Basi di Dati

Preparazione Appello - Slide + Approfondimenti

Ciao, questo è un riassunto che ho fatto PER ME STESSO quando ho dovuto studiare per l'esame, lo condivido sperando possa essere utile a qualcuno. Le informazioni al suo interno sono un misto tra quelle presenti nelle sue slide e quelle presenti nel libro; non mi assumo nessuna responsabilità riguardante la loro correttezza o precisione.

Basi di Dati **1**

Introduzione **5**
- Basi di dati e DBMS 5
  - Basi di dati 5
  - DBMS 5
- Modelli dei dati 6
  - Schemi e istanze 6
  - Architettura di un DBMS 6
- SQL 7
- Vantaggi e svantaggi dei DBMS 7

Il modello relazionale **7**
- Relazioni con attributi 8
- Relazioni e basi di dati 8
- Valori nulli 8
- Vincoli di integrità 8
  - Vincoli di chiave 9
  - Vincoli di integrità referenziale 9

Algebra relazionale **10**
- Operatori insiemistici 10
- Ridenominazione 10
- Selezione 10
- Proiezione 11
- Join 11
  - Join naturale 11
  - Join completi e incompleti 11
  - Outer join 12
  - Semijoin 12
  - Theta-join e equi-join 12
- Equivalenza di espressioni algebriche 12
- Viste 12

SQL **12**
- DDL 13

CREATE TABLE 13
Domini 13
Vincoli intrarelazionali 13
Vincoli interrelazionali 13
Politiche di reazione 13
DML 13
Interrogazioni 14
Operatori aggregati 14
Interrogazioni di tipo insiemistico 14
Interrogazioni nidificate 14
Aggiornamento 15
Inserimento 15
Cancellazione 15
Modifica 15

SQL: caratteristiche evolute 15
Vincoli di integrità generici 15
Asserzioni 15
Viste 15
Funzioni condizionali 17
Funzioni scalari 17
Controllo accessi 17

Metodologie e modelli per il progetto 18
Progettazione di basi di dati 18
Modello concettuale 19
Modello Entity-Relationship 19
Documentazione associata agli schemi concettuali 19

Progettazione concettuale 20
Analisi dei dati 20
Requisiti 20
Acquisizione e analisi 20
Pattern di progetto 20
Strategie di progetto 21
Qualità di uno schema concettuale 21
Una possibile metodologia 21

Progettazione logica 22
Ristrutturazione schema E-R 22
Analisi delle ridondanze 22
Eliminazione delle generalizzazioni 22
Partizionamento/accorpamento di entità e relazioni 22
Scelta degli identificatori primari 23
Traduzione verso il modello relazionale 23

Creazione e gestione degli indici 23

Metriche di valutazione degli indici 23
Indici primari 24
Indici secondari 24
Alberi binari di ricerca 25
B+ Tree 25
B- Tree 25
Indici Hash 25

Normalizzazione 26
Forme normali 26
Normalizzazione 27
Anomalie 27
Dipendenze funzionali 27
Forma normale di Boyce e Codd 27
Decomposizione in BCNF 27
Proprietà delle decomposizioni 28
Decomposizione senza perdita 28
Conservazione delle dipendenze 28
Terza forma normale 28
Decomposizione in terza forma normale 28
Teoria delle dipendenze e normalizzazione 28
Implicazione di dipendenze funzionali 29
Chiusura di un insieme di attributi 29
Coperture di dipendenze funzionali 29
Calcolo della copertura ridotta 30
Procedimento per ottenere schemi in terza forma normale 30

Transazioni 31
Proprietà 31
Atomicità 31
Consistenza 31
Isolamento 31
Persistenza 31
Controllo di affidabilità 32
Il log 32
Scrittura congiunta di log e base di dati 32
Modalità immediata 33
Modalità differita 33
Modalità mista 33
Checkpoint 33
Modello fail-stop 33
Processo di restart 34
Warm restart 34
Cold restart 34
Dump 34

Problemi e gestione della concorrenza 34
Gestione della concorrenza 35
Controllo di concorrenza 35
Equivalenza tra schedule 36
View-equivalenza 36
Conflict-equivalenza 36
Lock 36
Locking a due fasi (2PL) 37

Introduzione

Sistema informativo: Organizza e gestisce le informazioni necessarie per perseguire gli scopi dell'organizzazione.

Sistema informatico: Porzione automatizzata del sistema informativo con tecnologie informatiche.

Dato: Ciò che è immediatamente presente alla conoscenza, prima di ogni elaborazione; ha bisogno di essere interpretato per fornire informazione.

Informazione: Notizia, dato o elemento che consente di avere conoscenza più o meno esatta di fatti, situazioni, modi di essere; rappresentata per mezzo di dati.

Basi di dati e DBMS

Basi di dati

Insiemi organizzati di dati utilizzati per rappresentare le informazioni di interesse per un sistema informativo e per il supporto allo svolgimento delle attività di un ente, gestiti da un DBMS. Sono:

- **grandi**: possono avere dimensioni enormi, e i sistemi devono poter gestire i dati senza porre limiti alle dimensioni, a parte quelle fisiche dei dispositivi;
- **persistenti**: hanno un tempo di vita indipendente rispetto a quello delle singole esecuzioni dei programmi che le utilizzano;
- **condivise**: applicazioni e utenti diversi devono poter accedere a dati comuni; così facendo si riducono la ridondanza e le inconsistenze.

DBMS

Data Base Management System, sistema software in grado di gestire collezioni di dati che siano grandi, condivise e persistenti, assicurando:

- **affidabilità**: capacità del sistema di conservare sostanzialmente intatto il contenuto della base di dati (o almeno di permetterne la ricostruzione) in caso di malfunzionamenti software (e/o hardware); di questa caratteristica fa parte anche la gestione delle transazioni;
- **privatezza** dei dati: ciascun utente viene abilitato a svolgere solo determinate azioni sui dati, attraverso meccanismi di autorizzazione;
- **efficienza**: capacità di svolgere e operazioni utilizzando un insieme di risorse che sia accettabile per gli utenti;
- **efficacia**: capacità di rendere produttive le attività degli utenti.

Le quattro caratteristiche appena citate sono anche le principali che differenziano i DBMS dai file system, sopra i quali i DBMS sono costruiti.

Modelli dei dati

Un modello dei dati è un insieme di concetti utilizzati per organizzare i dati di interesse e descriverne la struttura in modo che essa risulti comprensibile a un elaboratore.
Esistono due tipi di modelli principali, e sono i modelli logici e i modelli concettuali. I **modelli logici** sono adottati nei DBMS esistenti per l'organizzazione dei dati e sono utilizzati dai programmi e indipendenti dalle strutture fisiche. Alcuni esempi di modelli logici sono i modelli relazionale, reticolare, gerarchico ecc. I **modelli concettuali** permettono invece di rappresentare i dati in maniera indipendente dalla scelta del modello logico, cercando di descrivere concetti del mondo reale; per questo motivo sono utilizzati nelle fasi preliminari della progettazione. Il modello concettuale più diffuso è il modello Entity-Relationship.

Il modello relazionale è attualmente il più diffuso modello di dati e permette di definire tipi per mezzo del costruttore relazione. Una relazione viene spesso rappresentata per mezzo di una tabella, le cui righe rappresentano record e le colonne corrispondono ai campi dei record.

Schemi e istanze

Nelle basi di dati esiste una parte sostanzialmente invariante nel tempo, detta **schema** della base di dati, costituita dalle caratteristiche dei dati, e una parte variabile nel tempo, detta **istanza** della base di dati, costituita dai valori effettivi. Lo schema di una relazione è costituito dalla sua intestazione, cioè dal nome della relazione seguito dai nomi dei suoi attributi (es. Docenza(Corso, NomeDocente)). L'istanza di una relazione, invece, è costituita dall'insieme, variante nel tempo, delle sue righe (che sono coerenti con io schema). Un esempio di istanza potrebbe essere:

Basi di dati	Rossi
Reti	Neri
Linguaggi	Verdi

Architettura di un DBMS

L'architettura considerata standard per DBMS si articola su tre livelli, per ciascuno dei quali esiste uno schema:

- **schema logico**: descrizione dell'intera base di dati per mezzo del modello logico;
- **schema interno**: rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione;
- **schema esterno**: descrizione di una porzione della base di dati di interesse, per mezzo del modello logico; utile per garantire privacy. Nei sistemi più moderni questo livello non è previsto; al suo posto è possibile definire delle viste.

Va ricordato che il livello logico è in ogni caso indipendente da quello fisico.

SQL

Un contributo all'efficacia dei DBMS è dato anche dalla disponibilità di vari linguaggi e interfacce, che possono presentarsi sotto forma di linguaggi testuali interattivi (SQL), comandi immersi in linguaggi ospite o anche interfacce amichevoli senza linguaggio testuale.
Tra i linguaggi testuali interattivi troviamo SQL, i cui comandi possono essere immersi in un linguaggio ospite (es. Java) o eseguiti attraverso interfacce grafiche (es. Postgres). SQL è un **Data Manipulation Language** (**DML**); è infatti utilizzato per creare query (es. SELECT) sui database, cioè per rendere possibile l'estrazione di informazioni dal database interrogando la base dei dati, nonché per l'aggiornamento di istanze di basi di dati. Oltre a ciò, ricopre anche il ruolo di **Data Definition Language** (**DDL**), cioè di linguaggio utile a definire gli schemi logici, esterni e fisici (es. CREATE TABLE) e le autorizzazioni per l'accesso. SQL è anche un linguaggio parzialmente dichiarativo.

Vantaggi e svantaggi dei DBMS

Tra i vantaggi dei DBMS troviamo:

- possibilità di considerare i dati come una risorsa comune e condivisa di una organizzazione;
- modello unificato e preciso della parte del mondo reale di interesse;
- gestione centralizzata dei dati con possibilità di standardizzazione ed "economia di scala";
- riduzione di ridondanze e inconsistenze;
- facilità di sviluppo e gestione di applicazioni.

