

Database relazionali SQL

1. Entità
2. Modelli
3. Definizione di database
4. Nascita e scopo dei database relazionali
5. Applicazioni dei database relazionali
6. Definizione di transazione
7. Integrità e coerenza dei dati
8. Le proprietà ACID dei database relazionali
9. Progettazione di un database
10. Modello Entità-Relazioni
11. Draw.io per (app online per creare gli schemi)
12. Modello Logico (tabelle righe e colonne)
13. Normalizzazione del database (le tre forme normali)
14. Modello fisico e XAMPP

Premessa

Entità

In informatica una entità , può essere una cosa reale che esiste e che si può toccare ma può essere anche una cosa astratta esempio:

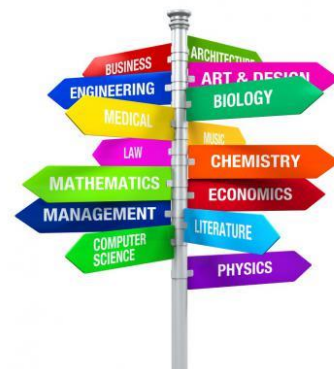
una materia scolastica, un corso universitario, una poesia, un commento ecc...

Esempi in informatica:

- Un personal computer, un cellulare, una stampante, un tablet, un server, un computer, ecc..
- Una automobile, un robot, uno scooter, una lavatrice, una nave, un trattore, un distributore di bevande o di carburante , ecc..
- Una persona, un animale
- Una poesia, una materia scolastica, un corso universitario, ecc...
- Un libro, autore, editore, prestito, ecc..
- Un dottore, una ricetta, diagnosi, percorso riabilitativo ecc..



Liceo Scientifico					
	1° BIENNIO		2° BIENNIO		
	1° ANNO	2° ANNO	3° ANNO	4° ANNO	5° ANNO
Lingua e letteratura italiana	4	4	4	4	4
Lingua e cultura latina	3	3	3	3	3
Lingua e cultura straniera	3	3	3	3	3
Diritto ed economia	1	1			
Storia e geografia	3	3			
Storia			2	2	2
Filosofia			3	3	3
Matematica	5	5	4	4	4
Fisica	2	2	3	3	3
Scienze naturali	2	2	3	3	3
Disegno e storia dell'arte	2	2	2	2	2
Scienze motorie e sportive	2	2	2	2	2
Religione cattolica o Att. alternative	1	1	1	1	1
Totale ore settimanali	28	28	30	30	30



I modelli

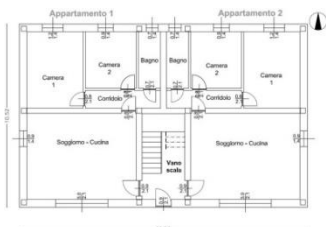
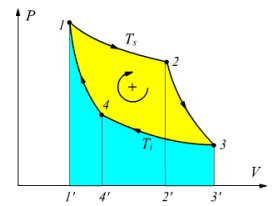
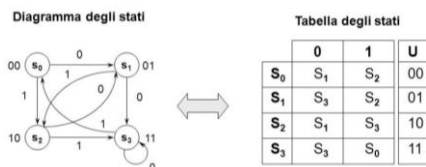
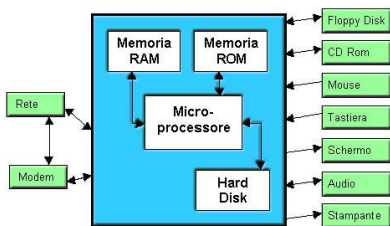
Un modello è una rappresentazione semplificata di una realtà che si vuole rappresentare :

Esempi di modelli possono essere:

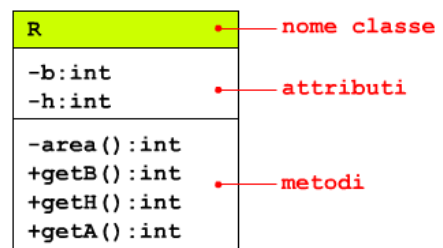
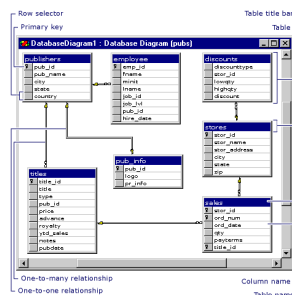
- Un diagramma, una tabella, un grafico, un grafo, una piantina di un edificio
- Un insieme di equazioni matematiche
- Un programma (software) con simulazione al computer
- Un disegno al cad, una simulazione al cad
- Un database, un modello OOP (classi ed oggetti)

L'obiettivo di un modello è di aiutarci a capire come funziona un sistema reale, a progettarlo, o a prevederne il comportamento

Vedremo che un database sarà un insieme di entità legate tra loro tramite relazioni



$$\begin{cases} y = 2x + 1 \\ 3x + y = 0 \end{cases}$$



Definizione di database (archivio elettronico)



Un database è un insieme di file in binario salvato su disco?"

No, la definizione di un database come "un insieme di file in binario salvato su disco" è una **descrizione molto semplificata e incompleta**. Sebbene i dati di un database alla fine vengano memorizzati su un supporto fisico come un disco rigido in formato binario, un database è **molto di più** di una semplice collezione di file.

Ecco una definizione più completa e precisa di cosa sia un database:

Un database è **una collezione organizzata di dati strutturati**, tipicamente memorizzata elettronicamente in un sistema informatico, in modo da poter essere facilmente accessibile, gestita e aggiornata.

Vediamo i punti chiave di questa definizione:

- **Collezione organizzata:** I dati in un database non sono semplicemente ammassati. Sono strutturati in modo logico per rappresentare entità, relazioni e informazioni del mondo reale. Questa organizzazione permette di trovare e gestire i dati in modo efficiente.
- **Dati strutturati:** I dati hanno un formato definito (ad esempio, tabelle con righe e colonne, documenti con campi specifici, grafi con nodi e archi). Questa struttura consente di interrogare e analizzare i dati in modo significativo.

- **Tipicamente memorizzata elettronicamente:** La maggior parte dei database moderni sono implementati su sistemi informatici e memorizzano i dati digitalmente.
- **Facilmente accessibile, gestita e aggiornata:** Un database è progettato per consentire agli utenti e alle applicazioni di interagire con i dati in modo controllato e sicuro. Questo include operazioni come l'inserimento, la lettura, la modifica e l'eliminazione dei dati.

Oltre alla semplice memorizzazione di file binari, un database include:

- **Schema del database:** La definizione della struttura del database, incluse le tabelle, le colonne, i tipi di dati, le relazioni tra le tabelle e i vincoli di integrità.
- **Sistema di gestione di database (DBMS):** Un software che permette di interagire con il database. Il DBMS fornisce funzionalità per definire, creare, interrogare, aggiornare e amministrare il database. Esempi di DBMS includono MySQL, PostgreSQL, Oracle, SQL Server e MongoDB.
- **Linguaggio di interrogazione:** Un linguaggio specifico (come SQL per i database relazionali) utilizzato per interagire con il database, recuperare informazioni e manipolare i dati.
- **Meccanismi di controllo e sicurezza:** Funzionalità per garantire l'integrità dei dati, la concorrenza degli accessi e la sicurezza contro accessi non autorizzati.
- **Indici e altre strutture di dati:** Ottimizzazioni per velocizzare le operazioni di ricerca e recupero dei dati.

In sintesi:

Un database è un sistema complesso che va ben oltre una semplice raccolta di file binari. È una struttura organizzata e gestita da un software specifico (DBMS) per archiviare, recuperare e manipolare dati in modo efficiente e affidabile. I file binari sono solo il modo in cui i dati vengono fisicamente memorizzati sul disco, ma la vera essenza di un database risiede nella sua organizzazione logica, nella sua struttura e nelle funzionalità offerte dal DBMS.

Nascita e scopo dei database relazionali

I database relazionali sono nati negli **anni 70**, circa 50 anni fa .

