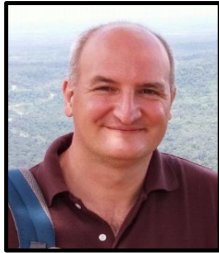


# Gestione Dati





## INFORMAZIONI SUGLI AUTORI



### **Dino Martinazzo**

Laurea in Statistica e Gestione delle Imprese presso l'ateneo di Padova  
Diploma di laurea in Statistica Informatica per la Gestione delle Imprese presso l'ateneo di Padova

Perito Elettronico industriale

Esperienza pluriennale come libero professionista in progettazione e realizzazioni di sistemi di controllo di processo ed acquisizione dati in ambito industriale, controllo e supervisione

Partecipazione a gruppi di ricerca MIUR in particolare col Ministero dell'Istruzione Universitaria e Ricerca U.R.S.T su

“Sperimentazione dei sistemi computer assisted per la rilevazione della valutazione della didattica universitaria da parte degli studenti e dell’inserimento lavorativo e professionale dei laureati e dei diplomati”.

Responsabile R&D per AruKu.it : Sistema per la rilevazione Posturale

E' stato fatto tutto il possibile per garantire l'accuratezza delle informazioni presentate. Tuttavia, le informazioni contenute in questo libro sono fornite senza garanzia,esplicita o implicita. Gli autori, rivenditori e distributori, non saranno ritenuti responsabili per eventuali danni causati, o presunti tali, direttamente o indirettamente, da questo libro.Nel testo sono presenti immagini utilizzate solo a scopo illustrativo. Alcune di queste immagini sono state reperite in internet e non siamo in grado di attribuirne l'origine. Non è nostra intenzione ledere i diritti di alcuno, e nel caso in cui venissero sollevate delle rivendicazioni motivate sulla paternità delle stesse, saremo pronti a indicarne la proprietà o a sostituirle con altre non coperte da diritto d'autore.

# CONTENUTI

<b>Introduzione .....</b>	<b>5</b>
La gestione dei dati .....	5
<b>La gestione dei dati .....</b>	<b>8</b>
Raccolta dei dati .....	9
Organizzazione dei dati .....	11
Archiviazione dei dati .....	13
Sicurezza dei dati .....	16
Gestione del ciclo di vita dei dati .....	19
Condivisione dei dati .....	21
<b>I sistemi di gestione dei database.....</b>	<b>24</b>
I sistemi di gestione dei database .....	25
Structured Query Language .....	27
DDL (Data Definition Language) .....	30
DML (Data Manipulation Language).....	33
DQL (Data Query Language) .....	35
DCL (Data Control Language).....	38
<b>SQL .....</b>	<b>40</b>
Linguaggio SQL: Fondamenti e Esempi .....	41
<b>MS Access.....</b>	<b>69</b>
Microsoft Access .....	69
<b>SQL Server .....</b>	<b>72</b>
MS SQL Server .....	73
Altri server SQL .....	75
<b>SQLITE.....</b>	<b>77</b>
SQLite è un server SQL Embedded .....	78

# INTRODUZIONE

## La gestione dei dati

La gestione dei dati, anche nota come gestione dell'informazione, si riferisce alle attività e ai processi che riguardano la raccolta, l'organizzazione, l'archiviazione, la conservazione, la protezione e la distribuzione dei dati all'interno di un'organizzazione. È un aspetto critico per il funzionamento efficace e efficiente di qualsiasi azienda o entità che lavori con dati.

La gestione dei dati comprende diversi componenti e processi, tra cui:

**Raccolta dei dati:** questo è il processo di acquisizione dei dati da diverse fonti, come formulari online, dispositivi IoT (Internet of Things), database, registri cartacei, sensori e così via. È importante assicurarsi che i dati siano accurati, completi e pertinenti alle esigenze dell'organizzazione.

**Organizzazione dei dati:** una volta raccolti, i dati devono essere organizzati in una struttura coerente per consentire una gestione efficiente. Ciò può includere la categorizzazione dei dati in base a determinati attributi o criteri, l'assegnazione di metadati per una facile identificazione e l'utilizzo di modelli di dati per stabilire relazioni tra diverse entità dati.

**Archiviazione dei dati:** l'archiviazione dei dati coinvolge l'allocazione e la conservazione dei dati in modo che siano accessibili e protetti. Ci sono diverse

opzioni di archiviazione disponibili, tra cui server locali, data center, cloud storage e servizi di archiviazione online. La scelta dipende dalle esigenze specifiche dell'organizzazione in termini di accessibilità, sicurezza e scalabilità.

**Sicurezza dei dati:** la sicurezza dei dati è un aspetto cruciale della gestione dei dati. Comprende misure di protezione per prevenire l'accesso non autorizzato, l'alterazione o la perdita dei dati. Queste misure possono includere l'implementazione di protocolli di autenticazione e autorizzazione, crittografia dei dati, backup regolari, monitoraggio degli accessi e delle attività, nonché politiche di gestione delle password.

**Gestione del ciclo di vita dei dati:** i dati seguono un ciclo di vita che va dalla loro creazione o acquisizione fino alla loro eliminazione. La gestione del ciclo di vita dei dati implica la definizione delle politiche e dei processi per determinare quando i dati devono essere archiviati, conservati, resi obsoleti o eliminati. Ciò aiuta a mantenere i dati rilevanti, a rispettare le normative sulla privacy e a ridurre la congestione dei sistemi di archiviazione.

**Condivisione dei dati:** la condivisione dei dati coinvolge la distribuzione o la divulgazione di dati a persone o entità autorizzate. Può implicare la creazione di report, l'accesso ai dati attraverso strumenti di analisi o la condivisione di dati con partner commerciali o clienti. È importante stabilire politiche e meccanismi per garantire che la condivisione dei dati avvenga in modo sicuro e conforme alle normative sulla privacy.







## LA GESTIONE DEI DATI

## Raccolta dei dati

La raccolta dei dati è il processo di acquisizione di informazioni e di dati grezzi da diverse fonti. Questi dati possono provenire da fonti interne all'organizzazione, come sistemi aziendali, database, registri o moduli compilati dagli utenti. Possono anche essere acquisiti da fonti esterne, come sondaggi, interviste, dati di mercato, dati provenienti dai social media o altre fonti pubbliche.

La raccolta dei dati può avvenire in diversi modi, tra cui:

**Metodi manuali:** Questi includono la compilazione di moduli cartacei o la registrazione manuale dei dati in fogli di calcolo o database. Può richiedere tempo e risorse significative, ma può essere appropriato per organizzazioni di piccole dimensioni o quando i dati sono disponibili in forma fisica.

**Metodi automatici:** Questi metodi coinvolgono l'uso di strumenti e tecnologie per acquisire automaticamente i dati da diverse fonti. Ad esempio, i sensori possono raccogliere dati ambientali, i dispositivi IoT possono fornire dati sulle prestazioni dei prodotti, e i sistemi informatici possono estrarre automaticamente dati da fonti digitali.

**Metodi online:** Con l'avvento di Internet, molti dati sono accessibili online. La raccolta dei dati può avvenire attraverso la scansione di siti web, il web scraping (l'estrazione automatica di dati da pagine web), la raccolta di dati dai social media o l'acquisizione di dati da moduli online.

**Metodi di ricerca:** La ricerca di dati può coinvolgere la consultazione di fonti specializzate, come banche dati, archivi, pubblicazioni scientifiche o report di settore. Questa ricerca può essere svolta manualmente o mediante l'uso di strumenti di ricerca avanzati.

Quando si raccoglie dei dati, è importante considerare alcuni aspetti fondamentali:

**Pertinenza:** assicurarsi che i dati raccolti siano rilevanti per gli obiettivi dell'organizzazione e per le domande di ricerca o di business poste.

**Qualità:** garantire che i dati siano accurati, affidabili e privi di errori. Ciò può richiedere procedure di controllo di qualità e validazione dei dati.

**Consenso e privacy:** rispettare le normative sulla privacy e ottenere il consenso appropriato quando si raccolgono dati personali o sensibili.

**Scalabilità:** considerare la scalabilità delle operazioni di raccolta dei dati, soprattutto se si prevede di gestire grandi volumi di dati nel tempo.

**Archiviazione e protezione:** pianificare l'archiviazione e la protezione dei dati raccolti per garantire l'accesso sicuro, la conservazione a lungo termine e la conformità alle normative sulla privacy.

Una corretta raccolta dei dati fornisce una base solida per una gestione efficace e significativa delle informazioni all'interno di un'organizzazione.

## Organizzazione dei dati

L'organizzazione dei dati è un componente essenziale della gestione dei dati ed è il processo di strutturazione e categorizzazione dei dati per facilitarne la ricerca, l'accesso e l'analisi. Una buona organizzazione dei dati permette di ottenere informazioni significative e di prendere decisioni informate. Di seguito sono descritte alcune delle principali pratiche di organizzazione dei dati:

**Categorizzazione:** La categorizzazione è il processo di classificazione dei dati in gruppi o categorie basate su caratteristiche comuni. Ad esempio, si possono creare categorie per classificare i clienti in base all'età, al luogo di residenza, all'interesse per un prodotto specifico o a qualsiasi altro attributo rilevante. La categorizzazione aiuta a organizzare i dati in modo logico e a facilitarne l'analisi.

**Gerarchia:** La gerarchia è una struttura di organizzazione dei dati in cui gli elementi sono organizzati in modo gerarchico, con livelli superiori e inferiori. Ad esempio, si possono creare gerarchie per organizzare i dati delle vendite in regioni, paesi, città e negozi. Ciò consente una visualizzazione strutturata dei dati e facilita l'analisi a diversi livelli di dettaglio.

**Metadati:** I metadati sono informazioni che descrivono i dati stessi. Possono includere informazioni come il nome del campo, il tipo di dati, l'origine, la data di creazione, il formato e altre informazioni pertinenti. L'aggiunta di metadati ai dati

permette di identificarli e di comprenderne il contesto, semplificando così la ricerca e l'utilizzo dei dati.

**Standardizzazione:** La standardizzazione è il processo di uniformazione dei dati in modo che siano coerenti e omogenei. Ciò può implicare l'uso di formati standard per le date, le valute, le unità di misura e altre convenzioni di rappresentazione dei dati. La standardizzazione dei dati rende più facile confrontare, combinare e analizzare i dati provenienti da diverse fonti.

**Normalizzazione:** La normalizzazione è il processo di riduzione della ridondanza e dell'incoerenza dei dati. Consiste nel suddividere i dati in tabelle separate e relazionate tra loro, eliminando la duplicazione delle informazioni. La normalizzazione aiuta a ridurre la ridondanza dei dati e a migliorare l'integrità e l'efficienza nella gestione dei dati.

**Indicizzazione:** L'indicizzazione è il processo di creazione di indici o puntatori che consentono un accesso rapido e efficiente ai dati. Gli indici vengono creati su colonne o attributi comuni e accelerano le operazioni di ricerca e filtraggio dei dati. L'indicizzazione è particolarmente utile quando si lavora con grandi volumi di dati.

**Strumenti di gestione dei database:** L'utilizzo di sistemi di gestione dei database (DBMS) fornisce funzionalità avanzate per organizzare e gestire i dati. I DBMS offrono strumenti per creare tabelle, definire relazioni tra le tabelle, eseguire query complesse e gestire l'integrità dei dati.

L'archiviazione dei dati è un aspetto fondamentale della gestione dei dati ed è il processo di conservazione dei dati in modo sicuro e accessibile. L'obiettivo dell'archiviazione dei dati è quello di garantire che i dati siano conservati a lungo termine, protetti da perdite, danni o accessi non autorizzati, e facilmente recuperabili quando necessario. Di seguito sono descritte alcune considerazioni importanti per l'archiviazione dei dati:

Scelta del sistema di archiviazione: Esistono diverse opzioni per l'archiviazione dei dati, tra cui:

Archiviazione su supporti fisici: Questa opzione prevede l'utilizzo di dispositivi di archiviazione fisici, come dischi rigidi interni o esterni, unità USB, nastro magnetico o dischi ottici. Tuttavia, questa soluzione richiede spazio fisico e può essere soggetta a guasti meccanici o danni fisici.

Archiviazione su server locali: Consiste nell'utilizzo di server dedicati all'interno dell'organizzazione per archiviare i dati. Questa opzione richiede la gestione e la manutenzione dei server, nonché un backup regolare per prevenire la perdita di dati.

Archiviazione in data center: I data center sono strutture specializzate che offrono servizi di archiviazione e gestione dei dati. Le organizzazioni possono utilizzare i servizi di un data center per ospitare i propri server o affidare l'archiviazione dei dati completamente al data center.

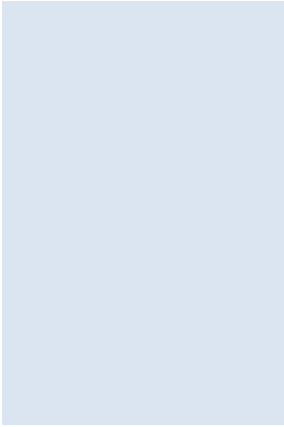
Archiviazione nel cloud: L'archiviazione nel cloud prevede il salvataggio dei dati su server remoti gestiti da fornitori di servizi cloud. Questa soluzione offre flessibilità, scalabilità e accessibilità da diverse posizioni geografiche, senza la necessità di gestire l'infrastruttura hardware.

Backup regolare dei dati: Il backup dei dati è un'attività cruciale per proteggere i dati da perdite accidentali o catastrofi. È consigliabile effettuare backup regolari dei dati e archivarli in un luogo separato per garantire una copia di sicurezza in caso di problemi con l'archiviazione primaria dei dati.

Sicurezza dei dati: La sicurezza dei dati è un aspetto critico dell'archiviazione dei dati. È importante implementare misure di sicurezza per proteggere i dati da accessi non autorizzati, furto, hacking o danni fisici. Queste misure possono includere crittografia dei dati, accesso controllato con autenticazione e autorizzazione, monitoraggio dei sistemi e delle attività, e protezione fisica dei server o dei dispositivi di archiviazione.

Gestione del ciclo di vita dei dati: I dati hanno un ciclo di vita che va dalla loro creazione o acquisizione alla loro eliminazione. È importante definire politiche e procedure per gestire il ciclo di vita dei dati, compresa l'archiviazione a lungo termine, l'eliminazione sicura dei dati obsoleti e la conservazione dei dati che devono essere mantenuti per ragioni normative o legali.

Conformità normativa: Le organizzazioni devono assicurarsi che l'archiviazione dei dati sia conforme alle normative e ai regolamenti applicabili, come le leggi sulla privacy dei dati o le normative settoriali. Ciò può richiedere l'implementazione di



misure aggiuntive di sicurezza e di procedure di gestione dei dati per garantire la conformità.

L'archiviazione dei dati deve essere progettata in modo accurato per garantire la disponibilità, l'integrità e la sicurezza dei dati a lungo termine, fornendo un accesso rapido ed efficiente quando necessario.



La sicurezza dei dati è un aspetto critico nella gestione dei dati e si riferisce alle misure e alle pratiche adottate per proteggere i dati da accessi non autorizzati, perdite, alterazioni o divulgazioni indebite. La sicurezza dei dati è di fondamentale importanza per garantire la privacy, la riservatezza e l'integrità delle informazioni sensibili.

Ecco alcuni concetti e pratiche chiave relativi alla sicurezza dei dati:

**Accesso controllato:** È importante stabilire controlli di accesso appropriati per limitare l'accesso ai dati solo alle persone autorizzate. Ciò può includere l'implementazione di autenticazione forte, come l'utilizzo di password complesse, l'autenticazione a due fattori o l'utilizzo di tecnologie biometriche. È anche importante definire ruoli e privilegi di accesso basati sul principio del "bisogno di sapere", in modo che solo le persone necessarie abbiano accesso ai dati sensibili.

**Crittografia dei dati:** La crittografia è un metodo di protezione dei dati tramite l'utilizzo di algoritmi per trasformare i dati in una forma illeggibile, chiamata testo cifrato, che può essere decifrato solo da chi dispone delle chiavi di crittografia corrette. La crittografia può essere applicata sia durante la trasmissione dei dati attraverso reti o canali non sicuri, come Internet, sia durante l'archiviazione dei dati per proteggerli da accessi non autorizzati in caso di violazione fisica o digitale.