Tuttavia, esistono anche delle situazioni nelle quali l'adozione di un DBMS può risultare sconveniente: applicazioni con uno o pochi utenti, senza necessità di accessi concorrenti e relativamente stabili nel tempo possono essere realizzate più proficuamente con file ordinari piuttosto che con DBMS.

Il modello relazionale

Prodotto cartesiano: In matematica, dati due insiemi D1 e D2, si chiama prodotto cartesiano di D1 e D2, in simboli D1 x D2, l'insieme delle coppie ordinate (v1, v2) tali che v1 è un elemento di D1 e v2 è un elemento di D2.

Relazione matematica: Una relazione matematica sugli insiemi D1 e D2 (chiamati domini della relazione) è un sottoinsieme di D1 x D2.

Grado: Il numero n delle componenti del prodotto cartesiano (e quindi di ogni n-upla) viene detto grado del prodotto cartesiano e della relazione.

Cardinalità: Il numero di elementi (cioè di n-uple) della relazione.

Relazione: Una relazione su X è un insieme di tuple su X.

Relazioni con attributi

Una relazione è un insieme, quindi:

- non è definito alcun ordinamento tra le n-uple;
- le n-uple di una relazione sono distinte l'una dall'altra: una tabella rappresenta una relazione solo in questo caso.

Nel caso in cui non siano presenti degli attributi, l'ordinamento interno delle n-uple è significativo ai fini dell'interpretazione dei dati; se invece attributi (che devono essere diversi tra loro) sono presenti, il loro ordine risulta irrilevante, e di conseguenza anche quello interno delle n-uple.

Relazioni e basi di dati

Una delle caratteristiche fondamentali del modello relazionale viene spesso indicata dicendo che esso è basato sui valori: i riferimenti tra i dati in relazioni diverse sono cioè rappresentati per mezzo di valori che compaiono nelle tuple e non, per esempio, per mezzo di puntatori.

Schema di relazione: costituito da un simbolo R, detto nome della relazione, e da un insieme di attributi X; il tutto si indica con R(X).

Schema di base di dati: insieme di schemi di relazione con nomi diversi.
Es: R = { Studenti (Matricola, Cognome, Nome, DataNascita),
Esami (Studente, Voto, Corso),
Corsi (Codice, Titolo, Docente) }

Istanza di relazione: (su uno schema R(X)) è un insieme r di tuple su X.

Istanza di base di dati: (su uno schema R) è un insieme di relazioni r dove ogni r_i è una relazione sullo schema $R_i(X_i)$.

Valori nulli

Per rappresentare in modo semplice l'eventuale non disponibilità di valori si usa un valore speciale, detto valore nullo, che denota l'assenza di informazione ma è un valore aggiuntivo rispetto a quelli del dominio e ben distinto da essi. È rappresentato dalla dicitura NULL ed è possibile imporre restrizioni sulla sua presenza. A seconda dei casi può rappresentare diversi tipi di valori (indistinguibili per il DBMS): valore sconosciuto, valore inesistente o valore senza informazione.

Vincoli di integrità

Per vincolo di integrità si intende una proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione; un vincolo è anche definibile come una funzione booleana, che per ogni istanza restituisce un valore vero o falso. I DBMS più usati generalmente supportano in modo nativo la presenza di vincoli, ma in caso contrario sarà l'applicazione in uso a dover garantire il non inserimento di dati non conformi.

Vari tipi di vincoli sono distinguibili in base agli elementi che coinvolgono:

- **vincoli intrarelazionali**, cioè vincoli che fanno riferimento a una singola relazione (tabella); fanno parte di questa categoria i:
  - **vincoli di tupla**: possono essere valutati su ciascuna tupla e si possono esprimere attraverso espressioni booleane;
  - **vincoli di dominio**: impongono restrizioni sul dominio dell'attributo; si possono definire come casi particolari di vincoli di tupla su un solo attributo;
  - **vincoli di chiave**.
- **vincoli interrelazionali**, cioè vincoli che coinvolgono più relazioni.
  - **vincoli di integrità referenziale**.

Vincoli di chiave

Chiave: Insieme di attributi utilizzato per identificare univocamente le tuple di una relazione. In particolare, un insieme di attributi è chiave di una relazione se è una sua superchiave minimale. Ogni relazione ha una chiave. Una chiave identifica una tupla di una relazione, e può quindi essere usata per essere referenziata da un'altra tabella.

Superchiave: Un insieme di attributi è superchiave di una relazione se questa non contiene due tuple distinte con lo stesso valore. Una superchiave identifica le tuple di una relazione.

Superchiave minimale: Se all'interno della superchiave (insieme di attributi) non sono presenti sottoinsiemi che possono essere superchiavi (es. superchiave formata da un solo attributo) ci troviamo di fronte a una superchiave minimale.

Chiave primaria: Chiave su cui si vieta la presenza di valori nulli; esattamente una per relazione.

Vincoli di integrità referenziale

Un vincolo di integrità referenziale fra un insieme di attributi X di una relazione R_1 e un'altra relazione R_2 è soddisfatto se i valori su X di ciascuna tupla dell'istanza di R_1 compaiono come valori della chiave (primaria) dell'istanza di R_2.
In altre parole, si ha un vincolo di integrità referenziale quando informazioni in relazioni diverse sono correlate attraverso valori comuni. I vincoli di integrità referenziale sono inoltre fondamentali per il concetto di "modello basato sui valori".
Si possono infine definire delle cosiddette **politiche di reazione**, cioè azioni che vengono intraprese in caso di violazione dei vincoli di integrità referenziale (CASCADE, SET NULL ecc.).

Algebra relazionale

Operatori insiemistici

L'algebra relazionale è un linguaggio procedurale costituito da un insieme di operatori che agiscono su e producono relazioni e possono essere composti. Nel contesto delle relazioni, si possono applicare tre operatori insiemistici:

- l'**unione** di due relazioni r_1 e r_2 definite sullo stesso insieme di attributi X è indicata con $r_1 \cup r_2$ ed è una relazione ancora su X contenente le tuple che appartengono a r_1 oppure a r_2, oppure a entrambe;
- l'**intersezione** di $r_1(X)$ e $r_2(X)$ è indicata con $r_1 \cap r_2$ ed è una relazione su X contenente le tuple che appartengono sia a r_1 sia a r_2;
- la **differenza** di $r_1(X)$ e $r_2(X)$ è indicata con $r_1 - r_2$ ed è una relazione su X contenente le tuple che appartengono a r_1 e non appartengono a r_2.

Laureati

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Specialisti

Matricola	Nome	Età
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Laureati ∪ Specialisti

Matricola	Nome	Età
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33

Laureati ∩ Specialisti

Matricola	Nome	Età
7432	Neri	54
9824	Verdi	45

Laureati – Specialisti

Matricola	Nome	Età
7274	Rossi	42

Ridenominazione

Un'altra operazione è inoltre possibile quando si utilizza uno degli operatori appena visti, e consiste nell'adeguare i nomi degli attributi a seconda delle necessità. Questa operazione è detta ridenominazione, perchè appunto cambia il nome degli attributi lasciando inalterato il contenuto delle relazioni. Le operazioni di ridenominazione si indicano col simbolo ρ.

Selezione

L'operazione di selezione, che si indica col simbolo σ, produce un sottoinsieme delle tuple su tutti gli attributi; si dice quindi che la selezione genera "decomposizioni orizzontali". Per effettuare una selezione si aggiunge una condizione di selezione, spesso espressa sotto forma di formula proposizionale. Possiamo quindi dire che la selezione $\sigma_F(r)$, in cui r è una relazione e F una formula proposizionale, produce una relazione sugli stessi attributi di r che contiene le tuple di r su cui F è vera.

Proiezione

L'operazione di proiezione si indica invece col simbolo π e produce "decomposizioni verticali", in quanto restituisce un risultato cui contribuiscono tutte le tuple, ma su un sottoinsieme degli attributi. Inoltre, possiamo dire che il risultato di una proiezione contiene al più tante tuple quante l'operando, ma può contenerne di meno perché i contributi uguali "collassano" in una sola tupla. In particolare, se X è una superchiave di R, allora $\pi_X(R)$ contiene esattamente tante tuple quante R.

Join

Il join è un operatore, indicato col simbolo ⋈, che permette di correlare dati contenuti in relazioni diverse, confrontando i valori contenuti in esse. Esiste in diverse versioni.

Join naturale

Il join naturale è un operatore che correla dati in relazioni diverse, sulla base di valori uguali in attributi con lo stesso nome. Potremmo definirlo come il join "standard", e dire che nel caso in cui si applichi a due relazioni senza attributi in comune coincide col prodotto cartesiano. Il join naturale può essere simulato per mezzo di ridenominazione, equi-join e proiezione.

Join completi e incompleti

Un join si dice completo nel caso in cui ciascuna tupla di ciascuno degli operandi contribuisca ad almeno una tupla del risultato; in caso contrario si parla di **join incompleto**. Se invece nessuna tupla dovesse essere combinabile, si otterrebbe una relazione vuota come risultato del join.
Per quanto riguarda la cardinalità nei join, possiamo dire che, considerando le relazioni $R_1(A,B)$ e $R_2(B,C)$:

- in generale vale $0 \leq |R_1 \bowtie R_2| \leq |R_1| \times |R_2|$;
- se B è chiave in R_2 vale $0 \leq |R_1 \bowtie R_2| \leq |R_1|$;
- se B è chiave in R_2 ed esiste vincolo di integrità referenziale tra B in R_1 e R_2 vale $|R_1 \bowtie R_2| = |R_1|$.

in generale	$0 \leq NumTuple \leq \|R_1\| \times \|R_2\|$
join completo	$NumTuple \geq Max(\|R_1\|, \|R_2\|)$
$X_1 \cap X_2$ contiene una chiave per r_2 (B)	$0 \leq NumTuple \leq \|R_1\|$
$X_1 \cap X_2$ coincide con una chiave per r_2 (B) e integrità referenziale con R1	$NumTuple = \|R_1\|$

Outer join

Questo tipo di join prevede che tutte le tuple diano un contributo al risultato, eventualmente estese con valori nulli ove non vi siano controparti opportune. Esiste in 3 varianti: **join esterno sinistro**, che estende solo le tuple del primo operando, **join esterno destro**, che estende solo le tuple del secondo operando, e **join esterno completo**, che le estende tutte.

