



# TRANSAZIONI

# CREDITS

- Paolo Ciaccia. Sistemi Informativi L-B  
Home Page del corso:
  - <http://www-db.deis.unibo.it/courses/SIL-B/>

# INTERAZIONE CON UNA BASE DI DATI

- Quando si interagisce con una base di dati, spesso è necessario eseguire più comandi SQL che corrispondono a una singola logica applicativa
- Vedremo in seguito come scrivere applicazioni per basi di dati arbitrariamente complesse utilizzando linguaggi general purpose, che interagiscono con il DBMS (connessione, esecuzione di comandi SQL) per l'esecuzione, attraverso opportune interfacce
- Per il momento: assumiamo di interagire con il sistema con sequenze di comandi SQL
- Quali garanzie sul risultato dell'esecuzione complessiva?

# ESEMPIO

- Trasferimento di una somma da un conto corrente ad un altro

```
UPDATE CC
```

```
SET Saldo = Saldo - 50
```

```
WHERE Conto = 123
```

```
UPDATE CC
```

```
SET Saldo = Saldo + 50
```

```
WHERE Conto = 235
```

- Che cosa succede se si verifica un problema dopo l'esecuzione della prima operazione e il sistema non è in grado di eseguire la seconda?

- Aggiornamento degli stipendi degli impiegati di una sede

```
UPDATE Imp
```

```
SET Stipendio = 1.1*Stipendio
```

```
WHERE Sede = 'S01'
```

- Che cosa succede se per qualche problema viene aggiornato lo stipendio solo di alcuni impiegati della sede considerata?

# CONCETTI DI BASE

- Una transazione è **un'unità logica di elaborazione** che corrisponde a **una serie di operazioni fisiche elementari** (letture/scritture) **sul DB**
- Esempi:
  - Trasferimento di una somma da un conto corrente ad un altro

```
UPDATE CC
```

```
SET Saldo = Saldo - 50
```

```
WHERE Conto = 123
```

```
UPDATE CC
```

```
SET Saldo = Saldo + 50
```

```
WHERE Conto = 235
```

- Aggiornamento degli stipendi degli impiegati di una sede

```
UPDATE Imp
```

```
SET Stipendio = 1.1*Stipendio
```

```
WHERE Sede = 'S01'
```

- In entrambi i casi **tutte le operazioni elementari devono essere eseguite**

# CONCETTI DI BASE

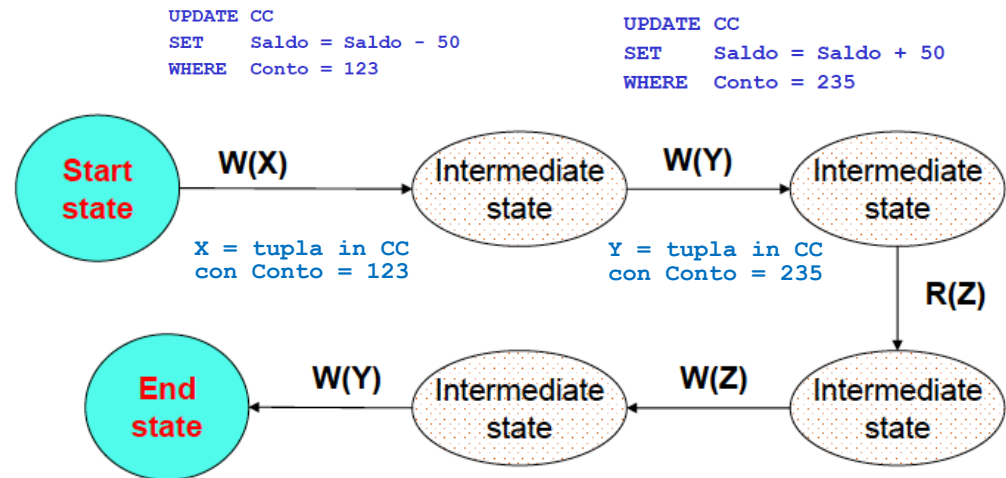
- Per avere garanzie su come l'applicazione viene eseguita e come i dati vengono acceduti e modificati, gli utenti interagiscono con la base di dati attraverso programmi applicativi ai quali viene dato il nome di **transazioni**
- **Transazione**: sequenza di operazioni di lettura e scrittura sulla base di dati a cui viene garantita un'esecuzione che soddisfa alcune buone proprietà
- *Esempi*:
  - Bonifico bancario
  - acquisto biglietto
  - prenotazione aerea
  - ...