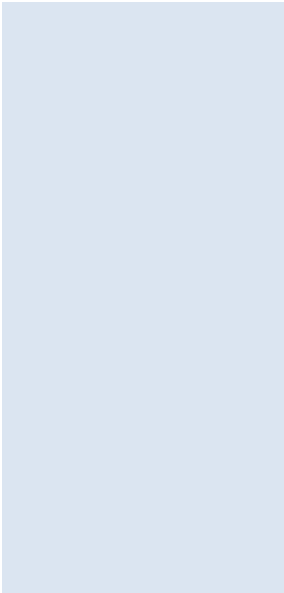
**Backup e ripristino dei dati:** I backup regolari dei dati sono fondamentali per garantire la disponibilità dei dati in caso di incidenti o catastrofi. È importante effettuare backup frequenti e memorizzarli in luoghi separati dal sistema primario. Inoltre, è necessario testare periodicamente i processi di ripristino per verificare l'efficacia del ripristino dei dati in caso di necessità.

**Monitoraggio e rilevamento delle minacce:** La sorveglianza costante dei sistemi e dei dati è essenziale per individuare tempestivamente eventuali violazioni o attività sospette. L'implementazione di strumenti di monitoraggio e rilevamento delle minacce consente di identificare comportamenti anomali o attacchi in corso e di adottare misure correttive immediatamente.

**Formazione e consapevolezza degli utenti:** La formazione degli utenti sulle best practice di sicurezza dei dati è fondamentale per prevenire incidenti causati da errori umani o da azioni non intenzionali. Gli utenti devono essere istruiti su come creare password forti, evitare phishing o altre truffe online, utilizzare in modo sicuro i dispositivi e condividere i dati solo con le persone autorizzate.

**Politiche di gestione dei dati:** Le politiche e le procedure di gestione dei dati devono essere definite e implementate per garantire un'adeguata protezione dei dati. Ciò può includere l'adozione di politiche di accesso e utilizzo dei dati, la gestione dei diritti di accesso degli utenti, la classificazione dei dati in base alla loro sensibilità e l'implementazione di controlli di sicurezza adeguati in base alle esigenze dell'organizzazione.

**Aggiornamenti e patch di sicurezza:** Mantenere i sistemi operativi, le applicazioni e gli strumenti di sicurezza aggiornati con le ultime patch di sicurezza è fondamentale per ridurre le



vulnerabilità e proteggere i dati da exploit noti. È importante adottare un processo regolare di aggiornamento e patching per mantenere i sistemi sicuri.

La sicurezza dei dati è un approccio continuo che richiede una combinazione di misure tecniche, processi di gestione e consapevolezza degli utenti. Le organizzazioni devono valutare costantemente i loro ambienti di dati, identificare le minacce potenziali e adottare misure adeguate per proteggere i dati da accessi non autorizzati o dannosi.

## Gestione del ciclo di vita dei dati

La gestione del ciclo di vita dei dati si riferisce alla pianificazione e all'esecuzione di attività che gestiscono i dati in tutte le fasi del loro ciclo di vita, dalla creazione o acquisizione, alla conservazione, alla distribuzione e infine all'eliminazione. La gestione del ciclo di vita dei dati è importante per garantire un utilizzo efficiente delle risorse, la conformità normativa e la riduzione del rischio associato alla conservazione e alla gestione dei dati.

Ecco le fasi principali del ciclo di vita dei dati e le relative attività:

**Creazione o acquisizione:** Questa fase riguarda la creazione dei dati da fonti interne o l'acquisizione di dati da fonti esterne. Durante questa fase, possono essere definite politiche per la raccolta dei dati e le pratiche di acquisizione per garantire la qualità e l'integrità dei dati.

**Archiviazione e conservazione:** Dopo la creazione o l'acquisizione, i dati vengono archiviati in un sistema di archiviazione appropriato. Durante questa fase, vengono definite politiche e procedure per l'archiviazione dei dati, come la scelta del sistema di archiviazione, la definizione dei tempi di conservazione dei dati e l'implementazione delle misure di sicurezza appropriate per proteggere i dati durante l'archiviazione.

**Utilizzo e distribuzione:** Durante questa fase, i dati vengono utilizzati per scopi specifici, come l'analisi, la generazione di

report o la condivisione con le parti interessate. È importante definire le regole di accesso e utilizzo dei dati, stabilire i diritti di accesso degli utenti e monitorare l'uso dei dati per garantire la sicurezza e la conformità normativa.

Manutenzione e aggiornamento: Durante il ciclo di vita dei dati, potrebbe essere necessario eseguire attività di manutenzione e aggiornamento per garantire l'integrità e la qualità dei dati. Ciò può includere l'applicazione di patch di sicurezza, la pulizia dei dati duplicati o obsoleti, l'aggiornamento delle informazioni e l'aggiornamento dei metadati associati ai dati.

Archiviazione a lungo termine e dismissione: A un certo punto, i dati potrebbero non essere più necessari per scopi operativi o legali. Durante questa fase, vengono definiti processi e procedure per l'archiviazione a lungo termine dei dati che devono essere conservati per ragioni normative o legali. Inoltre, quando i dati non sono più richiesti, è importante definire procedure di dismissione dei dati che garantiscono l'eliminazione sicura e definitiva dei dati.

La gestione del ciclo di vita dei dati richiede un'adeguata pianificazione e governance per garantire che i dati siano gestiti in modo appropriato in tutte le fasi del loro ciclo di vita. Ciò contribuisce a garantire la qualità dei dati, a ridurre i rischi di sicurezza e conformità e a ottimizzare l'utilizzo delle risorse.

## Condivisione dei dati

La condivisione dei dati si riferisce alla pratica di trasferire o consentire l'accesso ai dati da una parte all'altra, che può essere tra individui, organizzazioni o sistemi informatici. La condivisione dei dati può avvenire sia internamente, all'interno di un'organizzazione, che esternamente, con parti esterne all'organizzazione.

Ecco alcuni aspetti chiave da considerare nella condivisione dei dati:

**Autorizzazione e controllo dell'accesso:** È importante stabilire chiare politiche di accesso e definire i diritti di accesso agli utenti o alle parti interessate. Ciò implica determinare chi ha il permesso di accedere ai dati, quali dati possono essere acceduti e in che modo possono essere utilizzati. È consigliabile utilizzare strumenti di controllo dell'accesso come le liste di controllo degli accessi (ACL) o i meccanismi di gestione dei diritti per garantire che solo le persone autorizzate abbiano accesso ai dati.

**Conformità normativa e legale:** Nella condivisione dei dati è fondamentale rispettare le normative e le leggi applicabili. Questo può includere l'aderenza alle leggi sulla privacy dei dati, come il Regolamento generale sulla protezione dei dati (GDPR) o il California Consumer Privacy Act (CCPA), nonché altre normative settoriali o specifiche dell'industria. È necessario assicurarsi di ottenere il consenso appropriato e di

conformarsi ai requisiti di protezione dei dati al momento della condivisione dei dati.

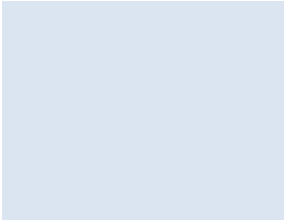
**Sicurezza dei dati:** La sicurezza dei dati è di vitale importanza nella condivisione dei dati per proteggere i dati da accessi non autorizzati, divulgazioni indebite o alterazioni. È necessario adottare misure di sicurezza adeguate, come la crittografia dei dati durante la trasmissione, l'uso di connessioni sicure e l'implementazione di politiche di sicurezza robuste per proteggere i dati durante la condivisione.

**Condivisione strutturata dei dati:** Per garantire una corretta interpretazione e utilizzo dei dati condivisi, è importante definire una struttura e un formato comuni per i dati. Ciò può includere l'utilizzo di standard di settore o la creazione di formati di dati specifici che facilitano l'interpretazione e l'integrazione dei dati da parte delle parti interessate.

**Gestione dei metadati:** I metadati sono informazioni aggiuntive associate ai dati che forniscono una descrizione e un contesto ai dati stessi. È consigliabile definire una strategia per la gestione dei metadati durante la condivisione dei dati, in modo che le parti interessate possano comprendere il significato e il contesto dei dati condivisi.

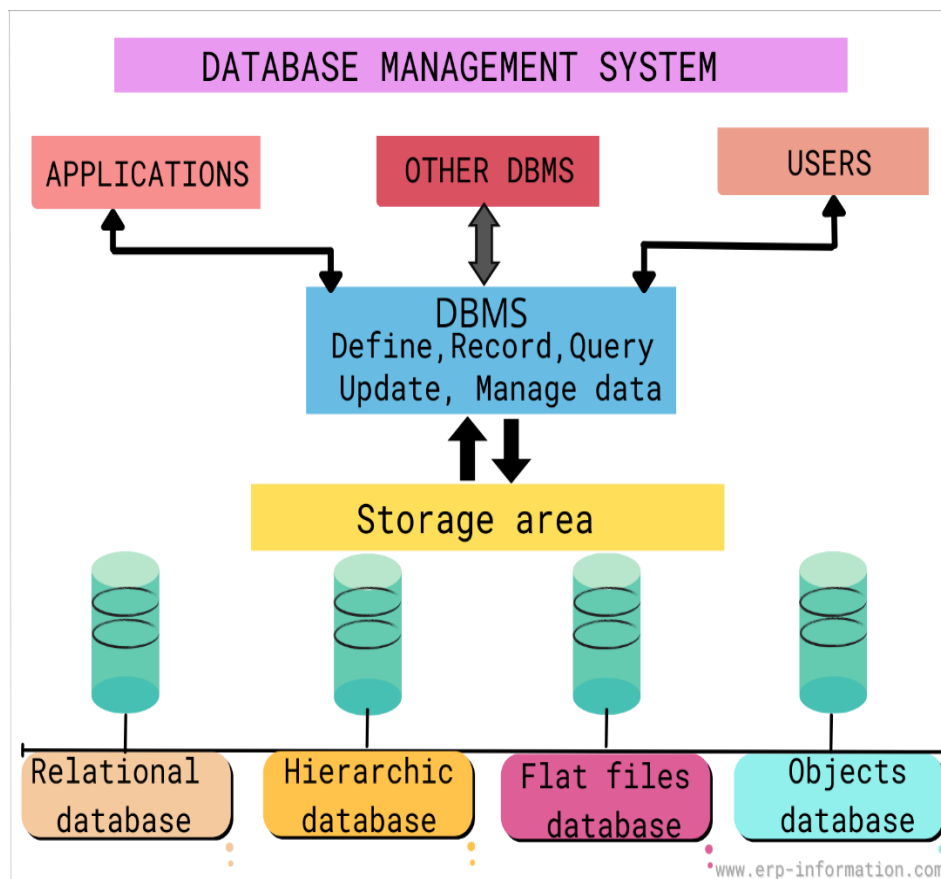
**Accordi di condivisione dei dati:** È consigliabile stabilire accordi o contratti di condivisione dei dati che delineino i termini e le condizioni della condivisione dei dati. Questi accordi possono includere aspetti come le finalità della condivisione dei dati, le responsabilità delle parti coinvolte, le restrizioni di utilizzo e le clausole di riservatezza.

La condivisione dei dati può offrire vantaggi significativi, come la collaborazione, l'innovazione e l'ottimizzazione delle risorse.



Tuttavia, è importante adottare misure adeguate per garantire la sicurezza, la conformità e la gestione corretta dei dati durante il processo di condivisione.





## I SISTEMI DI GESTIONE DEI DATABASE

## I sistemi di gestione dei database

I sistemi di gestione dei database (DBMS, Database Management Systems) sono software progettati per gestire l'organizzazione, l'archiviazione, l'accesso e la gestione dei dati all'interno di un database. I DBMS forniscono un'interfaccia tra gli utenti o le applicazioni e il database, consentendo loro di manipolare i dati in modo efficiente e sicuro. Ecco alcuni concetti chiave correlati ai DBMS:

**Struttura del database:** I DBMS consentono di definire la struttura dei dati nel database, come le tabelle, gli attributi e le relazioni. Gli utenti possono creare, modificare o eliminare tabelle e definire i vincoli di integrità per garantire la coerenza e la validità dei dati.

**Linguaggio di interrogazione:** I DBMS forniscono un linguaggio di interrogazione, come SQL (Structured Query Language), che consente agli utenti di formulare query per estrarre, inserire, aggiornare o eliminare dati dal database. Il linguaggio di interrogazione offre un'interfaccia standardizzata per interagire con il database.

**Gestione della concorrenza e transazionale:** I DBMS gestiscono la concorrenza quando più utenti o applicazioni cercano di accedere e modificare i dati contemporaneamente. Utilizzano tecniche come il locking per garantire la coerenza dei dati e prevenire problemi come la lettura sporca o

l'aggiornamento anomalo. Inoltre, i DBMS supportano le transazioni, che consentono di raggruppare un insieme di operazioni come un'unità indivisibile, garantendo la loro esecuzione corretta o il rollback in caso di errori.

Ottimizzazione delle query: I DBMS analizzano le query formulate dagli utenti e determinano il piano di esecuzione ottimale per recuperare i dati richiesti in modo efficiente. Utilizzano algoritmi di ottimizzazione per valutare diverse strategie di esecuzione e selezionare quella più efficiente in termini di tempo di risposta.

Sicurezza dei dati: I DBMS offrono funzionalità di sicurezza per proteggere i dati all'interno del database. Consentono di definire autorizzazioni di accesso per gli utenti o i gruppi di utenti, stabilendo chi può accedere a quali dati. Inoltre, i DBMS possono fornire meccanismi di crittografia dei dati per proteggere i dati sensibili durante la memorizzazione o la trasmissione.

Backup e ripristino: I DBMS consentono di eseguire backup periodici dei dati del database per garantire la disponibilità e la protezione dei dati in caso di guasti hardware, errori umani o altri eventi catastrofici. In caso di perdita di dati, i DBMS offrono funzionalità di ripristino per recuperare i dati dal backup e ripristinarli allo stato consistente più recente.

Scalabilità e performance: I DBMS devono essere in grado di gestire grandi quantità di dati e supportare un alto numero di utenti concorrenti. Devono fornire meccanismi per la scalabilità del database, come partizionamento dei dati o replica, per distribuire il carico di lavoro su più server e migliorare

## Structured Query Language

Structured Query Language (SQL) è un linguaggio di programmazione standardizzato utilizzato per la gestione dei database relazionali. SQL consente di creare, modificare e interrogare i dati all'interno di un database utilizzando un insieme di istruzioni. Ecco alcuni concetti fondamentali relativi a SQL:

Categorie di istruzioni SQL: Le istruzioni SQL possono essere classificate in diverse categorie:

DDL (Data Definition Language): Queste istruzioni vengono utilizzate per definire e modificare la struttura del database, come creare, modificare o eliminare tabelle, vincoli, indici, viste, ecc.

DML (Data Manipulation Language): Queste istruzioni consentono di manipolare i dati all'interno del database, come inserire, aggiornare, eliminare o recuperare record dalle tabelle.

DQL (Data Query Language): Queste istruzioni vengono utilizzate per interrogare i dati all'interno del database e recuperare informazioni specifiche, utilizzando comandi come SELECT, FROM, WHERE, ecc.

DCL (Data Control Language): Queste istruzioni gestiscono i diritti di accesso e le autorizzazioni sui dati, come concedere o revocare i privilegi di accesso agli utenti.

Istruzioni SQL comuni: Alcune delle istruzioni SQL più comuni includono:

SELECT: Utilizzata per recuperare dati specifici da una o più tabelle all'interno del database.

INSERT: Utilizzata per inserire nuovi dati in una tabella.

UPDATE: Utilizzata per modificare i dati esistenti all'interno di una tabella.

DELETE: Utilizzata per eliminare dati da una tabella.

CREATE: Utilizzata per creare nuove tabelle, viste, indici, ecc.

ALTER: Utilizzata per modificare la struttura di una tabella esistente.

DROP: Utilizzata per eliminare tabelle, viste, indici, ecc.

GRANT: Utilizzata per concedere diritti di accesso agli utenti.

REVOKE: Utilizzata per revocare i diritti di accesso agli utenti.

Clausole SQL: SQL utilizza varie clausole per filtrare e specificare i criteri di interrogazione dei dati. Alcune clausole comuni includono:

WHERE: Utilizzata per specificare i criteri di selezione dei dati in una query.

ORDER BY: Utilizzata per ordinare i risultati della query in base a una o più colonne.