Semijoin

Operatore che restituisce le tuple di una relazione che partecipano al join naturale di tale relazione con un'altra. Un semijoin si può ottenere anche attraverso un join e una proiezione.

Theta-join e equi-join

Il theta-join è definibile come un prodotto cartesiano seguito da una selezione. Nel caso in cui l'operatore di confronto usato nella condizione della selezione sia sempre quello di uguaglianza, si parla di **equi-join**.

Equivalenza di espressioni algebriche

L'algebra relazionale permette di formulare espressioni equivalenti tra loro, cioè che producono lo stesso risultato: in presenza di alternative equivalenti, però, viene scelta quella con costo minore. In particolare, i DBMS non eseguono davvero le interrogazioni come vengono formulate, ma usano regole di equivalenza per ottimizzarle e ridurre così la computazione.

Viste

Le **relazioni derivate** rappresentano un modo per creare rappresentazioni diverse degli stessi dati; il loro contenuto è infatti funzione del contenuto di altre relazioni. Esistono principalmente due tipi di relazioni derivate:

- **viste materializzate**, cioè relazioni effettivamente memorizzate nella base di dati;
- **relazioni virtuali** (**viste**), cioè relazioni definite per mezzo di funzioni, non memorizzate nella base di dati, ma utilizzabili nelle interrogazioni come se lo fossero.

Entrambe queste tipologie di relazioni derivate presentano vantaggi e svantaggi. Le viste materializzate, per esempio, hanno tra i vantaggi principali il fatto di essere immediatamente disponibili per le interrogazioni, ma hanno gli svantaggi di essere ridondanti, di appesantire gli aggiornamenti e di essere raramente supportate dai DBMS. Le viste invece sono supportate da tutti i DBMS, ma non sono immediatamente disponibili in quanto devono essere ricalcolate per ogni interrogazione; nonostante ciò, però, presentano comunque numerosi vantaggi:

- un utente può voler vedere o essere autorizzato a vedere solo una porzione della base di dati;
- è possibile semplificare la scrittura di espressioni complesse.

Gli aggiornamenti sulle viste sono fortemente limitati e possibili solo in pochi casi (es. quando il join è completo).

SQL

SQL (Structured Query Language) è un linguaggio con varie funzionalità, raggruppabili principalmente in due categorie: **Data Definition Language** (**DDL**) e **Data Manipulation Language** (**DML**).

DDL

CREATE TABLE

Istruzione SQL che definisce uno schema di relazione e ne crea un'istanza vuota; specifica anche attributi, domini e vincoli.

Domini

Si possono dividere in **domini elementari** e **domini definiti dall'utente**. I primi sono domini predefiniti e in pratica specificano i tipi degli attributi (es. char(lung), integer, date ecc.), i secondi si possono specificare attraverso l'istruzione **CREATE DOMAIN**, decidendo per esempio un massimo o un minimo per i valori di un certo attributo.

Vincoli intrarelazionali

Le diverse tipologie di vincoli intrarelazionali si possono specificare attraverso le seguenti istruzioni:

- **NOT NULL**: indica che l'attributo non può contenere valori nulli;
- **UNIQUE**: indica che l'attributo è una chiave;
- **PRIMARY KEY**: indica che l'attributo è chiave primaria;
- **CHECK**: specifica vincoli generici.

Vincoli interrelazionali

I vincoli di integrità referenziale (cioè i vincoli interrelazionali più diffusi) si specificano attraverso le istruzioni **REFERENCES** (nel caso di singoli attributi) e **FOREIGN KEY** (nel caso di attributi multipli).

Politiche di reazione

Le politiche di reazione sono azioni da intraprendere in caso di violazione dei vincoli di integrità referenziale; sia in caso di istruzione DELETE che in caso di istruzione UPDATE le politiche di reazione possibili sono le seguenti:

- **CASCADE**: si propagano le cancellazioni/aggiornamenti;
- **SET NULL**: all'attributo referente viene assegnato il valore nullo al posto del valore cancellato/modificato nella tabella;
- **SET DEFAULT**: all'attributo referente viene assegnato il valore di default al posto del valore cancellato/modificato nella tabella;
- **NO ACTION**: l'azione di cancellazione/modifica non viene consentita.

DML

Interrogazioni

SELECT è l'istruzione usata per le interrogazione, e nella sua versione base è composta nel seguente modo:

SELECT *ListaAttributi*
FROM *ListaTabelle*
[**WHERE** *Condizione*]

In algebra relazionale ciò corrisponde a operazioni di selezione e proiezione, che è possibile fare anche singolarmente. In questa istruzione è anche possibile definire degli alias per gli attributi; all'interno della clausola WHERE, inoltre si può usare la parola **LIKE** per specificare solo alcune caratteristiche della sequenza di caratteri da confrontare con i valori trovati nella selezione. È anche possibile specificare la presenza o meno di valori nulli attraverso **IS [NOT] NULL**; inoltre, dato che SELECT di per sé non rimuove eventuali duplicati, si può usare a questo proposito l'istruzione **DISTINCT**. È fondamentale ricordare anche che nella clausola FROM, se necessario, è possibile effettuare **JOIN** tra più tabelle, specificando anche eventuali attributi attraverso la parola **ON**. Si può anche specificare un ordinamento per i risultati ottenuti attraverso **ORDER BY**.

Operatori aggregati

Nelle espressioni della target list (SELECT) possiamo avere anche espressioni che calcolano valori a partire da insiemi di tuple; le principali espressioni di questo tipo sono le seguenti:

- **COUNT**;
- **MIN**;
- **MAX**;
- **AVG**;
- **SUM**.

Tutti questi operatori aggregati ignorano i valori null.
Una clausola che viene spesso affiancata agli operatori aggregati è **GROUP BY**, utile a specificare come dividere le tabelle in sottoinsiemi (gruppi per min, max ecc.); sui sottoinsiemi ottenuti è poi possibile applicare delle condizioni attraverso la clausola **HAVING**.

Interrogazioni di tipo insiemistico

Per effettuare unione, differenza ed intersezione non basta la clausola select, ma ci vogliono rispettivamente **UNION**, **EXCEPT** e **INTERSECT**; quest'ultima in particolare non è molto usata in quanto gli stessi risultati si possono ottenere con interrogazioni normali.

Interrogazioni nidificate

Si possono anche nidificare interrogazioni all'interno di altre, per esempio nella clausola WHERE. Nel contesto delle interrogazioni nidificate si possono utilizzare anche gli operatori **[NOT] IN**, **ANY** e **[NOT] EXISTS**, per verificare rispettivamente se il valore specificato nel WHERE [non] è presente nel risultato della query nidificata, se il valore specificato è uguale/maggiore/minore di uno qualsiasi dei risultati della query nidificata o per eseguire

l'interrogazione nidificata una volta per ciascuna tupla restituita dall'interrogazione esterna, controllando se il valore specificato [non] esiste nei risultati della query nidificata.

Aggiornamento

Per quanto riguarda l'aggiornamento dei dati in SQL, le principali operazioni che si possono effettuare sono 3 e sono inserimento, eliminazione e modifica.

Inserimento

Attraverso il comando **INSERT INTO** è possibile specificare i valori da inserire in una tabella sia in modo esplicito ("ex novo") attraverso **VALUES** sia estraendo il contenuto da altre parti della base di dati, attraverso una interrogazione SELECT. Nell'inserimento l'ordine degli attributi è significativo, è possibile omettere la lista degli attributi per fare riferimento a tutti gli attributi della relazione e se la lista degli attributi non contiene tutti gli attributi della relazione, per quelli mancanti viene inserito un valore nullo o di default.

Cancellazione

Il comando **DELETE** elimina righe, che soddisfano una eventuale condizione specificata con WHERE, dalle tabelle della base di dati.

Modifica

Attraverso il comando **UPDATE** si possono invece aggiornare uno o più attributi delle righe di una tabella che soddisfano l'eventuale condizione specificata anche in questo caso con WHERE. Il valore aggiornato può essere:

- il risultato di un'espressione sugli attributi della tabella;
- il risultato di un'interrogazione SQL;
- il valore nullo;
- il valore di default per il dominio.

SQL: caratteristiche evolute

Vincoli di integrità generici

In SQL, attraverso la sintassi **CHECK (Condizione)** è possibile specificare vincoli di tupla o fra tuple della stessa o di diverse relazioni.

Asserzioni

Sempre attraverso la clausola **CHECK** è possibile definire delle asserzioni, cioè dei vincoli a livello di schema. Un esempio di asserzione potrebbe per esempio imporre che in una tabella sia sempre presente almeno una riga.

Viste

Le viste sono tabelle "virtuali" il cui contenuto dipende dal contenuto delle altre tabelle di una base di dati. In particolare, una vista è rappresentata da una query (SELECT) il cui risultato può essere utilizzato come se fosse una tabella. L'utilizzo principale che si fa delle viste è volto a semplificare query complesse; le viste possono inoltre essere aggiornate con INSERT, UPDATE e DELETE. Gli aggiornamenti sono però generalmente ammessi solo su viste definite su una sola relazione, e possono prevedere l'imposizione di alcune verifiche con la clausola **CHECK OPTION**: le modifiche sono permesse solo a condizione che la tupla continui ad appartenere alla vista.

