

## Descrizione del Problema

- Esistono numerose applicazioni in cui le informazioni contenute in un database sono elaborate da più programmi o più esecuzioni dello stesso programma.
- Un esempio è il sistema di prenotazione in cui in uno stesso momento più agenti possono vendere i biglietti e quindi modificare le liste dei passeggeri.
- Se non si disciplina accesso ai dati, può capitare ottenere degli effetti indesiderati. Ad esempio vendere lo stesso posto a due persone diverse.

## **Definizione di Transazione**

- Una **transazione** è una unità elementare di lavoro di un'applicazione a cui si vogliono associare particolari caratteristiche di correttezza, robustezza ed isolamento.
- Un sistema che mette a disposizione meccanismi per la definizione e l'esecuzione di transazioni viene detto **sistema transazionale**.  
Ad esempio, un sistema di prenotazione voli è un sistema transazionale in cui la prenotazione e cancellazione dei voli sono transazioni.

## Proprietà delle transazioni

- Atomicità: una transazione può comportarsi in due soli modi:
  - terminare con successo (**commit**, e modificare i dati a cui accede;
  - terminare con fallimento (**abort**), nel qual caso le informazioni del database non vengono modificati.
- Persistenza: l'effetto di una transazione terminata con commit non deve essere perso;
- Serializzabilità: impone una condizione di correttezza sull'esecuzione concorrente delle transazioni. In particolare se  $n$  transazioni  $T_1, \dots, T_n$  vengono eseguite concorrentemente allora l'effetto di tale esecuzione deve coincidere con almeno una delle  $n!$  possibili esecuzioni sequenziali delle transazioni  $T_1, \dots, T_n$ .
- Isolamento: le transazioni devono operare indipendentemente l'una dell'altra. In particolare non si vuole che abort di una transazione possa causare l'abort di altre transazioni.

## **Controllo della Concorrenza**

- L'obiettivo è di controllare l'esecuzione concorrente delle transazioni così da garantire le proprietà di serializzabilità e di isolamento.
- La tecnica del **locking** è uno dei meccanismi più noti per il controllo della concorrenza.

L'idea di base:

- ogni volta che una transazione accede ad un dato esso viene **bloccato** (lock);
- quando il dato è bloccato solo la transazione per cui esso è stato bloccato può operarvi;
- quando la transazione ha finito di operare sul dato il dato viene **sbloccato** (unlock);
- le richieste di accesso ai dati vengono gestite da una parte del sistema transazionale detto **gestore dei lock**.

## Tipi di Lock (1)

- Condiviso: una transazione richiede questo tipo di lock quando vuole leggere un dato ma non necessita di modificarlo. Un dato può essere bloccato in lettura da più transazioni contemporaneamente
- Esclusivo: una transazione richiede questo tipo di lock quando vuole leggere e scrivere un dato. Un dato può essere bloccato in scrittura solo da una transazione e nessun altro tipo di lock può essere richiesto.

## Tipi di Lock (2)

Il gestore del lock:

- soddisfa una richiesta di lock condiviso su un dato già bloccato in lettura;
- non soddisfa una richiesta di lock esclusivo su un dato già bloccato in lettura;
- non soddisfa nessuna richiesta di lock su un dato già bloccato in scrittura;

## Locking a Due Fasi

- Per garantire serializzabilità ed isolamento le transazioni devono richiedere e rilasciare i lock secondo un determinato protocollo. Il più noto di questi è il **locking a due fasi**.
- Il codice delle transazioni deve essere scritto in modo da rispettare i seguenti vincoli:
  - non devono essere eseguite istruzioni di lock dopo aver eseguito delle istruzioni di unlock;
  - le istruzioni di unlock devono essere eseguite dopo l'istruzione di commit, e di eventuale scrittura dei dati, o di abort.

## Deadlock (o Stallo)

Può capitare che due o più transazioni si blocchino mutuamente nel senso che ciascuna per procedere nell'elaborazione aspetta che l'altra abbia rilasciato un dato bloccato. In tal caso si dice che le transazioni sono in **Deadlock**.

Due tipi di soluzioni:

- Assegnare ad ogni transazione una scadenza entro la quale deve terminare.
- Utilizzare un algoritmo per determinare periodicamente situazioni di deadlock.