GROUP BY: Utilizzata per raggruppare i risultati della query in base a una o più colonne.

HAVING: Utilizzata per filtrare i gruppi di dati sulla base di condizioni specificate.

JOIN: Utilizzata per combinare dati provenienti da più tabelle in una singola query.

SQL è un linguaggio molto potente e flessibile per la gestione dei dati nei database relazionali. La sua sintassi standardizzata consente agli sviluppatori di interagire con i dati in modo efficiente e coerente, indipendentemente dal DBMS specifico utilizzato.

## DDL (Data Definition Language)

DDL (Data Definition Language) è una categoria di istruzioni SQL utilizzate per definire e gestire la struttura degli oggetti di un database, come tabelle, viste, indici, vincoli e altri elementi di schema. Le istruzioni DDL consentono di creare, modificare o eliminare gli oggetti del database. Ecco alcune delle istruzioni DDL comuni:

**CREATE:** Utilizzata per creare nuovi oggetti nel database, come tabelle, viste, indici, vincoli, procedure, trigger, ecc.

Esempio di creazione di una tabella:

```
CREATE TABLE nome_tabella (  
    colonna1 tipo_dato,  
    colonna2 tipo_dato,  
    ...  
);
```

**ALTER:** Utilizzata per modificare la struttura degli oggetti esistenti nel database, come tabelle, viste o indici. Le modifiche possono includere l'aggiunta, la modifica o l'eliminazione di colonne o vincoli.

Esempio di aggiunta di una colonna a una tabella esistente:

`ALTER TABLE nome_tabella`

`ADD colonna3 tipo_dato;`

**DROP:** Utilizzata per eliminare gli oggetti dal database, come tabelle, viste, indici o vincoli.

Esempio di eliminazione di una tabella:

`DROP TABLE nome_tabella;`

**TRUNCATE:** Utilizzata per eliminare tutti i dati di una tabella, mantenendo la sua struttura.

Esempio di troncamento di una tabella:

`TRUNCATE TABLE nome_tabella;`

**RENAME:** Utilizzata per rinominare un oggetto nel database, come una tabella o una colonna.

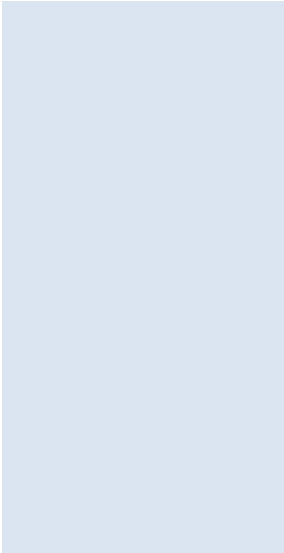
Esempio di rinominazione di una tabella:

`ALTER TABLE vecchio_nome RENAME TO  
nuovo_nome;`

**COMMENT:** Utilizzata per aggiungere commenti o descrizioni agli oggetti del database, come tabelle o colonne.

Esempio di aggiunta di un commento a una colonna:





```
COMMENT ON COLUMN nome_tabella.colonna IS  
'Descrizione della colonna';
```

Le istruzioni DDL consentono agli amministratori di database e agli sviluppatori di definire e modificare la struttura degli oggetti nel database in base alle esigenze del sistema. Forniscono flessibilità per la gestione dello schema del database e consentono di creare una struttura di dati coerente e ben organizzata.

## DML (Data Manipulation Language)

DML (Data Manipulation Language) è una categoria di istruzioni SQL utilizzate per manipolare i dati all'interno di un database. Le istruzioni DML consentono di inserire, modificare, eliminare o recuperare i dati dalle tabelle del database. Ecco alcune delle istruzioni DML più comuni:

**SELECT:** Utilizzata per recuperare dati da una o più tabelle del database. È possibile specificare le colonne da selezionare, le tabelle da cui ottenere i dati e le condizioni per filtrare i risultati. Esempio di selezione di tutte le colonne da una tabella:

```
SELECT * FROM nome_tabella;
```

**INSERT:** Utilizzata per inserire nuovi dati in una tabella. È possibile specificare le colonne in cui inserire i dati e i valori corrispondenti.

Esempio di inserimento di dati in una tabella:

```
INSERT INTO nome_tabella (colonna1, colonna2, ...)
VALUES (valore1, valore2, ...);
```

**UPDATE:** Utilizzata per modificare i dati esistenti all'interno di una tabella. È possibile specificare le colonne da aggiornare e i nuovi valori.

Esempio di aggiornamento di dati in una tabella:

```
UPDATE nome_tabella SET colonna1 = nuovo_valore  
WHERE condizione;
```

DELETE: Utilizzata per eliminare i dati da una tabella. È possibile specificare le righe da eliminare utilizzando una condizione.

Esempio di eliminazione di righe da una tabella:

```
DELETE FROM nome_tabella WHERE condizione;
```

Le istruzioni DML consentono di manipolare i dati all'interno del database in modo efficace. Possono essere utilizzate per inserire nuovi dati, aggiornare informazioni esistenti o eliminare dati non necessari. La flessibilità delle istruzioni DML consente di modificare il contenuto del database in base alle necessità dell'applicazione o dell'utente.

## DQL (Data Query Language)

DQL (Data Query Language) è una categoria di istruzioni SQL utilizzate per interrogare i dati all'interno di un database e recuperare informazioni specifiche. Le istruzioni DQL consentono di formulare query per selezionare e recuperare dati da una o più tabelle del database. Ecco alcune delle istruzioni DQL più comuni:

**SELECT:** È l'istruzione principale utilizzata per recuperare dati da una o più tabelle del database. È possibile specificare le colonne da selezionare, le tabelle da cui ottenere i dati e le condizioni per filtrare i risultati.

Esempio di selezione di tutte le colonne da una tabella:

```
SELECT * FROM nome_tabella;
```

Esempio di selezione specifica di colonne da una tabella:

```
SELECT colonna1, colonna2 FROM nome_tabella;
```

**WHERE:** Utilizzata per filtrare i dati basandosi su una o più condizioni. Le condizioni possono includere operatori di confronto, operatori logici e funzioni di confronto.

Esempio di selezione di dati con una condizione:

`SELECT * FROM nome_tabella WHERE condizione;`  
**ORDER BY:** Utilizzata per ordinare i risultati della query in base a una o più colonne. È possibile specificare l'ordine crescente (ASC) o decrescente (DESC).

Esempio di selezione di dati ordinati:

```
SELECT * FROM nome_tabella ORDER BY colonna  
ASC;
```

**GROUP BY:** Utilizzata per raggruppare i risultati della query in base a una o più colonne. È spesso utilizzata insieme alle funzioni di aggregazione, come SUM, COUNT, AVG, ecc.

Esempio di selezione di dati raggruppati:

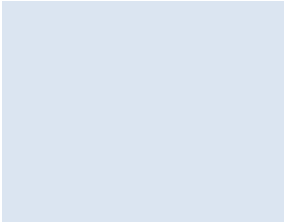
```
SELECT colonna, COUNT(*) FROM nome_tabella  
GROUP BY colonna;
```

**HAVING:** Utilizzata per filtrare i gruppi di dati basandosi su una o più condizioni dopo l'utilizzo di GROUP BY. Funziona in modo simile a WHERE, ma viene applicata ai gruppi invece che alle singole righe.

Esempio di selezione di dati raggruppati con una condizione:

```
SELECT colonna, COUNT(*) FROM nome_tabella  
GROUP BY colonna HAVING COUNT(*) > 10;
```

Le istruzioni DQL consentono di formulare query complesse per estrarre informazioni specifiche dai dati del database. Possono essere utilizzate per filtrare, ordinare, raggruppare e



limitare i risultati in base alle esigenze dell'applicazione o dell'utente. La potenza delle istruzioni DQL consente di ottenere dati rilevanti e significativi dal database.

## DCL (Data Control Language)

DCL (Data Control Language) è una categoria di istruzioni SQL utilizzate per gestire i diritti di accesso e le autorizzazioni sui dati all'interno di un database. Le istruzioni DCL consentono di concedere o revocare privilegi agli utenti e di controllare l'accesso ai dati. Ecco alcune delle istruzioni DCL più comuni:

**GRANT:** Utilizzata per concedere privilegi agli utenti o ai ruoli. I privilegi possono includere la possibilità di eseguire operazioni come SELECT, INSERT, UPDATE, DELETE, CREATE, DROP, ecc.

Esempio di concessione di privilegi ad un utente:

```
GRANT privilegio1, privilegio2, ... ON nome_tabella TO  
nome_utente;
```

**REVOKE:** Utilizzata per revocare i privilegi precedentemente concessi agli utenti o ai ruoli.

Esempio di revoca di privilegi ad un utente:

```
REVOKE privilegio1, privilegio2, ... ON nome_tabella  
FROM nome_utente;
```

DENY: Utilizzata per negare specifici privilegi agli utenti o ai ruoli. La negazione ha la priorità sui privilegi concessi e non può essere annullata.

Esempio di negazione di privilegi ad un utente:

```
DENY privilegio1, privilegio2, ... ON nome_tabella TO  
nome_utente;
```

SET ROLE: Utilizzata per impostare un ruolo attivo per l'utente corrente. I ruoli possono avere privilegi specifici associati ad essi.

Esempio di impostazione di un ruolo attivo:

```
SET ROLE nome_ruolo;
```

Le istruzioni DCL consentono di controllare i diritti di accesso e le autorizzazioni dei dati nel database. Consentono agli amministratori di database di gestire i privilegi degli utenti, garantendo che solo gli utenti autorizzati possano eseguire determinate operazioni sui dati. Questo contribuisce a garantire la sicurezza e l'integrità del database, proteggendo i dati sensibili e impedendo accessi non autorizzati.





## SQL

# Linguaggio SQL: Fondamenti e Esempi

1. Introduzione a SQL
2. Creazione di tabelle
3. Inserimento dei dati
4. Interrogazione dei dati: SELECT
5. Filtraggio dei dati: WHERE
6. Ordinamento dei dati: ORDER BY
7. Unione di tabelle: JOIN
8. Aggregazione dei dati: GROUP BY
9. Filtraggio avanzato: HAVING
10. Sottoselezioni: Subquery
11. Modifica dei dati: UPDATE
12. Eliminazione dei dati: DELETE
13. Transazioni e commit
14. Viste e stored procedure
15. Ottimizzazione delle query

## Introduzione:

Linguaggio SQL: Fondamenti e Esempi è un'ampia guida che ti accompagnerà nel mondo del linguaggio SQL, fornendoti una solida base di conoscenza e numerosi esempi pratici. Che tu sia un principiante assoluto o un programmatore esperto che vuole approfondire le proprie competenze.

Il linguaggio SQL (Structured Query Language) è uno strumento fondamentale per lavorare con i database relazionali. Con SQL, è possibile creare, modificare e interrogare i dati in modo efficiente e intuitivo. Questo libro ti guiderà passo dopo passo attraverso i concetti chiave del linguaggio SQL, offrendoti una comprensione approfondita delle sue caratteristiche principali.

Nel capitolo 1, "Introduzione a SQL", esploreremo i concetti di base del linguaggio SQL, compresi i tipi di dati, le istruzioni DDL (Data Definition Language) e le istruzioni DML (Data Manipulation Language). Ti mostreremo anche come connetterti a un database e come eseguire le prime semplici query.

Nel capitolo 2, "Creazione di tabelle", imparerai come progettare e creare tabelle per archiviare i dati. Discuteremo i diversi tipi di dati supportati da SQL e ti guideremo nella creazione di tabelle con vincoli di integrità referenziale.

Nel capitolo 3, "Inserimento dei dati", affronteremo il processo di inserimento dei dati nelle tabelle. Ti mostreremo diverse modalità per inserire i dati e ti forniremo consigli su come evitare errori comuni.

Nel capitolo 4, "Interrogazione dei dati: SELECT", ci concentreremo sulle istruzioni SELECT, che sono fondamentali per recuperare i dati da un database. Ti mostreremo come selezionare colonne specifiche, ordinare i risultati e utilizzare l'operatore DISTINCT per rimuovere i duplicati.

Nel capitolo 5, "Filtraggio dei dati: WHERE", imparerai a utilizzare l'istruzione WHERE per filtrare i dati in base a determinate condizioni. Ti guideremo attraverso l'uso di operatori logici, operatori di confronto e funzioni di confronto delle stringhe per ottenere i risultati desiderati.

Nel capitolo 6, "Ordinamento dei dati: ORDER BY", esploreremo come ordinare i risultati delle query in base a criteri specifici. Ti mostreremo come utilizzare l'istruzione ORDER BY per ordinare i dati in modo ascendente o discendente, sia per una singola colonna che per più colonne.

Nel capitolo 7, "Unione di tabelle: JOIN", imparerai a combinare i dati provenienti da diverse tabelle utilizzando le clausole JOIN. Esploreremo i diversi tipi di JOIN, come INNER JOIN, LEFT JOIN, RIGHT JOIN e FULL JOIN, e ti forniremo esempi pratici per illustrarne l'utilizzo corretto.

Nel capitolo 8, "Aggregazione dei dati: GROUP BY", affronteremo l'aggregazione dei dati utilizzando l'istruzione GROUP BY. Ti mostreremo come raggruppare i dati in base a determinate colonne e applicare funzioni di aggregazione come SUM, COUNT, AVG, MIN e MAX per ottenere informazioni sintetiche sui dati.

Nel capitolo 9, "Filtraggio avanzato: HAVING", esploreremo l'istruzione HAVING, che viene utilizzata insieme all'istruzione GROUP BY per filtrare i risultati dell'aggregazione. Ti mostreremo come applicare condizioni specifiche sui gruppi di dati utilizzando HAVING.

Nel capitolo 10, "Sottoselezioni: Subquery", imparerai a utilizzare le subquery, ovvero query annidate all'interno di altre query. Ti mostreremo come utilizzare le subquery per ottenere risultati più complessi, eseguire confronti tra le tabelle e filtrare i dati in base ai risultati di altre query.

Nel capitolo 11, "Modifica dei dati: UPDATE", affronteremo l'aggiornamento dei dati all'interno di una tabella utilizzando l'istruzione UPDATE. Ti guideremo attraverso il processo di aggiornamento di singole righe o di più righe contemporaneamente, utilizzando clausole di filtro specifiche.

Nel capitolo 12, "Eliminazione dei dati: DELETE", imparerai come eliminare dati da una tabella utilizzando l'istruzione DELETE. Ti forniremo le linee guida per eliminare singole righe o intere tabelle e ti avviseremo sui potenziali rischi associati a un'eliminazione non corretta.

Nel capitolo 13, "Transazioni e commit", esploreremo il concetto di transazione nel contesto del linguaggio SQL. Ti mostreremo come utilizzare le transazioni per eseguire una serie di operazioni come un'unica unità atomica e come utilizzare l'istruzione COMMIT per confermare le modifiche effettuate o l'istruzione ROLLBACK per annullarle in caso di errore.

Nel capitolo 14, "Viste e stored procedure", affronteremo le viste, che sono query salvate come oggetti di database, e le stored procedure, che sono insiemi di istruzioni SQL predefinite. Ti mostreremo come creare viste e stored

procedure e come utilizzarle per semplificare le operazioni di interrogazione e modifica dei dati.

Nel capitolo 15, "Ottimizzazione delle query", affronteremo le strategie per ottimizzare le prestazioni delle query SQL. Ti mostreremo come utilizzare gli indici per accelerare l'esecuzione delle query, come ottimizzare le clausole WHERE e JOIN, e ti forniremo consigli pratici per migliorare le prestazioni complessive del tuo database.

Questo libro ti condurrà attraverso un percorso completo di apprendimento del linguaggio SQL, partendo dalle nozioni di base e arrivando a concetti più avanzati. Ogni capitolo è strutturato in modo chiaro e accessibile, con spiegazioni dettagliate, esempi pratici e suggerimenti utili. Inoltre, troverai esercizi alla fine di ciascun capitolo per mettere in pratica le tue conoscenze e consolidare ciò che hai imparato.

Che tu sia uno sviluppatore software, un analista di dati o un appassionato di database, "Linguaggio SQL: Fondamenti e Esempi" sarà la tua guida affidabile per acquisire una solida comprensione del linguaggio SQL e delle sue applicazioni pratiche. Con questo libro, sarai in grado di gestire e manipolare i dati in modo efficiente e ottenere informazioni rilevanti dai tuoi database.