La loro nascita è principalmente dovuta alla necessità di superare le limitazioni dei modelli di **database precedenti, come i modelli gerarchici e reticolari, che presentavano diverse problematiche:**

- **Complessità nella gestione dei dati:** Era difficile rappresentare relazioni complesse tra i dati e apportare modifiche alla struttura del database senza influenzare le applicazioni.
- **Ridondanza dei dati:** Spesso le stesse informazioni venivano ripetute più volte, causando spreco di spazio e potenziali inconsistenze.
- **Difficoltà nell'interrogazione:** Recuperare informazioni specifiche richiedeva procedure complesse e poco flessibili.
- **Mancanza di standard:** Non esisteva un linguaggio di interrogazione standardizzato, rendendo difficile la portabilità delle applicazioni tra diversi sistemi di database.

Il modello relazionale, introdotto da **Edgar F. Codd mentre lavorava per IBM**, proponeva un approccio basato sulla teoria matematica degli insiemi e delle relazioni. L'idea fondamentale era di organizzare i dati in **tabelle**, dove **ogni tabella rappresenta un'entità** e le **colonne rappresentano gli attributi**. Le relazioni tra le entità vengono espresse attraverso **chiavi e vincoli**. **Le righe dette tuple rappresentano le istanze dell'entità**

Tabella Clienti:

ID_Cliente	Nome	Cognome	Email
1	Alice	Neri	aliceneri@gmail.com
2	Bob	Rossi	bobrossi@gmail.com

Tabella Ordini:

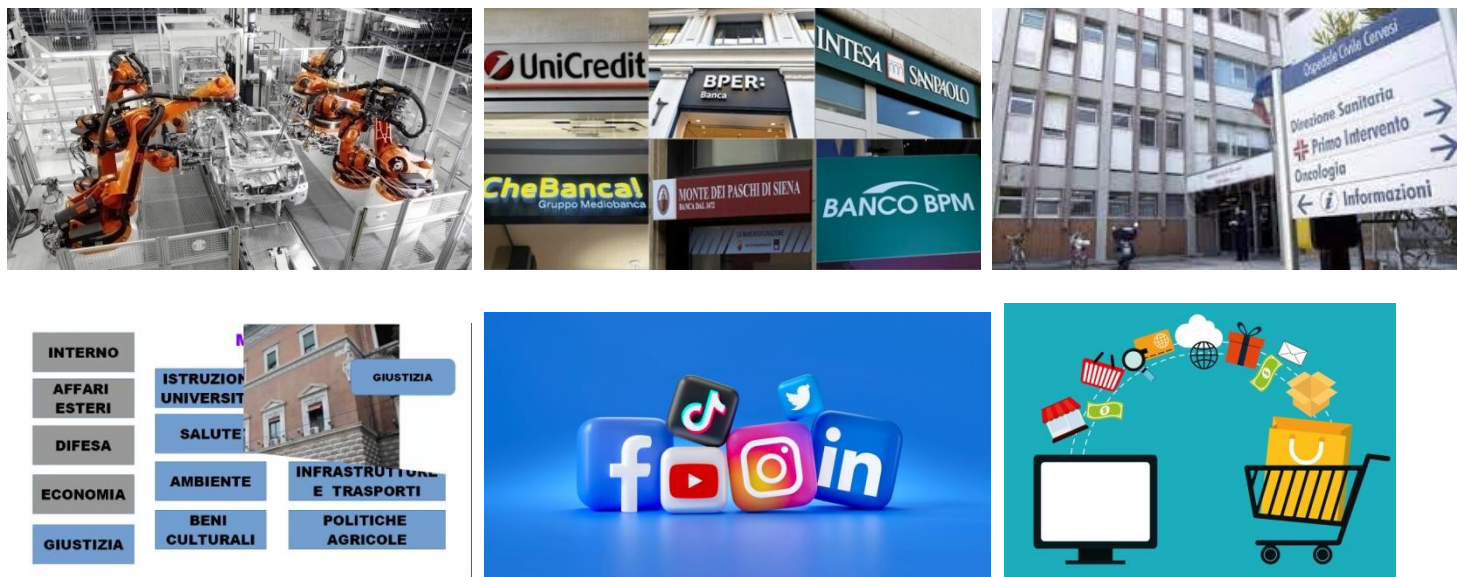
ID_Ordine	ID_Cliente	Data_Ordine	Totale
101	1	2024-04-20	55.00
102	2	2024-04-20	22.50

Questo modello offriva numerosi vantaggi:

- **Struttura semplice e intuitiva:** L'organizzazione dei dati in tabelle rendeva il modello facile da comprendere e utilizzare.
- **Eliminazione della ridondanza:** La normalizzazione delle tabelle permetteva di ridurre al minimo la duplicazione dei dati.
- **Flessibilità e facilità di modifica:** La struttura tabellare consentiva di apportare modifiche allo schema del database con un impatto minimo sulle applicazioni esistenti.
- **Potente linguaggio di interrogazione:** L'introduzione di **SQL (Structured Query Language)** forniva un linguaggio standardizzato per interrogare e manipolare i dati in modo efficiente e flessibile.

- **Integrità dei dati:** L'uso di chiavi e vincoli permetteva di garantire la coerenza e l'accuratezza dei dati. Grazie a questi vantaggi, il modello relazionale si è rapidamente affermato come lo standard per i sistemi di gestione di database (DBMS) ed è **tuttora** la tecnologia più diffusa per la gestione dei dati nella maggior parte delle applicazioni aziendali e non.

Applicazioni



I database relazionali sono la spina dorsale di moltissime applicazioni e sistemi informativi in svariati settori. La loro capacità di organizzare, gestire e interrogare grandi quantità di dati strutturati in modo efficiente li rende indispensabili in numerosi ambiti, tra cui:

Settore **Aziendale e Commerciale:**

- **Gestione delle Relazioni con i Clienti (CRM):** Tracciare interazioni, dati di contatto, storico degli acquisti e preferenze dei clienti.
- **Pianificazione delle Risorse Aziendali (ERP):** Integrare processi aziendali come finanza, contabilità, risorse umane, produzione e gestione della catena di approvvigionamento.
- **Gestione degli Ordini e Fatturazione:** Registrare e monitorare ordini, generare fatture e gestire i pagamenti.
- **Gestione dell'Inventario:** Tracciare i livelli di stock, le movimentazioni di magazzino e gestire gli approvvigionamenti.
- **Sistemi di Punto Vendita (POS):** Registrare le transazioni di vendita, gestire i prezzi e monitorare le vendite.
- **Business Intelligence e Data Warehousing:** Archiviare e analizzare dati provenienti da diverse fonti per generare report, identificare trend e supportare le decisioni aziendali.

Settore **Finanziario:**

- **Sistemi Bancari:** Gestire conti correnti, transazioni, prestiti, mutui e informazioni sui clienti.
- **Mercati Finanziari:** Registrare e analizzare dati di trading, gestire portafogli di investimento e monitorare le transazioni.
- **Assicurazioni:** Gestire polizze, sinistri, clienti e pagamenti.
- **Rilevamento di Frodi:** Analizzare pattern di dati per identificare attività sospette.

Settore **Sanitario:**

- **Cartelle Cliniche Elettroniche (EHR):** Archiviare e gestire la storia medica dei pazienti, i trattamenti, i farmaci e i risultati degli esami.
- **Gestione degli Appuntamenti:** Organizzare e monitorare gli appuntamenti dei pazienti.
- **Sistemi di Fatturazione Medica:** Gestire le richieste di rimborso e la fatturazione ai pazienti e alle compagnie assicurative.

Settore **Pubblico:**

- **Gestione dei Registri Civili:** Archiviare informazioni su nascite, matrimoni, decessi.
- **Sistemi di Identificazione:** Gestire database di cittadini e documenti di identità.
- **Gestione delle Imposte:** Tracciare i pagamenti e le dichiarazioni fiscali.
- **Sistemi di Votazione:** Registrare gli elettori e gestire i processi di voto.

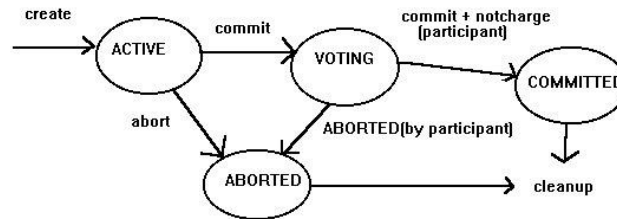
Altri Settori:

- **E-commerce:** Gestire cataloghi di prodotti, ordini, clienti e transazioni online.
- **Social Media:** Gestire profili utente, contenuti, interazioni e relazioni tra utenti.

