interviene un guasto/errore durante l'esecuzione della transazione

25

Why recovery is needed (What causes a Transaction to fail)

1. A computer failure (system crash):

A hardware or software error occurs in the computer system during transaction execution. If the hardware crashes, the contents of the computer's internal memory may be lost.

2. A transaction or system error:

Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error. In addition, the user may interrupt the transaction during its execution.

26

Why recovery is needed (What causes a Transaction to fail)

3. Local errors or exception conditions detected by the transaction:

Certain conditions necessitate cancellation of the transaction. For example, data for the transaction may not be found. A condition, such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal from that account, to be canceled.

A programmed abort in the transaction causes it to fail.

4. Concurrency control enforcement:

The concurrency control method may decide to abort the transaction, to be restarted later, because it violates serializability or because several transactions are in a state of deadlock

27

Why recovery is needed (What causes a Transaction to fail)

5. Disk failure:


Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction

6. Physical problems and catastrophes:

This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator

28

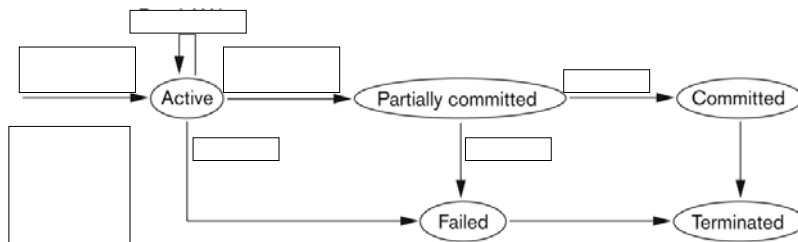
Why recovery is needed

- 
- 1. A computer failure (system crash)**
 - 2. A transaction or system error**
 - 3. Local errors or exception conditions detected by the transaction**
 - 4. Concurrency control enforcement**
 - 5. Disk failure**
 - 6. Physical problems and catastrophes**

29

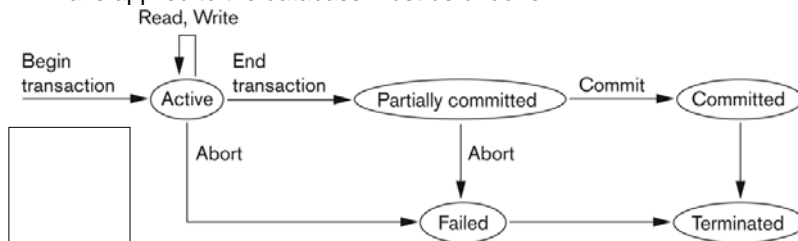
Transaction and System Concepts

- For recovery purposes, the system needs to keep track of when the transaction starts, terminates, commits or aborts.
- Transaction states
 - Active
 - Partially committed
 - Committed
 - Failed
 - Terminated



Transaction and System Concepts

- Recovery manager keeps track of the following operations:
 - begin_transaction: This marks the beginning of transaction execution.
 - read or write: These specify read or write operations in a transaction
 - end_transaction: This marks the end limit of transaction execution.
 - At this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database or whether the transaction has to be aborted
 - commit_transaction: This signals a successful end of the transaction so that any changes (updates) executed by the transaction can be safely committed to the database and will not be undone.
 - rollback (or abort): This signals that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.



Proprietà ACID

- **Una corretta gestione delle transazioni da parte del DBMS richiede che vengano garantite quattro proprietà, note con il nome di proprietà acide (ACID):**
 - **Atomicità (atomicity)**
 - **Consistenza (consistency)**
 - **Isolamento (isolation)**
 - **Persistenza (durability)**
- **Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione delle transazioni viene detto sistema transazionale**

32

Proprietà ACID

- **Atomicità: una transazione raggruppa un insieme di istruzioni in una unità indivisibile di esecuzione**
 - **Al termine della transazione il database potrà trovarsi in due soli stati:**
 - **Con tutte le modifiche previste dalla corretta esecuzione delle istruzioni della transazione**
 - **Come se nessuna delle istruzioni della transazione fosse mai stata eseguita**