# CONCETTI DI BASE

- Una transazione può essere vista come una sequenza di operazioni elementari di lettura (R) e scrittura (W) di oggetti (**tuple**) del DB che, a partire da uno stato iniziale consistente del DB, porta il DB in un nuovo stato finale consistente

```
UPDATE CC
SET     Saldo = Saldo - 50
WHERE   Conto = 123
```

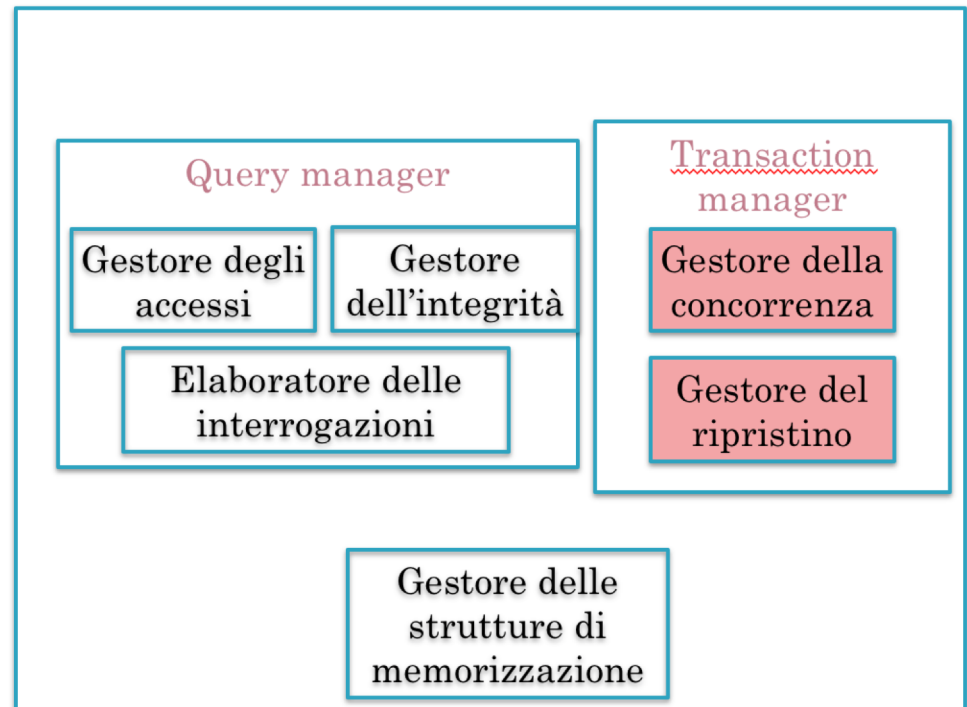
```
UPDATE CC
SET     Saldo = Saldo + 50
WHERE   Conto = 235
```



# CONCETTI DI BASE

- Il DBMS garantisce che le transazioni vengano eseguite nel rispetto delle **proprietà ACID**
  - **A**tomicità
  - **C**onsistenza
  - **I**solamento
  - **D**urabilità (persistenza)
- Le proprietà ACID sono garantite da specifiche componenti presenti del DBMS

DBMS





# CONCETTI DI BASE

- Il DBMS garantisce che le transazioni vengano eseguite nel rispetto delle **proprietà ACID**
- **Atomicity** = una transazione è **un'unità di elaborazione**
  - Il DBMS garantisce che la transazione venga eseguita come un tutt'uno
- **Consistency** = una transazione **lascia il DB in uno stato consistente**
  - Il DBMS garantisce che nessuno dei vincoli di integrità del DB venga violato
- **Isolation** = una transazione si **esegue indipendentemente dalle altre**
  - Se più transazioni eseguono in concorrenza, il DBMS garantisce che l'effetto netto è equivalente a quello di una qualche esecuzione sequenziale delle stesse
- **Durability** = gli **effetti** di una transazione che ha terminato correttamente la sua esecuzione devono essere **persistenti nel tempo**
  - Il DBMS deve proteggere il DB a fronte di guasti

