Le Basi di Dati

Uno dei principali compiti dei sistemi informatici è l'attività di raccolta, organizzazione e conservazione dei dati. Tali sistemi garantiscono che questi dati siano conservati in modo permanente su dispositivi per la loro memorizzazione, permettendone l'aggiornamento e rendendone possibile l'accesso da parte degli utenti.

Questo tutorial ha come argomento la gestione dei dati tramite sistemi informatici; ha come obiettivo trattare in maniera semplice ma esaustiva i concetti che stanno dietro ad una tale gestione.

Di seguito verranno introdotti i concetti di sistema informativo e di base di dati, definendo poi quali sono i requisiti che deve avere un sistema informatico per gestire una base di dati.

Introduzione

Sistemi informativi, informazioni e dati

Per sistema informativo si intende quel sistema che permette la disponibilità e la gestione delle informazione. L'esistenza di un sistema informativo è indipendente dalla sua automazione; lo dimostra il fatto che archivi e servizi anagrafici esistono da vari secoli. Per indicarne la porzione automatizzata viene utilizzato il termine sistema informatico.
La diffusione dell'informatica ha fatto si che la quasi totalità dei sistemi informativi siano anche sistemi informatici.

Le informazioni vengono rappresentate e scambiate in varie forme, quali la lingua, disegni, figure, numeri. In alcuni casi può anche non esistere una rappresentazione esplicita delle informazioni, come nel caso di informazioni trasmesse oralmente e ricordate a memoria.
Col progredire delle attività umane, tuttavia, è nata l'esigenza di individuare opportune codifiche per la memorizzazione dei dati.

Nei sistemi informatici il concetto di rappresentazione e codifica viene portato all'estremo: le informazioni vengono rappresentate per mezzo di dati, che hanno bisogno di essere interpretati per fornire informazioni.

Basi di dati, la definizione

La più generale definizione di una base di dati è collezione di dati utilizzati per rappresentare le informazioni di interesse per un sistema informativo.
Tale definizione è molto semplicistica e troppo generale. Nel paragrafo seguente si cercare di definire il termine in maniera più precisa.

Occorre, tuttavia, trarre una prima considerazione sulle basi di dati. Se prendiamo come esempio i dati relativi alle applicazioni bancarie noteremo che essi hanno una struttura sostanzialmente invariata da decenni, mentre le procedure che agiscono su di essi variano con una certa frequenza. Inoltre, quando viene introdotta una nuova procedura occorre, prima di tutto, "ereditare" (=importare) i dati dalla vecchia, se pur con le necessarie trasformazioni.

Questa caratteristica di stabilità porta ad affermare che i dati costituiscono una "risorsa" per l'organizzazione che li gestisce, un patrimonio significativo da sfruttare e proteggere. Le normative attuali in fatto di privacy e tutela delle basi di dati lo dimostra.

Sistemi di gestione di basi di dati

Sebbene la gestione dei dati abbia catalizzato, fin dalle origini dell'informatica, l'attenzione delle applicazioni, solo negli anni settanta nascono linguaggi specificatamente dedicati alla gestione dei dati. Un esempio di tali linguaggi è il COBOL, nato in quegli anni e ormai superato, che è presente ancor oggi in un numero incredibile di applicazioni.

L'approccio "convenzionale" alla gestione dei dati sfrutta la presenza di archivi (o file) per memorizzare i dati in modo persistente sulla memoria di massa.

Tale approccio presenta delle macroscopiche deficienze per quanto riguarda la ricerca e la condivisione dei dati, in pratica annullata; infatti con una simile metodologia di lavoro ogni utente lavora con la propria copia "locale", con i relativi problemi *ridondanza* e possibilità di *incoerenze*. Le basi di dati sono state concepite in buona parte per ovviare ad inconvenienti di questo tipo.

Un sistema di gestione di basi di dati, detto DBMS (Data Base Management System) è un sistema software in grado di gestire collezioni di dati che siano *grandi*, *condivise* e *persistenti* assicurando la loro *affidabilità* e *privatezza*. Inoltre, in quanto prodotto informatico, deve essere *efficiente* e *efficace*. Una base di dati è una collezione di dati gestita da un DBMS.

Riassumendo un DBMS si occupa di basi di dati con le seguenti caratteristiche:

***Grandi dimensioni**:*
nel senso che possono avere anche dimensioni enormi (terabyte e oltre) e quindi oltre le capacita della memoria centrale di un elaboratore. Di conseguenza un DBMS deve essere in grado di gestire memorie secondarie.

***Risorse Condivise**:*
perché un DBMS deve permettere a più utenti di accedere contemporaneamente ai dati comuni. In tal modo viene anche ridotta la *ridondanza* e *inconsistenza* dei dati, dato che esiste una sola copia dei dati. Per controllare l'accesso condiviso di più utenti il DBMS dispone di un meccanismo apposito, *detto controllo di concorrenza*.

***Affidabilità**:*
dato che un DBMS deve garantire l'integrità dei dati anche in caso di malfunzionamento hardware e software, prevedendo per lo meno procedure di recupero dei dati. I DBMS forniscono, per tali scopi, procedure di salvataggio e ripristino della base di dati (*backup* e *recovery*).

***Privatezza**:*
i DBMS gestiscono un sistema di *autorizzazioni* che definisce i *diritti* di ciascun utente (lettura, scrittura ecc.).

Modelli di dati

Un modello di dati è un insieme di concetti utilizzati per organizzati i dati di interesse e descrivere la struttura in modo che essa risulti comprensibile ad un elaboratore.

Ogni modello di dati fornisce meccanismi di strutturazione, analoghi ai costruttori di tipo dei linguaggi di programmazione, che permettono di definire nuovi tipi sulla base di tipi elementari predefiniti.