Funzioni condizionali

COALESCE: Funzione che ammette come argomento una sequenza di espressioni e restituisce il primo valore non nullo; può quindi essere usata per convertire valori nulli in valori definiti dal programmatore.

NULLIF: Funzione che richiede come argomento un'espressione e un valore costante; se l'espressione è pari al valore costante, la funzione restituisce il valore nullo, altrimenti restituisce il valore dell'espressione.

CASE: Funzione che permette di specificare strutture condizionali, il cui risultato dipende dalla valutazione del contenuto delle tabelle (simile a Switch-Case).

Funzioni scalari

Le funzioni scalari sono funzioni a livello di attributi di tuple che restituiscono singoli valori. Possono essere per esempio temporali (es. current_date, extract(), ecc.), volte alla manipolazione di stringhe (es. char_length, lower, ecc.) o utili ad effettuare delle conversioni (es. cast)

Controllo accessi

In SQL è possibile specificare chi (utente), come (lettura/scrittura) e dove (tabelle, viste, attributi) può utilizzare una base di dati o parte di essa, ricordando però che il creatore di una risorsa ha tutti i privilegi su di essa.
Un privilegio è caratterizzato da:

- la risorsa cui si riferisce;
- l'utente che concede il privilegio;
- l'utente che riceve il privilegio;
- l'azione che viene permessa;
- la trasmissibilità del privilegio.

I privilegi che si possono concedere in SQL riguardano INSERT, UPDATE, DELETE, SELECT, REFERENCES, USAGE.
Il controllo degli accessi e la gestione delle autorizzazioni servono a nascondere gli elementi cui un utente non può accedere, senza sospetti; per esempio, l'utente riceve lo stesso messaggio sia nel caso in cui una tabella non esista sia nel caso in cui una tabella esista ma non è autorizzato a vederlo. Inoltre, un modo per autorizzare un utente a vedere solo alcune tuple di una relazione potrebbe comprendere l'utilizzo di una vista, sulla quale applicheremo le autorizzazioni che ci interessano.

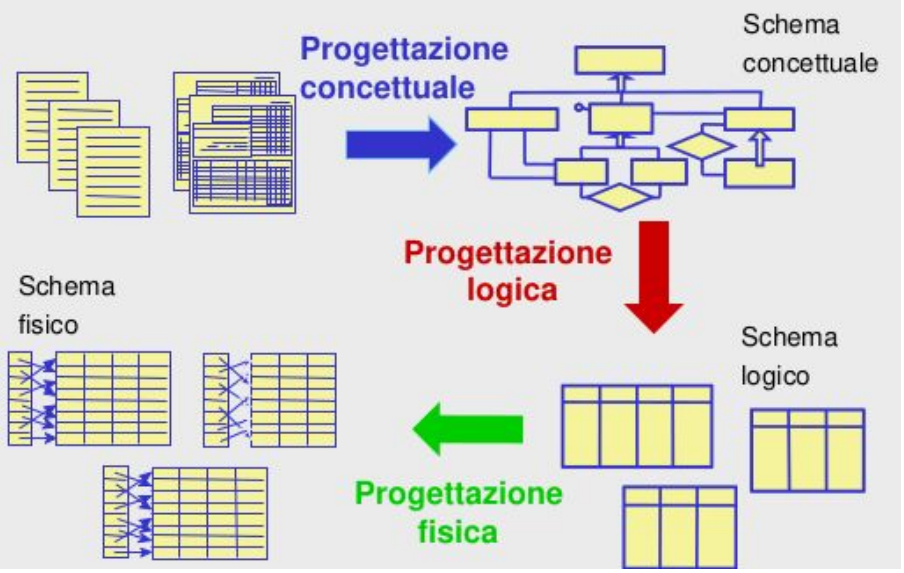
Metodologie e modelli per il progetto

Progettazione di basi di dati

La progettazione di una base di dati è un'attività del processo di sviluppo dei sistemi informativi che fa parte del loro ciclo di vita, le cui fasi sono le seguenti:

- **studio di fattibilità**: definizione di costi e priorità;
- **raccolta e analisi dei requisiti**: studio delle proprietà del sistema;
- **progettazione** delle funzionalità e dei dati manipolati
- **realizzazione**
- **validazione e collaudo**: verifica delle funzionalità
- **funzionamento**: il sistema diventa operativo

Durante le varie fasi della progettazione vengono prodotti diversi elementi:



Modello concettuale

I modelli concettuali hanno principalmente tre vantaggi: servono per ragionare sulla realtà di interesse, indipendentemente dagli aspetti realizzativi, permettono di rappresentare le classi di oggetti di interesse e le loro correlazioni e prevedono efficaci rappresentazioni grafiche (utili per documentazione e comunicazione).

Modello Entity-Relationship

Entità: Classi di oggetti (fatti, persone, cose) della realtà di interesse con proprietà comuni e con esistenza "autonoma".

Relazioni: Legami logici fra due o più entità, rilevanti nell'applicazione di interesse.

Attributi: Proprietà elementari di un'entità o di una relazione, di interesse ai fini dell'applicazione; associano ad ogni occorrenza di entità o relazione un valore appartenente a un insieme detto dominio dell'attributo. Possono essere **composti** nel momento in cui raggruppano attributi di una medesima entità o relazione che presentano affinità nel loro significato o uso.

Cardinalità: Coppia di valori associati a ogni entità che partecipa ad una relazione; il primo valore indica il numero minimo di occorrenze, il secondo il numero massimo. Tipicamente si usano i simboli 0 (partecipazione opzionale) e 1 (partecipazione obbligatoria) per la cardinalità minima e 1 e N (senza limiti) per quella massima. Con riferimento alle cardinalità massime, si possono avere relazioni uno a uno, uno a molti o molti a molti. È anche possibile associare delle cardinalità agli attributi, con due possibili scopi: indicare opzionalità o indicare attributi multivalore.

Identificatori: Usati per l'identificazione univoca delle occorrenze di un'entità (simili al concetto di chiave) e possono essere **interni** (costituiti da attributi dell'entità) o **esterni** (costituiti da entità esterne attraverso relazione con cardinalità (1, 1)). Ogni entità deve possedere almeno un identificatore.

Generalizzazioni: Mettono in relazione una o più entità "figlie" con una entità "genitore", che le comprende come casi particolari. In questo modo, ogni proprietà (attributi, relazioni, ecc.) dell'entità genitore è anche delle figlie, e ogni occorrenza delle figlie è anche del genitore.
Si parla inoltre di **generalizzazione totale** se ogni occorrenza dell'entità genitore è anche di almeno una delle entità figlie, altrimenti è **parziale**; una generalizzazione è **esclusiva** se ogni occorrenza dell'entità genitore è occorrenza di al più una delle entità figlie, altrimenti è **sovrapposta**.

Documentazione associata agli schemi concettuali

Insieme agli schemi concettuali, viene spesso prodotta una documentazione aggiuntiva composta da un dizionario dei dati che forniscono spiegazioni relative a entità e relazioni, e una spiegazione dei vincoli non esprimibili nell'ER.

Progettazione concettuale

Analisi dei dati

Comprende attività (interconnesse) di:

- **acquisizione dei requisiti**;
- **analisi dei requisiti**;
- **costruzione dello schema concettuale**;
- **costruzione del glossario**;

Requisiti

Possibili fonti:

- utenti e committenti, attraverso interviste o documentazione apposita;
- documentazione esistente, attraverso normative, regolamenti interni, procedure aziendali o realizzazioni preesistenti;
- modulistica.

Acquisizione e analisi

Nonostante il reperimento dei requisiti sia un'attività difficile e non standardizzabile, si possono definire le seguenti regole generali:

- scegliere il corretto livello di astrazione;
- standardizzare la struttura delle frasi;
- suddividere le frasi articolate;
- separare le frasi sui dati da quelle sulle funzioni;
- costruire un glossario dei termini;
- individuare omonimi e sinonimi;
- rendere esplicito il riferimento fra termini;
- riorganizzare le frasi per concetti.

È anche possibile stabilire delle regole generali per capire quale costrutto E-R utilizzare per rappresentare un determinato concetto:

- entità, se ha proprietà significative e descrive oggetti con esistenza autonoma;
- attributo, se è semplice e non ha proprietà;
- relazione, se correla due o più concetti;
- generalizzazione, se è caso particolare di un altro.

Pattern di progetto

I pattern di progetto sono soluzioni progettuali a problemi comuni, e sono largamente usati nell'ingegneria del software. I più comuni nella progettazione concettuale di basi di dati sono i seguenti:

- reificazione:
    - di attributo di entità;
    - di relazione binaria;
    - di relazione ricorsiva;

    - di attributo di relazione;
    - di relazione ternaria;
- part-of;
- instance-of;
- storicizzazione di concetto;
- evoluzione di concetto.

Strategie di progetto

Scegliere una strategia di progetto significa invece scegliere come si vuole procedere avendo tante specifiche (anche dettagliate). In questo senso, è possibile scegliere tra strategie **top-down**, **bottom-up** o **inside-out**; a livello pratico, però, si procede con una strategia mista: si individuano cioè i concetti principali (più importanti/cruciali/più citati) e si realizza uno schema scheletro. Successivamente, i concetti si decompongono e si procede a raffinare, espandere e integrare ciò che si è ottenuto.

Qualità di uno schema concettuale

Una volta conclusa la progettazione concettuale, si valuta la qualità dello schema ottenuto sulla base di 4 parametri:

- **correttezza**: si presta attenzione a eventuali errori sintattici e/o semantici e si ri-controlla il diagramma E-R;
- **completezza**: tutti i concetti devono essere tradotti in parti nel diagramma E-R;
- **leggibilità**: ciò che si è ottenuto dev'essere comprensibile anche da non addetti ai lavori; bisogna perciò mettere al centro i concetti chiave, insieme a quelli con più collegamenti, evitare intersezioni e sovrapposizioni e mettere le entità genitori sopra le figlie;
- **minimalità**: uno schema minimale è spesso più leggibile, quindi bisogna evitare generalizzazioni non necessarie e entità senza attributi.

