# Complementi di Basi di Dati

Ilaria Battiston

Anno scolastico 2018-2019

# Contents

# 1 Introduzione

I database non consistono solo in dati e tabelle: sono modelli centralizzati o distribuiti che permettono di gestire un carico anche significativo di utenti. La sicurezza è un aspetto importante, insieme all'affidabilità: i sistemi relazionali hanno grandi utilizzi nei settori bancari, e le operazioni devono giungere a termine senza guasti.

I DBMS sono quindi sistemi che devono garantire la gestione di dati di grandi dimensioni, persistenti, affidabili e condivisi. Oltre ai modelli relazionali esistono i NoSQL (Not Only SQL) e RDF, i quali hanno vantaggi e svantaggi a seconda dell'utilizzo.

Per garantire le precedenti caratteristiche, l'architettura di un DBMS deve avere una serie di funzionalità cooperanti, come un gestore delle transazioni, un query compiler e un gestore della memoria secondaria.

# 2 Strutture fisiche di accesso

La più piccola struttura di memorizzazione dei dati a cui gli utenti possano accedere è il file: così come le tabelle, ha un'intestazione fissa e un numero di righe.

I dati non devono solo essere memorizzati in modo persistente: devono anche essere facilmente recuperabili, e gli accessi da gestire sono sia in lettura che in scrittura. Un obiettivo fondamentale è la minnimizzazione del tempo di accesso e trasferimento da CPU alla memoria secondaria.

I dati sono memorizzati nei dischi magnetici, con blocchi da 4-32kbyte e un tempo di accesso di circa $10^{-8}$ millisecondi, con un tasso di trasferimento di 300 Mbit/secondo. Il problema è la grande differenza di ordini di grandezza tra queste operazioni.

L'organizzazione ottimale è un compromesso tra il tempo e lo spazio, di cui il tempo è la priorità considerando il costo ridotto della memoria.

## 2.1 Campi in SQL

Ogni campo in una tabella è memorizzato in una struttura fisica, ma lo spazio occupato cambia a seconda del tipo del dato. Le tuple sono pertanto record fisici organizzati in collezioni all'interno dei blocchi di memoria, e bisogna gestire anche modifica e cancellazione.

Esempi di tipi di dati:

- VARCHAR, che alloca $n+1$ bytes secondo un bit di carattere separatore o numero di caratteri;
- BLOB e GLOB, destinati a larghi file il cui spazio viene allocato solo al momento dell'inserimento.

I record possono avere formato e lunghezza fissi o variabili, con eventuali campi straordinari. Il record layout include informazioni supplementari con schemi o puntatori, lunghezza e timestamp di ultima lettura e scrittura.

I record sono organizzati in blocchi, unità di memoria trasferite dal disco alla memoria principale. La dimensione è generalmente fissa a $2^n$, di cui solitamente alcuni byte non vengono utilizzati.

Nel caso in cui la lunghezza sia variabile, lo header contiene anche la lunghezza del record e l'offset (distanza rispetto al byte iniziale) dei campi. I campi fissi sono allocati prima di quelli variabili.

Alcuni record sono rappresentati in più blocchi, cioè spanned: anche questa informazione è contenuta nell'header, e l'ordinamento è gestito tramite la contiguità fisica o collegamento tra record (modello a grafo, sistemi scalabili).

# 3 Strutture di files

I metodi di accesso sono algoritmi o moduli che forniscono primitive CRUD per ciascuna delle strutture.

Esempi di metodi di accesso sono:

- Strutture sequenziali;
- Strutture con accesso calcolato (hash);
- Strutture ad albero (indice).

L'accesso può essere sequenziale, binario o con indice. Il costo è determinato dallo spazio e dal tempo, ed è solitamente approssimato come il numero di blocchi acceduti. Per sceglere la struttura ottima è necessario distinguere se i file sono statici o dinamici e la frequenza delle operazioni CRUD.