Preparati a immergerti nel mondo del linguaggio SQL e ad ampliare le tue competenze nell'ambito della gestione dei dati. Sia che tu voglia creare nuovi database, eseguire complesse interrogazioni o ottimizzare le prestazioni delle tue applicazioni, "Linguaggio SQL: Fondamenti e Esempi" sarà il tuo compagno ideale per raggiungere i tuoi obiettivi.

Inizia il tuo viaggio di apprendimento ora e scopri il potenziale illimitato del linguaggio SQL per gestire e manipolare i dati in modo efficace. Buon viaggio nell'universo dell'SQL!

## **Capitolo 1: Introduzione a SQL**

SQL, acronimo di Structured Query Language, è un linguaggio di programmazione utilizzato per la gestione e l'interrogazione dei database relazionali. In questo primo capitolo, ti introdurremo ai concetti fondamentali di SQL e ti guideremo attraverso i primi passi per iniziare a utilizzarlo.

**1.1 Cosa è SQL?** SQL è un linguaggio standardizzato per lavorare con database relazionali. È ampiamente utilizzato in diversi contesti, come lo sviluppo di software, l'analisi dei dati e la gestione delle informazioni aziendali. Con SQL, è possibile creare, modificare e interrogare i dati all'interno di un database in modo efficiente e organizzato.

**1.2 Tipi di database supportati da SQL** SQL può essere utilizzato con diversi tipi di database relazionali, tra cui MySQL, PostgreSQL, Oracle, Microsoft SQL Server e molti altri. Anche

se ci possono essere alcune differenze sintattiche tra i vari sistemi di gestione dei database (DBMS), i principi fondamentali di SQL rimangono gli stessi.

1.3 Concetti fondamentali di SQL Prima di immergerci nei dettagli del linguaggio SQL, è importante comprendere alcuni concetti fondamentali:

- **Tabelle:** Le tabelle sono gli elementi centrali di un database relazionale. Rappresentano entità o oggetti del mondo reale e contengono i dati organizzati in righe e colonne. Ad esempio, potremmo avere una tabella "Clienti" che contiene informazioni su tutti i clienti di un'azienda.
- **Colonne:** Le colonne di una tabella rappresentano le diverse proprietà o attributi delle entità. Ad esempio, nella tabella "Clienti" potremmo avere colonne come "Nome", "Cognome", "Indirizzo" e così via.
- **Righe:** Le righe di una tabella contengono i dati specifici per ciascuna entità. Ogni riga rappresenta un record o un'istanza di un'entità. Ad esempio, una riga nella tabella "Clienti" potrebbe rappresentare un singolo cliente con i suoi dati personali.
- **Query:** Una query è una richiesta di informazioni o un'azione da eseguire su un database utilizzando il linguaggio SQL. Le query possono essere utilizzate per selezionare, inserire, aggiornare o eliminare dati all'interno delle tabelle.

1.4 Connessione a un database Per iniziare a utilizzare SQL, è necessario stabilire una connessione al database desiderato. Ciò può essere fatto utilizzando un client SQL o un'interfaccia di amministrazione fornita dal DBMS. Una volta stabilita la connessione, sarai pronto per iniziare a eseguire le prime query.

In questo capitolo, abbiamo fornito una panoramica generale di SQL e dei suoi concetti fondamentali. Nei capitoli successivi, esploreremo dettagliatamente i diversi aspetti del linguaggio SQL, fornendo esempi pratici e suggerimenti utili. Se sei pronto, proseguiamo con il capitolo successivo, in cui affronteremo la creazione di tabelle nel database

## **Capitolo 2: Creazione di tabelle**

Le tabelle sono gli elementi fondamentali di un database relazionale. Nel capitolo precedente, abbiamo introdotto brevemente il concetto di tabelle. In questo capitolo, approfondiremo la creazione di tabelle nel database utilizzando il linguaggio SQL.

**2.1 Sintassi della creazione della tabella** Per creare una tabella, utilizzeremo l'istruzione `CREATE TABLE` seguita dal nome desiderato per la tabella e l'elenco delle colonne con i rispettivi tipi di dati. La sintassi di base per la creazione di una tabella è la seguente:

```
CREATE TABLE nome_tabella (  
    colonna1 tipo_dato1,  
    colonna2 tipo_dato2,  
    ...  
);
```

Ad esempio, se volessimo creare una tabella chiamata "Clienti" con le colonne "ID", "Nome" e "Cognome", potremmo utilizzare la seguente istruzione:

```
CREATE TABLE Clienti (  
    ID INT,  
    Nome VARCHAR(50),  
    Cognome VARCHAR(50)  
);
```

**2.2 Tipi di dati delle colonne** Le colonne di una tabella possono essere di diversi tipi di dati, a seconda del tipo di informazioni che devono memorizzare. Alcuni dei tipi di dati comuni utilizzati in SQL includono:

- `INT`: un numero intero.
- `VARCHAR(n)`: una stringa di caratteri di lunghezza variabile, con una lunghezza massima di `n` caratteri.
- `DATE`: una data nel formato "YYYY-MM-DD".
- `DECIMAL(p, s)`: un numero decimale con `p` cifre totali e `s` cifre dopo la virgola.

Esistono molti altri tipi di dati supportati da SQL, come `FLOAT`, `BOOLEAN`, `TIMESTAMP`, e così via. È importante selezionare il tipo di dato appropriato in base alle esigenze del tuo database.

**2.3 Vincoli di integrità referenziale** Durante la creazione di una tabella, è possibile applicare vincoli di integrità referenziale per garantire la coerenza dei dati nel database. I vincoli di integrità referenziale definiscono le relazioni tra le tabelle e specificano come i dati devono essere trattati quando vengono effettuate operazioni di inserimento, aggiornamento o eliminazione.

Ad esempio, se abbiamo una tabella "Ordini" con una colonna "ID\_Cliente" che fa riferimento alla tabella "Clienti", possiamo definire un vincolo di chiave esterna per garantire che l'ID del cliente esista nella tabella "Clienti".

2.4 Esempi pratici Per comprendere meglio la creazione di tabelle, vediamo alcuni esempi pratici:

```
CREATE TABLE Prodotti (  
    ID INT PRIMARY KEY,  
    Nome VARCHAR(100),  
    Prezzo DECIMAL(8, 2),  
    Categoria VARCHAR(50)  
);
```

In questo esempio, stiamo creando una tabella "Prodotti" con le colonne "ID", "Nome", "Prezzo" e "Categoria". La colonna "ID" è definita come chiave primaria utilizzando il vincolo PRIMARY KEY.

```
CREATE TABLE Ordini (  
    ID INT PRIMARY KEY,  
    DataOrdine DATE,  
    ID_Cliente INT,  
    FOREIGN KEY (ID_Cliente) REFERENCES Clienti(ID)  
);
```

In questo secondo esempio, stiamo creando una tabella "Ordini" con le colonne "ID", "DataOrdine" e "ID\_Cliente". La colonna "ID\_Cliente" ha un vincolo di chiave esterna che fa riferimento all'ID nella tabella "Clienti".

2.5 Conclusioni La creazione di tabelle è un passo fondamentale nella progettazione di un database relazionale. In questo capitolo, abbiamo esplorato la sintassi per la creazione di tabelle utilizzando SQL, i tipi di dati delle colonne e l'applicazione di vincoli di integrità referenziale.

Ora che hai compreso come creare tabelle nel tuo database, sei pronto per passare al prossimo capitolo, in cui esploreremo come inserire dati all'interno delle tabelle utilizzando l'istruzione INSERT INTO.

### Capitolo 3: Inserimento dei dati

Nel capitolo precedente, abbiamo imparato a creare tabelle nel nostro database utilizzando SQL. Ora che abbiamo la struttura delle tabelle pronta, è il momento di inserire i dati effettivi all'interno di esse. In questo capitolo, esploreremo l'istruzione INSERT INTO per inserire nuove righe di dati nelle tabelle.

3.1 Sintassi dell'istruzione INSERT INTO L'istruzione INSERT INTO viene utilizzata per inserire nuove righe di dati all'interno di una tabella. La sintassi di base è la seguente:



```
INSERT INTO nome_tabella (colonna1, colonna2, ...)
VALUES (valore1, valore2, ...);
```

Dove "nome\_tabella" è il nome della tabella in cui desideri inserire i dati, "colonna1, colonna2, ..." sono le colonne a cui desideri assegnare i valori, e "valore1, valore2, ..." sono i valori da inserire nelle rispettive colonne.

3.2 Esempi pratici Per comprendere meglio l'istruzione INSERT INTO, vediamo alcuni esempi pratici:

```
INSERT INTO Clienti (Nome, Cognome, Email)
VALUES ('Mario', 'Rossi', 'mario@email.com');
```

In questo esempio, stiamo inserendo una nuova riga di dati nella tabella "Clienti". Stiamo specificando i valori per le colonne "Nome", "Cognome" ed "Email".

```
INSERT INTO Prodotti (Nome, Prezzo, Categoria)
VALUES ('iPhone 12', 999.99, 'Elettronica');
```

In questo secondo esempio, stiamo inserendo un nuovo prodotto nella tabella "Prodotti". Stiamo specificando i valori per le colonne "Nome", "Prezzo" e "Categoria".

3.3 Inserimento di dati da un'altra tabella In alcuni casi, potresti voler inserire dati nella tabella da un'altra tabella. Questo può essere fatto utilizzando un'istruzione SELECT insieme all'istruzione INSERT INTO. Ad esempio:

```
INSERT INTO ClientiPremium (ID_Cliente, Livello)
SELECT ID_Cliente, 'Premium'
FROM Clienti
WHERE ImportoTotaleSpeso > 1000;
```

In questo esempio, stiamo inserendo i clienti con un importo totale speso superiore a 1000 nella tabella "ClientiPremium". Stiamo selezionando l'ID del cliente dalla tabella "Clienti" e assegnando loro il livello "Premium".

3.4 Conclusioni L'istruzione INSERT INTO è uno strumento fondamentale per inserire dati all'interno delle tabelle di un database utilizzando SQL. In questo capitolo, abbiamo esplorato la sintassi di base per l'inserimento di dati e abbiamo visto alcuni esempi pratici.

Ora che sai come inserire i dati nel tuo database, sei pronto per passare al prossimo capitolo, in cui esploreremo

l'istruzione SELECT per interrogare i dati all'interno delle tabelle e ottenere informazioni specifiche.

## Capitolo 4: Interrogazione dei dati con SELECT

Nel capitolo precedente, abbiamo imparato a inserire dati all'interno delle tabelle del nostro database. Ora che i dati sono presenti, è il momento di imparare come interrogare i dati e ottenere informazioni specifiche. In questo capitolo, esploreremo l'istruzione SELECT, che è uno degli strumenti più potenti di SQL per l'interrogazione dei dati.

4.1 Sintassi dell'istruzione SELECT L'istruzione SELECT viene utilizzata per selezionare i dati da una o più tabelle del database. La sintassi di base è la seguente:

```
SELECT colonna1, colonna2, ...  
FROM nome_tabella;
```

Dove "colonna1, colonna2, ..." sono le colonne che desideri selezionare e "nome\_tabella" è il nome della tabella da cui desideri estrarre i dati. È anche possibile utilizzare l'istruzione SELECT con clausole aggiuntive come WHERE, ORDER BY e GROUP BY per filtrare, ordinare e raggruppare i dati.

4.2 Esempi pratici Per comprendere meglio l'istruzione SELECT, vediamo alcuni esempi pratici:

```
SELECT Nome, Cognome  
FROM Clienti;
```

In questo esempio, stiamo selezionando le colonne "Nome" e "Cognome" dalla tabella "Clienti". Questa query restituirà tutti i nomi e cognomi dei clienti presenti nel database.

```
SELECT *  
FROM Prodotti  
WHERE Prezzo > 1000;
```

In questo secondo esempio, stiamo selezionando tutte le colonne dalla tabella "Prodotti" per i prodotti con un prezzo superiore a 1000. Questa query restituirà tutti i dettagli dei prodotti che soddisfano il criterio specificato.

4.3 Utilizzo di clausole aggiuntive Come accennato in precedenza, è possibile utilizzare clausole aggiuntive per filtrare, ordinare e raggruppare i dati. Alcune delle clausole comuni che possono essere utilizzate con l'istruzione SELECT includono:

- WHERE: utilizzata per impostare condizioni per filtrare i dati in base a criteri specifici.
- ORDER BY: utilizzata per ordinare i risultati in base a una o più colonne.
- GROUP BY: utilizzata per raggruppare i dati in base ai valori di una o più colonne.

4.4 Conclusioni L'istruzione SELECT è uno strumento potente per interrogare i dati all'interno di un database utilizzando SQL. In questo capitolo, abbiamo esplorato la sintassi di base per l'utilizzo dell'istruzione SELECT e abbiamo visto alcuni esempi pratici.

Ora che sai come interrogare i dati, sei pronto per passare al prossimo capitolo, in cui esploreremo l'aggiornamento dei dati all'interno delle tabelle utilizzando l'istruzione UPDATE.

## Capitolo 5: Aggiornamento dei dati con UPDATE

Nel capitolo precedente, abbiamo appreso come interrogare i dati all'interno delle tabelle del nostro database. Ora è il momento di imparare come aggiornare i dati esistenti. In questo capitolo, esploreremo l'istruzione UPDATE, che ci consente di modificare i valori delle colonne all'interno delle tabelle.

5.1 Sintassi dell'istruzione UPDATE L'istruzione UPDATE viene utilizzata per aggiornare i valori delle colonne all'interno di una tabella. La sintassi di base è la seguente:

```
UPDATE nome_tabella
SET colonna1 = valore1, colonna2 = valore2, ...
WHERE condizione;
```

Dove "nome\_tabella" è il nome della tabella in cui desideri aggiornare i dati, "colonna1, colonna2, ..." sono le colonne che desideri modificare, "valore1, valore2, ..." sono i nuovi valori da assegnare alle rispettive colonne e "condizione" è una clausola opzionale che specifica quale righe devono essere aggiornate.

5.2 Esempi pratici Per comprendere meglio l'istruzione UPDATE, vediamo alcuni esempi pratici:

```
UPDATE Clienti
SET Email = 'nuovaemail@email.com'
WHERE ID = 1;
```

In questo esempio, stiamo aggiornando il valore della colonna "Email" nella tabella "Clienti" per il cliente con ID 1. Stiamo assegnando un nuovo indirizzo email al cliente.

```
UPDATE Prodotti
SET Prezzo = Prezzo * 1.1
WHERE Categoria = 'Elettronica';
```

In questo secondo esempio, stiamo aggiornando il valore della colonna "Prezzo" nella tabella "Prodotti" per i prodotti appartenenti alla categoria "Elettronica". Stiamo aumentando il prezzo di tutti questi prodotti del 10%.

**5.3 Utilizzo di clausole aggiuntive** Come accennato in precedenza, è possibile utilizzare clausole aggiuntive per specificare condizioni più complesse per l'aggiornamento dei dati. Alcune delle clausole comuni che possono essere utilizzate con l'istruzione UPDATE includono:

- **WHERE:** utilizzata per impostare condizioni per specificare quali righe devono essere aggiornate.
- **JOIN:** utilizzata per aggiornare le colonne basate su una combinazione tra più tabelle.

**5.4 Conclusioni** L'istruzione UPDATE è uno strumento essenziale per modificare i dati all'interno delle tabelle di un database utilizzando SQL. In questo capitolo, abbiamo esplorato la sintassi di base per l'utilizzo dell'istruzione UPDATE e abbiamo visto alcuni esempi pratici.

Ora che hai imparato come aggiornare i dati, sei pronto per passare al prossimo capitolo, in cui esploreremo l'eliminazione dei dati utilizzando l'istruzione DELETE.

## Capitolo 6: Eliminazione dei dati con DELETE

Nel capitolo precedente, abbiamo appreso come aggiornare i dati all'interno delle tabelle del nostro database. Ora è il momento di imparare come eliminare i dati che non sono più necessari. In questo capitolo, esploreremo l'istruzione DELETE, che ci consente di rimuovere righe di dati dalle tabelle.

**6.1 Sintassi dell'istruzione DELETE** L'istruzione DELETE viene utilizzata per eliminare righe di dati da una tabella. La sintassi di base è la seguente:

```
DELETE FROM nome_tabella
WHERE condizione;
```

Dove "nome\_tabella" è il nome della tabella da cui desideri eliminare le righe e "condizione" è una clausola opzionale che specifica quali righe devono essere eliminate. Se non viene specificata una condizione, tutte le righe della tabella verranno eliminate.