Una possibile metodologia

1. **Analisi dei requisiti**:
    a. analizzare i requisiti ed eliminare le ambiguità;
    b. costruire un glossario dei termini;
    c. raggruppare i requisiti in insiemi omogenei.
2. **Passo base**: schema scheletro con i concetti più rilevanti.
3. **Decomposizione**: decomporre i requisiti con riferimento ai concetti nello schema scheletro.
4. **Passo iterativo** (da ripetere finché non si è soddisfatti, per ogni sottoschema):
    a. raffinare i concetti presenti sulla base delle loro specifiche;
    b. aggiungere concetti per descrivere specifiche non descritte.
5. **Integrazione**: integrare i vari sottoschemi in uno schema complessivo, facendo riferimento allo schema scheletro.
6. **Analisi di qualità**: verificare le qualità dello schema e modificarlo.

Progettazione logica

Il principale scopo della progettazione logica è quello di tradurre lo schema concettuale in uno schema logico che rappresenti gli stessi dati in maniera corretta ed efficiente. Va osservato che alcuni aspetti non sono direttamente rappresentabili, e che spesso è necessario considerare le prestazioni per una traduzione adeguata.

Ristrutturazione schema E-R

Il primo passo da compiere durante la progettazione logica è la ristrutturazione dello schema E-R, che ha come scopi quello di semplificare la successiva traduzione in modello relazionale e quello di ottimizzare le prestazioni; a tal proposito, va tenuto conto del fatto che uno schema E-R ristrutturato non è più uno schema concettuale nel senso stretto del termine. Le principali attività in cui si può suddividere la fase di ristrutturazione sono: analisi delle ridondanze, eliminazione delle generalizzazioni, partizionamento/accorpamento di entità e relazioni e scelta degli identificatori primari.

Analisi delle ridondanze

In uno schema E-R, si chiama ridondanza un'informazione significativa ma derivabile da altre; per questo motivo, in questa fase ci si chiede se eliminare le eventuali ridondanze, mantenerle, o addirittura introdurne di nuove. Le ridondanze hanno l'unico vantaggio principale di semplificare le interrogazioni, che si paga però con appesantimento degli aggiornamenti, più spazio occupato e rischi di inconsistenze. Le principali forme di ridondanza che si possono trovare in uno schema E-R sono attributi derivabili (dalla stessa o da altre entità/relazioni) e relazioni derivabili dalla composizione di altre (più in generale cicli di relazioni).

Eliminazione delle generalizzazioni

In questa fase le generalizzazioni vanno sostituite con entità e relazioni, in quanto non direttamente rappresentabili nel modello relazionale. Le strategie possibili sono tre:

- **accorpamento delle figlie** della generalizzazione **nel genitore** (preferibile se gli accessi al padre e alle figlie sono contestuali);
- **accorpamento del genitore** della generalizzazione **nelle figlie** (preferibile se gli accessi al padre e alle figlie sono separati);
- **sostituzione della generalizzazione con relazione** (preferibile se gli accessi al padre non implicano anche ai figli e viceversa).

Sono anche possibili soluzioni ibride.

Partizionamento/accorpamento di entità e relazioni

Le ristrutturazioni sono spesso effettuate per rendere le operazioni più efficienti, cercando di ridurre gli accessi ai dati; per questo è importante dividere gli attributi a cui si accede spesso separatamente e raggruppare attributi di concetti diversi a cui si accede insieme. Per fare quanto appena descritto, solitamente si procede con **partizionamento verticale di entità**, **eliminazione di attributi multivalore** o **accorpamento di entità/relazioni**.

Scelta degli identificatori primari

Scegliere gli identificatori primari rappresenta una operazione indispensabile per la traduzione in modello relazionale, ed è basata su tre criteri principali: **assenza di opzionalità**, **semplicità** e **utilizzo nelle operazioni più frequenti** o importanti; inoltre, se nessuno degli attributi di un'entità ha queste caratteristiche, si procede con l'aggiunta di codici appositi.

Traduzione verso il modello relazionale

In questa fase, le entità diventano relazioni sugli stessi attributi, usando gli identificatori come chiavi primarie. Inoltre, in caso di relazioni molti a molti, la relazione si trasforma in una tabella che ha come attributi gli identificatori delle entità precedentemente collegate (che insieme formano la chiave) e gli attributi propri della relazione. Si aggiungono infine chiavi esterne e relativi vincoli di integrità referenziale.

Creazione e gestione degli indici

Gli indici favoriscono l'accesso in base al valore di uno o più campi, e il loro obiettivo nei DBMS è di velocizzare l'accesso a 0+ tuple del database; per fare ciò, si definisce una chiave di ricerca (costituita da uno o più attributi della relazione). Un indice è quindi un file (o una struttura dati) che consiste di un insieme di record (index entries) della seguente forma:

Chiave di ricerca	Pointer

La dimensione dell'indice è minore rispetto a quella delle tabella/relazione associate.
Il principale vantaggio degli indici risiede nel fatto che offrono benefici sostanziali quando si cercano dei record, ma di contro portano alla creazione di overhead in fase di aggiornamento di una relazione, in quanto ogni indice ad essa associato deve essere aggiornato di conseguenza.

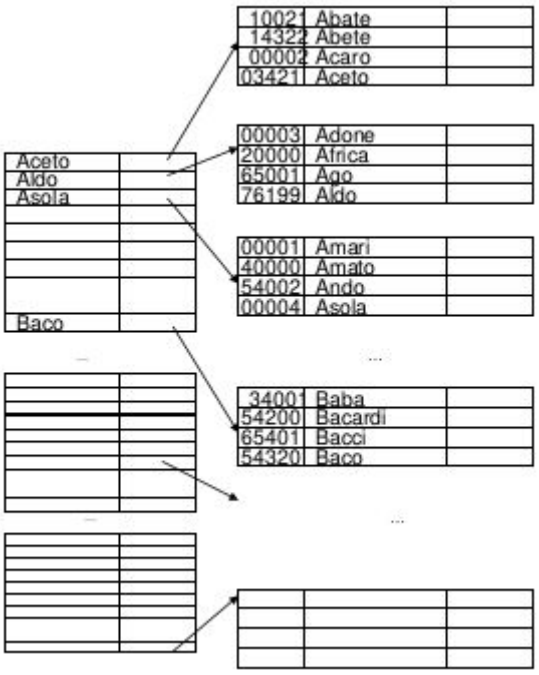
Metriche di valutazione degli indici

L'efficienza di un indice si valuta in base ai seguenti parametri:

- tempo di accesso (via query):
  - ai record con uno specifico valore per un attributo;
  - ai record con valori all'interno di uno specifico intervallo.
- tempo di inserimento;
- tempo di cancellazione;
- spazio occupato.

Indici primari

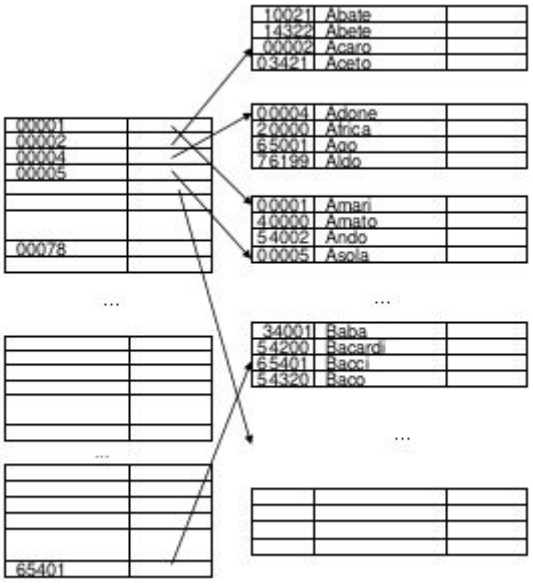
In un indice primario, le tuple sono mantenute ordinate in base ai valori delle chiavi primarie (che possono corrispondere alle chiavi di ricerca); un file può avere al più un indice primario.



Iterare risulta efficiente su indice primario, in quanto i blocchi dei file del database vengono letti in sequenza.

Indici secondari

In un indice secondario, le tuple sono mantenute in ordine diverso rispetto a quello della chiave di ricerca, e un file può avere più di un indice secondario.



A differenza di quanto accade nel caso di indici primari, iterare risulta costoso su un indice secondario, perché occorre potenzialmente saltare avanti e indietro nel file del database, e quindi i blocchi del file degli indici vengono letti più volte.

Alberi binari di ricerca

Per creare un indice bisogna fare uso di alberi binari di ricerca, cioè alberi in cui, per ogni nodo, il sottoalbero sinistro contiene solo etichette minori e il sottoalbero destro contiene solo etichette maggiori (come BST); di conseguenza, trovare un nodo ha un "costo" pari alla sua profondità. Per minimizzare il costo medio, è possibile bilanciare l'albero: ogni nodo o ha due figli o non ne ha nessuno; se l'albero è bilanciato e contiene n nodi, la profondità è circa $\log_2 n$.

B+ Tree

I B+ Tree sono alberi di ricerca bilanciati i cui nodi hanno più di un valore e le foglie sono collegate tra loro. I file che fanno uso di questo tipo di indici hanno diversi vantaggi:

- si riorganizzano con cambiamenti "locali" in caso di inserimenti e cancellazioni;
- la tabella non deve essere riorganizzata per garantire le performance quando tuple vengono aggiunte o cancellate.

Ci sono però anche alcuni svantaggi:

- occupazione di spazio extra, tempi extra per inserire e rimuovere;
- le query con "=" vengono solo parzialmente ottimizzate.

