Basi di Dati
Transazioni

Modifica dello stato di una BD

- Lo stato della base di dati è modificato da INSERT, DELETE e UPDATE

- Il DBMS accetta le modifiche solo se lo stato della base di dati successivo alla modifica risulta **corretto** rispetto ai **vincoli**
  - Vincoli di chiave
  - Vincoli di integrità referenziale
  - Vincoli di dominio
  - …

Esempio

Clienti di un istituto di credito e relativi conti correnti

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

con i vincoli di **integrità referenziale**:

- TITOLARE(CF) referenzia CLIENTE(CF)
- TITOLARE(CC) referenzia CONTO(CC)

Esempio

Clienti di un istituto di credito e relativi conti correnti

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Nuovo vincolo: quando la banca ha un nuovo cliente, questi deve **anche** essere titolare di un conto corrente, cioè CLIENTE(CF) referenzia TITOLARE(CF).

Esempio

Clienti di un istituto di credito e relativi conti correnti

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Vincoli:

- TITOLARE(CF) referenzia CLIENTE(CF)
- TITOLARE(CC) referenzia CONTO(CC)
- CLIENTE(CF) referenzia TITOLARE(CF)

Esempio

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Vincoli:

- TITOLARE(CF) referenzia CLIENTE(CF)
- TITOLARE(CC) referenzia CONTO(CC)
- CLIENTE(CF) referenzia TITOLARE(CF)

Operazioni da eseguire:

a. INSERT nuovo **conto** in CONTO

b. INSERT nuovo **cliente** in CLIENTE

c. INSERT **(nuovo cliente, nuovo conto)** in TITOLARE

Esempio

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Vincoli:

- TITOLARE(CF) referenzia CLIENTE(CF)
- TITOLARE(CC) referenzia CONTO(CC)
- CLIENTE(CF) referenzia TITOLARE(CF)

Operazioni da eseguire:

a. **INSERT nuovo conto in CONTO**

b. INSERT nuovo **cliente** in CLIENTE

c. INSERT **(nuovo cliente, nuovo conto)** in TITOLARE

Supponiamo di eseguire **prima a.**

Esempio

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Vincoli:

- TITOLARE(CF) referenzia CLIENTE(CF)
- TITOLARE(CC) referenzia CONTO(CC)
- CLIENTE(CF) referenzia TITOLARE(CF)

Operazioni da eseguire:

a. INSERT nuovo conto in CONTO

b. INSERT nuovo cliente in CLIENTE

c. INSERT **(nuovo cliente, nuovo conto)** in TITOLARE

Supponiamo di eseguire **poi b.**

Esempio

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Vincoli:

- TITOLARE(CF) referenzia CLIENTE(CF)
- TITOLARE(CC) referenzia CONTO(CC)
- *CLIENTE(CF) referenzia TITOLARE(CF)*

Operazioni da eseguire:

a. INSERT nuovo conto in CONTO

b. INSERT nuovo cliente in CLIENTE

c. INSERT **(nuovo cliente, nuovo conto)** in TITOLARE

Supponiamo di eseguire **poi b.** Il DBMS verifica i vincoli: il cliente non è titolare di alcun conto corrente, quindi non è possibile inserirlo.

Esempio

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Vincoli:

- TITOLARE(CF) referenzia CLIENTE(CF)
- TITOLARE(CC) referenzia CONTO(CC)
- CLIENTE(CF) referenzia TITOLARE(CF)

Operazioni da eseguire:

a. INSERT nuovo conto in CONTO

b. INSERT nuovo **cliente** in CLIENTE

c. INSERT (nuovo cliente, nuovo conto) in TITOLARE

Supponiamo di eseguire invece **c.**

Esempio

CONTO(CC,NAgenzia,Saldo,…)

CLIENTE(CF,Cognome,…)

TITOLARE(CF,CC)

Vincoli:

- *TITOLARE(CF) referenzia CLIENTE(CF)*
- TITOLARE(CC) referenzia CONTO(CC)
- CLIENTE(CF) referenzia TITOLARE(CF)

Operazioni da eseguire:

a. INSERT nuovo conto in CONTO

b. INSERT nuovo **cliente** in CLIENTE

c. INSERT (nuovo cliente, nuovo conto) in TITOLARE

Supponiamo di eseguire invece **c.** Il DBMS verifica i vincoli: il CF del titolare deve essere presente nella tabella CLIENTE, quindi non è possibile inserirlo.

Problema e soluzione

I vincoli possono non consentire a **singole istruzioni** di essere eseguite.

Se invece si consente al DBMS di eseguire provvisoriamente l'istruzione di modifica e di aspettare l'arrivo di **successive istruzioni** che pongano rimedio alla situazione di violazione dei vincoli, alla fine lo stato del DBMS può essere comunque corretto (cioè soddisfa tutti i vincoli).

La necessità di transazioni

Queste considerazioni hanno portato i progettisti dei DBMS a disciplinare il rapporto tra le applicazioni.