6.2 Esempi pratici Per comprendere meglio l'istruzione DELETE, vediamo alcuni esempi pratici:

```
DELETE FROM Clienti  
WHERE ID = 1;
```

In questo esempio, stiamo eliminando la riga corrispondente al cliente con ID 1 dalla tabella "Clienti". Questo significa che il cliente con ID 1 verrà completamente rimosso dal database.

```
DELETE FROM Prodotti  
WHERE Prezzo > 1000;
```

In questo secondo esempio, stiamo eliminando tutte le righe dalla tabella "Prodotti" per i prodotti con un prezzo superiore a 1000. Tutti i prodotti che soddisfano il criterio specificato verranno eliminati dal database.

6.3 Utilizzo di clausole aggiuntive Come accennato in precedenza, è possibile utilizzare clausole aggiuntive per specificare condizioni più complesse per l'eliminazione dei dati. Alcune delle clausole comuni che possono essere utilizzate con l'istruzione DELETE includono:

- WHERE: utilizzata per impostare condizioni per specificare quali righe devono essere eliminate.
- JOIN: utilizzata per eliminare righe basate su una combinazione tra più tabelle.

6.4 Conclusioni L'istruzione DELETE è uno strumento fondamentale per eliminare dati all'interno delle tabelle di un database utilizzando SQL. In questo capitolo, abbiamo esplorato la sintassi di base per l'utilizzo dell'istruzione DELETE e abbiamo visto alcuni esempi pratici.

Ora che hai appreso come eliminare i dati, sei pronto per passare al prossimo capitolo, in cui esploreremo l'importanza della creazione di indici per migliorare le prestazioni delle query.

## Capitolo 7: Creazione di indici per le prestazioni delle query

Nel capitolo precedente, abbiamo esplorato come eliminare dati dalle tabelle del nostro database. Ora è il momento di

concentrarci sul miglioramento delle prestazioni delle query. In questo capitolo, parleremo dell'importanza della creazione di indici e di come possono essere utilizzati per ottimizzare le query.

**7.1 Cosa sono gli indici** Gli indici sono strutture dati specializzate che vengono creati sulle colonne delle tabelle per accelerare le operazioni di ricerca e di interrogazione dei dati. Un indice fornisce un meccanismo di accesso rapido ai dati in base ai valori delle colonne indicizzate.

**7.2 Vantaggi degli indici** L'utilizzo degli indici offre diversi vantaggi per migliorare le prestazioni delle query. Alcuni dei principali vantaggi sono:

- **Miglioramento delle prestazioni delle query:** gli indici consentono di eseguire ricerche e interrogazioni sui dati in modo più efficiente, riducendo il tempo di esecuzione delle query.
- **Riduzione del carico sulle risorse del database:** grazie agli indici, le query richiedono meno risorse del database per l'esecuzione, migliorando l'efficienza complessiva del sistema.
- **Ottimizzazione dell'ordine dei dati:** gli indici consentono di organizzare i dati in un ordine specifico, facilitando l'accesso rapido ai dati ordinati.

**7.3 Creazione di indici** Per creare un indice su una colonna, è possibile utilizzare l'istruzione `CREATE INDEX`. Di seguito è riportata la sintassi di base:

```
CREATE INDEX nome_indice  
ON nome_tabella (colonna);
```

Dove "nome\_indice" è il nome da assegnare all'indice, "nome\_tabella" è il nome della tabella su cui creare l'indice e "colonna" è il nome della colonna da indicizzare.

**7.4 Utilizzo degli indici** Una volta creato un indice, il database utilizzerà automaticamente l'indice per ottimizzare le query quando possibile. Tuttavia, è importante considerare che l'utilizzo degli indici comporta anche dei costi in termini di spazio di archiviazione e di prestazioni durante l'aggiornamento dei dati.

È possibile utilizzare l'istruzione `EXPLAIN` per analizzare come il database sta eseguendo una query e identificare eventuali problemi di prestazioni. In base ai risultati ottenuti, è possibile valutare se creare nuovi indici o modificarne di esistenti per migliorare le prestazioni delle query.

**7.5 Conclusioni** La creazione di indici è fondamentale per ottimizzare le prestazioni delle query all'interno di un database. In questo capitolo, abbiamo discusso dei vantaggi degli indici, della loro creazione e del loro utilizzo. Utilizzando gli indici in modo appropriato, è possibile migliorare significativamente le prestazioni delle query e ottenere un database più efficiente. Nel prossimo capitolo, esploreremo l'importanza della gestione delle transazioni nel garantire l'integrità e la coerenza dei dati all'interno del database.

## **Capitolo 8: Gestione delle transazioni per l'integrità dei dati**

Nel capitolo precedente, abbiamo appreso l'importanza della creazione di indici per migliorare le prestazioni delle query. Ora è il momento di concentrarci sull'integrità dei dati. In questo capitolo, esploreremo l'importanza della gestione delle transazioni nel garantire che le operazioni di inserimento, aggiornamento ed eliminazione dei dati vengano eseguite in modo affidabile e coerente.

**8.1 Cosa sono le transazioni** Una transazione è un insieme di operazioni di database che devono essere eseguite come un'unica unità atomica. Le transazioni garantiscono che le operazioni vengano eseguite in modo coerente e affidabile, mantenendo l'integrità dei dati.

**8.2 Proprietà delle transazioni** Le transazioni possiedono quattro proprietà fondamentali, spesso indicate con l'acronimo ACID:

- **Atomicità (Atomicity):** garantisce che una transazione sia eseguita nella sua interezza o non venga eseguita affatto. Se anche una singola operazione all'interno della transazione fallisce, tutte le operazioni vengono annullate (rollback) e il database viene riportato allo stato precedente all'inizio della transazione.
- **Coerenza (Consistency):** garantisce che una transazione porti il database da uno stato consistente a un altro. Le operazioni all'interno di una transazione devono rispettare vincoli e regole definite sulle tabelle e le relazioni.
- **Isolamento (Isolation):** garantisce che le modifiche effettuate da una transazione siano isolate e non siano visibili ad altre transazioni fino al termine della transazione stessa. Ciò previene l'interferenza tra transazioni concorrenti.
- **Durabilità (Durability):** garantisce che le modifiche apportate da una transazione siano permanenti anche in

caso di guasto del sistema o riavvio del database. Le modifiche vengono salvate in modo permanente sul disco.

**8.3 Utilizzo delle transazioni** Le transazioni vengono gestite tramite due istruzioni principali: `BEGIN TRANSACTION` per avviare una transazione e `COMMIT` per confermare le modifiche e completare la transazione. In caso di necessità di annullare una transazione, è possibile utilizzare l'istruzione `ROLLBACK` per annullare tutte le modifiche effettuate all'interno della transazione.

Esempio di utilizzo delle transazioni:

```
BEGIN TRANSACTION;  
INSERT INTO Clienti (Nome, Cognome) VALUES ('Mario',  
'Rossi');  
UPDATE Prodotti SET Quantita = Quantita - 1 WHERE ID =  
1;  
COMMIT;
```

In questo esempio, stiamo avviando una transazione, inserendo un nuovo cliente nella tabella "Clienti" e aggiornando la quantità di un prodotto nella tabella "Prodotti". Dopo aver confermato le modifiche con `COMMIT`, la transazione viene completata.

**8.4 Conclusioni** La gestione delle transazioni è un aspetto fondamentale per garantire l'integrità e la coerenza dei dati all'interno di un database. In questo capitolo, abbiamo esplorato le proprietà delle transazioni e l'utilizzo delle istruzioni `BEGIN TRANSACTION`,

Utilizzando le transazioni in modo appropriato, è possibile assicurarsi che le operazioni di inserimento, aggiornamento ed eliminazione dei dati vengano eseguite in modo affidabile e coerente. Le transazioni sono particolarmente importanti quando si lavora con operazioni che coinvolgono più tabelle o richiedono un alto livello di consistenza dei dati.

Nel prossimo capitolo, esploreremo le viste (views) e le procedure (stored procedures), due strumenti potenti per l'organizzazione e l'ottimizzazione delle operazioni di interrogazione e manipolazione dei dati all'interno di un database.

## **Capitolo 9: Viste e procedure**

Nel capitolo precedente, abbiamo discusso dell'importanza della gestione delle transazioni per l'integrità dei dati. Ora è il momento di esplorare due strumenti potenti per l'organizzazione e l'ottimizzazione delle operazioni di



interrogazione e manipolazione dei dati: le viste (views) e le procedure (stored procedures).

9.1 Le viste Una vista è una rappresentazione virtuale di una tabella, creata mediante una query. Le viste consentono di definire un insieme specifico di colonne e righe di dati che possono essere utilizzate come una tabella reale. Le viste offrono diversi vantaggi, tra cui:

- Semplificazione delle query complesse: le viste possono nascondere la complessità delle query combinando più tabelle o applicando filtri specifici. In questo modo, è possibile eseguire interrogazioni più semplici e comprensibili.
- Sicurezza dei dati: le viste possono essere utilizzate per limitare l'accesso ai dati sensibili, consentendo agli utenti di visualizzare solo le informazioni rilevanti per loro.
- Organizzazione e modularità: le viste consentono di organizzare i dati in modo logico e modulare, facilitando la gestione e la manutenzione del database.

9.2 Le procedure Una procedura (stored procedure) è un insieme di istruzioni SQL predefinite che vengono salvate nel database e possono essere richiamate per eseguire operazioni specifiche. Le procedure offrono diversi vantaggi, tra cui:

- Riutilizzabilità del codice: le procedure consentono di scrivere una volta il codice per una determinata operazione e riutilizzarlo in diverse parti dell'applicazione. Ciò favorisce la ridondanza del codice e la facilità di manutenzione.
- Prestazioni: le procedure possono essere ottimizzate e memorizzate nella cache del database, migliorando le prestazioni dell'applicazione.
- Sicurezza: le procedure consentono di limitare l'accesso diretto alle tabelle del database, fornendo un'interfaccia controllata per l'esecuzione delle operazioni.

9.3 Creazione di viste e procedure Per creare una vista, è possibile utilizzare l'istruzione CREATE VIEW, specificando la query che definisce la vista. Ecco un esempio:

```
CREATE VIEW VistaClienti AS
SELECT Nome, Cognome, Email
FROM Clienti
WHERE Stato = 'Attivo';
```

In questo esempio, stiamo creando una vista chiamata "VistaClienti" che mostra solo i clienti attivi, includendo solo le colonne Nome, Cognome ed Email dalla tabella "Clienti". Per creare una procedura, è possibile utilizzare l'istruzione CREATE PROCEDURE, definendo il nome della procedura e il suo corpo. Ecco un esempio:

```
CREATE PROCEDURE SpeseTotali(IN anno INT, OUT totale
DECIMAL(10,2))
BEGIN
    SELECT SUM(Importo) INTO totale
    FROM Fatture
    WHERE YEAR(Data) = anno;
END;
```

In questo esempio, stiamo creando una procedura chiamata "SpeseTotali" che calcola il totale delle spese per un determinato anno, rest

9.3 Creazione di viste e procedure Per creare una vista, è possibile utilizzare l'istruzione CREATE VIEW, specificando la query che definisce la vista. Ecco un esempio:

```
CREATE VIEW VistaClienti AS
SELECT Nome, Cognome, Email
FROM Clienti
WHERE Stato = 'Attivo';
```

In questo esempio, stiamo creando una vista chiamata "VistaClienti" che mostra solo i clienti attivi, includendo solo le colonne Nome, Cognome ed Email dalla tabella "Clienti". Per creare una procedura, è possibile utilizzare l'istruzione CREATE PROCEDURE, definendo il nome della procedura e il suo corpo. Ecco un esempio:

```
CREATE PROCEDURE SpeseTotali(IN anno INT, OUT totale
DECIMAL(10,2))
BEGIN
    SELECT SUM(Importo) INTO totale
    FROM Fatture
    WHERE YEAR(Data) = anno;
END;
```

In questo esempio, stiamo creando una procedura chiamata "SpeseTotali" che calcola il totale delle spese per un

determinato anno, restituendo il risultato attraverso il parametro di output "totale".

9.4 Utilizzo di viste e procedure Una volta create, le viste possono essere utilizzate come tabelle virtuali nelle interrogazioni. Ad esempio, possiamo eseguire una semplice query sulla vista "VistaClienti":

```
SELECT *  
FROM VistaClienti;
```

Questa query restituirà tutti i clienti attivi con le colonne specificate nella definizione della vista.

Le procedure possono essere richiamate utilizzando l'istruzione CALL, specificando il nome della procedura e i parametri necessari. Ad esempio, per richiamare la procedura "SpeseTotali" e ottenere il totale delle spese per l'anno 2022, possiamo eseguire la seguente query:

```
CALL SpeseTotali(2022, @totale);  
SELECT @totale;
```

In questo modo, la procedura calcolerà il totale delle spese per l'anno specificato e lo restituirà tramite la variabile di output "@totale".

9.5 Conclusioni Le viste e le procedure sono strumenti potenti per organizzare e ottimizzare le operazioni di interrogazione e manipolazione dei dati in un database. Le viste consentono di semplificare le query complesse e di garantire la sicurezza dei dati, mentre le procedure favoriscono la riusabilità del codice e migliorano le prestazioni dell'applicazione.

Nel prossimo capitolo, esploreremo la sicurezza del database e le diverse tecniche per proteggere i dati sensibili da accessi non autorizzati.

## **Capitolo 10: Sicurezza del database e protezione dei dati sensibili**

Nel capitolo precedente, abbiamo discusso dell'utilizzo delle viste e delle procedure per l'organizzazione e l'ottimizzazione delle operazioni di database. Ora è il momento di affrontare un aspetto fondamentale: la sicurezza del database e la protezione dei dati sensibili.

10.1 Importanza della sicurezza del database La sicurezza del database è cruciale per proteggere i dati sensibili da accessi non autorizzati, modifiche indesiderate o perdita di informazioni. I dati all'interno del database possono includere informazioni personali, finanziarie o commerciali, che devono

essere protette per rispettare la privacy degli utenti e mantenere la fiducia dei clienti.

10.2 Principi di sicurezza del database Per garantire la sicurezza del database, è necessario adottare una serie di principi e misure di protezione. Alcuni principi fondamentali includono:

- **Accesso controllato:** è necessario limitare l'accesso al database solo agli utenti autorizzati, utilizzando autenticazione e autorizzazione. Ogni utente dovrebbe avere un account con credenziali uniche e privilegi appropriati per accedere e manipolare i dati.
- **Protezione dei dati:** i dati sensibili devono essere crittografati sia in transito che a riposo, per evitare che siano letti o manipolati da terze parti non autorizzate. È importante adottare tecniche di crittografia robuste per garantire la riservatezza dei dati.
- **Monitoraggio e auditing:** è fondamentale tenere traccia delle attività nel database, registrando gli accessi, le modifiche ai dati e le operazioni eseguite dagli utenti. Questo consente di rilevare eventuali anomalie o attività sospette e di fornire un registro delle attività per fini di revisione e conformità.
- **Backup e ripristino:** è necessario eseguire regolarmente backup dei dati e definire procedure di ripristino in caso di incidenti o perdita di dati. I backup consentono di recuperare i dati in caso di guasti hardware, errori umani o attacchi informatici.

10.3 Protezione dei dati sensibili Per proteggere i dati sensibili all'interno del database, è possibile adottare diverse misure di sicurezza, tra cui:

- **Ruoli e privilegi:** assegnare ruoli e privilegi appropriati agli utenti per garantire che possano accedere solo ai dati necessari per svolgere le loro funzioni. In questo modo, si riduce il rischio di accessi non autorizzati o di manipolazione dei dati.
- **Mascheramento dei dati:** mascherare o sostituire i dati sensibili con valori non riconoscibili o anonimi durante le operazioni di query o di estrazione dei dati. Ad esempio, è possibile mascherare i numeri di carta di credito o i codici fiscali per evitare che siano visualizzati integralmente.
- **Firewalls e sistemi di rilevamento delle intrusioni:** utilizzare firewall per proteggere il database da accessi non

autorizzati dall'esterno e implementare sistemi di rilevamento delle intrusioni per monitorare l'attività del database e rilevare eventuali tentativi di accesso non autorizzati o attività sospette.