# CONCETTI DI BASE

## Atomicità

- è detta anche proprietà tutto-o-niente
- tutte le operazioni di una transazione devono essere trattate come una singola unità: o vengono eseguite tutte, oppure non ne viene eseguita alcuna
- in caso di fallimento (abort) della transazione lo stato risultante della base di dati è quello precedente l'inizio della transazione
- *l'atomicità delle transazioni è assicurata dal gestore del ripristino (recovery)*

## Esempio

```
UPDATE CC  
SET      Saldo = Saldo - 50  
WHERE    Conto = 123
```

```
UPDATE CC  
SET      Saldo = Saldo + 50  
WHERE    Conto = 235
```

- è **atomica** se o effettua entrambe le operazioni (prelievo e versamento) o non ne effettua alcuna

# CONCETTI DI BASE

## Consistenza

- una transazione deve agire sulla base di dati in modo corretto
- eseguita su una base di dati in assenza di altre transazioni, la transazione trasforma la base di dati da uno stato consistente (cioè che soddisfa i vincolo di integrità e riflette lo stato reale del mondo che la base di dati deve modellare) ad un altro stato ancora consistente
- l'esecuzione di un insieme di transazioni corrette e concorrenti deve a sua volta mantenere consistente la base di dati
- il gestore della concorrenza (concurrency control) garantisce la consistenza (coadiuvato dal gestore dell'integrità)*

## Esempio

```
UPDATE CC
SET      Saldo = Saldo - 50
WHERE    Conto = 123
```

```
UPDATE CC
SET      Saldo = Saldo + 50
WHERE    Conto = 235
```

- è **consistente** se l'importo prelevato è lo stesso versato sul conto (quindi la somma dei saldi è sempre costante)

# CONCETTI DI BASE

## Isolamento

- ogni transazione deve sempre osservare una base di dati consistente e il suo esito non deve essere influenzato da esecuzioni concorrenti di altre transazioni
- una transazione non può, quindi, leggere risultati intermedi di altre transazioni, deve quindi essere **isolata**
- *la proprietà di isolamento è assicurata dal gestore della concorrenza che isola gli effetti di una transazione fino alla terminazione della stessa*

## Esempio

```
UPDATE CC
SET      Saldo = Saldo - 50
WHERE    Conto = 123
```

```
UPDATE CC
SET      Saldo = Saldo + 50
WHERE    Conto = 235
```

- è **isolata** se il programma transazione non vede gli effetti di altre transazioni che leggono e scrivono sul conto 123 e sul conto 235, se non sono ancora state concluse

# CONCETTI DI BASE

## Durabilità/persistenza

- i risultati di una transazione terminata con successo devono essere resi permanenti nella base di dati nonostante possibili malfunzionamenti del sistema
- *la persistenza è assicurata dal gestore del ripristino*
- tale sottosistema può inoltre fornire misure aggiuntive per garantire la persistenza anche a fronte di guasti ai dispositivi di memorizzazione

## Esempio

```
UPDATE CC
SET      Saldo = Saldo - 50
WHERE    Conto = 123
```

```
UPDATE CC
SET      Saldo = Saldo + 50
WHERE    Conto = 235
```

- è **duratura (persistente)** se, terminata la transazione, i conti correnti 123 e 235 riflettono il trasferimento effettuato

# TRANSAZIONI FLAT

- Esistono diversi tipi di transazioni
- Il tipo più semplice corrisponde alle **transazioni flat**
- Usate nei DBMS disponibili in commercio (pur con qualche estensione)
- Tecniche di implementazione e limitazioni sono ben note
- Transazioni semplici, di breve durata

# TRANSAZIONI FLAT

- Una transazione flat viene iniziata
  - **implicitamente** all'esecuzione del primo comando SQL o
  - **esplicitamente** con il comando **BEGIN [TRANSACTION]**  
o **START TRANSACTION**
- Può avere solo 2 esiti:
  - Terminare correttamente
  - Terminare non correttamente