Da qui l'introduzione delle transazioni.

Il concetto di transazione

Una transazione è una **unità di programma**

- Una transazione ha un **begin transaction** che comunica al DBMS la richiesta di interazione con esso da parte dell'applicazione
- Il DBMS identifica l'inizio della transazione T_i e la abbina in modo univoco con l'utente/applicazione che ne ha fatto richiesta
- Il DBMS, nell'ambito di T_i, riceve dei comandi DML in sequenza e li abbina alla transazione
- …

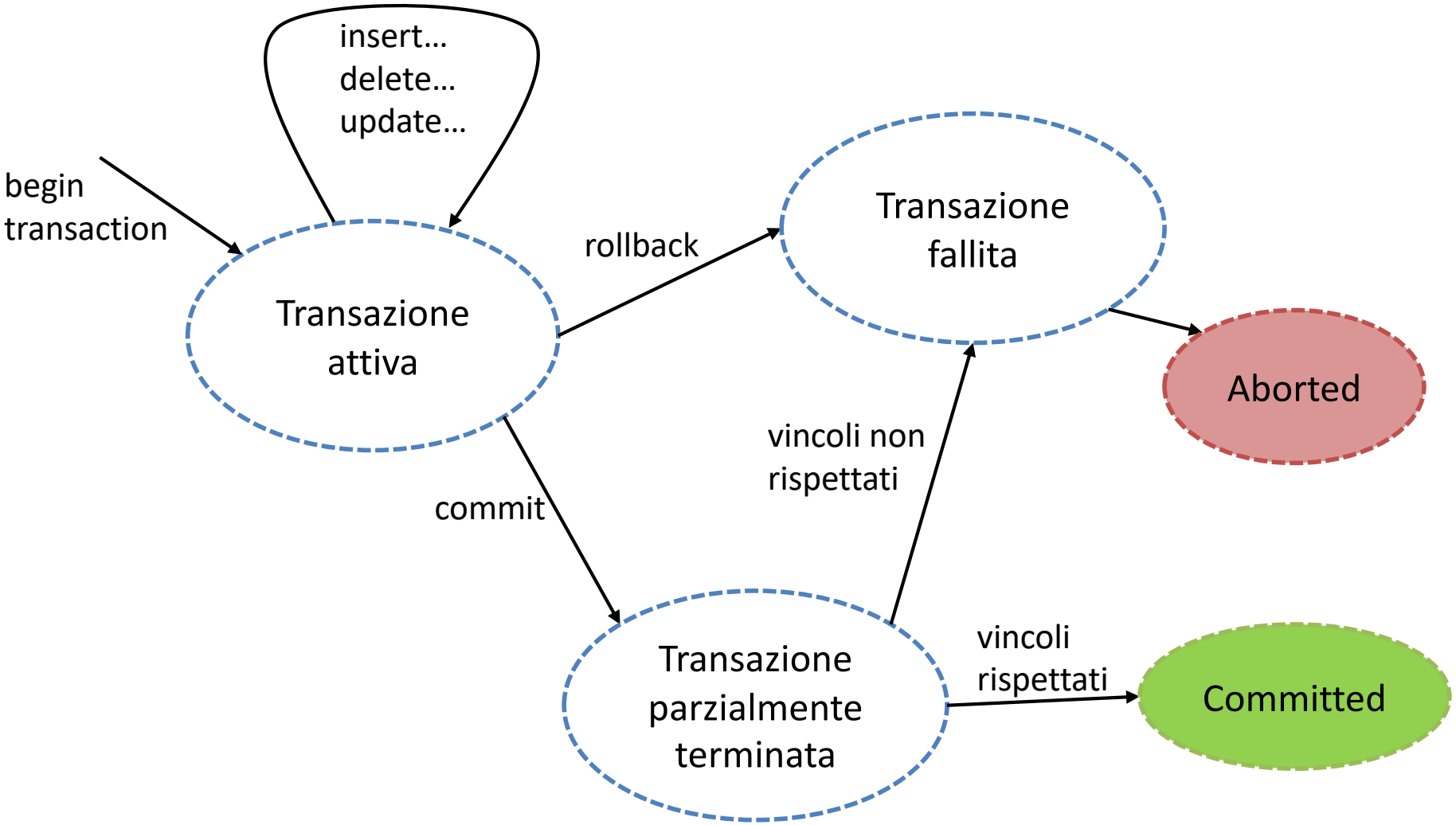
Il concetto di transazione

- …
- La terminazione dell'unità di programma è decisa dall'applicazione attraverso il comando:
  - **commit work**