- **Aggiornamenti e patch:** mantenere il database aggiornato con gli ultimi aggiornamenti e patch di sicurezza forniti dal fornitore del database. Questi aggiornamenti correggono vulnerabilità note e migliorano la sicurezza complessiva del sistema.
- **Audit dei dati sensibili:** tenere traccia delle attività di accesso e manipolazione dei dati sensibili, registrando informazioni come l'utente che ha eseguito l'operazione, la data e l'ora dell'accesso, e le modifiche apportate. Questo consente di identificare eventuali violazioni o abusi e facilita la responsabilità e la tracciabilità delle azioni.
- **Backup e ripristino:** eseguire regolarmente backup dei dati sensibili e definire procedure di ripristino per garantire la disponibilità dei dati in caso di guasti, perdita di dati o attacchi informatici. È importante conservare i backup in luoghi sicuri e testare periodicamente il processo di ripristino per assicurarsi che i dati possano essere recuperati correttamente.

La protezione dei dati sensibili è un aspetto cruciale per garantire la sicurezza del database. Adottando misure di sicurezza adeguate e seguendo buone pratiche, è possibile proteggere i dati da accessi non autorizzati e mantenerli al sicuro. Nel prossimo capitolo, esploreremo le best practice per l'ottimizzazione delle query e delle prestazioni del database.

## **Capitolo 11: Ottimizzazione delle query e delle prestazioni del database**

**11.1 Importanza dell'ottimizzazione delle query**  
L'ottimizzazione delle query è un aspetto fondamentale per migliorare le prestazioni del database e garantire una risposta rapida ed efficiente alle richieste degli utenti. Le query inefficienti possono causare rallentamenti, sovraccaricare il sistema e generare tempi di risposta elevati, compromettendo l'esperienza dell'utente.

**11.2 Analisi delle query** Prima di ottimizzare le query, è importante analizzare e comprendere il loro comportamento. È possibile utilizzare strumenti di monitoraggio delle prestazioni per identificare le query lente o inefficienti e analizzare il loro piano di esecuzione. Il piano di esecuzione mostra come il database sta elaborando la query e fornisce informazioni utili per identificare i punti critici e le aree di miglioramento.

**11.3 Utilizzo degli indici** Gli indici sono strutture di dati che migliorano la velocità di ricerca e di recupero dei dati all'interno del database. Utilizzare gli indici in modo appropriato può notevolmente migliorare le prestazioni delle query. È consigliabile creare indici su colonne frequentemente utilizzate nelle clausole di ricerca, join o ordinamento delle query. Tuttavia, è importante bilanciare l'utilizzo degli indici, in quanto un eccessivo numero di indici può rallentare le operazioni di inserimento, aggiornamento o cancellazione dei dati.

**11.4 Ottimizzazione delle query** Esistono diverse tecniche per ottimizzare le query e migliorare le prestazioni del database. Alcune delle più comuni includono:

- **Riduzione del numero di record restituiti:** limitare il numero di record restituiti da una query utilizzando clausole di filtraggio o di limitazione come WHERE o TOP. In questo modo, si riduce il carico di lavoro sul database e si ottiene una risposta più veloce.
- **Utilizzo di join efficienti:** utilizzare join appropriati per combinare le tabelle in modo efficiente. Utilizzare join interni solo quando necessario e considerare l'utilizzo di join esterni solo se richiesto. Utilizzare anche gli indici sulle colonne coinvolte nelle clausole di join per migliorare le prestazioni.
- **Evitare query nidificate:** evitare l'utilizzo di query nidificate, in quanto possono causare un alto carico computazionale. Invece, utilizzare clausole JOIN o subquery per ottenere gli stessi risultati in modo più efficiente.
- **Ottimizzazione delle istruzioni SQL:** scrivere istruzioni SQL efficienti, evitando l'uso di funzioni costose o complesse, e preferendo l'utilizzo di funzioni native del database. Utilizzare anche le funzioni di aggregazione quando necessario, piuttosto che elaborare manualmente i risultati.

**11.5 Monitoraggio e tuning delle prestazioni** Una volta implementate le ottimizzazioni, è importante monitorare le prestazioni del database per identificare eventuali aree di miglioramento. Utilizzare strumenti di monitoraggio delle prestazioni per monitorare le query lente, gli indici non utilizzati o altre problematiche che possono influire sulle prestazioni complessive del sistema. Eseguire reg

**11.6 Scalabilità del database** La scalabilità del database è un altro aspetto da considerare per garantire prestazioni efficienti nel tempo. È importante progettare il database in modo da poter gestire un aumento del carico di lavoro e dei dati nel tempo. Questo può includere la distribuzione su più server,

l'utilizzo di partizionamento dei dati o l'implementazione di soluzioni di replica per migliorare la disponibilità dei dati.

**11.7 Cache dei dati** L'utilizzo di cache dei dati può contribuire notevolmente all'ottimizzazione delle prestazioni. La cache dei dati memorizza i risultati delle query frequenti in memoria, consentendo di recuperare rapidamente i dati senza eseguire la query sul database. Questo riduce il carico sul sistema e migliora i tempi di risposta.

**11.8 Ottimizzazione dello schema del database** Infine, ottimizzare lo schema del database può avere un impatto significativo sulle prestazioni. Ridurre il numero di tabelle, normalizzare il modello di dati e gestire correttamente le relazioni tra le tabelle può migliorare l'efficienza delle query e ridurre i tempi di risposta.

**Conclusione** L'ottimizzazione delle query e delle prestazioni del database è un processo continuo e necessario per garantire un funzionamento efficiente del sistema. Utilizzando correttamente gli indici, ottimizzando le query, monitorando le prestazioni e adottando soluzioni di scalabilità, è possibile migliorare notevolmente le prestazioni complessive del database. Nel prossimo capitolo, esploreremo le best practice per la gestione dei backup e il ripristino dei dati.

## **Capitolo 12: Gestione dei backup e ripristino dei dati**

**12.1 Importanza dei backup** La gestione dei backup è un aspetto critico nella sicurezza e nella continuità operativa del database. I backup consentono di creare copie dei dati in modo da poterli ripristinare in caso di perdita, guasti hardware, errori umani o attacchi informatici. È fondamentale garantire la disponibilità dei backup e testare regolarmente il processo di ripristino per assicurarsi che i dati possano essere recuperati correttamente.

**12.2 Tipi di backup** Esistono diversi tipi di backup che è possibile eseguire:

- **Backup completo:** copia di tutti i dati presenti nel database. Questo tipo di backup è utile per ripristinare l'intero database in caso di perdita totale dei dati.
- **Backup differenziale:** copia solo i dati che sono stati modificati dalla data dell'ultimo backup completo. Questo tipo di backup è più veloce rispetto al backup completo, ma richiede anche il backup completo per il ripristino.
- **Backup incrementale:** copia solo i dati che sono stati modificati dalla data dell'ultimo backup, indipendentemente dal tipo di backup precedente. Questo tipo di backup è il più veloce, ma richiede il ripristino

sequenziale di tutti i backup incrementali per ottenere lo stato finale dei dati.

- Backup del log delle transazioni: copia dei log delle transazioni del database. Questo tipo di backup consente di ripristinare il database a un determinato punto nel tempo, consentendo il recupero delle modifiche apportate dopo l'ultimo backup completo.

**12.3 Pianificazione dei backup** È importante stabilire una pianificazione regolare per i backup del database. Questo può includere la frequenza dei backup (giornaliera, settimanale, mensile), l'orario di esecuzione e il tipo di backup da eseguire. È consigliabile conservare i backup in luoghi sicuri, separati dal server di produzione, per proteggerli da eventi catastrofici come incendi o danni fisici al server.

**12.4 Verifica dei backup** La verifica dei backup è un passaggio essenziale per assicurarsi che siano completi e integri. Periodicamente, è necessario eseguire test di ripristino utilizzando i backup per verificare che i dati possano essere recuperati correttamente. In caso di errori o corruzioni, è importante risolverli tempestivamente e correggere eventuali problemi di backup.

**12.5 Archiviazione dei backup** L'archiviazione dei backup deve essere gestita in modo sicuro e conforme alle politiche di sicurezza dell'azienda. È possibile utilizzare supporti di archiviazione offline, come unità esterne o servizi di archiviazione cloud, per proteggere i backup da accessi non autorizzati o da eventi fisici che potrebbero danneggiare il server principale.

**12.6 Ripristino dei dati** Nel caso di perdita o danneggiamento dei dati, è necessario eseguire il processo di ripristino utilizzando i backup disponibili. È importante seguire le procedure corrette per ripristinare i backup in modo da garantire l'integrità dei dati e riprist

## **Capitolo 13: Sicurezza dei dati nel database**

**13.1 Importanza della sicurezza dei dati** La sicurezza dei dati nel database è un aspetto fondamentale per proteggere le informazioni sensibili e garantire la privacy degli utenti. Con l'aumento delle minacce informatiche e degli attacchi hacker, è essenziale adottare misure di sicurezza per prevenire l'accesso non autorizzato, la violazione dei dati e le perdite di informazioni critiche.

**13.2 Autenticazione e autorizzazione** L'autenticazione e l'autorizzazione sono due concetti chiave nella sicurezza dei dati. L'autenticazione verifica l'identità degli utenti che accedono al database, utilizzando credenziali come nome utente e password. È importante utilizzare politiche di



password robuste, scadenze periodiche delle password e meccanismi di autenticazione a più fattori per aumentare la sicurezza.

L'autorizzazione, d'altra parte, definisce i privilegi e i livelli di accesso che gli utenti hanno all'interno del database. È consigliabile assegnare i privilegi in base al principio del privilegio minimo, in modo che gli utenti abbiano solo l'accesso necessario per svolgere le proprie attività.

**13.3 Crittografia dei dati** La crittografia dei dati è una pratica comune per proteggere le informazioni sensibili all'interno del database. La crittografia può essere applicata sia ai dati in transito (durante la comunicazione tra client e server) che ai dati inattivi (archiviati nel database). Utilizzare algoritmi di crittografia robusti e tenere le chiavi di crittografia al sicuro per garantire la confidenzialità dei dati.

**13.4 Monitoraggio e audit** Il monitoraggio e l'audit delle attività nel database sono importanti per rilevare potenziali violazioni di sicurezza o comportamenti anomali. Utilizzare strumenti di monitoraggio per registrare le attività degli utenti, analizzare i log e identificare eventuali pattern sospetti. In caso di rilevamento di attività non autorizzate, è fondamentale rispondere tempestivamente per mitigare i rischi.

**13.5 Backup e ripristino sicuro dei dati** Come discusso nel capitolo precedente, i backup sono fondamentali per garantire la sicurezza dei dati. Assicurarsi che i backup siano crittografati e conservati in luoghi sicuri per prevenire accessi non autorizzati o perdite fisiche dei dati. In caso di violazione dei dati, è possibile utilizzare i backup per ripristinare le informazioni e ripristinare lo stato precedente del database.

**13.6 Aggiornamenti e patch di sicurezza** Mantenere il database aggiornato con gli ultimi aggiornamenti e patch di sicurezza è essenziale per proteggere il sistema da vulnerabilità note. Monitorare regolarmente le nuove versioni del software del database e applicare le patch di sicurezza consigliate per ridurre il rischio di attacchi basati su vulnerabilità note.

**13.7 Consapevolezza della sicurezza dei dati** Infine, promuovere la consapevolezza della sicurezza dei dati tra gli utenti e il person

## **Capitolo 14: Disaster Recovery nel database**

**14.1 Introduzione al Disaster Recovery** Il Disaster Recovery (DR) è un insieme di procedure e strategie volte a ripristinare il sistema e i dati in caso di un evento catastrofico che compromette l'operatività del database. Gli eventi catastrofici possono includere guasti hardware, catastrofi naturali, attacchi informatici o errori umani. Il DR è essenziale per garantire la continuità operativa e ridurre al minimo l'impatto negativo di tali eventi.

**14.2 Pianificazione del Disaster Recovery** La pianificazione del DR richiede un'analisi dei rischi e l'identificazione dei potenziali

scenari di disastro. È importante valutare l'impatto che un'interruzione potrebbe avere sul sistema e definire obiettivi di ripristino (RTO) e obiettivi di punto di ripristino (RPO). L'RTO rappresenta il tempo massimo accettabile per ripristinare il sistema, mentre l'RPO indica l'intervallo di tempo massimo accettabile per la perdita di dati.

**14.3 Strumenti di replica e mirroring** Per implementare un'efficace strategia di DR, è possibile utilizzare strumenti di replica e mirroring del database. La replica del database coinvolge la creazione di copie dei dati in tempo reale su un server di replica, consentendo un rapido switch al server di replica in caso di guasto del server principale. Il mirroring, d'altra parte, è una tecnica di replica che mantiene due server sincronizzati in modo che uno possa prendere il controllo se l'altro fallisce.

**14.4 Backup e archiviazione esterna** Come discusso nei capitoli precedenti, i backup sono un elemento essenziale del DR. È importante conservare i backup in un luogo esterno sicuro, separato dal server di produzione, per proteggerli da eventi catastrofici che potrebbero colpire l'intero sistema. L'utilizzo di servizi di archiviazione cloud o di supporti di archiviazione offline può garantire la disponibilità dei backup anche in caso di disastri fisici.

**14.5 Test di ripristino e procedure di emergenza** Per garantire l'efficacia del piano di DR, è fondamentale testare regolarmente le procedure di ripristino e le procedure di emergenza. Questi test consentono di valutare la tempestività e l'efficacia del ripristino dei dati e dell'operatività del sistema in situazioni di emergenza. È importante coinvolgere tutte le parti interessate e documentare i risultati dei test per apportare eventuali miglioramenti.

**14.6 Ripristino graduale e priorità delle operazioni** Durante un evento di disastro, è importante stabilire una sequenza di ripristino graduale e stabilire la priorità delle operazioni. Ciò implica l'identificazione dei componenti critici del sistema e l'assegnazione di risorse prioritarie per il loro ripristino. Ad esempio, è possibile stabilire che il ripristino dei dati sia prioritario rispetto alla riparazione del hardware o delle applicazioni non essenziali.

**14.7 Continuità operativa e ripristino post-disastro** Una volta che il sistema è stato ripristinato dopo un disastro, è necessario stabilire una strategia di continuità operativa per garantire che l'attività possa riprendere normalmente. Ciò può includere la verifica dell'integrità dei dati ripristinati, il ripristino delle configurazioni di rete, l'implementazione di misure di sicurezza aggiuntive e la valutazione dei danni causati dall'evento di disastro.

## **Capitolo 15: Tuning delle prestazioni del database**

**15.1 Introduzione al tuning delle prestazioni** Il tuning delle prestazioni del database è un processo continuo e iterativo che mira a ottimizzare l'efficienza e le prestazioni del sistema. L'obiettivo è ridurre i tempi di risposta delle query, migliorare la capacità di gestione del carico e ottimizzare l'utilizzo delle risorse hardware disponibili.

**15.2 Analisi delle prestazioni** La prima fase del tuning delle prestazioni consiste nell'analizzare le prestazioni attuali del database. Questo può includere il monitoraggio delle metriche di prestazione, come il tempo di risposta delle query, l'utilizzo delle risorse di memoria e CPU, e l'identificazione delle query lente o inefficaci. L'analisi delle prestazioni fornisce una base per identificare le aree che richiedono ottimizzazione.

**15.3 Ottimizzazione dello schema del database** Un aspetto importante del tuning delle prestazioni è l'ottimizzazione dello schema del database. Ciò implica la progettazione di tabelle e indici in modo efficiente, la normalizzazione dei dati, l'uso di vincoli e relazioni appropriati e l'eliminazione di duplicazioni o ridondanze inutili. Un'adeguata progettazione dello schema del database può migliorare notevolmente le prestazioni delle query.

**15.4 Ottimizzazione delle query** Le query inefficaci sono spesso una causa comune di ridotte prestazioni del database. L'ottimizzazione delle query coinvolge l'analisi del piano di esecuzione delle query, l'uso corretto degli indici, la scrittura di query ottimizzate e l'utilizzo di strumenti per l'ottimizzazione automatica delle query. L'obiettivo è ridurre il numero di scansioni di tabelle, minimizzare il traffico di rete e utilizzare le risorse del database in modo efficiente.