- **Telecomunicazioni:** Gestire informazioni sugli abbonati, registri delle chiamate e fatturazione.
- **Istruzione:** Gestire informazioni sugli studenti, corsi, voti e iscrizioni.
- **Trasporti e Logistica:** Gestire orari, prenotazioni, tracciamento delle spedizioni e informazioni sui clienti.

In sintesi, i database relazionali sono una tecnologia fondamentale per qualsiasi organizzazione che necessiti di gestire grandi quantità di dati strutturati in modo affidabile, efficiente e sicuro. La loro versatilità e la potenza del linguaggio SQL li rendono adatti a una vasta gamma di applicazioni in quasi tutti i settori.

Transazione



Una **transazione** in un database è una **sequenza di una o più operazioni** (come letture, scritture, modifiche, eliminazioni) che vengono trattate come una **singola unità logica di lavoro**. Pensa a una transazione come a un "tutto o niente": o tutte le operazioni all'interno della transazione vengono completate con successo e le modifiche vengono rese permanenti nel database (questa azione è chiamata **commit**), oppure, se qualcosa va storto durante l'esecuzione, tutte le modifiche vengono annullate e il database torna allo stato precedente all'inizio della transazione (questa azione è chiamata **rollback**).

L'obiettivo principale delle transazioni è garantire l'**integrità** e la **consistenza** dei dati nel database, specialmente in ambienti multiutente dove più operazioni possono avvenire contemporaneamente.

Ecco alcuni esempi per rendere il concetto più chiaro:

Esempio 1: Trasferimento di denaro da un conto bancario all'altro

Immagina di dover trasferire 100€ dal tuo conto A al conto B. Questa operazione può essere vista come una transazione composta da due passaggi:

1. **Decrementare** il saldo del conto A di 100€.
2. **Incrementare** il saldo del conto B di 100€.

È fondamentale che entrambe queste operazioni avvengano con successo. Se, per esempio, il sistema dovesse bloccarsi dopo aver decrementato il conto A ma prima di incrementare il conto B, i dati del database sarebbero inconsistenti (avresti 100€ in meno ma non sarebbero finiti da nessuna parte). Una transazione garantisce che o entrambi i passaggi vengano completati (il trasferimento avviene con successo) o nessuno dei due venga eseguito (il database rimane nello stato iniziale).

Esempio 2: Ordine di un prodotto in un negozio online

Quando effettui un ordine online, diverse operazioni devono avvenire in sequenza:

1. **Verificare** la disponibilità degli articoli richiesti.
2. **Aggiornare** l'inventario, diminuendo la quantità degli articoli ordinati.
3. **Creare** un nuovo record dell'ordine nel database.
4. **Registrare** il pagamento.
5. **Generare** una notifica per la spedizione.

Tutte queste operazioni dovrebbero essere trattate come una singola transazione. Se, per esempio, il pagamento fallisce dopo che l'inventario è stato aggiornato, la transazione dovrebbe essere annullata per ripristinare l'inventario e non creare l'ordine.

Esempio 3: Registrazione di un nuovo utente

La registrazione di un nuovo utente in un sistema potrebbe comportare:

1. **Inserire** un nuovo record nella tabella degli utenti con le informazioni fornite.
2. **Inserire** un record correlato nella tabella dei profili utente (se presente).
3. **Assegnare** eventuali ruoli o permessi predefiniti.

Se una qualsiasi di queste operazioni fallisce, l'intera registrazione dovrebbe essere annullata per evitare di avere dati parziali o inconsistenti nel database.

In sintesi, una transazione è un meccanismo fondamentale nei sistemi di gestione di database per assicurare l'affidabilità e la coerenza dei dati, raggruppando una serie di operazioni correlate in un'unica unità indivisibile di lavoro.

Integrità e coerenza dei dati

In un contesto di database, **integrità dei dati** e **coerenza dei dati** sono concetti strettamente correlati ma con sfumature distinte. Entrambi sono fondamentali per garantire l'affidabilità e l'utilità di un database.

Integrità dei Dati: Si riferisce all'accuratezza, alla validità e alla completezza dei dati presenti nel database. Un database con elevata integrità contiene **dati corretti, non corrotti, non duplicati** (a meno che non sia intenzionale), e che rispettano i vincoli e le regole definiti per la sua struttura. L'integrità si concentra sulla qualità intrinseca dei singoli dati e delle loro relazioni.

Coerenza dei Dati: Si riferisce allo stato del database in cui tutti i dati sono in accordo tra loro e riflettono la realtà che modellano in modo logico e sensato. **Un database coerente rispetta tutte le regole e i vincoli definiti, mantenendo uno stato valido attraverso le transazioni.** La coerenza si concentra sul mantenimento di uno stato valido del database nel tempo, specialmente durante e dopo le operazioni di modifica.

In sintesi:

- **Integrità:** I dati sono "giusti" e "completi" secondo le definizioni e i vincoli.
- **Coerenza:** Il database passa da uno stato "valido" a un altro stato "valido" dopo ogni transazione.

Esempio dove integrità e coerenza sono rispettate:

Considera un database per la gestione degli ordini di un negozio online con due tabelle correlate: "Clienti" e "Ordini".

Tabella Clienti:

ID_Cliente	Nome	Email
1	Alice	[indirizzo email rimosso]
2	Bob	[indirizzo email rimosso]

Tabella Ordini:

ID_Ordine	ID_Cliente	Data_Ordine	Totale
101	1	2024-04-20	55.00
102	2	2024-04-20	22.50

Integrità rispettata:

- Il campo ID_Cliente in entrambe le tabelle è definito come chiave primaria (in Clienti) e chiave esterna (in Ordini), garantendo l'integrità referenziale: ogni ordine è associato a un cliente esistente.
- Il campo Email nella tabella Clienti potrebbe avere un vincolo di formato per assicurare che sia un indirizzo email valido.
- Il campo Totale nella tabella Ordini potrebbe avere un vincolo per essere un valore numerico positivo.
- Non ci sono clienti o ordini duplicati (basandosi sulla chiave primaria).

Coerenza rispettata:

Se un nuovo ordine viene inserito nella tabella "Ordini" con un ID_Cliente che esiste nella tabella "Clienti", il database rimane in uno stato coerente. La transazione di inserimento dell'ordine rispetta il vincolo di chiave esterna. Se una transazione tentasse di inserire un ordine con un ID_Cliente inesistente, il sistema (grazie all'integrità referenziale) impedirebbe l'operazione, mantenendo la coerenza del database (non ci sarebbero ordini "orfani").

Esempio dove integrità e coerenza NON sono rispettate:

Riprendiamo lo stesso scenario, ma immaginiamo che non siano stati definiti vincoli di chiave esterna tra le tabelle.

Tabella Clienti:

ID_Cliente	Nome	Email
1	Alice	[indirizzo email rimosso]
2	Bob	[indirizzo email rimosso]

Tabella Ordini:

ID_Ordine	ID_Cliente	Data_Ordine	Totale
101	1	2024-04-20	55.00

ID_Ordine	ID_Cliente	Data_Ordine	Totale
102	3	2024-04-20	22.50
103	1	2024-04-21	-10.00
104	NULL	2024-04-21	100.00

Integrità NON rispettata:

- L'ordine con ID_Ordine 102 ha un ID_Cliente (3) che non esiste nella tabella "Clienti". Questo viola l'integrità referenziale.
- L'ordine con ID_Ordine 103 ha un Totale negativo (-10.00), violando un potenziale vincolo che richiederebbe valori positivi.
- L'ordine con ID_Ordine 104 ha un ID_Cliente NULL, il che potrebbe non essere consentito se ogni ordine deve essere associato a un cliente.

Coerenza NON rispettata:

Il database si trova in uno stato incoerente perché la tabella "Ordini" contiene record che non hanno una corrispondenza valida nella tabella "Clienti" (ordine 102). Questo rompe la logica del sistema di gestione degli ordini, dove ogni ordine dovrebbe appartenere a un cliente. Inoltre, la presenza di un totale negativo (ordine 103) va contro la logica di un costo e rende il database semanticamente incoerente.