B- Tree

Questo tipo di indici è simile a B+ Tree con la differenza che in questo caso anche i nodi interni contengono valori e le foglie non sono connesse tra di loro. Tra i vantaggi dei B- Tree troviamo però il fatto che tende generalmente ad usare meno nodi di un B+ Tree e che è possibile (non molto spesso) trovare una coppia chiave-valore anche prima di raggiungere una foglia. Gli svantaggi principali risiedono invece in una maggiore complessità di inserimenti, cancellazioni e manutenzione, e non c'è nessun vantaggio sostanziale nelle query per intervallo.

Indici Hash

Gli indici Hash si basano sul principio delle funzioni di Hash: una funzione f:string → string è di hash se, fissato un valore n, per ogni stringa s di lunghezza arbitraria, f(s) restituisce una stringa di lunghezza n; un caso particolare di funzione hash è tale che restituisce una stringa esadecimale (cioè un numero). La funzione appena vista non è necessariamente invertibile.
Un indice hash organizza le chiavi di ricerca in un file hash basato sul concetto di "bucket" (insieme di record chiave-puntatore). A questo punto, dato un valore K della chiave di ricerca, una funzione hash h associa ogni valore K a esattamente un bucket.
La funzione hash è quindi utilizzata per trovare i record di accesso, inserimento e cancellazione; inoltre, dati due valori diversi è possibile che le loro stringhe hash siano uguali: si parla in questo caso di collisione e occorre poi cercare il valore di interesse iterando nel bucket.
Gli indici Hash sono sempre secondari, ma è possibile avere un indice primario separato; di fatto si può quindi avere sia indici hash che B- Tree.

Rispetto ai B+- Tree gli indici hash hanno diversi vantaggi e svantaggi. Vantaggi:

- la ricerca di un valore ha un costo di circa 1 (più basso dei B+ Tree);
- l'inserimento e la cancellazione hanno un costo di circa 1, in quanto occorre inserire/rimuovere la entry dal bucket, creando/eliminando bucket di collisione.

Svantaggi:

- possono essere usati solo per WHERE con atomi del tipo variabile=valore;
- non possono essere usati per velocizzare operazioni di ORDER BY, in quanto non è possibile cercare la entry successiva in ordine.

La sintassi per definire indici in SQL è la seguente:

CREATE INDEX NomeIndex
ON NomeTabella(ListaAttributi)

B+ Tree	mediamente bene per = consigliati per intervalli < o > bene per cancellazione o inserimento
B- Tree	mediamente bene per = male per cancellazione o inserimento mediocri in generale
Hash	migliori per uguaglianze male ORDER BY

uguaglianza	Hash
intervallo	B+ Tree
uguaglianza e intervallo	B+ Tree
cancellazione/inserimento	B+ Tree
ORDER BY	B+ Tree

Normalizzazione

Forme normali

Una forma normale è una proprietà di una base di dati relazionale che ne garantisce l'assenza di determinati difetti; quando una relazione non è normalizzata, infatti, presenta ridondanze e si presta a comportamenti poco desiderabili durante gli aggiornamenti.

Normalizzazione

La normalizzazione è una procedura che permette di trasformare schemi non normalizzati in schemi che soddisfano una forma normale, e va utilizzata come tecnica di verifica dei risultati della progettazione di una base di dati.

Anomalie

All'interno di una base di dati possono esserci diversi tipi di anomalie:

- ridondanze;
- anomalie di aggiornamento;
- anomalia di cancellazione;
- anomalia di inserimento.

Spesso questi fenomeni si verificano nei casi in cui si cerca di rappresentare informazioni eterogenee attraverso un'unica relazione.

Dipendenze funzionali

Per studiare in maniera sistematica i concetti introdotti informalmente nel paragrafo precedente, è necessario far uso di uno specifico strumento di lavoro: la dipendenza funzionale. Si tratta di un particolare vincolo di integrità per il modello relazionale che, come ci suggerisce il nome, descrive legami di tipo funzionale tra gli attributi di una relazione.
Questo concetto può essere formalizzato come segue: data una relazione r su uno schema R(X) e due sottoinsiemi di attributi non vuoti Y e Z di X, diremo che esiste su r una dipendenza funzionale tra Y e Z , se, per ogni coppia di tuple t1 e t2 di r aventi gli stessi valori sugli attributi Y, risulta che t1 e t2 hanno gli stessi valori anche sugli attributi Z (si scrive quindi $Y \rightarrow Z$).
Una dipendenza funzionale si dice banale se, prendendo per esempio il caso $Y \rightarrow A$, A compare tra gli attributi di Y; una dipendenza funzionale si considera invece "buona" se è tra una chiave e altri attributi in quanto in questo caso non causa anomalie.

Forma normale di Boyce e Codd

La forma normale di Boyce e Codd (**BCNF**) è la più importante tra le forme normali.
Una relazione r è in forma normale di Boyce e Codd se per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, X contiene una chiave K di r, cioè X è superchiave per r.
Anomalie e ridondanze non si presentano per relazioni in forma normale di Boyce e Codd, perché i concetti indipendenti sono separati, uno per relazione.

Decomposizione in BCNF

Intuitivamente, per ogni dipendenza $X \rightarrow Y$ che viola la BCNF, si definisce una relazione su XY e si elimina Y dalla relazione originale. Questo tipo di decomposizione non è sempre raggiungibile; in caso di dipendenze funzionali che coinvolgono tutti gli attributi, per esempio, nessuna decomposizione potrebbe conservare tale dipendenza.

Proprietà delle decomposizioni

Le seguenti proprietà garantiscono un livello di qualità delle decomposizioni, in quanto garantiscono rispettivamente che sia possibile ricreare le informazioni originali e che i vincoli di integrità originali siano mantenuti.

Decomposizione senza perdita

Data una relazione r su un insieme di attributi X e X_1 e X_2 sottoinsiemi di X, diciamo che r si decompone senza perdita su X_1 e X_2 se il join delle due proiezioni è uguale a r stessa.
E possibile individuare una condizione che garantisce la decomposizione senza perdita di una relazione, come segue. Sia r una relazione su X e siano X_1 e X_2 sottoinsiemi di X tali che $X_1 \cup X_2 = X$; inoltre, sia $X_0 = X_1 \cap X_2$; allora: r si decompone senza perdita su X_1 e X_2 se soddisfa la dipendenza funzionale $X_0 \rightarrow X_1$ oppure la dipendenza funzionale $X_0 \rightarrow X_2$.
In altre parole, possiamo dire che r si decompone senza perdita su due relazioni se l'insieme degli attributi comuni alle due relazioni è chiave per almeno una delle relazioni decomposte; in caso contrario, si parla di decomposizione con perdita.

Conservazione delle dipendenze

Si dice che una decomposizione conserva le dipendenze dello schema originario se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti. In questo modo, è possibile garantire, sullo schema decomposto, il soddisfacimento degli stessi vincoli il cui soddisfacimento è garantito dallo schema originario.

Terza forma normale

Diciamo che una relazione r è in terza forma normale se, per ogni dipendenza funzionale (non banale) $X \rightarrow A$ definita su di essa, almeno una delle seguenti condizioni è verificata:

- X contiene una chiave K di r;
- A appartiene ad almeno una chiave di r.

Questa forma normale risulta "sempre raggiungibile", ma meno restrittiva di quella di Boyce e Codd: esclude infatti alcune dipendenze, ammettendo anomalie.

Decomposizione in terza forma normale

Per decomporre in terza forma normale si crea una relazione per ogni gruppo di attributi coinvolti in una dipendenza funzionale e si verifica che alla fine una relazione contenga una chiave della relazione originaria.
Una possibile strategia generale di normalizzazione potrebbe essere la seguente:

- se la relazione non è normalizzata si decompone in terza forma normale;
- alla fine si verifica se lo schema ottenuto è anche in BCNF.

Teoria delle dipendenze e normalizzazione

In questo paragrafo mostriamo, sia pure in modo schematico, come i più importanti concetti discussi nei paragrafi precedenti possano essere formalizzati, arrivando a un processo di normalizzazione realizzabile in modo algoritmico. Ci poniamo cioè il seguente problema:

data una relazione e un insieme di dipendenze funzionali definite su di essa, generare una decomposizione della relazione che contenga solo relazioni in forma normale e soddisfi le qualità di decomposizione senza perdita e conservazione delle dipendenze. Poiché, come abbiamo visto, questo obiettivo non è raggiungibile per la forma normale di Boyce e Codd, lo perseguiremo per la terza forma normale.
La teoria della normalizzazione può essere usata nella progettazione logica per verificare lo schema relazionale finale, ma si può usare anche durante la progettazione concettuale per verificare la qualità dello schema concettuale.

Implicazione di dipendenze funzionali

Diciamo che un insieme di dipendenze funzionali F implica un'altra dipendenza f se ogni relazione che soddisfa tutte le dipendenze in F soddisfa anche f. Es:

Impiegato → Categoria
Categoria → Stipendio
implicano
Impiegato → Stipendio

Chiusura di un insieme di attributi

Siano dati uno schema di relazione R(U) e un insieme di dipendenze funzionali F definite sugli attributi in U e sia X un insieme di attributi contenuti in U (cioè X ⊆ U): la chiusura di X rispetto a F, indicata con X_F^+, è l'insieme degli attributi che dipendono funzionalmente da X (esplicitamente o implicitamente):

$$X_F^+ = \{A \mid A \in U \; e \; F \; implica \; X \rightarrow A\} .$$

Se A appartiene a X_F^+ allora X → A è implicata da F.
Esiste un algoritmo semplice ed efficiente per il calcolo di X_F^+:

- Input: un insieme X di attributi e un insieme F di dipendenze.
- Output: un insieme X_p di attributi.
- Algoritmo:
  - inizializziamo X_p con l'insieme di input X;
  - esaminiamo le dipendenze in F: se esiste una dipendenza Y → A con Y ⊆ X_p e A ∉ X_p allora aggiungiamo A a X_p;
  - Ripetiamo il secondo passo fino al momento in cui non vi sono ulteriori attributi che possono essere aggiunti a X_p.