Il modello relazionale dei dati (modello su cui si concentra l'attenzione di questo tutorial) permette di definire tipi per mezzo del costruttore di *relazione*, che consente di organizzare i dati in insiemi di record a struttura fissa. Una relazione viene spesso rappresentata mediante una tabella in cui le righe rappresentano i specifici record e le colonne corrispondono ai campi dei record.

Schemi ed istanze

Esistono, oltre al modello relazionale, altri modelli di database quali il modello gerarchico, il modello reticolare, il modello ad oggetti. Tutti i modelli di basi di dati sono, però, accomunati da dalla presenza di una parte che rimane invariata nel tempo, detta *schema*, e da una parte, detta istanza o stato della base di dati, costituita dai valori effettivi.

Si dice anche che lo schema è la parte intensionale della base di dati mentre l'istanza è la parte estensionale.

Livelli di astrazione nei DBMS

Esiste una proposta di struttura standardizzata per i DBMS articolata su tre livelli, detti *esterno, logico e interno*; per ciascun livello esiste uno schema:

Lo ***schema logico*** (o ***concettuale***), che costituisce la descrizione dell'intera base di dati per mezzo del modello logico adottato dal DBMS (cioè tramite uno dei modelli citati in precedenza, relazionale, gerarchico, reticolare o a oggetti).

Lo ***schema interno*** costituisce la rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione.

Uno ***schema esterno*** costituisce la descrizione di una porzione della base di dati di interesse, per mezzo del modello logico. Uno schema esterno può prevedere organizzazioni dei dati diverse rispetto a quelle utilizzate nello schema logico, che riflettono il punto di vista di un particolare utente o insieme di utenti. Pertanto, è possibile associare ad uno schema logico vari schemi esterni.

Nei sistemi moderni il livello esterno non è esplicitamente presente, ma è possibile definire relazioni derivate (o viste, dall'inglese *views*).

Indipendenza dei dati

L'architettura così definita garantisce l'indipendenza dei dati, ovvero la principale proprietà dei DBMS.
Questa proprietà permette agli utenti ed ai programmi applicativi di utilizzare una base di dati ad un elevato livello di astrazione, che prescinde dai dettagli realizzativi utilizzati per la base di dati stessa.
In particolare, l'indipendenza dei dati può essere caratterizzata ulteriormente come indipendenza fisica e logica:

L'indipendenza fisica consente di interagire con il DBMS in modo indipendente dalla struttura fisica dei dati. In base a questa proprietà è possibile modificare le strutture fisiche senza influire sulle descrizioni dei dati ad alto livello e quindi sui programmi che utilizzano i dati stessi.

L'indipendenza logica consente di interagire con il livello esterno della base di base in modo indipendente dal livello logico.

I Database Relazionali

Rappresenta il modello su cui si basa la maggior parte dei sistemi di basi di dati oggi sul mercato. Tale modello fu proposto in una pubblicazione scientifica nel 1970 al fine di superare le limitazioni logiche dei modelli allora utilizzati, che non permettevano di realizzare efficacemente la proprietà di indipendenza dei dati, già riconosciuta come fondamentale.
Sebbene i primi prototipi di db basati sul modello relazionale risalgano ai primi anni settanta bisognerà aspettare la metà degli anni ottanta perché tale modello acquisisca una frazione significativa di mercato.

La lentezza di affermazione del modello relazionale deriva principalmente dal suo alto livello di astrazione: non è stato immediato per gli operatori del settore imparare ad individuare relazioni efficienti.

Il modello relazionale

Vengono qui illustrate le modalità secondo cui esso questo modello permette di organizzare i dati, come il concetto di relazione possa essere mutuato dalla teoria degli insiemi ed utilizzato, con le debite varianti, per rappresentare le informazioni di interesse in una base di dati. Vengono approfonditi i concetti di corrispondenza fra dati in strutture diverse, informazione completa e vincoli di integrità.

Modelli logici nei sistemi di basi di dati

Il modello relazionale si basa su due concetti fondamentali: relazione e tabella.
Mentre il concetto di tabella è facilmente intuibile, quello di relazione proviene dalla matematica, ed in particolare dalla teoria degli insiemi.

E' opinione diffusa che parte del successo del modello relazionale derivi dalla presenza contemporanea di questi due concetti, uno intuitivo ed uno formale.
Infatti, mentre le tabelle risultano naturali e facilmente comprensibili le relazioni garantiscono una formalizzazione semplice e chiara che ha permesso uno sviluppo teorico del modello finalizzato al raggiungimento di risultati di interesse concreto.

Il modello relazionale risponde al requisito dell'indipendenza dei dati e, pertanto, prevede un livello fisico ed un livello logico. Utenti e programmatori interagiscono solo col livello logico e quindi non è necessario che essi conoscano le strutture fisiche della base di dati.
Anche questo aspetto è responsabile del suo successo dato che i suoi principali concorrenti (reticolare e gerarchico) obbligavano gli utilizzatori a conoscerne, almeno a grandi linee, la struttura realizzativa.

Relazioni

Il concetto di relazione è legato al concetto puramente matematico di prodotto cartesiano tra due insiemi.
Avendo due insiemi D_1 e D_2 il prodotto cartesiano (D_1 x D_2) è l'insieme delle coppie ordinate (v_1 e v_2) tale che v_1 è un elemento di D_1 e v_2 è un elemento di D_2.
Quindi il prodotto cartesiano è l'insieme di tutti le combinazioni tra gli insiemi dati.

La relazione è un sottoinsieme della relazione matematica tra due insiemi, detti domini della relazione, rappresentato da un insieme di tuple omogenee, dove per tuple (traslitterazione dell'inglese "tuple") si intende un elemento definito tramite i suoi attributi.
Le tuple sotto questo aspetto sono diverse dal concetto matematico di n-uple, elemento individuato tramite posizione, e tupla, in cui l'elemento è individuato tramite i suoi attributi.