In conclusione, l'integrità si concentra sulla correttezza dei singoli dati e delle loro relazioni statiche, mentre la coerenza si concentra sul mantenimento di uno stato valido del database durante le transazioni e nel tempo, rispettando le regole aziendali e i vincoli definiti. Un database con elevata integrità è un prerequisito per raggiungere una buona coerenza.

ACID

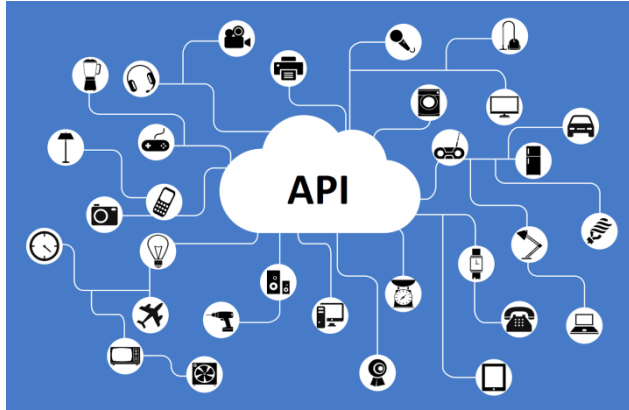
In ambito di database, l'acronimo **ACID** si riferisce a un **insieme di proprietà** che garantiscono che le transazioni di database siano elaborate in modo affidabile. Ogni lettera dell'acronimo rappresenta una proprietà fondamentale:

- **Atomicità (Atomicity):** Una transazione viene trattata come una singola unità indivisibile di lavoro. O tutte le modifiche all'interno della transazione vengono completate con successo (commit), oppure nessuna di esse viene applicata al database (rollback). Non ci possono essere stati intermedi.
 - **Esempio pratico:** Immagina un trasferimento di denaro da un conto bancario A a un conto bancario B. L'operazione consiste in due passaggi:
 1. Decrementare l'importo dal conto A.
 2. Incrementare l'importo nel conto B. Se il sistema fallisce dopo il primo passaggio (ad esempio, un'interruzione di corrente), l'atomicità garantisce che la transazione venga annullata (rollback). L'importo non viene detratto dal conto A senza essere accreditato sul conto B, mantenendo l'integrità dei dati.
- **Consistenza (Consistency):** Una transazione porta il database da uno stato valido a un altro stato valido. Tutte le regole e i vincoli definiti nel database (come chiavi primarie, chiavi esterne, vincoli di integrità referenziale, ecc.) devono essere rispettati.
 - **Esempio pratico:** Considera una tabella "Ordini" con un vincolo che impone che il campo "quantità" sia sempre un numero intero positivo. Se una transazione tenta di inserire un ordine con una "quantità" negativa o non numerica, la proprietà di consistenza farà sì che la transazione fallisca, impedendo l'inserimento di dati non validi nel database.
- **Isolamento (Isolation):** Transazioni multiple che vengono eseguite contemporaneamente devono essere isolate l'una dall'altra. Il risultato finale dell'esecuzione concorrente di più transazioni deve essere lo stesso che si otterrebbe se le transazioni fossero eseguite in sequenza (serialmente).
 - **Esempio pratico:** Supponiamo che due utenti stiano tentando di aggiornare contemporaneamente il saldo di un prodotto disponibile in un magazzino.
 - L'utente 1 acquista 5 unità.
 - L'utente 2 acquista 3 unità. Senza un adeguato isolamento, potrebbe verificarsi una situazione in cui entrambi gli utenti leggono lo stesso saldo iniziale, e dopo gli aggiornamenti, il saldo finale non riflette la somma corretta delle sottrazioni (potrebbe essere stato sottratto solo 5 o 3 invece di 8). L'isolamento garantisce che le transazioni vengano eseguite in modo tale che il risultato finale sia corretto, come se fossero avvenute una dopo l'altra. Diversi livelli di isolamento (ad esempio, Read Committed, Repeatable Read, Serializable) offrono diversi compromessi tra concorrenza e coerenza.
- **Durabilità (Durability):** Una volta che una transazione è stata commessa (commit), le modifiche apportate al database sono permanenti e non devono andare perse a causa di guasti del sistema (come interruzioni di corrente, crash del sistema operativo, ecc.).
 - **Esempio pratico:** Dopo che un cliente ha completato con successo un ordine online e la transazione è stata commessa, i dettagli dell'ordine devono essere memorizzati in modo permanente nel database. Anche se il server del database dovesse subire un guasto subito dopo il commit, al

riavvio, i dati dell'ordine devono essere ancora presenti e integri. Questo di solito si ottiene attraverso meccanismi come la registrazione delle transazioni (transaction logging) e i backup del database. In sintesi, le proprietà ACID sono cruciali per garantire l'affidabilità, l'integrità e la coerenza dei dati in un sistema di gestione di database, specialmente in ambienti multiutente e transazionali.

Le applicazioni client- server si compongono di tre parti

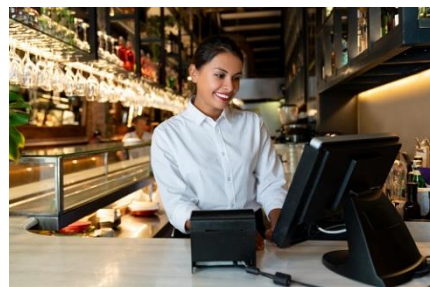
Il SERVER, il CLIENT e l' API

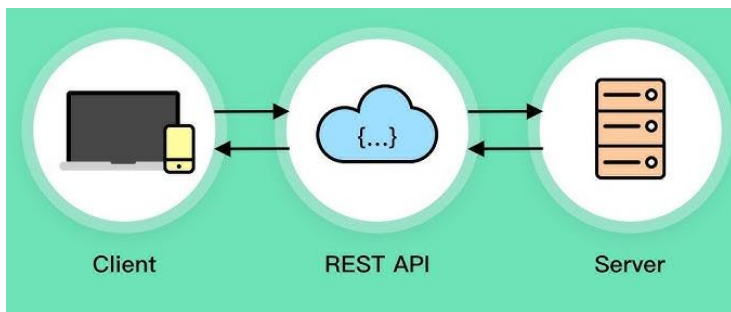


Viviamo in un mondo dove tutti sono interconnessi tra di loro, dove in pochi secondi riusciamo a mandare un messaggio ad una persona distante anche migliaia di chilometri e dove, comodamente dal letto con il nostro smartphone, potremmo anche programmare l'attivazione della macchina del caffè; ma **come avviene questa connessione**? Come fanno diversi dispositivi (tablet, pc, smartphone) e applicazioni a permetterlo?

L'eroe silenzioso e poco citato che permette tutto questo sono le **Application Programming Interface** o meglio conosciute come **API**.

Immagina di andare al ristorante e di ordinare del cibo o chiedere informazioni in merito ad un piatto. Sicuramente, per fare una di queste azioni interagirai con un **cameriere**, rivolgendoti a lui anche per chiedere e pagare il conto oppure per altre informazioni sul menù.





API



Progettazione database

La progettazione di un database relazionale è un processo cruciale per garantire che il sistema sia efficiente, affidabile e soddisfi le esigenze degli utenti. Generalmente, questo processo è suddiviso in diverse fasi logiche. Vediamole insieme:

1. Analisi dei Requisiti:

Questa è la fase iniziale e fondamentale. L'obiettivo è comprendere a fondo le esigenze degli utenti e i dati che il database dovrà contenere e gestire. Questo implica:

- **Raccolta di informazioni:** Interviste con gli utenti, analisi di documenti esistenti, osservazione dei processi aziendali.
- **Identificazione delle entità:** Individuazione degli oggetti o concetti significativi di cui si devono memorizzare informazioni (ad esempio, clienti, ordini, prodotti).
- **Identificazione degli attributi:** Definizione delle proprietà di ciascuna entità (ad esempio, per il cliente: nome, cognome, indirizzo).
- **Identificazione delle relazioni:** Comprensione di come le diverse entità sono collegate tra loro (ad esempio, un cliente può effettuare più ordini).
- **Definizione dei vincoli:** Individuazione delle regole che devono essere rispettate dai dati (ad esempio, l'ID cliente deve essere univoco).

