# SQL Transazioni

Prof. Francesco Accarino

IIS Altiero Spinelli Sesto San Giovanni

# Transazioni

Le Transazioni sono unità elementari di lavoro sulla base di dati di cui si vogliono garantire proprietà di *correttezza*, *robustezza* e *isolamento*

- I DBMS prevedono meccanismi per gestire la definizione e l'esecuzione di transazioni in ambienti concorrenti

- Sintatticamente una transazione `e contornata dai comandi **begin transaction e end transaction; all'interno possono** comparire i comandi di **commit work e rollback work.**

- Un comando di **commit trasferisce gli effetti della transazione** sulla base di dati

- Un comando di **rollback (*abort) annulla gli effetti della*** transazione e lascia inalterata la base di dati

# Transazioni: esempio

```
begin transaction
X := X - 10;
Y := Y + 10;
commit work;
end transaction
```

- Esegue il trasferimento di dieci unità da X a Y (p.es. conti correnti)

- Si vuole garantire che *vengano eseguite entrambe le azioni, o nessuna delle due*

# Transazioni ben formate

Una transazione si dice *ben formata se:*

– inizia con **begin transaction (bot);**
– termina con **end transaction (eot);**
– nel suo corso viene eseguito solo uno dei due comandi **commit work o rollback work.**

- In alcuni sistemi una coppia **begin transaction, end transaction viene automaticamente eseguita dopo ciascun** commit o abort.

- L'esecuzione risulta così automaticamente segmentata in una sequenza di transazioni ben formate.

# Proprietà ACIDE

Dall'acronimo inglese:

**ACID: Atomicity, Consistency, Isolation, Durability**

- Sono le proprietà di cui vogliamo le transazioni godano
- ***Atomicità: ciascuna transazione è un'unità indivisibile di*** esecuzione
- ***Consistenza: l'esecuzione di una transazione non deve*** violare l'integrità della base di dati
- ***Isolamento: il risultato dell'esecuzione di una transazione*** deve essere indipendente dall'esecuzione di altre transazioni
- ***Persistenza (Durability): gli effetti dell'esecuzione di una*** transazione andata in commit non devono essere persi

# Proprietà ACIDE e moduli del DBMS

L'architettura del DBMS è strutturata in modo da garantire il rispetto delle proprietà acide

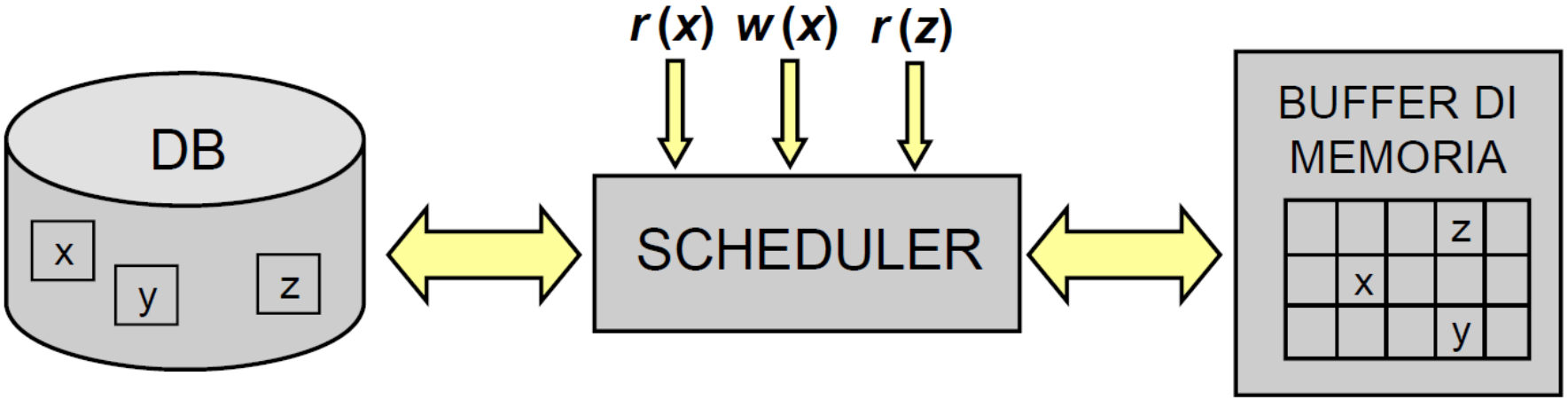
- L'*atomicità e la persistenza sono garantite dal controllo di affidabilità*

- L'*isolamento è garantito dal controllo della concorrenza, che* regola l'accesso ai dati da parte delle transazioni

- La *consistenza è garantita dai compilatori del DDL che* introducono opportuni controlli che vengono poi eseguiti run-tume dalle transazioni

- Prima dei DBMS esistevano comunque *sistemi transazionali* che supportavano l'esecuzione di transazioni

# Controllo della concorrenza




Le transazioni accedono ai dati *x, y, z tramite operazioni di* lettura *r(x) e scrittura w(x)*

• Ciascuna operazione comporta un trasferimento tra un *blocco* su disco e una *pagina di memoria centrale*

• Tutte le operazioni di lettura e scrittura,vengono gestite dallo *scheduler che determina se ciascuna richiesta può essere* soddisfatta o meno

# Transazioni concorrenti: esempio

- Supponiamo che in una base di dati bancaria vengano sottomesse due transazioni di bonifico:
  - **$T_1$ bonifico dal conto A al conto C**
  - **$T_2$ bonifico dal conto B al conto C**

- Le transazioni operano sulle tabelle:

**Bonifico(Cod_bo,Conto1,Conto2,Importo,Data)**
**Conto(Cod_cc,Cod_age,Saldo,Max_scoperto)**

# Sequenza di operazioni

- Ciascuna transazione tra l'istante di inizio e quello in cui chiede di andare in commit compie la stessa sequenza di accessi elementari alla base dati:

  1. **Lettura in Conto del Saldo del conto debitore**
  2. **Aggiornamento in Conto del Saldo del conto debitore**
  3. **Lettura in Conto del Saldo del conto creditore**
  4. **Aggiornamento in Conto del Saldo del conto creditore**
  5. **Inserimento in Bonifico di una nuova tupla**

- Se $T_1$ e $T_2$ vengono eseguite concorrentemente, e senza controllo,le loro azioni elementare si possono intercalare in un ordine qualsiasi con effetti nefasti

# Sequenza scorretta 1

1. $T_1$ legge il Saldo del conto A
2. $T_1$ aggiorna il Saldo del conto A
3. $T_2$ legge il Saldo del conto B
4. $T_2$ aggiorna il Saldo del conto B
5. $T_1$ legge il Saldo del conto C
6. $T_2$ legge il Saldo del conto C
7. $T_2$ aggiorna il Saldo del conto C
8. $T_1$ aggiorna il Saldo del conto C
9. $T_1$ inserisce in Bonifico una nuova tupla
10. $T_2$ inserisce in Bonifico una nuova tupla

# Sequenza scorretta 2

L'esecuzione complessiva è scorretta

- Sia **$T_1$ che $T_2$ leggono lo stesso valore per il saldo del** conto **C**
- Alla fine il conto **C risulterà incrementato del solo bonifico** effettuato da **$T_1$,, e non di entrambi come sarebbe corretto**
- In un DBMS il controllo della concorrenza avrebbe impedito il verificarsi di questa situazione
- **$T_1$ e $T_2$ avrebbero dovuto chiedere allo *scheduler il*** permesso di accedere ai dati della base di dati
- Lo scheduler può in presenza di conflitti porre una transazione in attesa