Ad esempio dati due insiemi A e B dove A= {1,2,3} e B={h,k} il prodotto cartesiano è uguale all'insieme A x B={(1,h),(2,h),(3,h),(1,k),(2,k),(3,k)} mentre una relazione possibile è {(1,h),(1,k),(3,h)}.

Generalizzando la relazione ad un numero di insiemi n>0 avremmo D_1,D_2,…,D_n il prodotto D_1 x D_2 x … x D_n ed un sottoinsieme che descriverà la relazione.
Il numero n delle componenti dell'insieme è detto grado del prodotto cartesiano e della relazione.
Il numero degli elementi che della relazione è detta cardinalità della relazione.

Le tabelle nascono dall'esigenza di rappresentare graficamente le relazioni presentandole in una forma più facilmente comprensibile. In questo caso le righe della tabella saranno rappresentate dalle tuple mentre le colonne ne rappresentano i campi.

E' importante chiarire che in una relazione, in quanto insieme, non vi è alcun ordinamento fra le tuple che lo compongono; nelle tabelle che la rappresentano l'ordine c'è per necessità, ma è occasionale in quanto due tabelle con le stesse righe, ma in ordine diverso, rappresentano la stessa relazione.
Inoltre le tuple di una relazione sono distinte l'una dall'altra, in quanto tra gli elementi di un insieme non possono essere presenti due elementi uguali; da cui si deduce che una tabella può rappresentare una relazione solo se le righe che la formano sono diverse l'una dall'altra.

Informazioni incompleta e valori nulli

Un'altra caratteristica importante dei sistemi relazionali è la presenza di un particolare valore che può assumere un'istanza di una tabella.
Tale valore è detto null e viene utilizzato per indicare una serie di situazioni che è possibile trovare in un campo di una tabella.
Ad esempio ci si può trovare di fronte ad una tabella del tipo:

Città	**Indirizzo prefettura**
Roma	Via Quattro Novembre
Firenze	Null
Tivoli	Null
Prato	Null

In questo esempio il valore null indica tre diverse situazioni:

· **Valore sconosciuto**: nel primo caso dato che Firenze è un capoluogo di provincia ed avrà sicuramente una prefettura; significa che il DBMS non dispone del suo indirizzo.
· **Valore inesistente**: nel caso di Tivoli, dato che questa città non ha di Prefettura.
· **Senza informazione**: siccome la provincia di Prato è di recente istituzione significa che non sappiamo se la mancanza di un indirizzo dipenda dal fatto che essa ancora non esista, oppure da un deficit del DBMS.

Vincoli di integrità

I vincoli di integrità sono quei vincoli, caratteristica fondamentale delle basi di dati relazionali, che indicano la "bontà" delle informazioni. Ci possono essere, infatti, casi in cui i dati non rispettino, per una serie di motivi che andremo ad analizzare, l'integrità logica del database che, in quanto insieme formalizzato di informazioni, ha delle regole molto precise (e spesso rigide) che devono essere obbligatoriamente rispettate.

I vincoli di un DBMS si dividono in intrarelazionali, se riguardano l'interno nella relazioni, ed extrarelazionali, più pesanti perché riguardano i legami fra le relazioni e quindi la natura stessa di un DBMS relazionale.

Il più caratterizzante esempio di violazione di un vincolo extrarelazionale si ha qualora non vi sia corrispondenza tra le istanze di due tabelle che, per ragioni intrinseche allo schema della base di dati, sono legate tra loro da un vincolo detto di integrità referenziale, il quale impone che per ogni valore di una tabella vi sia un corrispondente nell'altra.

Un vincolo intrarelazionale, al contrario, trova il suo soddisfacimento rispetto alle singole relazioni del DBMS; esso può essere:

· **vincolo di tupla**; spiegato di seguito
· **vincolo su valori**, o **vincolo di dominio**: quando si impone a certi valori del database di rientrare in determinate caratteristiche (ad esempio il voto di un esame universitario deve obbligatoriamente rientrare tra il 18 ed il 30, ed un DBMS incaricato raccogliere tali dati deve prevederlo).

Vincoli di tupla

Il vincolo di tupla è un vincolo che esprime delle condizioni sui valori di ciascuna tupla, indipendentemente dalle altre tuple.

Ad esempio consideriamo l'esempio dei voti degli esami universitari e ipotizziamo di avere un DBMS che debba memorizzarli; poniamo, inoltre, di destinare un campo all'eventuale lode (campo di tipo booleano, vero/falso).

In questo caso avremmo che l'attributo "lode" potrà essere vero solo se il voto di quella tupla è uguale a trenta. Cercando di dare una forma a tale vicolo potremmo scrivere che:

se (voto <> 30) allora (lode = falso)

mentre in questo caso il vincolo di dominio (citato nel paragrafo precedente) potrebbe essere espresso con la formula:

(voto >= 18) and (voto <= 30)

Chiavi

Una chiave è un insieme di attributi (anche uno solo) utilizzato per identificare in maniera univoca le tuple di una relazione. Questo per la natura stessa del modello relazionale che pone questo vincolo come suo assioma.

Ne consegue che la violazione del vincolo di chiave, inammissibile in un DBMS relazionale, è la presenza di due o più tuple aventi la stessa chiave.

Cenni di algebra e calcolo relazionale

L'algebra relazionale è un linguaggio procedurale, basato su concetti di tipo algebrico che deriva da quella parte della matematica che si occupa degli insiemi.
Siccome i concetti trattati sono molto astratti, poco legati al pratico utilizzo dei DBMS quanto a quello realizzativo, in questo capitolo ci limiteremo a descrivere i concetti generali senza entrare nel dettaglio delle formule e delle dimostrazioni.
Di seguito descriveremo gli operatori fondamentali dell'algebra relazionale, lo stretto necessario per comprendere da cosa deriva l' SQL.