Coperture di dipendenze funzionali

Può essere utile sostituire a un insieme di dipendenze funzionali un altro che specifichi, nella sostanza, le stesse proprietà e che sia più semplice da gestire.
Due insiemi di dipendenze funzionali F_1 e F_2 sono equivalenti se F_1 implica ciascuna dipendenza in F_2 e viceversa; se due insiemi sono equivalenti diciamo anche che ognuno è una copertura dell'altro. Ovviamente, tra due insiemi equivalenti, è meglio scegliere quello più semplice.
In particolare, un insieme F di dipendenze è:

- **non ridondante** se non esiste dipendenza f ∈ F tale che F - { f } implica f;
- **ridotto** se è non ridondante e non esiste un insieme F' equivalente a F ottenuto eliminando attributi dai primi membri di una o più dipendenze di F.

Ecco un esempio:

- {A → B; AB → C; A → C} è ridondante;
- {A → B; AB → C} non è ridondante né ridotto;
- {A → B; A → C} è ridotto.

Calcolo della copertura ridotta

1. Sostituiamo l'insieme dato con quello equivalente che ha tutti i secondi membri costituiti da singoli attributi;
2. per ogni dipendenza verifichiamo se esistono attributi eliminabili dal primo membro;
3. eliminiamo le dipendenze ridondanti.

Procedimento per ottenere schemi in terza forma normale

Dati uno schema R(U) e un insieme di dipendenze F su U, con chiavi K_1, … , K_n.

1. si calcola una copertura ridotta G di F, seguendo i 3 passaggi visti sopra;
2. G viene partizionato in sottoinsiemi tali che due dipendenze funzionali X→A e Y→B sono insieme se $X_G^+ = Y_G^+$;
3. viene costruita una relazione per ogni sottoinsieme;
4. se esistono due relazione S(X) and T(Y) con X ⊆ Y, S viene eliminata;
5. se, per qualche i, non esiste una relazione S(X) con $K_i \subseteq X$, viene aggiunta una relazione T(K_i).

Transazioni

Per transazione si intende una parte di programma caratterizzata da:

- un inizio di transazione (**begin-transaction**);
- un corpo di transazione (serie di insert/delete/update in SQL);
- una fine di transazione (**end-transaction**), che può portare:
    - **commit work**, per terminare correttamente per rendere i cambiamenti definitivi;
    - **rollback work** (o **abort**), per abortire la transazione, come se non fosse mai avvenuta.

Un sistema transazionale (OLTP) è in grado di definire ed eseguire transazioni per conto di un certo numero di applicazioni concorrenti.

Una transazione si può anche definire come una unità di elaborazione (ce ne possono essere più di una allínterno di una applicazione e diverse applicazioni possono effettuare transazioni in parallelo sullo stesso DB) che gode delle proprietà **ACID**: atomicità, consistenza, isolamento, durabilità (persistenza).

Proprietà

Atomicità

Una transazione è un'unità atomica di elaborazione, in quanto o è compiuta interamente o non è compiuta per niente: non può lasciare la base di dati in uno stato intermedio. Perché ciò non accada, un guasto o un errore prima del commit devono causare l'annullamento (**UNDO**) delle operazioni svolte, mentre un guasto o un errore dopo il commit non deve avere conseguenze, e se necessario bisogna poter ripetere le operazioni (**REDO**).

Consistenza

Le transazioni rispettano i vincoli del DB (di chiave, di integrità referenziale ecc.) e li verificano alla fine della transazione (durante la transazione quindi hanno la possibilità di violarli): è importante infatti che lo stato finale sia corretto; nel caso in cui i vincoli risultino violati alla fine della transazione, infatti, non c'è possibilità di commit ma solo di rollback.

Isolamento

Una transazione non risente degli effetti di eventuali transazioni concorrenti, e l'esecuzione concorrente di una collezione di transazioni deve produrre un risultato che si potrebbe ottenere con una esecuzione sequenziale: per questi motivi, una transazione non espone i suoi stati intermedi.

Persistenza

Gli effetti di una transazione andata in commit non vanno perduti, anche in presenza di guasti di dispositivo (es. rottura del disco) o di sistema (crash del software).

Controllo di affidabilità

Il controllo dell'affidabilità garantisce due proprietà fondamentali delle transazioni: l'atomicità e la persistenza. In pratica, esso garantisce che le transazioni non vengano lasciate incomplete, con alcune operazioni eseguite e le altre no, e che gli effetti di ciascuna transazione conclusa con un commit siano mantenuti in modo permanente.
Il controllore dell'affidabilità è responsabile di realizzare i comandi transazionali (start transaction, commit work, rollback work) e di realizzare le operazioni di ripristino (recovery) in seguito a malfunzionamenti.

Il log

Il controllore dell'affidabilità svolge il proprio compito attraverso il log, un file sequenziale che svolge la funzione di "diario di bordo", riportando tutte le operazioni in ordine. Il log è memorizzato su memoria stabile (difficilmente danneggiabile), e su di esso vengono registrate le varie azioni svolte dal DBMS.
Nel log possiamo trovare diversi tipi di record:

- operazioni delle transazioni:
  - B(T): begin transazione T;
  - I(T, O, AS): T inserisce l'oggetto O con valore AS;
  - D(T, O, BS): T cancella l'oggetto O con valore BS;
  - U(T, O, BS, AS): T aggiorna il valore dell'oggetto O da BS a AS;
  - C(T): commit transazione T;
  - A(T): abort transazione T.
- record di sistema
  - dump;
  - checkpoint.

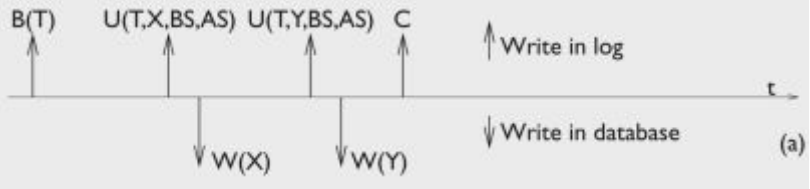
Durante il funzionamento normale delle transazioni, il controllore dell'affidabilità deve garantire che siano seguite due regole, che definiscono i requisiti minimi che consentono di ripristinare la correttezza della base di dati a fronte di guasti:

- la regola **WAL** (**Write Ahead Log**) impone che la parte before state dei record di un log venga scritta nel log prima di effettuare la corrispondente operazione sulla base di dati. Questa regola consente di disfare le scritture già effettuate da parte di una transazione che non ha ancora effettuato un commit, poiché per ogni aggiornamento viene reso disponibile in modo affidabile il valore precedente la scrittura;
- la regola di **Commit-Precedenza** impone che la parte after state dei record di un log venga scritta nel log prima di effettuare il commit. Questa regola consente di rifare le scritture già decise da una transazione che ha effettuato il commit.

Scrittura congiunta di log e base di dati

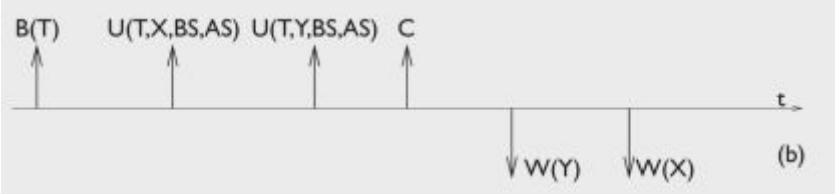
Le due regole WAL e di Commit-Precedenza impongono alcune restrizioni ai protocolli per la scrittura del log e della base di dati, ma lasciano aperte varie possibilità.

Modalità immediata



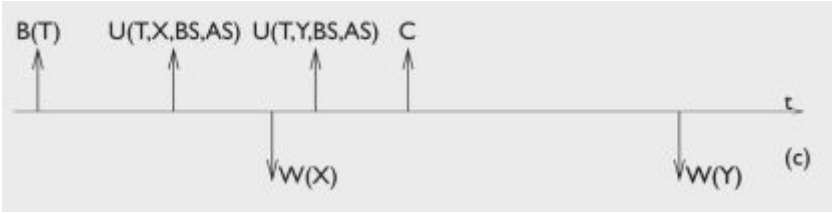
In questa modalità il DB contiene i nuovi valori aggiornati (AS nella figura) provenienti da transazioni uncommitted, di cui è richiesto l'UNDO al momento del guasto.

Modalità differita



In questa modalità il DB non contiene i nuovi valori aggiornati (AS) provenienti da transazioni uncommitted, e in caso di ABORT non è necessario fare niente.

Modalità mista



In questa modalità, la scrittura può avvenire sia in modo immediato che differito; inoltre, è consentita l'ottimizzazione delle operazioni di FLUSH.

Checkpoint

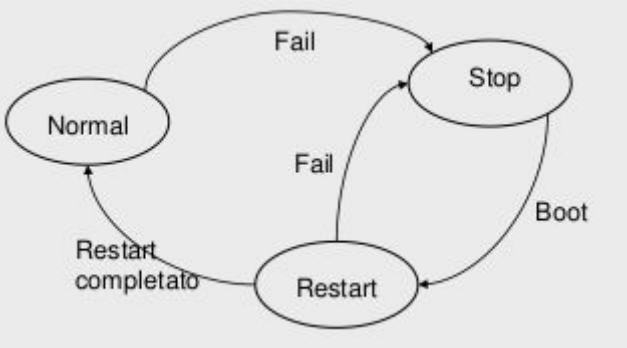
Il checkpoint è un'operazione che serve a registrare quali operazioni sono attive in un certo istante. La modalità più semplice per compiere questa operazione è la seguente:

1. si sospende l'accettazione di richieste di ogni tipo;
2. si trasferiscono in memoria di massa (tramite force) tutte le pagine "sporche" relative a transazioni andate in commit;
3. si registrano sul log in modo sincrono (force) gli identificatori delle transazioni in corso;
4. si riprende l'accettazione delle operazioni.