**15.5 Gestione delle risorse** Un altro aspetto chiave del tuning delle prestazioni è la gestione delle risorse del database. Ciò può includere l'allocazione adeguata di memoria, l'ottimizzazione delle configurazioni di cache, l'uso di partizionamento dei dati per distribuire il carico e l'ottimizzazione delle configurazioni di I/O. La corretta gestione delle risorse può contribuire a migliorare l'efficienza complessiva del sistema.

**15.6 Monitoraggio delle prestazioni** Dopo aver implementato le ottimizzazioni, è fondamentale monitorare continuamente le prestazioni del database per valutare l'impatto delle modifiche apportate. Il monitoraggio delle prestazioni consente di rilevare eventuali degradazioni delle prestazioni, identificare nuove opportunità di ottimizzazione e garantire che il sistema funzioni in modo efficiente nel tempo.

**15.7 Scalabilità e pianificazione futura** Infine, il tuning delle prestazioni dovrebbe considerare la scalabilità del sistema e la pianificazione per il futuro. Ciò implica l'analisi delle esigenze di crescita del database, la valutazione delle opzioni di scalabilità, come l'uso di clustering o di architetture distribuite,

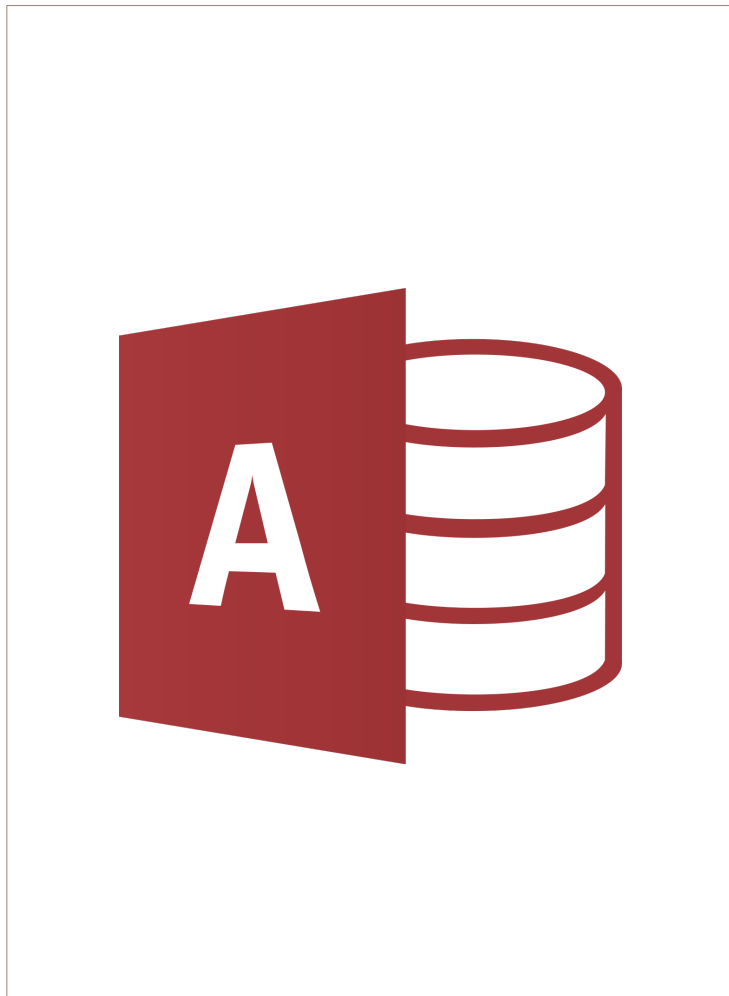
e l'adeguamento delle risorse per gestire l'aumento del carico in modo efficace. È importante pianificare in anticipo per evitare problemi di prestazioni futuri e garantire che il database possa adattarsi alle esigenze in continua evoluzione dell'organizzazione.

#### Conclusione

In questo libro, abbiamo esplorato i fondamenti del linguaggio SQL e la sua applicazione pratica nella gestione dei database. Abbiamo imparato le basi per creare, modificare e interrogare tabelle, nonché per gestire gli indici, le viste e le transazioni. Abbiamo anche approfondito concetti avanzati come la sicurezza dei dati, il ripristino in caso di disastro e il tuning delle prestazioni.

Il linguaggio SQL è uno strumento potente che consente agli sviluppatori e agli amministratori di database di gestire e manipolare i dati in modo efficiente. Con una buona comprensione dei concetti e delle best practice presentate in questo libro, sarete in grado di sfruttare appieno le potenzialità del linguaggio SQL e di costruire e gestire database robusti e performanti.

Speriamo che questo libro sia stato utile nel fornire una panoramica completa del linguaggio SQL e delle sue applicazioni. Ora siete pronti per intraprendere il vostro percorso nell'utilizzo di SQL per la gestione dei database. Buon apprendimento e buon lavoro con i vostri progetti di database!



# MS ACCESS

## Microsoft Access

Microsoft Access è un sistema di gestione di database relazionale (DBMS) sviluppato da Microsoft. Fa parte della suite di prodotti Microsoft Office ed è progettato per la gestione di database desktop. Ecco alcune informazioni chiave su Microsoft Access:

**Interfaccia utente:** Microsoft Access fornisce un'interfaccia utente intuitiva e basata su finestre. Gli utenti possono creare, modificare e interagire con i database utilizzando oggetti come tabelle, query, moduli e report.

**Creazione di database:** Gli utenti possono creare nuovi database da zero o utilizzare modelli predefiniti per avviare rapidamente un database. Access offre anche la possibilità di importare dati da altre origini, come fogli di calcolo Excel o file di testo.

**Tabelle:** Le tabelle sono gli elementi fondamentali di un database in Access. Gli utenti possono creare tabelle per archiviare i dati e definire la struttura dei campi (colonne) e dei record (righe).

**Query:** Le query in Access consentono di interrogare il database e recuperare dati specifici. Gli utenti possono creare query utilizzando l'editor visuale di Access o scrivendo istruzioni SQL.

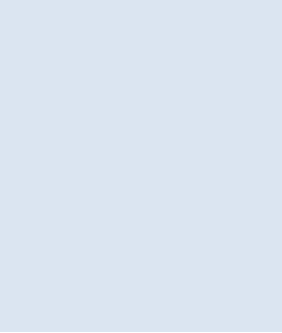
**Moduli:** I moduli in Access permettono di creare interfacce personalizzate per l'inserimento e la visualizzazione dei dati. Possono contenere controlli, logica di programmazione e azioni personalizzate.

**Report:** Access offre funzionalità di creazione di report per presentare i dati in formato stampabile o visuale. Gli utenti possono progettare report personalizzati e includere elementi come intestazioni, piedi di pagina, grafici e immagini.

**Automazione:** Access supporta la creazione di macro e di codice VBA (Visual Basic for Applications) per automatizzare compiti e personalizzare il comportamento del database.

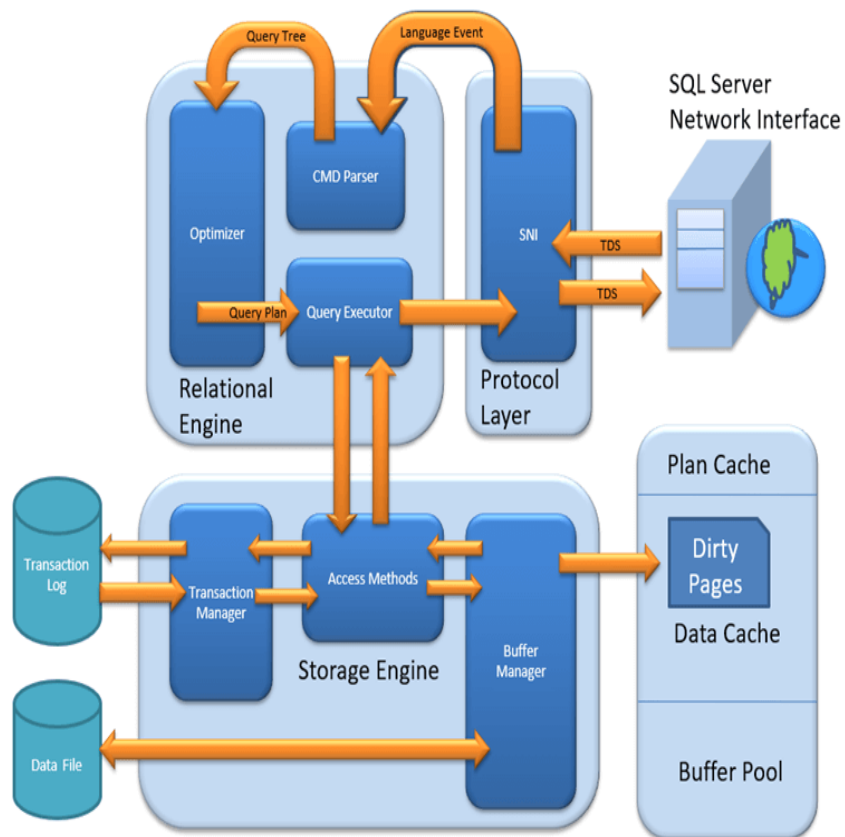
**Condivisione dei dati:** Microsoft Access consente la condivisione dei dati attraverso la creazione di file di database condivisibili (.accdb o .mdb) o tramite la pubblicazione del database su Microsoft SharePoint.

**Integrazione con altri prodotti Microsoft:** Access può essere integrato con altri prodotti Microsoft come Excel, Word e Outlook per facilitare lo scambio di dati e migliorare la produttività.



Microsoft Access è una soluzione popolare per la gestione di database desktop, soprattutto per utenti con esigenze di database relativamente semplici. È ampiamente utilizzato nelle piccole e medie imprese per la creazione di applicazioni personalizzate, la gestione dei dati e la generazione di report.





# SQL SERVER

## MS SQL Server

SQL Server è un sistema di gestione di database relazionale (DBMS) sviluppato da Microsoft. È una soluzione potente e scalabile progettata per la gestione di grandi volumi di dati e per supportare applicazioni aziendali critiche. Ecco alcune caratteristiche e funzionalità chiave di SQL Server:

**Architettura:** SQL Server utilizza un'architettura client-server, in cui un'istanza di SQL Server funge da server e accetta richieste di connessione dai client. Supporta connessioni simultanee da più client e offre funzionalità per l'accesso e la gestione dei dati.

**Lingaggio SQL:** SQL Server supporta il linguaggio SQL (Structured Query Language) per l'interazione con il database. Consente di eseguire query, definire e modificare schemi, gestire autorizzazioni e altro ancora.

**Gestione dei dati:** SQL Server offre una vasta gamma di funzionalità per la gestione dei dati, inclusa la creazione di tabelle, l'indicizzazione dei dati per migliorare le prestazioni delle query, la definizione di vincoli di integrità referenziale, la gestione delle transazioni e la gestione delle viste.

**Sicurezza:** SQL Server fornisce robuste funzionalità di sicurezza per proteggere i dati sensibili. Supporta l'autenticazione degli utenti, il controllo degli accessi basato su

ruoli, la crittografia dei dati, l'audit delle attività degli utenti e altre misure di sicurezza per garantire l'integrità dei dati.

**Backup e ripristino:** SQL Server offre strumenti e funzionalità per eseguire il backup dei database in modo regolare e per ripristinare i dati in caso di guasti hardware, errori umani o altri problemi. I backup possono essere eseguiti in diversi modi, come backup completo, backup differenziale o backup delle transazioni.

**Replicazione dei dati:** SQL Server supporta la replicazione dei dati per consentire la distribuzione di dati su più server. Ciò consente di mantenere copie dei dati sincronizzate in diversi ambienti e di migliorare le prestazioni delle applicazioni distribuite.

**Alta disponibilità:** SQL Server offre funzionalità di alta disponibilità per garantire la continuità del servizio. Queste funzionalità includono la replica del database, i gruppi di disponibilità, il mirroring del database e il clustering per assicurare che il database sia sempre accessibile e ridurre al minimo i tempi di inattività.

**Business Intelligence:** SQL Server include funzionalità per l'elaborazione e l'analisi dei dati aziendali. Offre strumenti per l'integrazione dei dati, l'analisi dei dati, la creazione di report e la visualizzazione dei dati in modo significativo.

SQL Server è una soluzione completa per la gestione dei dati aziendali, ampiamente utilizzata in ambienti aziendali di medie e grandi dimensioni. Offre prestazioni elevate, scalabilità, sicurezza avanzata e una vasta gamma di funzionalità per soddisfare le esigenze di gestione dei dati delle organizzazioni.

## Altri server SQL

Oltre a SQL Server di Microsoft, esistono diversi altri server SQL disponibili sul mercato. Ecco alcuni esempi di server SQL popolari:

**MySQL:** MySQL è un server SQL open-source ampiamente utilizzato per applicazioni web e applicazioni di piccole e medie dimensioni. È noto per la sua velocità, affidabilità e facilità d'uso. MySQL è sviluppato da Oracle Corporation.

**PostgreSQL:** PostgreSQL è un potente server SQL open-source noto per la sua conformità agli standard ANSI SQL. Offre una vasta gamma di funzionalità avanzate, inclusa la gestione delle transazioni, la replicazione dei dati e la scalabilità. PostgreSQL è noto per la sua affidabilità e capacità di gestire grandi volumi di dati.

**Oracle Database:** Oracle Database è un server SQL commerciale sviluppato da Oracle Corporation. È ampiamente utilizzato in ambienti aziendali per gestire grandi volumi di dati e supportare applicazioni aziendali critiche. Oracle Database offre funzionalità avanzate come la gestione dei dati gerarchici, la scalabilità e l'affidabilità.

**IBM Db2:** IBM Db2 è un server SQL commerciale sviluppato da IBM. Offre funzionalità avanzate come l'elaborazione analitica in memoria, la gestione di dati JSON, la sicurezza avanzata e la scalabilità. Db2 è noto per le sue prestazioni

elevate e la capacità di gestire grandi carichi di lavoro aziendali.

SQLite: SQLite è un server SQL incorporabile che può essere utilizzato come parte integrante di applicazioni desktop o mobili. È noto per la sua leggerezza, velocità e facilità di integrazione. SQLite è una scelta comune per applicazioni locali o mobili che richiedono un database SQL leggero e portatile.

Questi sono solo alcuni esempi di server SQL disponibili. Ogni server ha le sue caratteristiche, vantaggi e casi d'uso specifici. La scelta del server SQL dipende dalle esigenze dell'applicazione, dalla scalabilità richiesta, dalla complessità dei dati e da altri fattori specifici al contesto di utilizzo.



SQLITE

## SQLite è un server SQL Embedded

SQLite è un server SQL incorporabile e senza server che è noto per la sua leggerezza, facilità d'uso e portabilità. A differenza della maggior parte dei server SQL, SQLite non funziona come un processo server separato, ma è incorporato direttamente nelle applicazioni che lo utilizzano. Ecco alcune caratteristiche chiave di SQLite:

**Leggerezza:** SQLite è progettato per essere un database SQL leggero e senza server. L'intero database SQLite è contenuto in un singolo file, che lo rende facile da distribuire e da gestire. Questo lo rende una scelta popolare per applicazioni desktop, mobili e incorporabili che richiedono un database leggero.

**Zero configurazione:** Non è richiesta alcuna configurazione complessa per iniziare a utilizzare SQLite. Non ci sono processi di installazione o configurazione del server separati. È sufficiente includere la libreria SQLite nelle proprie applicazioni e iniziare a utilizzare il database.

**Transazionalità:** SQLite supporta le transazioni ACID (Atomicità, Coerenza, Isolamento, Durabilità), che garantiscono la consistenza e l'affidabilità dei dati. È possibile eseguire operazioni di commit e rollback delle transazioni per garantire la coerenza dei dati.

**Portabilità:** I file di database SQLite possono essere facilmente copiati o trasferiti tra diversi sistemi senza dover ripetere il

processo di installazione o configurazione del server. Ciò lo rende altamente portatile e adatto per l'utilizzo su diverse piattaforme.

Ampia compatibilità: SQLite è conforme agli standard SQL e supporta la maggior parte delle funzionalità SQL standard, inclusi JOIN, trigger, viste e sottoselezioni. Tuttavia, a differenza di server SQL più complessi, potrebbe mancare alcune funzionalità avanzate come la gestione di grandi volumi di dati o la scalabilità in cluster.

Ampia disponibilità: SQLite è ampiamente utilizzato ed è supportato da una vasta gamma di linguaggi di programmazione, compresi C/C++, Python, Java, .NET e molti altri. Ciò consente agli sviluppatori di utilizzare SQLite nelle loro applicazioni utilizzando il linguaggio di programmazione preferito.