Algebra relazionale

L'algebra relazionale definisce una serie di operatori e regole per la creazione e la modifica delle relazioni. Di seguito vengono trattati i più comuni.

Unione

Appurato che le relazioni sono insiemi, ha senso intervenire sulle relazioni con gli operatori dell'algebra insiemistica, quali l'unione la differenza e l'intersezione.
Tuttavia, dato che una relazione è un insieme di tuple omogenee (ovvero definite dagli stessi attributi) questi operatori potranno essere usati solo con relazioni aventi gli stessi attributi, o almeno degli attributi comuni.

Detto questo è possibile dire che l'unione tra due relazioni r1 ed r2, definite sullo stesso insieme di attributi X (si scrive r1(X) ed r2(X)), è indicata con r1Ur2 ed è una relazione ancora su X contenente le tuple che appartengono ad r1 oppure ad r2, oppure ad entrambe.

Nella fattispecie delle basi di dati l'unione di due tabelle, che come abbiamo visto sono le corrispondenti delle relazione, è l'insieme delle righe che appartengono alle tabelle (alla prima, alla seconda o ad entrambe).

Intersezione

L'intersezione di r1(X) ed r2(X) è indicata con r1 r2 ed è una relazione su X contenente le tuple che appartengono sia ad r1 che ad r2.

Riferito alle tabelle l'intersezione di due di queste è l'insieme delle righe che appartengono ad entrambe.

Differenza

La differenza di r1(X) ed r2(x) è indicata con r1 - r2 ed è una relazione su X contenente le tuple che appartengono ad r1 e non appartengono ad r2.

Quindi la differenza tra due tabelle è l'insieme delle righe che appartengono alla prima tabella e non alla seconda.

Ridenominazione
Per risolvere il problema derivante dal fatto che gli operatori sopra citati possono essere usati solo con attributi uguali, l'algebra relazionale ci mette a disposizione l'operatore di ridenominazione.
Questo operatore ci permette di ridefinire il nome dell'attributo in modo da operare operazioni di unione, differenza ed intersezioni anche su relazioni con diversi attributi (sempre ammesso che ciò possa avere un senso).

Selezione
La selezione è quell'operatore dell'algebra relazionale che produce un sottoinsieme di tuple, producendo una relazione che ha gli stessi attributi dell'operando ma un numero minore (o uguale in casi particolari) di istanze.

La selezione introduce un concetto fondamentale dell'algebra relazionale e, di conseguenza, delle basi di dati : il concetto di condizione (trattato in maniera più esauriente nel capitolo successivo). Infatti la discriminante che permette di stabilire quali tuple andranno a formare il risultato della selezione è il fatto che essa rispondano meno ad una determinata condizione.

Proiezione
La proiezione è un concetto simile a quello della selezione. Essa produce, alla pari della selezione, un sottoinsieme della relazione genitrice; a differenza della selezione, però, il sottoinsieme prodotto contiene tutte le tuple originali ma con un numero inferiore di attributi.

Join
L'operatore di join è il più caratteristico dell'algebra relazionale.
Il join naturale è un operatore che correla i dati di due relazioni sulla base di valori uguali in attributi con lo stesso nome.

Esistono diversi tipi di join, a seconda delle diverse situazione di relazioni, che sono tuttavia riconducibili al modello del join naturale. Un'analisi più approfondita esula dai compiti di questo lavoro, volto all'aspetto pratico delle basi di dati. Ulteriori informazioni circa il join si possono trovare nel capitolo successivo, che mostra l'implementazione di tale operatore nel linguaggio SQL.

Viste
La vista non è un operatore dell'algebra relazionale; più semplicemente è una tecnica dell'algebra che permette di mettere a disposizione dell'utente rappresentazioni diverse degli stessi dati.

La tecnica che ci permette di raggiungere questo scopo è quella delle relazioni derivate, relazioni il cui contenuto è funzione del contenuto di altre relazioni. Queste relazioni si contrappongono alle relazioni di base (le relazioni vere e proprie), il cui contenuto è autonomo. A livello di implementazioni si possono dividere le viste in due tipi:

· **viste materializzate**: relazioni derivate effettivamente memorizzate sulla base di dati;
· **viste virtuali**: relazioni definite per mezzo di funzioni, non memorizzate sulla base di dati, ma utilizzabili nelle interrogazioni come se effettivamente lo fossero.

Creazione di Database

I database di un qualunque Motore DBMS sono rappresentati da un insieme di tabelle contenenti dati e da altri oggetti quali viste, indici, stored procedure e trigger, definiti per supportare le attività eseguite con i dati.
I dati archiviati in un database sono in genere correlati a un oggetto o processo particolare, quali le informazioni dell'inventario del magazzino di una fabbrica.

Ogni database consente di archiviare dati correlati o dati non correlati a quelli presenti in altri database. Ad esempio, è possibile che in un server sia disponibile un database in cui sono archiviati i dati sul personale e un altro in cui sono archiviati i dati correlati ai prodotti.
In alternativa, è possibile che in un database siano archiviati i dati aggiornati sugli ordini dei clienti e in un altro database correlato siano archiviati gli ordini precedenti dei clienti da utilizzare per la creazione di report annuali.

Prima di creare un database, è importante conoscere le parti che lo compongono e come progettarle per ottenere prestazioni ottimali dal database dopo l'implementazione.

I database includono un insieme di tabelle in cui viene archiviato uno specifico set di dati strutturati.
Le tabelle contengono un insieme di righe e colonne (a volte denominate attributi).
Le singole colonne delle tabelle vengono impostate in modo che contengano un tipo di informazioni specifico, ad esempio date, nomi, importi in valuta o numeri.
Alle tabelle sono associati numerosi tipi di controlli (vincoli*,* regole*,* trigger*,* valori predefiniti e tipi di dati definiti dall'utente) che garantiscono la validità dei dati.