## 3.1 Sequenziali

Le strutture sequenziali hanno in comune il mantenimento di un ordinamento fisico in memoria. Esistono organizzazioni per righe o per colonne.

**Non ordinata**: i nuovi record sono appesi in fondo al file nell'ordine in cui arrivano al DB, e l'ordine di visualizzazione è arbitrario. Il dato non è persistente finchè non viene scritto sul FS, quindi il blocco dev'essere copiato nel buffer e il record aggiunto.
La scansione è necessariamente sequenziale, quindi lettura e aggiornamento sono lente, e i record vengono periodicamente riorganizzata per sovrascrivere le cancellazioni.

**Ordinata**: la posizione fisica è determinata dal campo chiave, e l'ordinamento fisico è l'ordinamento delle chiavi.
La ricerca (anche per range) è semplice, ma la creazione impiega più tempo. La cancellazione dipende dallo schema di allocazione, ma richiede riorganizzazione periodica.

Alcuni metodi per rimediare al tempo di inserimento sono l'append in coda con riordinamento periodico, (accesso logaritmico) oppure l'allocazione al posto di record cancellati. Può essere lasciato spazio libero per uso futuro, o possono esserci file di overflow con linked list.

## 3.2 OLAP

OLAP (On Line Analytical Processing) è la raccolta di dati il cui uso primario è per le decisioni organizzative. Un data warehouse è un carico di lavoro subject-oriented, integrato, non volatile e variabile con il tempo.

Le informazioni sono rappresentate tramite un cubo dimensionale di cui gli assi indicano il tempo, i prodotti e i punti vendita. L'organizzazione è gerarchica, in categorie.

Non esiste un metodo generale per la modellazione di un DW: il modello ER dev'essere trasformato ed esteso, i dati devono essere permanenti e sono memorizzati in blocchi contenenti righe vicine tra loro.

I dati sono salvati colonna per colonna per evitare scansioni multiple in query come COUNT, ma l'inserimento viene effettuato comunque con la riga (strutture row-oriented) che poi viene spostata.

## 3.3 Hash

Le tabelle hash permettono l'accesso diretto sulla base del campo chiave, in casi in cui una struttura ad array sarebbe inefficiente perché i possibili valori della chiave sono molti di più di quelli utilizzati.

I record sono organizzati in bucket (blocchi), e la funzione hash, generalmente il modulo, associa a ogni valore della chiave un indirizzo. Lo spazio delle chiavi è più grande dello spazio degli indirizzi (funzione non iniettiva) e ci sono possibilità di collisioni.

Le collisioni vengono gestite tramite tabella di overflow o extensible hashing. La tabella di overflow funziona come bucket aggiuntivo per gli hash duplicati, e l'eventuale spostamento con la cancellazione dei record.

Questo comporta la decisione della grandezza dei bucket e dei file: bisogna tenere in considerazione il numero medio di accessi. La probabilità di overflow cresce con il numero di record e decresce con la dimensione del blocco.

Con $T$ records e $F$ record per bucket in media, il numero dei bucket $B$ dovrebbe essere $B = T/(0.8 \cdot F)$ per utilizzare il 50-80% dello spazio disponibile.

Le strutture hash sono efficienti per accesso diretto con il valore della chiave, ma non funzionano per ricerche basate su range o campi diversi. La dimensione dei file non deve subire variazioni significative nel tempo.

L'extensible hashing consente la crescita indefinita della struttura ed elimina l'overflow: essa aumenta dinamicamente con accesso tramite directory, la quale definisce la funzione hash.

Lo spreco di spazio è limitato, le riorganizzazioni sono solo a livello locale e il tempo di accesso è veloce. La directory va tenuta nella memoria principale.

## 3.4 Indicizzazione

Un indice è una struttura ausiliaria, il cui scopo è l'accesso efficiente ai record di un file sulla base dei valori di un campo o un insieme di campi (chiave o pseudochiave), non necessariamente identificanti.