2. Progettazione Concettuale:

In questa fase, si crea un modello concettuale del database, indipendente da qualsiasi specifica implementazione tecnologica. Il risultato è un diagramma che rappresenta le entità, i loro attributi e le relazioni tra di esse. Il modello Entità-Relazione (ER) è uno strumento comune utilizzato in questa fase.

- **Creazione del diagramma ER:** Rappresentazione grafica delle entità come rettangoli, degli attributi come ellissi e delle relazioni come rombi con linee di collegamento.
- **Definizione della cardinalità delle relazioni:** Specificare quanti elementi di un'entità possono essere collegati a quanti elementi di un'altra entità (uno-a-uno, uno-a-molti, molti-a-molti).

3. Progettazione Logica:

Questa fase trasforma il modello concettuale in un modello logico, che è specifico per il tipo di sistema di gestione di database (DBMS) relazionale che si intende utilizzare. L'obiettivo è definire la struttura delle tabelle, le colonne e le chiavi.

- **Mappatura delle entità in tabelle:** Ogni entità del modello concettuale viene trasformata in una tabella.
- **Mappatura degli attributi in colonne:** Gli attributi di ciascuna entità diventano le colonne della corrispondente tabella.
- **Definizione delle chiavi primarie:** Scelta di uno o più attributi che identificano univocamente ogni riga di una tabella.
- **Definizione delle chiavi esterne:** Identificazione delle colonne utilizzate per stabilire e mantenere le relazioni tra le tabelle.
- **Normalizzazione:** Processo di ottimizzazione della struttura delle tabelle per ridurre la ridondanza dei dati e migliorare l'integrità. Questo implica l'applicazione di diverse forme normali (1NF, 2NF, 3NF, ecc.).

4. Progettazione Fisica:

In questa fase, si decide come il modello logico verrà implementato fisicamente nel DBMS scelto. Questo include la scelta delle strutture di memorizzazione, degli indici e delle tecniche di ottimizzazione delle prestazioni.

- **Scelta del DBMS:** Selezione del sistema di gestione di database relazionale più adatto alle esigenze (ad esempio, MySQL, PostgreSQL, Oracle).
- **Definizione delle tabelle e delle colonne nel DBMS:** Creazione effettiva delle tabelle e delle colonne nel sistema di database, specificando i tipi di dati, le dimensioni e i vincoli.
- **Creazione degli indici:** Definizione degli indici per accelerare le operazioni di ricerca e recupero dei dati.
- **Ottimizzazione delle prestazioni:** Considerazioni sulla gestione dello spazio di archiviazione, sulle strategie di backup e recovery, e sull'ottimizzazione delle query.

5. Implementazione:

Questa è la fase in cui il database viene effettivamente creato e popolato con i dati.

- **Creazione dello schema del database:** Esecuzione degli script SQL (Structured Query Language) per creare le tabelle, le chiavi, i vincoli e gli indici definiti nella fase di progettazione fisica.
- **Popolamento del database:** Inserimento dei dati iniziali nelle tabelle. Questo può avvenire manualmente, tramite script o importando dati da altri sistemi.
- **Sviluppo delle applicazioni:** Creazione delle applicazioni che interagiranno con il database per inserire, modificare, eliminare e recuperare i dati.

6. Test e Valutazione:

Una volta implementato, il database deve essere testato per verificarne la funzionalità, le prestazioni e l'affidabilità.

- **Test funzionali:** Verifica che il database soddisfi i requisiti degli utenti e che le operazioni di base (inserimento, modifica, eliminazione, ricerca) funzionino correttamente.
- **Test di performance:** Misurazione dei tempi di risposta delle query e delle transazioni per assicurarsi che il sistema sia efficiente.
- **Test di stress:** Valutazione del comportamento del database sotto carichi di lavoro elevati.
- **Test di sicurezza:** Verifica che il database sia protetto da accessi non autorizzati e da potenziali minacce.

7. Manutenzione ed Evoluzione:

Dopo la messa in produzione, il database richiede una manutenzione continua e potrebbe dover evolvere nel tempo per adattarsi a nuove esigenze.

- **Monitoraggio:** Controllo costante delle prestazioni del database e dell'utilizzo delle risorse.
- **Backup e recovery:** Implementazione di procedure per il backup regolare dei dati e per il ripristino in caso di guasti.
- **Ottimizzazione continua:** Tuning delle query e della struttura del database per migliorare le prestazioni.
- **Aggiornamenti e patch:** Applicazione degli aggiornamenti e delle patch del DBMS per risolvere bug e migliorare la sicurezza.
- **Modifiche allo schema:** Adattamento della struttura del database per supportare nuove funzionalità o modifiche ai requisiti.

Queste fasi non sono sempre strettamente sequenziali e spesso si sovrappongono o richiedono iterazioni. Tuttavia, seguire un approccio strutturato alla progettazione di un database relazionale è fondamentale per creare un sistema robusto e efficace.

Modello Entità-relazioni

Parliamo del modello Entità-Relazione (E/R) e creiamo un esempio pratico per una biblioteca.

Il **modello Entità-Relazione (E/R)** è un modello concettuale di alto livello utilizzato nella progettazione di database. Il suo scopo principale è quello di descrivere la struttura di un database in modo indipendente dal sistema di gestione di database (DBMS) specifico che verrà utilizzato. In altre parole, è un "blueprint" che rappresenta le diverse entità coinvolte in un sistema informativo e le relazioni che intercorrono tra di esse. L'idea alla base del modello E/R è di rappresentare il mondo reale in termini di:

- **Entità:** Oggetti o concetti distinti del mondo reale che hanno un'esistenza indipendente e che sono di interesse per il sistema che stiamo modellando. Le entità sono rappresentate graficamente da rettangoli. Esempi di entità in una biblioteca potrebbero essere "Libro", "Autore", "Utente".
- **Attributi:** Proprietà che descrivono un'entità. Ogni entità ha un insieme di attributi. Gli attributi sono rappresentati graficamente da ellissi collegate all'entità a cui appartengono. Ad esempio, l'entità "Libro" potrebbe avere gli attributi "Titolo", "ISBN", "AnnoPubblicazione", "NumeroCopie". Alcuni attributi possono essere identificatori univoci di un'entità (chiavi primarie), spesso sottolineati nel diagramma.
- **Relazioni:** Associazioni tra due o più entità. Le relazioni descrivono come le entità interagiscono tra loro. Le relazioni sono rappresentate graficamente da rombi collegati alle entità coinvolte. Il tipo di relazione

(uno-a-uno, uno-a-molti, molti-a-molti) viene spesso indicato sul rombo o sulle linee di collegamento. Ad esempio, tra le entità "Libro" e "Autore" potrebbe esserci una relazione "ScrittoDa".

Esempio di Modello E/R per una Biblioteca:

Immaginiamo di voler progettare un database per gestire una biblioteca. Identifichiamo alcune entità chiave e le loro relazioni:

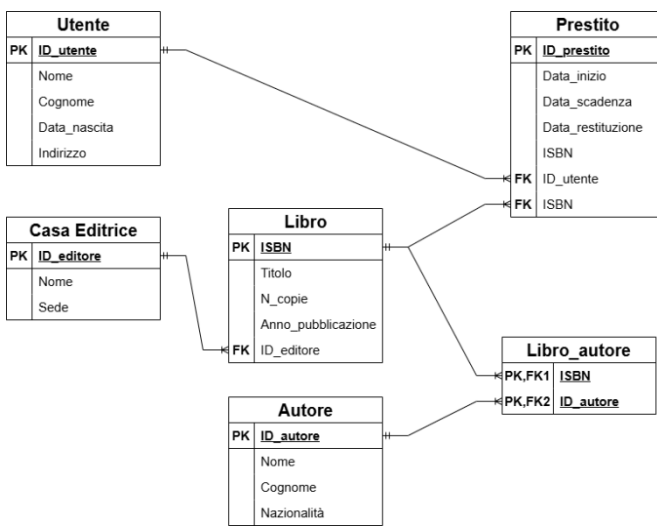
Entità:

- **Libro:**
 - Attributi:
 - ISBN (Chiave Primaria)
 - Titolo
 - AnnoPubblicazione
 - NumeroCopie
 - ID_Editore (Chiave Esterna, riferimento all'entità Editore)
- **Autore:**
 - Attributi:
 - ID_Autore (Chiave Primaria)
 - Nome
 - Cognome
 - Nazionalità
- **Utente:**
 - Attributi:
 - ID_Utente (Chiave Primaria)
 - Nome
 - Cognome
 - DataNascita
 - Indirizzo
- **Prestito:**
 - Attributi:
 - ID_Prestito (Chiave Primaria)
 - DataInizio
 - DataScadenza
 - DataRestituzione (può essere nullo)
 - ISBN (Chiave Esterna, riferimento all'entità Libro)
 - ID_Utente (Chiave Esterna, riferimento all'entità Utente)
- **Editore:**
 - Attributi:
 - ID_Editore (Chiave Primaria)
 - Nome
 - Sede

Relazioni:

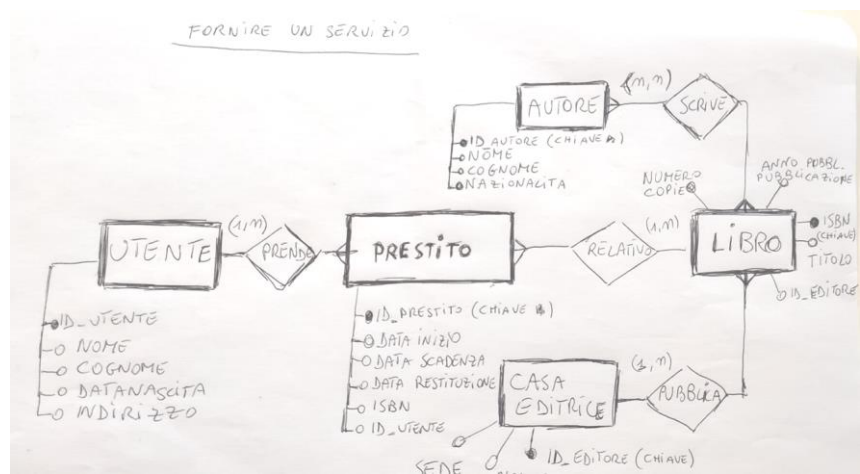
- **ScrittoDa:** Relazione tra l'entità **Libro** e l'entità **Autore**. Un libro può essere scritto da uno o più autori, e un autore può aver scritto più libri (relazione molti-a-molti). Per gestire questa relazione molti-a-molti, spesso si introduce una tabella associativa chiamata "Scrittura" con le chiavi esterne ID_Libro (riferimento a Libro) e ID_Autore (riferimento ad Autore) come chiave primaria composta.
- **PubblicatoDa:** Relazione tra l'entità **Libro** e l'entità **Editore**. Un libro è pubblicato da un solo editore, e un editore può pubblicare più libri (relazione uno-a-molti).
- **PrestatoA:** Relazione tra l'entità **Prestito** e l'entità **Utente**. Un prestito è associato a un solo utente, e un utente può avere più prestiti (relazione uno-a-molti).
- **Riguarda:** Relazione tra l'entità **Prestito** e l'entità **Libro**. Un prestito riguarda un solo libro, e un libro può essere oggetto di più prestiti (relazione uno-a-molti).

Rappresentazione Grafica (concettuale):

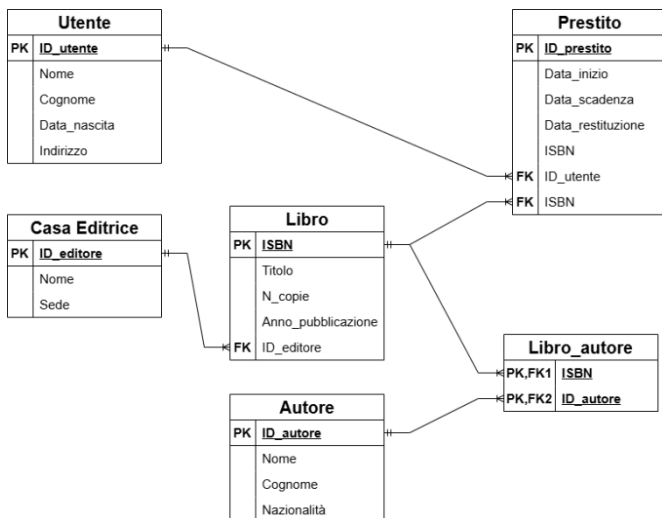
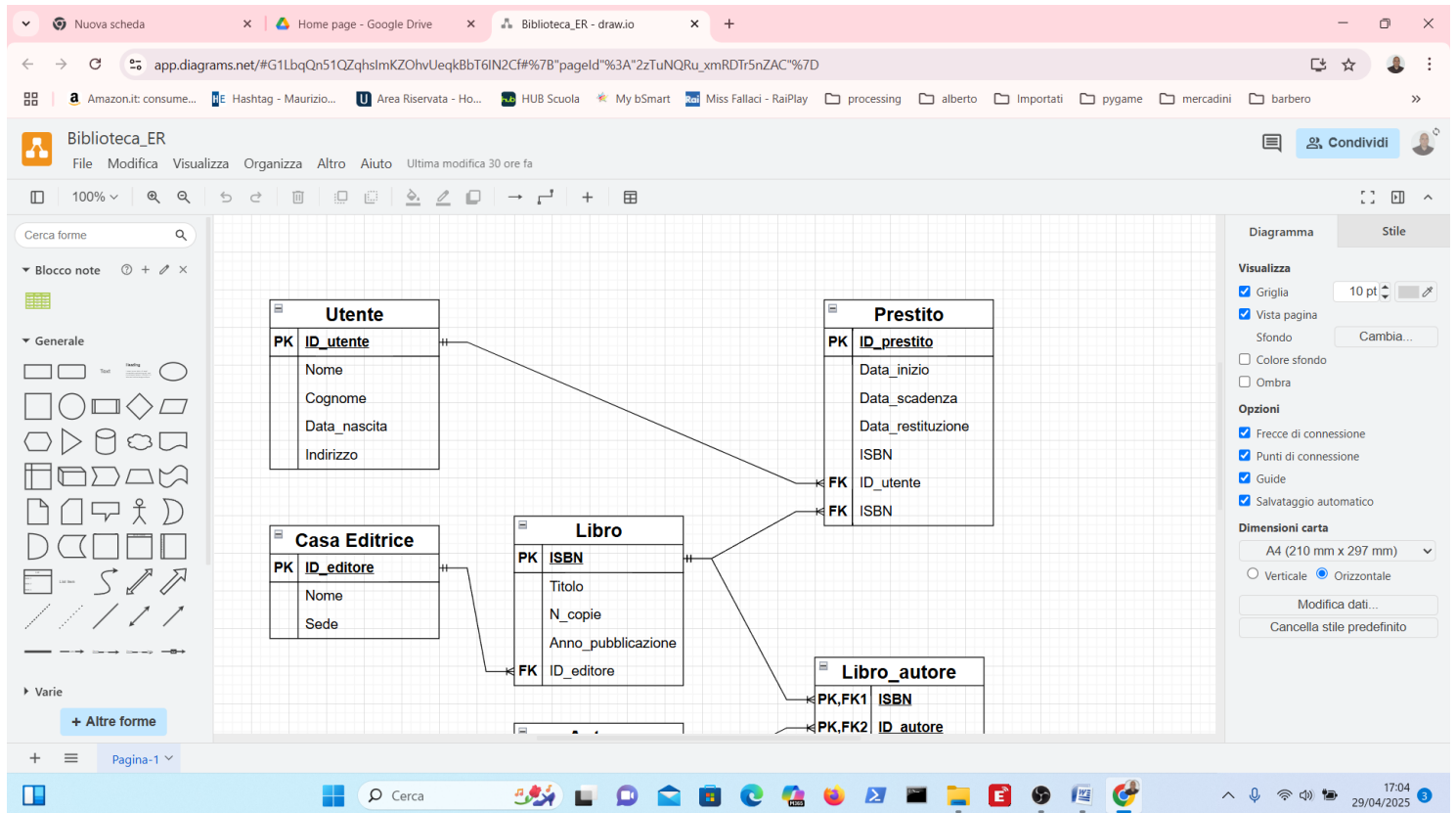


Servizi che un database può mettere a disposizione:

- Prestito
- Ordine
- Prenotazione
- Biglietto
- Fattura
- Scorta di un prodotto
- Acquisto
- Evento
- Post
- Spettacolo
- Proiezione di film
- Vendita
- Assicurazione
- Scambio di beni
- Condivisione di beni
- Ecc..



