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**

Premessa

Entità

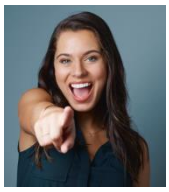
In informatica una entità , può essere una cosa reale che esiste e che si può toccare ma può essere anche una cosa astratta esempio:
una materia scolastica, un corso universitario, una poesia, un commento ecc...

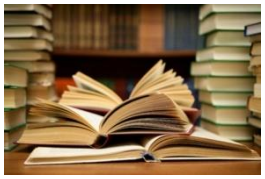
Esempi in informatica:

- 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..



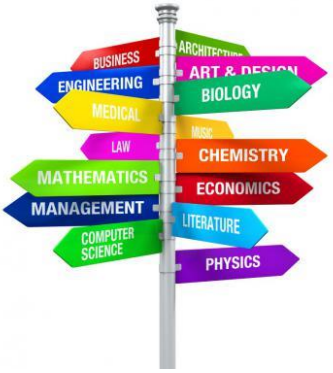






Liceo Scientifico

	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



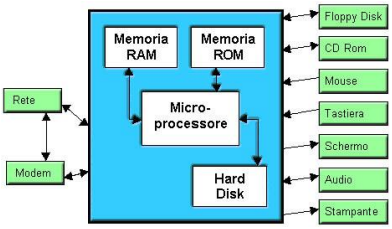
I modelli

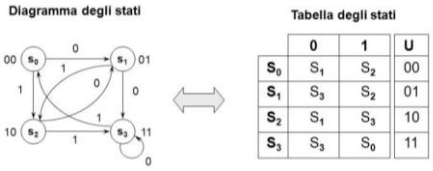
Un modello è una rappresentazione semplificata di una realtà che si vuole rappresentare :
Esempi di modelli possono essere:

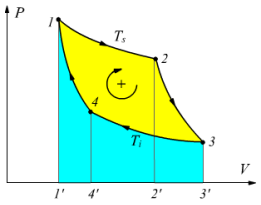
- Un diagramma, una tabella, un grafico, un grafo, una piantina di un edificio
- Un insieme di equazioni matematiche
- Un programma (software) con simulazione al computer
- Un disegno al cad, una simulazione al cad
- Un database, un modello OOP (classi ed oggetti)

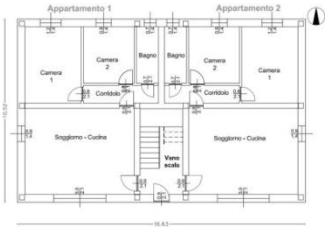
L'obiettivo di un modello è di aiutarci a capire come funziona un sistema reale, a progettarlo, o a prevederne il comportamento

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

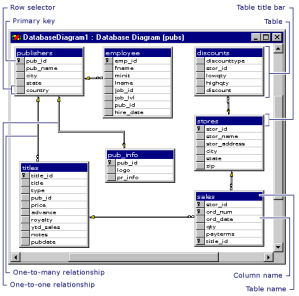


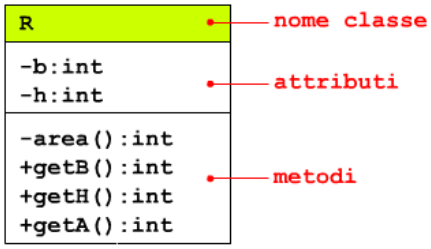






$$\begin{cases} y = 2x + 1 \\ 3x + y = 0 \end{cases}$$





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:

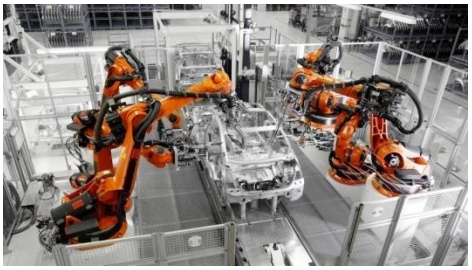
ID_Ordine	ID_Cliente	Data_Ordine	Totale
101	1	2024-04-20	55.00
102	2	2024-04-20	22.50

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:

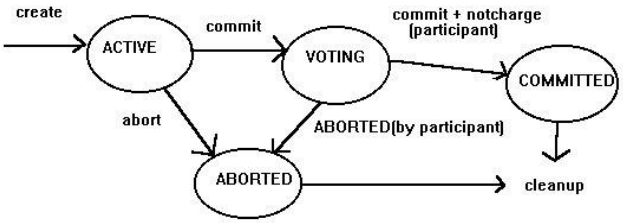
- **Gestione dei Registri Civili:** Archiviare informazioni su nascite, matrimoni, decessi.
- **Sistemi di Identificazione:** Gestire database di cittadini e documenti di identità.
- **Gestione delle Imposte:** Tracciare i pagamenti e le dichiarazioni fiscali.
- **Sistemi di Votazione:** Registrare gli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

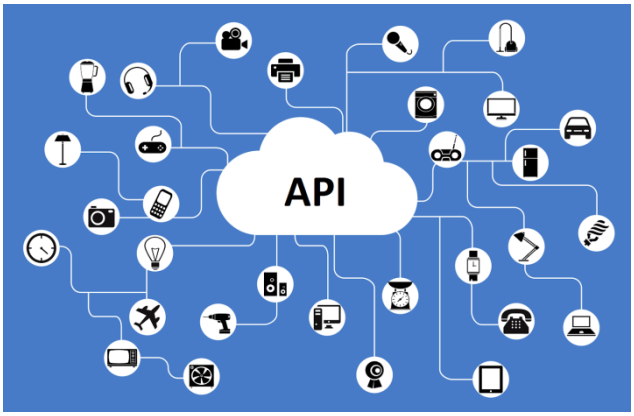
- **Atomicità (Atomicity):** Una transazione viene trattata come una singola unità indivisibile di lavoro. O tutte le modifiche all'interno della transazione vengono completate con successo (commit), oppure nessuna di esse viene applicata al database (rollback). Non ci possono essere stati intermedi.
    - **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

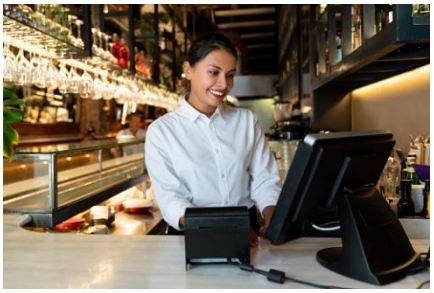


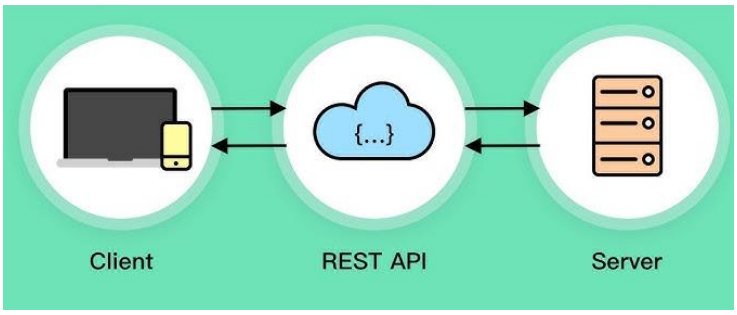
Viviamo in un mondo dove tutti sono interconnessi tra di loro, dove in pochi secondi riusciamo a mandare un messaggio ad una persona distante anche migliaia di kilometri e dove, comodamente dal letto con il nostro smartphone, potremmo anche programmare l'attivazione della macchina del caffè; ma **come 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