Un indice è un altro file con record contenenti chiave e indirizzo, ordinato secondo i valori della chiave (indice analitico di un libro). Viene rappresentato come una struttura a cui è collegato il file principale.

### 3.4.1 Indice primario

L'indice primario è costituito da due campi, di cui il primo è dello stesso tipo della chiave di ordinamento del file dati e il secondo è un puntatore al blocco dei file dati. Generalmente il campo dev'essere anche chiave primaria, altrimenti l'indice si definisce di cluster.

Non è necessario che tutti i valori delle chiavi siano rappresentati, perciò lo spazio occupato è minimo. Questo concetto rende l'indice sparso (talvolta). Ogni file può avere al più un indice primario.

### 3.4.2 Indice secondario

L'indice secondario è definito su un campo chiave diverso da quello di ordinamento. I puntatori puntano ai record (o ai blocchi) che corrispondono al valore della chiave.

L'indice secondaria è per sua natura denso, cioè tutti i puntatori puntano necessariamente a tutti i record. Il numero di indici secondari è variabile.

### 3.4.3 Caratteristiche degli indici

La dimensione del file indice dipende dal numero di blocchi occupati. Si definiscono $L$ record nel file dati, blcchi di dimensione $B$, $R$ lunghezza dei record, $K$ lunghezza del campo chiave e $P$ lunghezza dei puntatori.

Si ha:
Numero di blocchi per file dati $\rightarrow N_F = \lceil L/(\lfloor B/R \rfloor) \rceil$
Numero di blocchi per un file di indice denso $\rightarrow N_D = \lceil L/(\lfloor B/(K+P) \rfloor) \rceil$
Numero di blocchi per un file di indice sparso $\rightarrow N_F = \lceil N_F/(\lfloor B/(K+P) \rfloor) \rceil$

Gli indici sono efficienti perché permettono accesso diretto efficiente sulla chiave, sia puntuale sia per intervalli. Questo significa che le operazioni CRUD sono lente, quindi sono soggetti a riorganizzazione periodiche o flag per le eliminazioni.

Una possibilità per evitare le ricerche tra blocchi diversi è la costruzione di indici sugli indici, quindi multilivello. Possono esistere livelli fino ad avere quello più alto con un solo blocco, ma solitamente sono pochi essendo i record piccoli e ordinati.
Numero di blocchi al livello $j$ dell'indice $\rightarrow N_j = \lceil N_{j-1}/(\lfloor B/(K+P) \rfloor) \rceil$

## 3.5 Alberi

Gli indici sono strutture ordinate sequenziali, quindi poco flessibili e scarsamente adattabili alla dinamicità e ai cambiamenti. Gli alberi rimediano a questo problema.

### 3.5.1 Alberi di ricerca di ordine P

Un esempio è l'albero di ricerca: ogni nodo contiene un numero fisso $F$ di chiavi e $F+1$ puntatori, con valori delle chiavi ordinate all'interno di un nodo. La rappresentazione sequenziale affianca puntatori e nodi.

Ogni chiave $K_j$ con $0 \leq j \leq F$ è seguita da un puntatore $P_j$ e preceduta dal puntatore $P_{j-1}$. Il primo puntatore è $P_0$. Ognuno di essi punta a un sottoalbero:

- $P_0$ punta al sottoalbero con valori di chiave $< K_1$;
- $P_F$ punta al sottoalbero con valori di chiave $>= F$;
- $P_j$ con $0 \leq j \leq F$ punta al sottoalbero con valori di chiave comprese nell'intervallo $K_j \leq K < K_{j+1}$.

Il valore $F + 1$ è il fan-out dell'albero, cioè il numero di input.

Ci sono due tipi di alberi utilizzati:

- Key-sequenced trees, con i record dati contenuti nelle foglie, tipici quando la chiave corrisponde al campo identificante;
- Indirect trees, con puntatori al record contenuti nelle foglie, qualsiasi altro meccanismo di allocazione delle tuple e tipici quando a un valore corrispondono più record (indice secondario, chiave non primaria).