Nelle tabelle è possibile includere indici, con caratteristiche molto simili a quelli dei libri, che consentono di trovare rapidamente le righe.

È possibile aggiungere i vincoli di integrità referenziale dichiarativa (DRI, Declarative Referential Integrity) alle tabelle per garantire che i dati interrelati delle diverse tabelle rimangano consistenti.

Nei database è inoltre possibile archiviare viste che consentono l'accesso personalizzato ai dati della tabella.

Considerazioni sulla progettazione di database

Per progettare un database è necessario conoscere le funzioni business che si desidera modellare e le caratteristiche e i concetti relativi al database utilizzati per rappresentare le funzioni business.
È importante progettare un database in modo accurato per adattarlo alle proprie esigenze perché una significativa modifica alla struttura di un database già implementato può richiedere tempi lunghi.
Inoltre, le prestazioni di un database progettato in modo appropriato saranno migliori.
Durante la progettazione di un database, è necessario considerare:

· Le finalità del database e l'effetto di queste sulla struttura. Creare un piano di database adeguato alle finalità.
· Le regole di normalizzazione del database che impediscono di commettere errori durante la progettazione del database.
· La protezione dell'integrità dei dati.
· I requisiti di protezione del database e le autorizzazioni per gli utenti.
· I requisiti dell'applicazione in termini di prestazioni.

Creazione di piani di database

Il primo passaggio della procedura di creazione di un database è la creazione di un piano da utilizzare come guida quando il database viene implementato e come specifica funzionale per il database dopo l'implementazione. La complessità e i dettagli della struttura di un database dipendono dalla complessità e dalle dimensioni dell'applicazione di database e dal tipo di utenti.

La natura e la complessità di un'applicazione di database e il processo di pianificazione possono variare notevolmente. Un database può essere relativamente semplice e progettato per essere utilizzato da un singolo utente oppure può essere di grandi dimensioni, complesso e progettato, ad esempio, per consentire la gestione di migliaia di client. Nel primo caso, per progettare il database potrebbe essere sufficiente creare una semplice bozza su un foglio di carta.
Nel secondo caso, è probabile che il progetto sia un documento formale con centinaia di pagine che includono tutti i dettagli sul database.
Durante la pianificazione di un database è consigliabile attenersi ai passaggi seguenti, indipendentemente dalle dimensioni e dalla complessità:

· Raccolta di informazioni.
· Identificazione degli oggetti.
· Modellazione degli oggetti.
· Identificazione dei tipi di informazioni per ogni oggetto.
· Identificazione delle relazioni tra oggetti.

Raccolta di informazioni

Prima di creare un database, è necessario determinare in modo chiaro quale funzione dovrà essere svolta dal database. Se il database sostituirà un sistema di informazioni basato su materiale cartaceo o gestito manualmente, il sistema esistente fornirà la maggior parte delle informazioni necessarie.
È importante consultare tutti gli utenti che utilizzano il sistema per conoscere le loro mansioni e determinare in quale modo il database può soddisfare le loro esigenze. È inoltre importante identificare le funzioni che gli utenti desiderano vengano svolte dal nuovo sistema nonché individuare problemi, limiti e colli di bottiglia dei sistemi esistenti.
Infine, raccogliere copie di note informative sui clienti, di elenchi di inventario, di report amministrativi e altri documenti che fanno parte del sistema esistente perché saranno utili per progettare il database e le interfacce.
Identificazione degli oggetti
Durante il processo di raccolta delle informazioni, è necessario identificare gli oggetti o le entità chiave che verranno gestiti dal database.

L'oggetto può essere un elemento tangibile, ad esempio una persona o un prodotto, oppure può essere un elemento meno tangibile, quale una transazione business, il reparto di un'azienda o un periodo di retribuzione.
Esiste in genere un numero limitato di oggetti primari. Dopo che sono stati identificati, gli elementi correlati diventano evidenti. A ogni elemento distinto del database dovrebbe corrispondere una tabella.

Supponiamo che l'oggetto primario di un database di un'azienda sia un libro.
Gli oggetti correlati ai libri dell'attività di questa azienda sono gli autori che scrivono i libri, gli editori che li producono, i negozi che li vendono e le transazioni di vendita eseguite con i negozi. Ognuno di questi oggetti rappresenta una tabella nel database.

Modellazione di oggetti

Quando gli oggetti del sistema vengono identificati, è importante registrarli in modo da rappresentare il sistema visivamente. È possibile utilizzare il modello di database come riferimento durante l'implementazione del database.

A tale scopo, gli sviluppatori di database utilizzano strumenti che per complessità variano da semplici appunti su carta a programmi di elaborazione di testi, fogli di calcolo e programmi dedicati alla modellazione di dati per la progettazione di database. Indipendentemente dallo strumento utilizzato, è importante mantenerlo aggiornato.

Identificazione dei tipi di informazioni per ogni oggetto

Dopo aver identificato gli oggetti di database primari che si desidera inserire nelle tabelle, è necessario identificare i tipi di informazioni che si desidera archiviare per ogni oggetto, che diventeranno le colonne nella tabella dell'oggetto.
Le colonne di una tabella di database contengono alcuni tipi di informazioni comuni:

· Colonne di dati non elaborati
Tali colonne contengono informazioni tangibili, quali i nomi, determinate da un'origine esterna al database.

· Colonne di categoria
Tali colonne classificano o raggruppano i dati e contengono una selezione limitata di dati quali true/false, married/single, VP/Director/Group Manager e così via.