Draw.io



Modello logico

Lo schema logico di un database è come un **progetto dettagliato** che descrive la struttura del database stesso. Definisce le tabelle, le colonne all'interno di ciascuna tabella, i tipi di dati che ogni colonna può contenere e le relazioni tra le diverse tabelle. In sostanza, è una rappresentazione astratta di come i dati sono organizzati e collegati.

Lo schema logico di un database specifica come i dati sono strutturati per essere archiviati e recuperati in modo efficiente.

Esempio di schema logico per un database di prestito libri di una biblioteca:

Potremmo avere le seguenti tabelle:

1. **Libri**: Contiene le informazioni sui libri disponibili.
2. **Utenti**: Contiene le informazioni sugli utenti registrati.
3. **Prestiti**: Registra i prestiti dei libri agli utenti.

Ecco una possibile definizione delle colonne per ciascuna tabella e le loro relazioni:

Tabella: Libri

- ID_Libro (Chiave Primaria): Identificativo unico per ogni libro (es. INT)
- Titolo: Titolo del libro (es. VARCHAR)
- Autore: Autore del libro (es. VARCHAR)
- ISBN: Codice ISBN del libro (es. VARCHAR)
- AnnoPubblicazione: Anno di pubblicazione del libro (es. INT)
- NumeroCopie: Numero totale di copie disponibili (es. INT)

Tabella: Utenti

- ID_Utente (Chiave Primaria): Identificativo unico per ogni utente (es. INT)
- Nome: Nome dell'utente (es. VARCHAR)
- Cognome: Cognome dell'utente (es. VARCHAR)
- DataNascita: Data di nascita dell'utente (es. DATE)
- Indirizzo: Indirizzo dell'utente (es. VARCHAR)
- NumeroTelefono: Numero di telefono dell'utente (es. VARCHAR)

Tabella: Prestiti

- ID_Prestito (Chiave Primaria): Identificativo unico per ogni prestito (es. INT)
- ID_Libro (Chiave Esterna): Fa riferimento all'ID_Libro nella tabella Libri (es. INT)
- ID_Utente (Chiave Esterna): Fa riferimento all'ID_Utente nella tabella Utenti (es. INT)
- DataInizioPrestito: Data in cui il libro è stato preso in prestito (es. DATE)
- DataScadenzaPrestito: Data prevista per la restituzione del libro (es. DATE)
- DataRestituzioneEffettiva: Data effettiva di restituzione del libro (es. DATE, può essere NULL se il libro non è ancora stato restituito)

Relazioni tra le tabelle:

- **Uno-a-molti (One-to-Many):** Un libro può essere preso in prestito più volte da diversi utenti. Quindi, nella tabella Prestiti ci saranno più record con lo stesso ID_Libro. (La chiave esterna ID_Libro in Prestiti fa riferimento alla chiave primaria ID_Libro in Libri).
- **Uno-a-molti (One-to-Many):** Un utente può prendere in prestito più libri contemporaneamente o in momenti diversi. Quindi, nella tabella Prestiti ci saranno più record con lo stesso ID_Utente. (La chiave esterna ID_Utente in Prestiti fa riferimento alla chiave primaria ID_Utente in Utenti).

Questo è un esempio di schema logico concettuale. In un'implementazione reale, potremmo avere ulteriori dettagli come vincoli di integrità (ad esempio, un libro non può essere preso in prestito se non ci sono copie disponibili) e indici per ottimizzare le query.

Spero che questo esempio ti chiarisca il concetto di schema logico!

Utenti						
ID_utente	Nome	Cognome	Via	N_civico	Città	Provincia
1	Alberto	Rossi	Titano	101	Rimini	RN
2	Luigi	Verdi	Meduda	22	Riccione	RN
3	Maria	Neri	Pini	133	Parma	PR
4	Elena	Bianchi	Luna	55	Bologna	BO

Normalizzazione del database

Le Tre Regole di Normalizzazione Fondamentali

Queste regole, dette anche forme normali (FN), sono un insieme di criteri utilizzati per progettare schemi di database relazionali ben strutturati. L'obiettivo principale è minimizzare la ridondanza dei dati e migliorare l'integrità, rendendo il database più facile da gestire e modificare.

1. Prima Forma Normale (1FN)

Una relazione è in 1FN se ogni attributo (colonna) contiene solo valori atomici, cioè indivisibili. In altre parole, non ci devono essere attributi multi valore o attributi composti ripetuti all'interno di una singola tupla (riga).

2. Seconda Forma Normale (2FN)

Una relazione è in 2FN se è già in 1FN e tutti gli attributi non chiave (cioè gli attributi che non fanno parte della chiave primaria) dipendono funzionalmente dall'intera chiave primaria. Questo significa che se la chiave primaria è composta da più attributi, un attributo non chiave non può dipendere solo da una parte di essa.

3. Terza Forma Normale (3FN)

Una relazione è in 3FN se è già in 2FN e nessun attributo non chiave dipende transitivamente da un altro attributo non chiave. In altre parole, tutti gli attributi non chiave devono dipendere direttamente dalla chiave primaria e non attraverso un altro attributo non chiave.

Esempio di Database Prestito Libro di una Biblioteca Non Normalizzato

Immagina una singola tabella chiamata Prestiti per gestire i prestiti dei libri in una biblioteca. Questa tabella non è normalizzata e presenta diversi problemi:

ID_Prestito	Nome_Utente	Indirizzo_Utente	Titolo_Libro	Autore_Libro	Data_Prestito	Data_Scadenza
1	Alice Rossi	Via Roma 10, Rimini	Il Nome della Rosa	Umberto Eco	2025-04-20	2025-05-04
2	Bob Bianchi	Piazza Cavour 5, Riccione	Cent'anni di solitudine	Gabriel García Márquez	2025-04-22	2025-05-06
3	Alice Rossi	Via Roma 10, Rimini	La Divina Commedia	Dante Alighieri	2025-04-25	2025-05-09
4	Carlo Verdi	Viale Dante 15, Cattolica	Il Signore degli Anelli	J.R.R. Tolkien	2025-04-27	2025-05-11
5	Bob Bianchi	Piazza Cavour 5, Riccione	L'Amica Geniale	Elena Ferrante	2025-04-28	2025-05-12

Problemi di questa tabella non normalizzata:

- **Ridondanza dei dati:** Il nome e l'indirizzo dell'utente si ripetono per ogni libro che lo stesso utente prende in prestito (vedi Alice Rossi e Bob Bianchi). Questo spreca spazio di archiviazione e rende più difficile l'aggiornamento delle informazioni dell'utente. Se Alice cambia indirizzo, dovremmo aggiornare più righe.
- **Anomalie di inserimento:** Se volessimo inserire un nuovo utente nel sistema prima che prenda in prestito un libro, avremmo dei problemi perché non avremmo informazioni sul libro da inserire.
- **Anomalie di cancellazione:** Se l'ultimo libro preso in prestito da un utente viene restituito e la relativa riga viene eliminata, perdiamo anche le informazioni sull'utente.
- **Anomalie di modifica:** Modificare l'indirizzo di un utente richiede di aggiornare tutte le righe in cui compare il suo nome, con il rischio di inconsistenza se non tutte le righe vengono aggiornate correttamente.

Questo è un chiaro esempio di come un database non normalizzato possa portare a inefficienze e problemi di integrità dei dati. Nelle prossime interazioni, se vuoi, possiamo vedere come normalizzare questo schema di database applicando le tre forme normali.

Processo di normalizzazione

Vediamo come possiamo normalizzare il database dei prestiti della biblioteca applicando le tre forme normali.