Data la chiave $V$, la ricerca viene effettuata similmente alla ricerca binaria:

1. Se $V < K$ si procede da $P_0$;
2. Se $V \geq K_F$ si procede da $P_F$;
3. Altrimenti viene seguito il puntatore $P_j$ tale che $K_j \leq V \leq K_{j+1}$.

L'accesso è pertanto associativo per chiave, dove il numero di blocchi per accesso singolo rappresenta la profondità dell'albero. Questo tipo di struttura permette accesso efficiente sia per valore che per intervallo, quindi è la più comune nei DBMS.

### 3.5.2 Alberi bilanciati

Si ricorda che un albero bilanciato ha un numero variabile di foglie tale che la lunghezza dei cammini verso ognuna di esse sia la stessa. Ciò è ottenuto tramite operazioni di split e merge, e permette un tempo quasi costante (logaritmico) per gli accessi. Il numero di blocchi corrisponde alla profondità dell'albero.

Le strutture ad albero bilanciate si basano sull'utilizzo di un indice multilivello, struttura ausiliaria per l'accesso efficiente ai record tramite un campo chiave (non necessariamente identificante).

Ci sono due tipi di alberi bilanciati:

- B+ trees, con foglie concatenate secondo l'ordine della chiave. I valori possono essere ripetuti, e la ricerca è ottimizzata.
  Potenziale maggiore occupazione di memoria, e supporto efficiente di range queries;
- B trees, senza concatenamento sequenziale delle foglie. I nodi interni usano due puntatori per ogni valore $K_k$ della chiave, di cui uno punta al blocco che contiene il record e uno punta al sottoalbero con chiavi comprese tra $K_j$ e $K_{j+1}$.
  Inserimento e cancellazione sono preceduti da una ricerca fino a una foglia, e se non c'è spazio libero il nodo va diviso.

Minore occupazione di memoria, ma le ricerche relative a chiavi che puntano direttamente al record sono più efficienti.

L'occupazione di memoria, attraverso l'esecuzione delle operazioni di aggiornamento, è mantenuta tra il 50% e il 100%. L'efficienza è alta, perché le pagine ai primi livelli (accedute spesso) sono spesso nel buffer: ciò può comportare racing e deadlock, ma è poco probabile dato che le transazioni sono generalmente letture.

# 4 Architettura DBMS e MySQL

MySQL è un esempio di DBMS che utilizza le strutture fisiche e i metodi di accesso menzionati precedentemente. L'architettura è composta da elementi come query compiler, gestore delle transazoni, gestore del buffer e gestore dei metodi di accesso.

MySql Pluggable Storage Engine Architecture consente ai DBA di selezionare sistemi di storage diversi, di cui ognuno è specializzato per particolari applicazioni.

Storage engine disponibili:

1. MyISAM, motore default, utilizzato per applicazioni web;
2. InnoDB, implementa sistemi transazionali e garantisce alcune proprietà ACID;
3. Memory, memorizza i dati in memoria centrale per migliorare l'accesso;
4. Merge, supporta l'integrazione in formato MyISAM per considerare più tabelle come un oggetto;
5. Archive, per archivi di grandi dimensioni poco acceduti;
6. Federated, per dati distribuiti con un unico schema logico;
7. Cluster/NDB, per high performances e alta disponibilità.

MyISAM è stato il sistema di storage di default fino alla versione 3.2, non è transazionale: ogni database era una directory e ogni tabella un file. Gestisce record in formato fisso e dinamico (dati variabili), con meccanismo di accesso a matrice. Gli indici sono BTREE, RTREE e FULLTEXT, e la concorrenza è solo a livello di tabella.

InnoDB usa il concetto di tablespace per cui la struttura, la tabella e gli indici sono memorizzati insieme. Sono implementati gli indici BRTEE, e le interrogazioni più frequenti hanno tabelle di hash corrispondenti. Il controllo di concorrenza ha caratteristiche multi-versioning, low-level locking e foreign key constraints.