· Colonne identificatore
Tali colonne consentono di identificare ogni elemento archiviato nella tabella. I nomi di queste colonne indicano in genere che le colonne includono un ID o un numero (ad esempio, **ID_Impiegato**, **Num_Telefono**).
La colonna identificatore è il componente primario utilizzato dagli utenti e dagli strumenti di elaborazione interni al database per accedere a una riga di dati nella tabella. A volte l'oggetto si presenta nella forma tangibile di ID che è possibile utilizzare nella tabella (ad esempio, un numero di previdenza sociale), ma nella maggior parte delle situazioni è possibile definire la tabella in modo da creare un ID artificiale affidabile per la riga.

· Colonne relazionali o referenziali
Queste colonne stabiliscono un collegamento tra le informazioni di una tabella e le informazioni correlate di un'altra tabella. Ad esempio, una tabella che tiene traccia delle transazioni di

vendita viene in genere collegata alla tabella dei clienti per consentire di associare le informazioni complete sui clienti alla transazione di vendita.

Identificazione delle relazioni tra oggetti

Un punto di forza dei database relazionali è rappresentato dalla possibilità di correlare o associare informazioni su elementi diversi del database.
È possibile archiviare in modo distinto tipi isolati di informazioni, ma il motore di database consente di combinare i dati se necessario.
Per identificare le relazioni tra oggetti durante il processo di progettazione, è necessario esaminare le tabelle, determinare in che modo sono correlate logicamente e aggiungere colonne relazionali che stabiliscono i collegamenti tra le singole tabelle.

Ad esempio, durante la progettazione di un database di libri verranno create le tabelle per i titoli e gli editori.
La tabella **Libri** include informazioni per ogni libro: una colonna identificatore denominata **ID_Libro**, colonne di dati non elaborati per il titolo, il prezzo del libro e la relativa data di pubblicazione nonché alcune colonne con le informazioni di vendita del libro.
La tabella include una colonna di categoria denominata **tipo** che consente di raggruppare i libri in base al tipo di contenuto.
A ogni libro è associato un editore, ma le informazioni sull'editore sono incluse in un'altra tabella. Pertanto, la colonna **ID_Editore** della tabella **Libri** includerà soltanto l'ID dell'editore. Quando viene aggiunta una riga di dati per un libro, l'ID dell'editore viene archiviato con le altre informazioni relative al libro.

Tipi di Applicazione

Molti programmi rientrano in due categorie principali di applicazioni di database:

- Elaborazione delle transazioni in linea (OLTP, Online Transaction Processing)
- Supporto decisionale

Le caratteristiche di questi tipi di applicazioni hanno un effetto determinante sulle considerazioni relative alla progettazione di un database.

Elaborazione delle transazioni in linea

Le applicazioni di database per l'elaborazione delle transazioni in linea sono appropriate per la gestione dei dati che vengono modificati e in genere supportano un numero elevato di utenti che eseguono contemporaneamente più transazioni per la modifica dei dati in tempo reale.

Sebbene le singole richieste di dati tendano a fare riferimento a pochi record, molte delle richieste vengono eseguite contemporaneamente. Esempi comuni di questi tipi di database sono rappresentati dai sistemi di biglietteria aerea e dai sistemi di transazione bancaria.
Gli aspetti più importanti da considerare per questo tipo di applicazioni sono la concorrenza e l'atomicità.

I controlli della concorrenza in un sistema di database assicurano che gli stessi dati non

vengano modificati da due utenti contemporaneamente oppure che un utente non modifichi determinati dati prima che un altro utente abbia terminato di modificarli.
Ad esempio, se un cliente chiede all'impiegato di una biglietteria aerea di prenotare l'ultimo posto disponibile su un volo e l'impiegato inizia il processo di prenotazione del posto a nome del cliente, un altro impiegato non dovrebbe essere in grado di trovare tale posto disponibile per assegnarlo a un altro passeggero.

L'atomicità assicura che tutti i passaggi di una transazione vengono ritenuti correttamente completati solo se considerati tutti come un unico gruppo.
Se un passaggio ha esito negativo, non sarà possibile completare gli altri passaggi. Ad esempio, una transazione bancaria può comportare due passaggi: prelievo di fondi da un conto corrente e deposito su un conto di risparmio.

Se il passaggio durante il quale vengono prelevati i fondi dal conto corrente ha esito positivo, è necessario verificare che l'importo venga depositato sul conto di risparmio o ridepositato sul conto corrente.

Considerazioni sulla progettazione per i sistemi di elab. delle trans. in linea
I database dei sistemi di elaborazione delle transazioni dovrebbero essere progettati in modo da garantire:

· Il posizionamento corretto dei dati.
I colli di bottiglia delle operazioni di I/O rappresentano un problema rilevante per i sistemi OLTP a causa del numero di utenti che modificano i dati in tutto il database. È opportuno determinare quali saranno le più probabili modalità di accesso ai dati e raggruppare i dati a cui si accede più di frequente. A tale scopo, utilizzare i filegroup e i sistemi RAID (matrice ridondante di dischi indipendenti, Redundant Array of Independent Disks).

· Transazioni brevi per ridurre al minimo i blocchi a lungo termine e ottimizzare la concorrenza.
Evitare l'interazione dell'utente durante le transazioni. Quando possibile, eseguire una singola stored procedure per elaborare l'intera transazione. L'ordine con cui si fa riferimento alle tabelle nelle transazioni può avere effetto sulla concorrenza. Inserire nella parte finale della transazione i riferimenti alle tabelle a cui si accede di frequente per ridurre al minimo la durata di mantenimento dei blocchi.

· Backup in linea.
I sistemi OLTP sono spesso caratterizzati da operazioni continue (24 ore al giorno, 7 giorni alla settimana) con periodi di inattività minimi. Sebbene in molti motori DBMS sia consentita l'esecuzione del backup di un database mentre viene utilizzato, è opportuno pianificare il processo di backup in modo che venga eseguito durante i periodi di minore attività per ridurre al minimo gli effetti sugli utenti.