# TRANSAZIONI FLAT – TERMINAZIONE CORRETTA

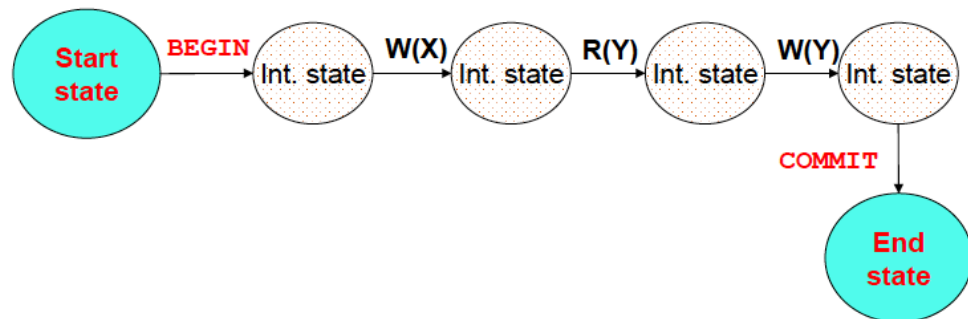
- Una transazione termina correttamente quando, dopo aver eseguito tutte le proprie operazioni, esegue una particolare istruzione SQL, detta **COMMIT [WORK]**, che comunica “ufficialmente” al Transaction Manager il termine delle operazioni

**BEGIN**

```
UPDATE CC
SET     Saldo = Saldo - 50
WHERE   Conto = 123
```

```
UPDATE CC
SET     Saldo = Saldo + 50
WHERE   Conto = 235
```

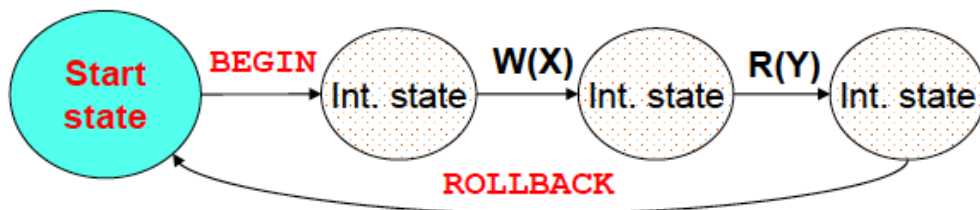
**COMMIT**





# TRANSAZIONI FLAT – TERMINAZIONE NON CORRETTA (ANTICIPATA)

- **Terminazione non corretta esplicita.**  
È la transazione che, per qualche motivo, decide che non ha senso continuare e quindi “abortisce” eseguendo l’istruzione SQL **ROLLBACK [WORK]**
- **Terminazione non corretta implicita.**  
È il sistema che non è in grado (ad es. per un **guasto** o per la **violazione di un vincolo**) di garantire la corretta prosecuzione della transazione, che viene quindi abortita



- Se per qualche motivo la transazione non può terminare correttamente la sua esecuzione il DBMS deve “disfare”(**UNDO**) le eventuali modifiche da essa apportate al DB

**BEGIN**

```
UPDATE CC
SET      Saldo = Saldo - 50
WHERE    Conto = 123
```

```
UPDATE CC
SET      Saldo = Saldo + 50
WHERE    Conto = 235
```

**ROLLBACK**

# TRANSAZIONI FLAT – TERMINAZIONE NON CORRETTA IMPLICITA PER GUASTO


- Si verifica un guasto e la transazione non può essere portata a termine

```
BEGIN

UPDATE CC
SET     Saldo = Saldo - 50
WHERE   Conto = 123

UPDATE CC
SET     Saldo = Saldo + 50
WHERE   Conto = 235

COMMIT
```



Guasto (ad es., cade il sistema)  
Il sistema esegue in autonomia **ROLLBACK** ed effettua **UNDO**, ritornando allo stato esistente prima dell'inizio della transazione

# TRANSAZIONI FLAT – TERMINAZIONE NON CORRETTA IMPLICITA PER VIOLAZIONE VINCOLO

- Un secondo caso di terminazione non corretta implicita si verifica nel caso di violazione di vincoli di integrità da parte della transazione
- In presenza di transazioni, la verifica dei vincoli di integrità (CHECK e asserzioni) può essere gestita in modo **flessibile**

# TRANSAZIONI FLAT E VINCOLI DI INTEGRITÀ

- Due diverse modalità di controllo che stabiliscono quando effettuare la valutazione del vincolo e ripristinare la consistenza:
  - **Valutazione immediata**: viene effettuata dopo ogni comando di manipolazione dei dati (**quindi anche tanti controlli all'interno della stessa transazione**)
  - **Valutazione differita**: il controllo viene effettuato al **termine dell'esecuzione della transazione**
- In entrambi i casi, in caso di violazione, la transazione viene abortita