Memory non prevede memorizzazione in memoria persistente, solo centrale. Supporta indici hash e tree-based.

Il DBA può decidere quale storage engine usare per ogni tabella tramite CREATE TABLE. I vari storage differiscono per funzionalità e per velocità di accesso ai dati.

# 5 Esecuzione e ottimizzazione di query

L'esecuzione delle query, insieme all'ottimizzazione, sono operazioni di cui si occupano il compiler, i gestori delle interrogazioni, dei metodi d'accesso e del buffer. L'approccio può comportare compilazione e memorizzazione oppure esecuzione diretta dopo la compilazione.

Tramite statistiche, parsing e calcolo del piano logico e fisico, le tabelle logiche vengono trasformate in strutture fisiche con metodi di accesso alla memoria e vengono implementate su esse le operazioni algebriche.

Il primo step è il parsing: il data catalog effettua un'analisi lessicale, sintattica e semantica usando il dizionario. Le query testuali vengono tradotte in algebra relazionale e trasformate in un query tree. Le foglie corrispondono alle tabelle (strutture dati), e i nodi intermedi sono le operazioni algebriche di selezione, proiezione, join ecc.

Il query tree viene successivamente convertito in un query plan logico (sempre con algebra relazionale) che è ottimizzato rispetto al costo computazionale. A un'interrogazione possono corrispondere diverse espressioni, ma l'ordine delle clausole è rilevante per determinare la più efficiente tra quelle equivalenti.

Esempio: le istruzioni WHERE (selezioni) con operatori logici vengono sempre effettuate prima dei JOIN, essendo quest'ultima molto pesante (prodotto cartesiano), ed è meglio ridurre il numero di tuple coinvolte.

Regole di trasformazione di equivalenza:

1. Atomizzazione, una soluzione congiuntiva può essere sostituita da una sequenza di operazioni di selezione individuali;
2. Commutatività della selezione, interscambiabilità delle clausole;
3. Commutatività di selezione e prodotto cartesiano;
4. Commutatività di selezione e proiezione;
5. Commutatività di proiezione e JOIN;
6. Equivalenza di prodotto cartesiano e selezione con JOIN;
7. **Anticipazione della selezione rispetto al JOIN** (JOIN con tuple filtrate o meno);
8. Anticipazione della proiezione rispetto al JOIN, sotto opportune condizioni.

La trasformazione pertanto avviene seguendo queste proprietà, e stimando i costi delle operazioni fondamentali per diversi metodi di accesso. Il problema ha generalmente complessità esponenziale ma si ricorre alle metaeuristiche.

Le informazioni quantitative che vengono prese in considerazione sono per esempio la cardinalità delle relazioni, le dimensione di tuple e valori o il numero di valori distinti. Tutte queste sono memorizzate nel catalogo delle statistiche e aggiornate periodicamente.

## 5.1 Buffer management

L'implementazione degli operatori relazionali e i metodi di accesso dispongono di un numero (sufficiente) di blocchi o pagine nella memoria principale. Ogni pagina corrisponde a un blocco in memoria secondaria, ed è allocata e gestita nel buffer (non persistente).

Il buffer è organizzato in pagine, con dimensioni multiple rispetto ai blocchi nel file system. Si ha che il rapporto tra il tempo di accesso al FS e quello al buffer è $\geq 10^6$. Vale il principio di località e la valutazione della hit ratio.

Il buffer manager media tutte le richieste di lettura e scrittura dei blocchi. Gli indirizzi vengono tradotti, dal disco al buffer e viceversa, e quelli logici diventano fisici. Il numero di pagine disponibili è un parametro definito in fase di creazione del DB, detto buffer pool.

## 5.2 Valutazione dei costi

I costi si dividono in spazio e tempo. Lo spazio si indica come il costo di utilizzo della memoria principale; il costo di una query in termini di tempo di esecuzione, invece, viene valutato considerando:

1. Costi di accesso alla memoria secondaria, per acquisire i dati in memoria centrale e memorizzare i risultati;
2. Costi di memorizzazione di files intermedi;
3. Costi di elaborazione.