33

Proprietà ACID

- **Atomicità: una transazione raggruppa un insieme di istruzioni in una unità indivisibile di esecuzione**
 - Si possono individuare tre diversi scenari
 - La transazione va a buon fine: nessuna delle istruzioni previste genera un errore o una condizione che richieda l'annullamento delle istruzioni eseguite sino a quel momento.
La transazione termina con il comando COMMIT
 - La transazione non va a buon fine perché si verifica una condizione che richiede l'annullamento delle istruzioni eseguite sino a quel momento
La transazione termina con il comando ROLLBACK
 - La transazione non va a buon fine perché viene generato un errore imprevisto e non gestibile all'interno della transazione
Il DBMS esegue un ROLLBACK

34

Proprietà ACID

- **Consistenza: il database deve trovarsi in uno stato consistente sia prima che dopo l'esecuzione di una transazione**
 - Una transazione non verrà completata se al suo termine dovesse lasciare il database in uno stato inconsistente, per esempio memorizzando dei dati che violano i vincoli di integrità del database:
 - Vincoli di chiave, di integrità referenziale, di dominio...

35

Proprietà ACID

- All'atto della specifica di un vincolo questo può essere definito come immediato (è il valore di default) o differito (deferrable)
- **Vincoli immediati:**
 - Generano un errore appena viene eseguita una istruzione che causa la loro violazione: l'errore può essere intercettato (exception) e gestito all'interno della transazione
- **Vincoli differiti:**
 - Vengono verificati solo al termine della transazione (COMMIT): non possono più essere gestiti all'interno della transazione. Una loro violazione viene gestita dal DBMS che impone un ROLLBACK della transazione

36

Vincoli differiti

- **Non tutti i DBMS supportano i vincoli differiti (e.g. MySQL non li supporta)**
- **La proprietà di differibilità di un vincolo deve essere specificata alla creazione del vincolo**

```
CREATE TABLE Dipartimento(  
    Codice varchar(50) PRIMARY KEY,  
    Nome varchar(50),  
    Direttore varchar(20),  
    CONSTRAINT fk_direttore FOREIGN KEY (Direttore)  
        REFERENCES Impiegato(Matricola) DEFERRABLE );
```

37

Vincoli differiti

- **Quando necessario, la momentanea disabilitazione del vincolo può essere impostata con il comando**
`SET CONSTRAINT <nome_vincolo> DEFERRED;`


- **Per esempio:**

```
SET CONSTRAINT fk_direttore DEFERRED;
```

```
INSERT INTO Dipartimento VALUES('D001', 'Amministrazione', 'AA998CC');
```

```
INSERT INTO Impiegato VALUES('AA998CC', 'Carlo', 'Cancelli', 'D001', 88000);
```

```
COMMIT;
```



Questo comando avrebbe provocato un errore (in assenza dell'impiegato con matricola AA998CC) se eseguito senza il costrutto DEFERRED.

Proprietà ACID

- **Persistenza: gli effetti di una transazione che ha eseguito correttamente il COMMIT non devono essere persi, devono essere memorizzati in modo permanente nel database**
 - Una volta accettato il COMMIT, il sistema deve garantire che anche in caso di guasto non verrà compromessa la memorizzazione delle operazioni effettuate

Proprietà ACID

- **Isolamento:** in un ambiente in cui più transazioni vengono eseguite in modo concorrente, ciascuna transazione dovrebbe essere isolata dalle altre
 - Il risultato di n transazioni eseguite in modo concorrente deve essere lo stesso che si otterrebbe eseguendo le transazioni in modo sequenziale, una alla volta