# TRANSAZIONI FLAT E VINCOLI DI INTEGRITÀ

- Due momenti per effettuare la scelta della modalità di controllo
  - Al momento della dichiarazione del vincolo, è possibile specificare se la sua **valutazione**
    - **debba** essere immediata **NOT DEFERRABLE** (default) o
    - **possa** essere differita **DEFERRABLE**, in questo caso si può specificare una modalità di controllo iniziale per le transazioni:
      - **DEFERRABLE INITIALLY IMMEDIATE** (default): se non altrimenti specificato dalle transazioni, il vincolo verrà valutato in modo immediato
      - **DEFERRABLE INITIALLY DEFERRED**: se non altrimenti specificato dalle transazioni, il vincolo verrà valutato in modo differito
  - Nel codice di una transazione, per modificare la modalità di valutazione dei vincoli dichiarati DEFERRABLE  
**SET CONSTRAINTS {<lista nomi vincoli> | ALL}**  
**{DEFERRED | IMMEDIATE}**

# QUANDO È UTILE DICHIARARE VINCOLI DEFERRABLE?

- Nell'esempio precedente è sufficiente invertire l'ordine degli inserimenti per evitare violazioni di chiave esterno
- Ci sono situazioni in cui la modalità differita è l'unica possibile
- Esempio
  - Chiavi esterne cicliche
  - $\text{Madre}(\underline{\text{IdM}}, \text{NomeM}, \text{IdF}^{\text{Figlio}})$
  - $\text{Figlio}(\underline{\text{IdF}}, \text{NomeF}, \text{IdM}^{\text{Madre}})$

# ESEMPIO

## FK



BEGIN

INSERT INTO Video VALUES (4000, 'A  
Star is Born', 'Bradley Cooper',...);

INSERT INTO Film VALUES ('A Star is  
Born', 'Bradley Cooper',...);

COMMIT;

- FOREIGN KEY (titolo, regista)  
REFERECES Film NOT DEFERRABLE
  - Valutazione immediata → FK dopo primo INSERT è violata → ROLLBACK transazione
- FOREIGN KEY (titolo, regista)  
REFERECES Film DEFERRABLE  
INITIALLY DEFERRED
  - Valutazione differita → COMMIT transazione
- FOREIGN KEY (titolo, regista)  
REFERECES Film DEFERRABLE  
INITIALLY IMMEDIATE
  - Valutazione immediata → ROLLBACK transazione



# ESEMPIO

## FK



BEGIN

SET CONSTRAINTS ALL DEFERRED;

INSERT INTO Video VALUES (4000, 'A  
Star is Born', 'Bradley Cooper',...);

INSERT INTO Film VALUES ('A Star is  
Born', 'Bradley Cooper',...);

COMMIT;

- FOREIGN KEY (titolo, regista)  
REFERECES Film DEFERRABLE  
INITIALLY IMMEDIATE
  - La transazione ha reso il vincolo DEFERRED
  - Valutazione differita → COMMIT transazione





# ESEMPIO

## FK



BEGIN

SET CONSTRAINTS ALL IMMEDIATE;

INSERT INTO Video VALUES (4000, 'A Star is Born', 'Bradley Cooper',...);

INSERT INTO Film VALUES ('A Star is Born', 'Bradley Cooper',...);

COMMIT;

- FOREIGN KEY (titolo, regista) REFERENCES Film DEFERRABLE INITIALLY DEFERRED

- La transazione ha reso il vincolo IMMEDIATE
- Valutazione immediata → ROLLBACK transazione



# COSA È SUCCESSO FINORA?

- Nei laboratori svolti finora non abbiamo mai parlato di transazioni
- Abbiamo detto che una transazione implicitamente inizia all'esecuzione del primo comando SQL
- Ma può terminare implicitamente?
- La modalità di terminazione implicita è gestita da una proprietà chiamata **AUTOCOMMIT**
  - Se **AUTOCOMMIT = TRUE**, ogni comando costituisce implicitamente una transazione a sé stante (= eseguire **COMMIT** dopo ogni comando SQL)
  - Se **AUTOCOMMIT = FALSE**, tutti i comandi eseguiti all'interno di una sessione costituiscono implicitamente una transazione (= eseguire **COMMIT** dopo l'ultimo comando della sessione)