Il primo fattore è il più importante: si misura con il numero di blocchi caricati nel buffer durante il calcolo delle operazioni, trascurando i costi di memorizzazione (caricamenti dei blocchi su disco).

Il costo dominante, pertanto, è il numero di operazion di I/O, cioè il trasferimento di blocchi dal disco al buffer, che dev'essere minimizzato sfruttando al minimo le $M$ pagine disponibili.

## 5.3 Rappresentazione fisica delle tabelle

### 5.3.1 Metodi di accesso ai dati

La scansione realizza un accesso sequenziale a tutti i record, tramite:

- Proiezione su insieme di attributi, senza eliminazione dei duplicati;
- Selezione su un predicato semplice del tipo $Ai = v$;
- Inserimento, cancellazione e aggiornamento delle tuple accedute durante lo scan.

L'ordinamento è il metodo per ordinare i dati in memoria principale. I DBMS non possono caricare tutta la base di dati nel buffer, quindi ordinano separatamente e poi effettuano il merge con lo spazio disponibile.

L'accesso diretto può essere eseguito solo se le strutture fisiche (indici, hash) lo permettono.

### 5.3.2 Piano di esecuzione fisico

Un piano di esecuzione fisico descrive l'implementazione delle operazioni di algebra relazionale.

La selezione può avere predicati semplici, con congiunzione o disgiunzione di condizione (AND, OR). Quelle semplici vengono svolte utilizzando i metidi di accesso, o la ricarca binaria, ma gli altri casi hanno in più la valutabilità del predicato.

Per valutabilità si intende la selettività, e il coinvolgimento di attributi su cui è definito un indice, che permettono un accesso più efficiente. La selettività è definita come il rapporto tra il numero di record che soddisfano il predicato e il numero totale (relazione).

In altre parole, la selettività è la probabilità che un record soddisfi la condizione su un attributo, assumendo uniformità di distribuzione dei valori. Questo valore viene stimato dal DBMS in base a statistiche raccolte durante l'esecuzione di query.

L'attributo più selettivo è quello con probabilità minima, e il DBMS ordina i predicati in base alla loro selettività: in questo modo, il totale dei record su cui viene effettuato il controllo diminuisce drasticamente. Normalmente è sufficiente scegliere il primo predicato per avere una query ottimizzata.

Se qualche predicato non è valutabile durante le operazioni di disgiunzione, è necessario uno scan in cui su ogni n-pla di valuta tutta la selezione disgiuntiva. Altrimenti, si possono anche utilizzare gli indici con eliminazione dei duplicati.

I due metodi sono più o meno efficienti a seconda di come sono composti i record: quando l'interrogazione è poco selettiva conviene uno scan, perché l'eliminazione dei duplicati è inefficiente.

JOIN (binario) è un'altra operazione che necessita di estensiva ottimizzazione, essendo molto pesante computazionalmente e frequente. Esistono vari metodi per il calcolo:

- Nested-loop, per ogni record $r$ in $R$ e per ogni record $s$ in $S$ verificare se $r(A) = s(B)$. Conviene usare il file più piccolo per il ciclo esterno, per ottimizzare l'uso del buffer, con blocchi caricati $b = b_S + b_R \cdot \lceil b_S/(n-1) \rceil$ dove $n$ è la capienza del buffer;
- Single-loop, variante del nested-loop con una struttura di accesso diretto su $B$ di $S$, influenzato dalla percentuale di record uniti, dalla cardinalità e dal numero di livelli degli indici;
- Merge-scan, che assume le tabelle ordinate in base agli attributi di join e richiede lo scan lineare di ciascun file;
- Hash, con tabelle implementate mediante hash sugli attributi di join (confronto degli hash e unione blocco per blocco).