Passo 1: Portare alla Prima Forma Normale (1FN)

La nostra tabella Prestiti attuale non è in 1FN perché ogni attributo non contiene valori atomici. Scomponiamo nome utente in nome e cognome. Stessa cosa con indirizzo utente in via, numero, città e provincia. Solo per la data viene ammesso un valore composto da anno, mese e giorno

Passo 2: Portare alla Seconda Forma Normale (2FN)

Per essere in 2FN, la tabella deve essere in 1FN e tutti gli attributi non chiave devono dipendere funzionalmente dall'intera chiave primaria. Nella nostra tabella Prestiti, la chiave primaria potrebbe essere ID_Prestito. Tutti gli altri attributi dipendono da ID_Prestito per identificare uno specifico prestito. Tuttavia, notiamo che alcuni attributi dipendono solo da una parte di una potenziale chiave primaria *composita* (anche se non l'abbiamo definita esplicitamente così).

Per risolvere questo, separiamo le informazioni sugli utenti e sui libri in tabelle separate:

- **Tabella Utenti:** Contiene informazioni sugli utenti.
- **Tabella Libri:** Contiene informazioni sui libri.
- **Tabella Prestiti:** Conterrà le informazioni specifiche sul prestito, collegando utenti e libri.

Ecco le nuove tabelle in 2FN:

Tabella Utenti:

ID_Utente	Nome	Cognome	Via	N_civico	Citta	Provincia
1	Alice	Rossi	Roma	10	Rimini	RN
2	Bob	Bianchi	Cavour	5	Riccione	RN
3	Carlo	Verdi	Dante	15	Bologna	BO

Tabella Libri:

ID_Libro	Titolo	Autore
101	Il Nome della Rosa	Umberto Eco
102	Cent'anni di solitudine	Gabriel García Márquez
103	La Divina Commedia	Dante Alighieri
104	Il Signore degli Anelli	J.R.R. Tolkien
105	L'Amica Geniale	Elena Ferrante

Tabella Prestiti:

ID_Prestito	ID_Utente	ID_Libro	Data_Prestito	Data_Scadenza
1	1	101	2025-04-20	2025-05-04
2	2	102	2025-04-22	2025-05-06
3	1	103	2025-04-25	2025-05-09
4	3	104	2025-04-27	2025-05-11
5	2	105	2025-04-28	2025-05-12

Ora, la tabella Prestiti ha come chiave primaria ID_Prestito, e gli attributi non chiave (ID_Utente, ID_Libro, Data_Prestito, Data_Scadenza) dipendono interamente da essa. Le informazioni sugli utenti e sui libri sono state spostate nelle rispettive tabelle, eliminando la ridondanza.

Passo 3: Portare alla Terza Forma Normale (3FN)

Per essere in 3FN, la tabella deve essere in 2FN e nessun attributo non chiave deve dipendere transitivamente da un altro attributo non chiave. Nel nostro schema attuale, non ci sono dipendenze transitive evidenti. Gli attributi non chiave in ciascuna tabella dipendono direttamente dalla chiave primaria della loro rispettiva tabella.

- Nella tabella Utenti, Nome_Utente e Indirizzo_Utente dipendono direttamente da ID_Utente.
- Nella tabella Libri, Titolo_Libro e Autore_Libro dipendono direttamente da ID_Libro.
- Nella tabella Prestiti, ID_Utente, ID_Libro, Data_Prestito e Data_Scadenza dipendono direttamente da ID_Prestito.

Pertanto, il nostro schema di database è ora in **Terza Forma Normale (3FN)**.

Riepilogo delle modifiche:

Abbiamo scomposto la tabella Prestiti originale in tre tabelle separate:

- Una tabella Utenti per memorizzare le informazioni sugli utenti.
- Una tabella Libri per memorizzare le informazioni sui libri.
- Una tabella Prestiti per registrare quali utenti hanno preso in prestito quali libri e quando.

Queste tabelle sono collegate tramite le chiavi esterne (ID_Utente che fa riferimento a Utenti.ID_Utente e ID_Libro che fa riferimento a Libri.ID_Libro nella tabella Prestiti).

Questo design normalizzato elimina la ridondanza dei dati, previene le anomalie di inserimento, cancellazione e modifica, e rende il database più efficiente e mantenibile.

Download XAMPP

apachefriends.org/it/download.html

Verifica la tua identità

Apache Friends

Scarica

Hosting

Comunità

Chi siamo

Cerca..

Cerca

IT

Scarica

XAMPP è una distribuzione di Apache semplice da installare che contiene MySQL, PHP e Perl. Scarica ed avvia il file di installazione. È davvero Installers created using InstallBuilder.

XAMPP per Windows 8.0.30, 8.1.25 & 8.2.12

Versione	Cosa è incluso?	Codice di controllo	Dimensione
8.0.30 / PHP 8.0.30	Cosa è incluso?	md5 sha1	Scarica (64 bit) 144 Mb
8.1.25 / PHP 8.1.25	Cosa è incluso?	md5 sha1	Scarica (64 bit) 148 Mb
8.2.12 / PHP 8.2.12	Cosa è incluso?	md5 sha1	Scarica (64 bit) 149 Mb

Documentazione/Domande frequenti

Non esiste un vero e proprio manuale o una guida per XAMPP. Abbiamo scritto la documentazione sotto forma di FAQ. Avete qualche domanda importante a cui non avete trovato risposta qui? Provate i Forum o Stack Overflow.

- Linux Domande frequenti
- Windows Domande frequenti
- OS X Domande frequenti



XAMPP Control Panel v3.3.0 [Compiled: Apr 6th 2021]

Modules

Service	Module	PID(s)	Port(s)	Actions
<input type="checkbox"/>	Apache			Start Admin Config Logs
<input type="checkbox"/>	MySQL			Start Admin Config Logs
<input type="checkbox"/>	FileZilla			Start Admin Config Logs
<input type="checkbox"/>	Mercury			Start Admin Config Logs
<input type="checkbox"/>	Tomcat			Start Admin Config Logs

Config

Netstat

Shell

Explorer

Services

Help

Quit

16:35:45 [main] XAMPP Installation Directory: "d:\xampp\"

16:35:45 [main] Checking for prerequisites

16:35:46 [main] All prerequisites found


16:35:46 [main] Initializing Modules

16:35:46 [main] The Mercury module is disabled

16:35:46 [main] The Tomcat module is disabled

16:35:46 [main] Starting Check-Timer

16:35:46 [main] Control Panel Ready

**XAMPP Control Panel v3.3.0**

Modules

Service	Module	PID(s)	Port(s)	Actions
<input type="checkbox"/>	Apache	5036 14892	80, 443	Stop Admin Config Logs
<input type="checkbox"/>	MySQL	14888	3306	Stop Admin Config Logs
<input type="checkbox"/>	FileZilla			Start Admin Config Logs
<input type="checkbox"/>	Mercury			Start Admin Config Logs
<input type="checkbox"/>	Tomcat			Start Admin Config Logs

Config

Netstat

Shell

Explorer

Services

Help

Quit

16:35:46 [main] The Mercury module is disabled

16:35:46 [main] The Tomcat module is disabled

16:35:46 [main] Starting Check-Timer

16:35:46 [main] Control Panel Ready

16:36:32 [Apache] Attempting to start Apache app...

16:36:33 [Apache] Status change detected: running

16:36:36 [mysql] Attempting to start MySQL app...

16:36:37 [mysql] Status change detected: running

localhost / 127.0.0.1 / biblio / utenti

localhost/phpmyadmin/index.php?route=/sql&pos=0&db=biblio&table=utenti

Server: 127.0.0.1 > Database: biblio > Tabella: utenti

Mostra Struttura SQL Cerca Inserisci Esporta Importa Privilegi Operazioni Monitoraggio Trigger

Mostrare le righe 0 - 0 (1 del totale, La query ha impiegato 0,0002 secondi.)

SELECT * FROM `utenti`

Profiling [Modifica inline] [Modifica] [Spiega SQL] [Crea il codice PHP] [Aggiorna]

Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella

Opzioni extra

ID_utente Nome Cognome Via N_civico Citta Provincia

1 Alice Rossi Roma 10 Rimini RN

Seleziona tutto Se selezionati: Modifica Copia Elimina Esporta

Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella

Operazioni sui risultati della query

Stampa Copia negli appunti Esporta Mostra diagramma Crea vista

Aggiungi ai preferiti questa query SQL

Etichetta: ☐ Permetti ad ogni utente di accedere a questo bookmark

Aggiungi ai preferiti questa query SQL

Console