  oppure
  - **rollback work**

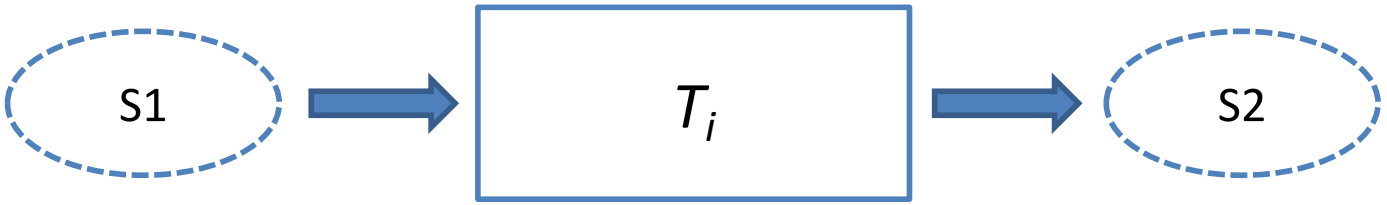
- Si tratta di due comandi standardizzati dall'SQL

Ciclo di vita di una transazione



Commit work

- Il comando **commit work** chiede al DBMS di rendere effettive tutte le modifiche inviate dall'inizio della transazione T_i sino al comando **commit work**
- Il DBMS non verifica i vincoli istruzione per istruzione, ma solo alla ricezione del **commit work**
- Se i vincoli sono verificati, il DBMS dà corso alla variazione di stato complessiva della base di dati



Fallimento del commit

- Se i vincoli non sono verificati, il DBMS annulla tutta la transazione, disfacendo le eventuali modifica apportate alla base di dati, e invia un segnale di errore all'applicazione

- Tutte le applicazioni ben ingegnerizzate devono sempre fare l'analisi degli errori inviati dal DBMS dopo la chiusura della transazione

Rollback work

- Dopo aver eseguito parte della transazione, l'applicazione può rendersi conto di non potere procedere alla computazione e chiedere all'applicazione di disfare tutti comandi fin lì eseguiti
- L'applicazione può cioè richiedere un **rollback**

- Esempio pratico: prenotazione voli in cui il posto viene riservato finché non si decide di confermare (commit) o annullare (rollback) la prenotazione

Buone pratiche

- Le transazioni, per loro natura, devono essere brevi
- Quindi un'applicazione ben fatta apre e chiude transazioni brevi (eventualmente anche in concorrenza)

Una transazione

```
start transaction;
update ContoCorrente
  set Saldo = Saldo + 10 where NumConto =
  12202;
update ContoCorrente
  set Saldo = Saldo - 10 where NumConto =
  42177;
commit work;
```

Una transazione con varie decisioni

```
start transaction;
update ContoCorrente
  set Saldo = Saldo + 10 where NumConto =
  12202;
update ContoCorrente
  set Saldo = Saldo - 10 where NumConto =
  42177;
select Saldo into A
  from ContoCorrente
  where NumConto = 42177;
if (A>=0)  then commit work
           else rollback work;
```

Proprietà delle transazioni

I DBMS gestiscono le transazioni garantendo, per ogni transazione T_i, il soddisfacimento delle proprietà ACID

Una transazione deve essere:

- **A**tomica
- **C**onsistente
- **I**solata
- **D**urabile (persistente)

Atomicità

- Una transazione o è eseguita **interamente** (e tutte le azioni, nessuna esclusa, sono eseguite) o non è eseguita **per niente** (nessuna delle azioni è eseguita)
- Non può lasciare la base di dati in uno stato intermedio
  - un guasto o un errore prima del commit devono causare l'annullamento (UNDO) delle operazioni svolte
  - un guasto o errore dopo il commit non deve avere conseguenze; se necessario vanno ripetute (REDO) le operazioni
- Esito
  - Commit = caso "normale" e più frequente (99%?)
  - Abort (o rollback)
    - richiesto dall'applicazione = suicidio
    - richiesto dal sistema (violazione dei vincoli, concorrenza, incertezza in caso di fallimento) = omicidio

Consistenza

Consistenza: l'esecuzione di una transazione non deve violare i vincoli della base di dati (integrità referenziale, unicità, applicativi, …)

Se la transazione opera con uno stato della base di dati **corretto** in **ingresso**, deve garantire la **correttezza** dello stato in **uscita**

È **responsabilità (dei programmatori) della transazione** mantenere la base di dati in uno stato corretto

Il DBMS interviene in parte **verificando i vincoli** (come minimo lo stato finale della base di dati deve verificare tutti i vincoli dichiarati)

Isolamento

- I DBMS ricevono azioni da transazioni diverse, provenienti da diverse applicazioni

- Tali azioni possono agire sulla stessa **identica tupla**
  - Esempio: azione contemporanea di cliente e operatore bancario sullo stesso conto corrente

- **Isolamento**: il DBMS garantisce che **l'esecuzione concorrente di transazioni** dia lo stesso risultato sulla base di dati che si avrebbe se ognuna venisse eseguita da sola

Durabilità (persistenza)

Tutte le modifiche andate a **buon fine** devono essere **persistenti**, anche in presenza di guasti