Il problema è sempre minimizzare il trasferimento di blocchi nel buffer, usando in modo combinato i metodi di accesso ai dati.

# 6 Tecnologie per DBMS

Le applicazioni vengono costantemente sviluppate e aggiornate, possono esserci imprevisti nella vita reale, ma i dati devono restare sempre atomici e consistenti (persistenti). Queste due proprietà sono assicurate e rispettate dai DBMS attraverso operazioni apposite e gestione dei fallimenti.

Una sequenza di esecuzioni di un insieme di transazioni è detta schedule, seriale se una transazione termini prima che la successiva inizi. L'ordine, pertanto, è importante: possono esserci risultati diversi anche se il sistema gestisce le esecuzioni nello stesso modo.

Proprietà di isolamento (schedule serializzabile): un insieme di transazioni eseguite concorrentemente produce lo stesso risultato che produrrebbe un'esecuzione sequenziale, quindi è come se ogni ogni transazione non avesse concorrenza. Questo processo è poco efficiente.

Dato uno schedule seriale con $n$ transazioni, ci sono $n!$ permutazioni possibili. Una transazione può essere rappresentata come una sequenza di parentesi begin/commit, azioni elementari di scrittura e lettura di un dato oppure con operazioni analoghe.

Per garantire l'isolamento è necessario garantire la serializzabilità di un insieme di transazioni. Ciò va a discapito del livello di concorrenza possibile, quindi ci vuole un giusto compromesso tra throughput e isolamento.

Il commit non viene eseguito finché non ci sono le condizioni necessarie per assicurare la corretta gestione della transazione. La concorrenza può causare diversi problemi, come i dati incoerenti in lettura o update inesistenti. Non tutti i DBMS gestiscono la concorrenza (es. MyISAM).

Lo scheduler ha il compito di garantire l'isolamento, assegnando identificatori univoci a ogni transazione e controllando se l'insieme può essere effettuato senza problemi.

Ci sono diversi modi di rappresentare il concetto di serializzabilità: oltre la garanzia dell'equivalenza dell'esecuzione, è necessario definire le classi di equivalenza e i rispettivi livelli. L'idea di base è trovare il maggior numero di schedule serializzabili all'interno di quelli seriali e verificabili in complessità ragionevole.

## 6.1 Algoritmi di controllo della concorrenza

Esistono diversi algoritmi per il controllo della concorrenza. Un tipo di controllo è basato sui conflitti: un conflitto è una situazione in cui scambiando l'ordine di due transazioni, l'output o il comportamento non è lo stesso.

Non tutte le transazioni avranno conflitti, quindi esse possono essere scambiate senza problemi. Anticipando alcune transazioni si ottiene una maggior concorrenza, per l'accesso a risorse diverse.

Un altro modo è tramite grafi di precedenza: questo deve tenere conto della lettura o scrittura, e in base a esse verificare che le scritture non siano in contemporanea e tutte le letture avvengano sulla stessa risorsa aggiornata.

Se esiste almeno un ciclo nel grafo, l'insieme di transazioni non rispetta la condizione di equivalenza e di conseguenza non è serializzabile. Questa verifica è effettuabile in tempo lineare, ma con molti nodi diventa comunque poco vantaggiosa.

La soluzione a ciò è l'uso dei lock: una transazione acquisisce il lock su una risorsa e lo rilascia quando ha completato la sua esecuzione, e ogni transazione ben formata è contenuta in una sezione critica.

Uno schedule si definisce legale se ogni coppia di lock-unlock è esclusiva, cioè nessun'altra transazione richiede la risorsa. Questa condizione non è sufficiente per assicurare l'isolamento, perché le ghost updates possono comunque avvenire.

Alcuni scheduler sono serializzabili senza conflitti, ma solo con 2PL: il sottoinsieme si riduce ulteriormente, perché talvolta il 2PL è troppo stretto per essere gestito facilmente. Il lock è granulare a livello di struttura logica o fisico (pagine intere).