40

Livelli di isolamento

- In assenza di isolamento l'esecuzione di più transazioni in un ambiente concorrente può generare diverse anomalie
- In particolare, i livelli di isolamento vengono classificati in relazione a tre distinte anomalie:
 - Dirty read: è possibile che una transazione T1 legga un dato modificato da una transazione T2 prima che T2 esegua il COMMIT
 - Nonrepeatable read: è possibile che all'interno di una transazione T1 un dato, letto due volte in momenti diversi, abbia valori diversi perché tra le due letture è stato modificato da una transazione T2 che ha eseguito il COMMIT (altrimenti sarebbe un dirty read)
 - Phantom read: è possibile che all'interno di una transazione T1 una SELECT, eseguita in due momenti diversi M1 ed M2, in M2 recuperi delle righe in più rispetto a quelle recuperate in M1, perché tra le due letture una transazione T2 ha aggiunto dei dati alla tabella oggetto della SELECT ed eseguito il COMMIT

41

Livelli di isolamento

- Con riferimento a tali anomalie, lo standard SQL definisce quattro livelli di isolamento (ANSI isolation levels)

| Isolation level | Dirty read | Nonrepeatable read | Phantom read |
|------------------|------------|--------------------|--------------|
| READ UNCOMMITTED | Permitted | Permitted | Permitted |
| READ COMMITTED | --- | Permitted | Permitted |
| REPEATABLE READ | --- | --- | Permitted |
| SERIALIZABLE | --- | --- | --- |

42

Livelli di isolamento

- Supponiamo di avere creato la seguente tabella:

```
CREATE TABLE CONTO(  
    id number primary key,  
    saldo number not null default 0.0)
```

- ...di averla così popolata...

| CONTO | |
|-------|-------|
| ID | SALDO |
| 1 | 100 |
| 2 | 200 |
| ... | ... |
| 100 | 50 |

- ...e che nel sistema siano in esecuzione due transazioni che leggono/modificano il contenuto della tabella

43

Dirty reads

CONTO

| ID | SALDO |
|-----|-------|
| 1 | 100 |
| 2 | 200 |
| ... | ... |
| 100 | 50 |

- Se la prima select di T1 legge saldo=100 ed anche la seconda select di T1 legge saldo=100 allora si ha una dirty read

T
E
M
P
O

```
/* Transaction 1 */
totale :=0.0;
SELECT saldo INTO tmp FROM CONTO
WHERE id=1;
totale := totale+tmp;

/* Transaction 2 */
/* Query 2 */
UPDATE CONTO SET saldo=saldo-50 WHERE id=1;
UPDATE CONTO SET saldo=saldo+50 WHERE id=100;

SELECT saldo INTO tmp FROM CONTO
WHERE id=100;
totale := totale+tmp;

COMMIT;
```

44

Nonrepeatable reads

CONTO

| ID | SALDO |
|-----|-------|
| 1 | 100 |
| 2 | 200 |
| ... | ... |
| 100 | 50 |

- Se la prima select di T1 legge saldo=100 ed anche la seconda select di T1 legge saldo=100 allora si ha una nonrepeat. read

T
E
M
P
O

```
/* Transaction 1 */
totale :=0.0;
SELECT saldo INTO tmp FROM CONTO
WHERE id=1;
totale := totale+tmp;

/* Transaction 2 */
/* Query 2 */
UPDATE CONTO SET saldo=saldo-50 WHERE id=1;
UPDATE CONTO SET saldo=saldo+50 WHERE id=100;
COMMIT;

SELECT saldo INTO tmp FROM CONTO
WHERE id=100;
totale := totale+tmp;
```

45

Phantom reads

CONTO

| ID | SALDO |
|-----|-------|
| 1 | 100 |
| 2 | 200 |
| ... | ... |
| 100 | 50 |

- Se la seconda select di T1 legge una riga in più della prima (quella aggiunta dalla T2) si ha una phantom read

T
E
M
P
O

```
/* Transaction 1 */
/* Query 1 */
SELECT * FROM CONTO
WHERE saldo BETWEEN 100 AND 300;

/* Query 1 */
SELECT * FROM CONTO
WHERE saldo BETWEEN 100 AND 300;

/* Transaction 2 */
/* Query 2 */
INSERT INTO CONTO VALUES ( 110, 250 );
COMMIT;
```