· Elevata normalizzazione del database.
Ridurre il più possibile le informazioni ridondanti per aumentare la velocità degli aggiornamenti e quindi ottimizzare la concorrenza. La riduzione dei dati aumenta inoltre la velocità di esecuzione dei backup perché è necessario eseguire il backup di una minore quantità di dati.

· Assenza o riduzione al minimo di dati storici o aggregati.
I dati a cui si fa riferimento solo di rado possono essere archiviati in altri database oppure spostati dalle tabelle aggiornate più frequentemente alle tabelle che includono soltanto dati storici. In questo modo, le dimensioni delle tabelle vengono limitate al massimo e vengono

pertanto ottimizzati i tempi di esecuzione dei backup e le prestazioni delle query.

· Utilizzo accurato degli indici.
È necessario aggiornare gli indici ogni volta che una riga viene aggiunta o modificata. Per evitare una indicizzazione eccessiva delle tabelle aggiornate con maggiore frequenza, mantenere basso il numero di colonne degli indici. Per progettare gli indici, utilizzare l'Ottimizzazione guidata indici.

· Configurazione hardware ottimale per gestire il numero elevato di utenti concorrenti e i tempi di risposta veloci necessari in un sistema OLTP.

Supporto decisionale

Le applicazioni di database per il supporto decisionale sono indicate per le query che non modificano i dati.

Ad esempio, un'azienda può riepilogare periodicamente i dati di vendita per data, area di vendita o per prodotto e archiviare queste informazioni in un database distinto da utilizzare per le analisi da parte dei responsabili.

Per prendere decisioni strategiche, gli utenti devono essere in grado di determinare rapidamente le tendenze delle vendite eseguendo query sui dati in base a vari criteri, ma non è necessario che modifichino questi dati.
Le tabelle in un database per il supporto decisionale sono intensamente indicizzate e i dati non elaborati vengono in genere pre-elaborati e organizzati per supportare i vari tipi di query da utilizzare. Poiché gli utenti non modificano i dati, la concorrenza e l'atomicità non rappresentano un problema.
I dati vengono modificati soltanto in occasione di aggiornamenti di massa periodici eseguiti durante gli orari di minore attività del database.

Considerazioni sulla progettazione per i sistemi di supporto decisionale

I database dei sistemi di supporto decisionale dovrebbero essere progettati in modo da garantire:

· Un'intensa indicizzazione.
I sistemi per il supporto decisionale gestiscono una quantità di aggiornamenti limitata ma volumi di dati estesi. Utilizzare numerosi indici per ottimizzare le prestazioni delle query.

· Denormalizzazione del database.
Introdurre dati preaggregati o riepilogati per soddisfare i requisiti comuni delle query e ottimizzare i tempi di risposta delle query stesse.

· Utilizzare uno schema a stella o a fiocco di neve per organizzare i dati all'interno del database.

Normalizzazione

L'attività di progettazione logica del database, incluse le tabelle e le relazioni tra tabelle, è di fondamentale importanza per ottenere un database relazionale ottimizzato.
Se si esegue una progettazione logica corretta, si hanno buone probabilità di ottenere prestazioni ottimali dal database e dall'applicazione. Se si esegue una progettazione logica inadeguata, potrebbero verificarsi effetti negativi sulle prestazioni dell'intero sistema.

La normalizzazione della progettazione logica del database comporta l'utilizzo di metodi formali per la suddivisione dei dati in più tabelle correlate. Un numero maggiore di tabelle compatte (con un numero minore di colonne) è tipico di un database normalizzato.
Un numero minore di tabelle di grandi dimensioni (con un numero maggiore di colonne) è tipico di un database non normalizzato.

Una normalizzazione ragionevole spesso consente di ottimizzare le prestazioni. Quando sono disponibili indici utili, il motore DBMS seleziona in modo efficace join tra tabelle rapidi ed efficienti.

Di seguito vengono indicati alcuni vantaggi garantiti dalla normalizzazione:

· Ordinamento e creazione dell'indice più veloci.
· Indici cluster più numerosi.
· Indici più compatti e con un minor numero di colonne.
· Un numero minore di indici per tabella con conseguente ottimizzazione delle prestazioni delle istruzioni INSERT, UPDATE e DELETE.
· Un numero minore di valori Null e una minore probabilità di inconsistenze, con conseguente aumento della compattezza del database.

Aumentando il grado di normalizzazione, aumentano il numero e la complessità dei join necessari per recuperare i dati.

Un numero troppo elevato di join relazionali complessi tra un numero troppo elevato di tabelle può rallentare le prestazioni. Una normalizzazione ragionevole è spesso caratterizzata da un numero basso di query eseguite regolarmente le quali utilizzano join che coinvolgono più di quattro tabelle.

A volte la progettazione logica del database è già stata eseguita e una riprogettazione totale non è realizzabile. Anche in questo caso, tuttavia, sarà possibile normalizzare selettivamente una tabella di grandi dimensioni in diverse tabelle più piccole.

Se per accedere al database si utilizzano stored procedure, è possibile implementare questa modifica dello schema senza interessare le applicazioni. In caso contrario, è possibile creare una vista che nasconde la modifica dello schema alle applicazioni.

Progettazione ottimale dei database

Nella teoria della progettazione di database relazionali, le regole di normalizzazione identificano alcuni attributi che devono essere presenti o assenti in un database ben progettato.
Una completa discussione sulle regole di normalizzazione esula dall'ambito di questo argomento.

Tuttavia, esistono alcune regole che consentono di ottenere una struttura di database affidabile:

· Includere un identificatore in ogni tabella.
La regola fondamentale della teoria della progettazione di database prevede di includere in ogni tabella un identificatore di riga univoco, una colonna o un set di colonne che è possibile utilizzare per distinguere ogni record dagli altri record della tabella. A ogni tabella è necessario associare una colonna di ID e non sarà possibile assegnare lo stesso ID a record diversi. Una o più colonne utilizzate come identificatore di riga univoco per una tabella rappresentano la chiave primaria della tabella.