SQLite è particolarmente adatto per applicazioni che richiedono un database leggero e incorporabile, come applicazioni mobili, software desktop e dispositivi embedded. È un'opzione popolare quando è necessario gestire piccoli a medi volumi di dati in modo semplice e portatile, senza richiedere la complessità di un server SQL separato.



## Esercizi SQLITE

CMD

```
cd 'C:\Users\Administrator\Desktop\ITS Gestione Dati\sqlite\'
```

```
.help
```

Creating a Database

```
sqlite3 test.db
```

```
create table test (id integer primary key, value text);
```

```
insert into test (id, value) values(1, 'eenie');
```

```
insert into test (id, value) values(2, 'meenie');
```

```
insert into test (value) values('miny');
```

```
insert into test (value) values('mo');
```

```
.mode column
```

```
.headers on
```

```
select * from test;
```

```
id value
```

```
-- -----
```

```
1 eenie
```

```
2 meenie
```

```
3 miny
```

4 mo

```
select last_insert_rowid();
```

```
create index test_idx on test (value);
```

```
create view schema as select * from sqlite_master;
```

```
.exit
```

```
.tables
```

```
.indices test
```

```
.schema test
```

```
.schema
```

```
.mode column
```

```
.headers on
```

```
select type, name, tbl_name, sql from sqlite_master order by  
type;
```

### Exporting Data

```
.output file.sql
```

```
.dump
```

```
.output stdout
```

This will create the file file.sql in your current working directory if it does not already exist. If a file by that name does exist, it will be overwritten.

```
.output file.csv
```

```
.separator ,
```

```
select * from test;
```

```
.output stdout
```

```
.output text.csv
```

```
.separator ,  
select * from test where value like 'm%';  
.output stdout
```

## Importing Data

```
.show
```

```
drop table test;  
drop view schema;  
.read file.sql
```

If you want to then import this CSV data into a similar table with the same structure as the test table (call it test2), do the following:

```
create table test2(id integer primary key, value text);  
.import text.csv test2
```

-----

```
sqlite3.exe test.db .dump  
sqlite3 test.db .dump > test.sql  
sqlite3 test.db "select * from test"  
sqlite3 test2.db < test.sql
```

```
sqlite3 test.db vacuum
```

will free up any unused space created from deleted objects

-----

esercizio

```
CREATE TABLE COMPANY(  
  ID INT PRIMARY KEY NOT NULL,  
  NAME TEXT NOT NULL,  
  AGE INT NOT NULL,  
  ADDRESS CHAR(50),  
  SALARY REAL  
);
```

```
CREATE TABLE DEPARTMENT(  
  ID INT PRIMARY KEY NOT NULL,  
  DEPT CHAR(50) NOT NULL,  
  EMP_ID INT NOT NULL  
);
```

.tables

schema COMPANY

```
INSERT          INTO          COMPANY  
(ID,NAME,AGE,ADDRESS,SALARY)  
VALUES (1, 'Paul', 32, 'California', 20000.00 );
```

```
INSERT          INTO          COMPANY  
(ID,NAME,AGE,ADDRESS,SALARY)  
VALUES (2, 'Allen', 25, 'Texas', 15000.00 );
```

```
INSERT          INTO          COMPANY  
(ID,NAME,AGE,ADDRESS,SALARY)  
VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );
```

```
INSERT          INTO          COMPANY
(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 );
```

```
INSERT          INTO          COMPANY
(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (5, 'David', 27, 'Texas', 85000.00 );
```

```
INSERT          INTO          COMPANY
(ID,NAME,AGE,ADDRESS,SALARY)
VALUES (6, 'Kim', 22, 'South-Hall', 45000.00 );
```

```
INSERT INTO COMPANY VALUES (7, 'James', 24, 'Houston',
10000.00 );
```

.header on

.mode column

```
SELECT * FROM COMPANY;
```

```
SELECT ID, NAME, SALARY FROM COMPANY;
```

.width 10, 20, 10

```
SELECT * FROM COMPANY;
```

.mode line

```
select 10 + 20;
```

```
10 + 20 = 30
```

```
select 10 - 20;
```

```
10 - 20 = -10
```

```
select 10 * 20;
```

```
10 * 20 = 200
```

```
select 10 / 5;  
10 / 5 = 2  
select 12 % 5;  
12 % 5 = 2
```

```
SELECT * FROM COMPANY WHERE SALARY > 50000;  
SELECT * FROM COMPANY WHERE SALARY = 20000;  
SELECT * FROM COMPANY WHERE SALARY != 20000;  
SELECT * FROM COMPANY WHERE SALARY <> 20000;  
SELECT * FROM COMPANY WHERE SALARY >= 65000;  
SELECT * FROM COMPANY WHERE AGE >= 25 AND  
SALARY >= 65000;  
SELECT * FROM COMPANY WHERE AGE >= 25 OR  
SALARY >= 65000;  
SELECT * FROM COMPANY WHERE AGE IS NOT NULL;  
SELECT * FROM COMPANY WHERE NAME LIKE 'Ki%';  
SELECT * FROM COMPANY WHERE NAME GLOB 'Ki*';  
SELECT * FROM COMPANY WHERE AGE IN ( 25, 27 ); value  
is either 25 or 27:  
SELECT * FROM COMPANY WHERE AGE NOT IN ( 25, 27 );  
SELECT * FROM COMPANY WHERE AGE BETWEEN 25  
AND 27;
```

```
SELECT AGE FROM COMPANY  
WHERE EXISTS (SELECT AGE FROM COMPANY WHERE  
SALARY > 65000);
```

```
SELECT * FROM COMPANY  
WHERE AGE > (SELECT AGE FROM COMPANY WHERE  
SALARY > 65000);
```

```
SELECT * FROM COMPANY WHERE SALARY = 10000;
```

```
SELECT COUNT(*) AS "RECORDS" FROM COMPANY;  
SELECT CURRENT_TIMESTAMP;
```

```
SELECT * FROM COMPANY;  
UPDATE COMPANY SET ADDRESS = 'Texas' WHERE ID =  
6;
```

```
modify all ADDRESS and SALARY  
UPDATE COMPANY SET ADDRESS = 'Texas', SALARY =  
20000.00;
```

```
DELETE FROM COMPANY WHERE ID = 7;  
DELETE FROM COMPANY;
```

```
SELECT FROM table_name  
WHERE SALARY LIKE '200%' Finds any values that start with  
200  
WHERE SALARY LIKE '%200%' Finds any values that have  
200 in any position  
WHERE SALARY LIKE '_00%' Finds any values that have 00  
in the second and third positions  
WHERE SALARY LIKE '2_%_%' Finds any values that start  
with 2 and are at least 3 characters in length  
WHERE SALARY LIKE '%2' Finds any values that end with 2  
WHERE SALARY LIKE '_2%3' Finds any values that have a 2  
in the second position and end with a 3  
WHERE SALARY LIKE '2___3' Finds any values in a five-digit  
number that start with 2 and end with 3
```

```
SELECT * FROM COMPANY LIMIT 6;
```

```
SELECT * FROM COMPANY ORDER BY SALARY ASC;
```

```
SELECT * FROM COMPANY ORDER BY NAME, SALARY  
ASC;
```

```
SELECT * FROM COMPANY ORDER BY NAME DESC;
```

If you want to know the total amount of salary on each customer

```
SELECT NAME, SUM(SALARY) FROM COMPANY GROUP  
BY NAME;
```

```
INSERT INTO COMPANY VALUES (8, 'Paul', 24, 'Houston',  
20000.00 );
```

```
INSERT INTO COMPANY VALUES (9, 'James', 44, 'Norway',  
5000.00 );
```

```
INSERT INTO COMPANY VALUES (10, 'James', 45, 'Texas',  
5000.00 );
```

```
SELECT NAME, SUM(SALARY) FROM COMPANY GROUP  
BY NAME;
```

```
SELECT NAME, SUM(SALARY) FROM COMPANY GROUP  
BY NAME ORDER BY NAME;
```

HAVING clause enables you to specify conditions that filter which group results appear in the final results.

name count is less than 2:



```
SELECT * FROM COMPANY GROUP BY name HAVING  
count(name) < 2;  
SELECT * FROM COMPANY GROUP BY name HAVING  
count(name) > 2;
```

DISTINCT keyword is used in conjunction with SELECT statement to eliminate all the duplicate records and fetching only unique records

```
SELECT name FROM COMPANY;  
SELECT DISTINCT name FROM COMPANY;
```

Constraints are the rules enforced on data columns on table.

NOT NULL

```
CREATE TABLE COMPANY(  
ID INT PRIMARY KEY NOT NULL,  
NAME TEXT NOT NULL,  
AGE INT NOT NULL,  
ADDRESS CHAR(50),  
SALARY REAL  
);
```

```
CREATE TABLE COMPANY(  
ID INT PRIMARY KEY NOT NULL,  
NAME TEXT NOT NULL,  
AGE INT NOT NULL,
```

```
ADDRESS CHAR(50),  
SALARY REAL DEFAULT 50000.00  
);
```

```
CREATE TABLE COMPANY(  
ID INT PRIMARY KEY NOT NULL,  
NAME TEXT NOT NULL,  
AGE INT NOT NULL UNIQUE,  
ADDRESS CHAR(50),  
SALARY REAL DEFAULT 50000.00  
);
```

```
CREATE TABLE COMPANY3(  
ID INT PRIMARY KEY NOT NULL,  
NAME TEXT NOT NULL,  
AGE INT NOT NULL,  
ADDRESS CHAR(50),  
SALARY REAL CHECK(SALARY > 0)  
);
```

### SQLite Joins

? The CROSS JOIN

? The INNER JOIN

? The OUTER JOIN

```
SELECT * FROM COMPANY;
```

```
CREATE TABLE DEPARTMENT(  
ID INT PRIMARY KEY NOT NULL,  
DEPT CHAR(50) NOT NULL,  
EMP_ID INT NOT NULL
```

);

```
INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)
VALUES (1, 'IT Billing', 1 );
INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)
VALUES (2, 'Engineering', 2 );
INSERT INTO DEPARTMENT (ID, DEPT, EMP_ID)
VALUES (3, 'Finance', 7 );
```

.header on

.mode column

```
SELECT * FROM department;
```

A CROSS JOIN matches every row of the first table with every row of the second table

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY CROSS
JOIN DEPARTMENT;
```

A INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows, which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER
JOIN DEPARTMENT
ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

The OUTER JOIN is an extension of the INNER JOIN. Though SQL standard defines three types of OUTER JOINS: LEFT, RIGHT and FULL but SQLite only supports the LEFT OUTER JOIN.

The SQLite UNION clause/operator is used to combine the results of two or more SELECT statements without returning any duplicate rows.

To use UNION, each SELECT must have the same number of columns selected, the same number of column expressions, the same data type, and have them in the same order, but they do not have to be the same length.

```
INSERT INTO DEPARTMENT(ID,DEPT,EMP_ID)
VALUES (1, 'IT Billing', 1);
INSERT INTO DEPARTMENT(ID,DEPT,EMP_ID)
VALUES (2, 'Engineering', 2);
INSERT INTO DEPARTMENT(ID,DEPT,EMP_ID)
VALUES (3, 'Finance',7);
INSERT INTO DEPARTMENT(ID,DEPT,EMP_ID)
VALUES (4, 'Engineering',3);
INSERT INTO DEPARTMENT(ID,DEPT,EMP_ID)
VALUES (5, 'Finance', 4);
INSERT INTO DEPARTMENT(ID,DEPT,EMP_ID)
VALUES (6, 'Engineering', 5);
INSERT INTO DEPARTMENT(ID,DEPT,EMP_ID)
VALUES (7, 'Finance', 6);
```

```
select * from COMPANY;
SELECT * FROM department;
```

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER  
JOIN DEPARTMENT  
ON COMPANY.ID = DEPARTMENT.EMP_ID  
UNION  
SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT  
OUTER JOIN DEPARTMENT  
ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

```
SELECT EMP_ID, NAME, DEPT FROM COMPANY INNER  
JOIN DEPARTMENT  
ON COMPANY.ID = DEPARTMENT.EMP_ID  
UNION ALL  
SELECT EMP_ID, NAME, DEPT FROM COMPANY LEFT  
OUTER JOIN DEPARTMENT  
ON COMPANY.ID = DEPARTMENT.EMP_ID;
```

```
UPDATE COMPANY SET ADDRESS = NULL, SALARY =  
NULL where ID IN(6,7);
```

```
SELECT ID, NAME, AGE, ADDRESS, SALARY  
FROM COMPANY  
WHERE SALARY IS NOT NULL;
```

```
SELECT ID, NAME, AGE, ADDRESS, SALARY  
FROM COMPANY  
WHERE SALARY IS NULL;
```

```
SELECT C.ID, C.NAME, C.AGE, D.DEPT  
FROM COMPANY AS C, DEPARTMENT AS D
```

```
WHERE C.ID = D.EMP_ID;
```

```
SELECT  C.ID    AS    COMPANY_ID,  C.NAME    AS  
        COMPANY_NAME, C.AGE, D.DEPT  
FROM COMPANY AS C, DEPARTMENT AS D  
WHERE C.ID = D.EMP_ID;
```

```
-----  
ALTER TABLE COMPANY RENAME TO OLD_COMPANY;  
ALTER TABLE OLD_COMPANY ADD COLUMN SEX char(1);
```

```
DELETE FROM COMPANY;  
VACUUM;
```

```
-----  
  
SELECT *  
FROM COMPANY  
WHERE ID IN (SELECT ID  
FROM COMPANY  
WHERE SALARY > 45000) ;
```

```
-----  
-----  
-----  
  
CREATE TABLE COMPANY(  
ID INTEGER PRIMARY KEY AUTOINCREMENT,  
NAME TEXT NOT NULL,  
AGE INT NOT NULL,  
ADDRESS CHAR(50),  
SALARY REAL  
);
```

```

INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY)
VALUES ( 'Paul', 32, 'California', 20000.00 );
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY)
VALUES ( 'Allen', 25, 'Texas', 15000.00 );
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY)
VALUES ( 'Teddy', 23, 'Norway', 20000.00 );
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY)
VALUES ( 'Mark', 25, 'Rich-Mond ', 65000.00 );
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY)
VALUES ( 'David', 27, 'Texas', 85000.00 );
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY)
VALUES ( 'Kim', 22, 'South-Hall', 45000.00 );
INSERT INTO COMPANY (NAME,AGE,ADDRESS,SALARY)
VALUES ( 'James', 24, 'Houston', 10000.00 );

```

.header on

.mode column

```
SELECT * FROM COMPANY;
```

-----

```
SELECT date('now');
```

```
SELECT date('now','start of month','+1 month','-1 day');
```

### SQLite COUNT Function

The SQLite COUNT aggregate function is used to count the number of rows in a database table.

### SQLite MAX Function

The SQLite MAX aggregate function allows us to select the highest (maximum) value for a certain column.

#### SQLite MIN Function

The SQLite MIN aggregate function allows us to select the lowest (minimum) value for a certain column.

#### SQLite AVG Function

The SQLite AVG aggregate function selects the average value for certain table column.

#### SQLite SUM Function

The SQLite SUM aggregate function allows selecting the total for a numeric column.

#### SQLite RANDOM Function

The SQLite RANDOM function returns a pseudo-random integer between -9223372036854775808 and +9223372036854775807.

```
SELECT random() AS Random;
```

#### SQLite ABS Function

The SQLite ABS function returns the absolute value of the numeric argument.

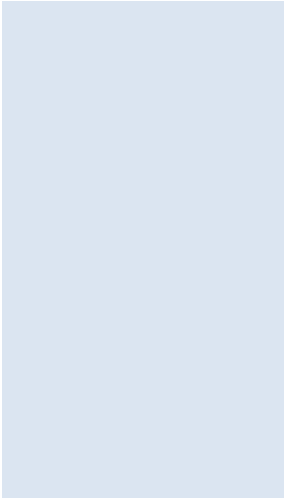
```
SELECT abs(5), abs(-15), abs(NULL), abs(0), abs('ABC');
```

#### SQLite UPPER Function

The SQLite UPPER function converts a string into upper-case letters.

#### SQLite LOWER Function





The SQLite LOWER function converts a string into lower-case letters.

### SQLite LENGTH Function

The SQLite LENGTH function returns the length of a string.