Modello fail-stop

Quando il sistema individua un guasto, esso forza immediatamente un arresto completo delle transazioni (**stop**) e il successivo ripristino del corretto funzionamento del sistema operativo (**boot**). Quindi, viene attivata una procedura di ripresa, denominata **ripresa a caldo** (**warm restart**) nel caso di guasto di sistema e **ripresa a freddo** (**cold restart**) nel caso di guasto di dispositivo. Al termine delle procedure di ripresa, il sistema diventa

nuovamente utilizzabile dalle transazioni; il buffer è completamente vuoto e può riprendere a caricare pagine della base di dati o del log.



Processo di restart

Un processo di restart ha come obiettivo quello di classificare le transazioni in:

- completate (tutti i dati in memoria stabile);
- in commit ma non necessariamente completate (può servire REDO);
- senza commit (vanno annullate, UNDO).

Warm restart

Il processo di ripresa a caldo si può suddividere in quattro fasi:

1. trovare l'ultimo checkpoint (ripercorrendo il log a ritroso);
2. costruire gli insiemi UNDO (transazioni da disfare) e REDO (transazioni da rifare);
3. ripercorrere il log all'indietro, fino alla più vecchia azione delle transazioni in UNDO e REDO, disfacendo tutte le azioni delle transazioni in UNDO;
4. ripercorrere il log in avanti, rifacendo tutte le azioni delle transazioni in REDO.

Cold restart

Il processo di ripresa a freddo avviene nel seguente modo:

1. si ripristinano i dati a partire dal backup;
2. si eseguono le operazioni registrate sul log fino all'istante del guasto;
3. si esegue una ripresa a caldo.

Dump

Per dump si intende una copia completa (backup) della base di dati, prodotta solitamente mentre il sistema non è operativo. Un dump è salvato in memoria stabile, e un record di dump nel log indica il momento in cui il dump è stato effettuato, con i relativi dettagli pratici.

Problemi e gestione della concorrenza

L'esecuzione concorrente di varie transazioni può causare alcuni problemi di correttezza o anomalie; la presenza di queste anomalie motiva la necessità di controllare la concorrenza.
Le anomalie più frequenti sono:

ANOMALIA	POTENZIALI CAUSE
Perdita di aggiornamento	Due transazioni scrivono lo stesso dato
Lettura sporca	Transazione legge un dato scritto da un'altra transazione che poi ha abortito
Letture inconsistenti	Transazione legge lo stesso dato in due momenti ma la seconda volta legge un dato aggiornato da un'altra transazione
Aggiornamento fantasma	Un dato appare "improvvisamente" aggiornato
Inserimento fantasma	Un nuovo dato appare "improvvisamente"

Gestione della concorrenza

Per evitare queste anomalie, le transazioni possono essere definite **read-only**, e il livello di isolamento può essere scelto per ogni transazione tra **read uncommitted**, **read committed**, **repeatable read**, **serializable**. La sintassi SQL per definire il livello di isolamento di una transazione è:

BEGIN TRANSACTION ISOLATION LEVEL valore.

ANOMALIA	RU	RC	RR	S
Perdita di aggiornamento	X	X	X	X
Lettura sporca	-	X	X	X
Letture inconsistenti	-	-	X	X
Aggiornamento fantasma	-	-	X	X
Inserimento fantasma	-	-	-	X

X = non permette

I livelli di isolamento diversi da serializable vanno usati esclusivamente con transazioni di sola lettura; quando una transazione effettua letture e scritture è opportuno usare comunque il livello di isolamento massimo. In generale, occorre definire il livello che serve in funzione delle operazioni che accadono nella transazione.

Controllo di concorrenza

Schedule: Rappresenta la sequenza di operazioni di ingresso/uscita presentate da transazioni concorrenti.

Scheduler: Sistema che accetta, rifiuta o riordina le operazioni richieste dalle transazioni.

Schedule seriale: Schedule in cui le azioni di ciascuna transazione compaiono in sequenza, senza essere inframmezzate da istruzioni di altre transazioni.

Schedule serializzabile: Schedule che produce lo stesso risultato prodotto da un qualunque schedule seriale delle stesse transazioni.

Equivalenza tra schedule

Definiamo dapprima una relazione che lega coppie di operazioni di lettura e scrittura: un'operazione di lettura $r_i(x)$ legge d a una scrittura $w_j(x)$ quando $w_j(x)$ precede $r_i(x)$ e non vi è alcun $w_k(x)$ compreso tra le due operazioni. Una operazione di scrittura $w_j(x)$ viene detta una **scrittura finale** se è l'ultima scrittura dell'oggetto x che appare nello schedule.

View-equivalenza

Due schedule vengono detti **view-equivalenti** se possiedono la stessa relazione "legge-da" e le stesse scritture finali. Uno schedule viene detto **view-serializzabile** se esiste uno schedule seriale view-equivalente ad esso. Indichiamo con **VSR** l'insieme degli schedule view-serializzabili. La view-serializzabilità di uno schedule garantisce assenza di perdita di aggiornamento, letture inconsistenti, aggiornamento fantasma.
Determinare la view-equivalenza di due schedule è un problema con complessità lineare, mentre decidere sulla view serializzabilità di uno schedule S è un problema "difficile" perché occorre provare tutte i possibili schedule seriali, ottenuti per permutazioni dell'ordine delle transazioni; tali difficoltà rendono la nozione di view-equivalenza e la verifica della view-serializzabilità non applicabili nella pratica.

Conflict-equivalenza

Una nozione di equivalenza più facilmente utilizzabile si basa sulla definizione di conflitto. Date due azioni a_i e a_j, si dice che a_i è in conflitto con a_j se esse operano sullo stesso oggetto e almeno una di esse è una scrittura. Possono quindi esistere conflitti lettura-scrittura (rw o wr) e conflitti scrittura-scrittura (ww). Si dice che lo schedule S_i è **conflict-equivalente** allo schedule S_j se i due schedule presentano le stesse operazioni e ogni coppia di operazioni in conflitto è nello stesso ordine nei due schedule. Uno schedule risulta quindi **conflict-serializzabile** se esiste uno schedule seriale a esso conflict-equivalente. Chiamiamo **CSR** l'insieme degli schedule conflict-serializzabili.

Ogni schedule conflict-serializzabile (CSR) è view-serializzabile (VSR), ma non tutti gli schedule view-serializzabili (VSR) sono conflict-serializzabili (CSR); quindi la conflict-serializzabilità è condizione sufficiente, ma non necessaria, per la view-serializzabilità.

E possibile determinare se uno schedule è conflict-serializzabile tramite il **grafo dei conflitti**. Il grafo è costruito facendo corrispondere un nodo a ogni transazione. Si traccia quindi un arco orientato da t_i a t_j se esiste almeno un conflitto tra un'azione a_i e un'azione a_j e si ha che a_i precede a_j. Si può dimostrare che uno schedule è in CSR se c solo se il suo grafo dei conflitti è aciclico.
L'analisi di ciclicità di un grafo ha una complessità lineare rispetto alle dimensioni del grafo stesso, però la conflict-serializzabilità risulta ancora eccessivamente onerosa in pratica.

Lock

Nell'esecuzione di operazioni di lettura e scrittura si devono rispettare i seguenti vincoli:

- ogni operazione di lettura deve essere preceduta da un **r_lock** e seguita da un **unlock**; il lock si dice in questo caso condiviso, perché su un dato possono essere contemporaneamente attivi più lock di questo tipo;
- ogni operazione di scrittura deve essere preceduta da un **w_lock** e seguita da un **unlock**; il lock si dice in tal caso esclusivo, perché non può coesistere con altri lock (esclusivi o condivisi) sullo stesso dato.

Quando una transazione segue queste regole si dice ben formata rispetto al locking; in genere, le transazioni sono automaticamente ben formate rispetto al locking, poiché emettono le opportune richieste di lock e unlock in modo trasparente al programma applicativo. Il gestore della concorrenza riceve le richieste di lock provenienti dalle transazioni, e concede i lock in base ai lock precedentemente concessi alle altre transazioni. Quando una richiesta di lock non viene concessa, la transazione richiedente viene messa in stato di attesa; l'attesa termina quando la risorsa viene sbloccata e diviene disponibile. I lock già concessi vengono memorizzati in tabelle di lock, gestite dal lock manager.
La politica che viene seguita dal lock manager per concedere i lock è rappresentata nella **tabella dei conflitti**, in cui le righe identificano le richieste, le colonne lo stato della risorsa richiesta, il primo valore nella cella l'esito della richiesta e il secondo valore nella cella lo stato che verrà assunto dalla risorsa dopo l'esecuzione della primitiva; inoltre, per ogni risorsa ci sono anche un contatore che tiene conto del numero di "lettori" (la risorsa è rilasciata quando il contatore scende a zero) e un valore booleano che tiene conto se c'è un w_lock).

	libera	***r_locked***	***w_locked***
r_lock(x)	OK / r_locked conta(x)++	OK / r_locked conta(x)++	NO / w_locked
w_lock(x)	OK / w_locked	NO / r_locked	NO / w_locked
unlock(x)	error	OK / if (--conta(x)=0) libera else r_locked	OK / not w_locked

Locking a due fasi (2PL)

I meccanismi di locking possono essere usati per garantire che le transazioni che lavorano sulla base di dati accedano ai dati in mutua esclusione. Per avere però la garanzia che le transazioni seguano uno schedule serializzabile è necessario porre una restrizione sull'ordinamento delle richieste di lock, che prende il nome di principio del **locking a due fasi**. Secondo questo principio una transazione, dopo aver rilasciato un lock, non può acquisirne altri.
In relazione agli schedule CSR, possiamo dire che tutti gli schedule 2PL sono CSR, ma non vale il contrario.
Esiste inoltre una variante del locking a due fasi, che prevede il rilascio dei lock solo dopo il commit o abort; tale variante prende il nome di **locking a due fasi stretto**.