· In una tabella è opportuno archiviare soltanto i dati di un solo tipo di entità.
Se si tenta di archiviare troppe informazioni in una tabella, la gestione efficiente e affidabile dei dati della tabella può essere ostacolata. Nel database dei Libri le informazioni su titoli ed editori sono archiviate in due tabelle distinte.
Nonostante nella tabella **Libri** sia possibile includere colonne con informazioni sia per il libro sia per l'editore del libro, questa struttura comporta diversi problemi.
Le informazioni sull'editore devono essere aggiunte e archiviate in modo ridondante per ogni libro pubblicato da un editore e questo comporta l'utilizzo di spazio di archiviazione aggiuntivo nel database. Se l'indirizzo dell'editore viene modificato, la modifica deve essere apportata per ogni libro. Se l'ultimo libro di un editore viene rimosso dalla tabella, le informazioni su tale editore vengono perdute.
Poiché nel database dei Libri le informazioni relative ai libri e agli editori sono archiviate nelle tabelle **Libri** e **Editori**, è possibile immettere le informazioni sull'editore una sola volta e quindi collegarle a ogni libro. Pertanto, se le informazioni sull'editore vengono modificate, la modifica deve essere apportata una sola volta e le informazioni sull'editore saranno presenti anche se nel database non sono inclusi libri associati a tale editore.

· In una tabella non includere colonne che supportano valori Null.
Nelle tabelle è possibile definire colonne che supportano valori Null. Un valore Null indica che non è presente alcun valore.
In alcuni casi può essere utile consentire l'inserimento di valori Null, ma in genere è preferibile utilizzarli solo se necessario perché richiedono una gestione speciale che aumenta la complessità delle operazioni sui dati.
Se è disponibile una tabella con più colonne che supportano valori Null e in diverse righe delle colonne sono inclusi valori Null, è consigliabile inserire tali colonne in un'altra tabella collegata alla tabella primaria. L'archiviazione dei dati in due tabelle distinte consente di ottenere una tabella primaria con una struttura semplice ma che consente di archiviare queste informazioni se si presenta la necessità.

· In una tabella non includere colonne o valori ripetuti.
La tabella di un elemento del database non deve includere l'elenco di valori relativi a un'informazione specifica. Ad esempio, è possibile che un libro nel database dei Libri sia stato scritto da più autori. Se nella tabella **Libri** è presente una colonna per il nome dell'autore, si verifica un problema.
Una soluzione consiste nell'archiviare i nomi dei coautori nella colonna, ma in questo modo è più difficile visualizzare un elenco dei singoli autori. Un'altra soluzione consiste nel modificare la struttura della tabella aggiungendo un'altra colonna per il nome del secondo autore, ma in questo modo è possibile specificare soltanto due autori. Se il libro è stato scritto da tre autori, sarà infatti necessario aggiungere un'altra colonna.
Se è necessario archiviare un elenco di valori in un'unica colonna o se sono disponibili più colonne per un unico dato (**Autore1**, **Autore2** e così via), è consigliabile inserire i dati duplicati in un'altra tabella con un collegamento alla tabella primaria.
Nel database dei Libri verrà creata una tabella per inserire le informazioni sui libri e un'altra

tabella in cui sono archiviati soltanto gli ID dei libri e gli ID degli autori dei libri.
Questa struttura consente di specificare un numero qualsiasi di autori per un libro senza modificare la definizione della tabella e non assegna lo spazio di archiviazione inutilizzato ai libri con un unico autore.

Integrità dei dati

Se si ottiene l'integrità dei dati si garantisce la qualità dei dati nel database.
Ad esempio, se a un dipendente viene assegnato il numero di matricola (**ID_Impiegato**) **123** , il database non dovrebbe consentire che a un altro dipendente venga assegnato un ID con lo stesso valore.
Se è disponibile una colonna **Valutazione_Impiegato** nella quale si prevede di includere valori da **1** a **5**, il database non dovrebbe accettare il valore **6**. Se nella tabella è disponibile una colonna **ID_Reparto** nella quale viene archiviato il numero di reparto del dipendente, il database dovrebbe consentire soltanto l'inserimento dei valori validi per i numeri di reparto dell'azienda.
Due operazioni importanti per la pianificazione di tabelle sono identificare i valori validi per una colonna e determinare come ottenere l'integrità dei dati della colonna.
L'integrità dei dati rientra nelle seguenti categorie:

- Integrità di entità
- Integrità di dominio
- Integrità referenziale

Integrità di entità

L'integrità di entità definisce una riga come entità univoca per una tabella specifica. L'integrità di entità determina l'integrità di una o più colonne identificatore o della chiave primaria di una tabella (tramite indici, vincoli UNIQUE, vincoli PRIMARY KEY o proprietà IDENTITY).

Integrità di dominio

L'integrità di dominio è la validità delle voci di una colonna specifica. È possibile ottenere l'integrità di dominio limitando il tipo (tramite i tipi di dati), il formato (tramite i vincoli CHECK e le regole) o l'intervallo di valori possibili (tramite i vincoli FOREIGN KEY, i vincoli CHECK, le definizioni DEFAULT, le definizioni NOT NULL e le regole).

Integrità referenziale

L'integrità referenziale mantiene le relazioni definite tra le tabelle quando i record vengono inseriti o eliminati. Solitamente in un motore DBMS, l'integrità referenziale si basa sulle relazioni tra le chiavi esterne e le chiavi primarie o tra le chiavi esterne e le chiavi univoche.

L'integrità referenziale garantisce che i valori di chiave siano consistenti tra le varie tabelle. Per ottenere tale consistenza, è necessario non fare riferimento a valori inesistenti e se viene modificato un valore di chiave, tutti i riferimenti a tale valore devono essere modificati in modo consistente in tutto il database.