Basi di dati

Un **database** è un insieme organizzato di dati che serve a supportare lo svolgimento di attività di un ente, un'azienda, un ufficio o una persona. I dati possono riguardare vari aspetti, come le informazioni personali dei clienti, I prodotti venduti, ecc.....

Per gestire i dati in modo efficiente ed efficace, si usano dei sistemi **informativi** ed **informatici**.

Informatica

L'informatica è la scienza che si occupa del trattamento razionale dell'informazione, soprattutto mediante l'uso di macchine automatiche, come i computer; ha due aspetti principali:

- **Metodologico**: riguarda i metodi per risolvere problemi e gestire le informazioni
- **Tecnologico**: riguarda i dispositivi elettronici e i sistemi che li utilizzano.

Come detto prima, abbiamo a disposizione due diversi tipi di sistemi:

- **Sistema informativo**: componente di un'organizzazione che gestisce le informazioni di interesse per l'organizzazione stessa, cioè quelle che servono per il raggiungimento dei suoi obiettivi.
    - Le funzioni principali sono **acquisire** e **memorizzare** i dati, **aggiornarli** in base alle variazioni, **interrogarli** per ottenere informazioni utili e **elaborarli** per produrre nuove conoscenze
- **Sistema informatico**: parte automatizzata di un sistema informativo, che usa la tecnologia informatica per gestire le informazioni. Garantisce che i **dati siano conservati** in modo permanente sui dispositivi di memorizzazione, permette un **rapido aggiornamento** dei dati per riflettere le loro variazioni, rende i dati **accessibili alle interrogazioni** degli utenti, e può essere **distribuito sul territorio**

Gestione delle informazioni

Nel corso del tempo, i sistemi informativi complessi hanno visto l'introduzione di **metodi sempre più sofisticati** per organizzare e codificare le informazioni.

Nei sistemi informatici, le informazioni vengono rappresentate in modo essenziale attraverso i **dati**: questo tipo ha bisogno di essere interpretato. Viene usato questo metodo perché la rappresentazione precisa di forme più ricche di informazione e conoscenza è **difficile**, e i dati

costituiscono spesso una risorsa strategica, perché sono **più stabili** nel tempo rispetto ad altri componenti. Inoltre, i dati rimangono gli stessi nella **migrazione** da un sistema all'altro.

Per fare questo, usiamo due diversi prodotti.

Database

I database sono **collezioni di dati** usati per rappresentare le informazioni di interesse di un sistema informativo. Ogni dato è rappresentato logicamente **una sola volta**, e può essere utilizzato da un insieme di applicazioni da diversi utenti secondo opportuni criteri di riservatezza. I dati all'interno di un database sono *molti*, *hanno un formato definito*, sono *permanenti*, sono *raggruppati per insiemi omogenei di dati*, ed *esistono relazioni specifiche tra gli insiemi di dati*.

Inoltre:

- La ridondanza è **minima e controllata**; è assicurata la consistenza delle informazioni
- I dati sono disponibili per utenze diverse e concorrenti
- I dati sono controllati: protetti da malfunzionamenti hardware e software
- Indipendenza dei dati dal programma

Le basi di dati sono:

- **Grandi**: le dimensioni sono molto maggiori della memoria centrale dei sistemi di calcolo utilizzati. Il limite deve essere solo quello *fisico* dei dispositivi
- **Persistenti**: hanno un tempo di vita indipendente dalle singole esecuzioni dei programmi che le utilizzano
- **Condivise**: ogni organizzazione (specie se grande) è divisa in *settori*, o svolge diverse attività. Ciascuno di essi ha un sottosistema informativo
    - È una risorsa integrata, condivisa fra applicazioni.
    - Hanno bisogno di **meccanismi di autorizzazione**, poiché ci possono essere attività diverse su dati condivisi
    - Ci deve essere un **controllo della concorrenza**, perché ci sono accessi di più utenti ai dati condivisi

Schemi e istanze

Ogni database è composto da due parti:

- **Schema**: la struttura sostanzialmente in variante nel tempo che descrive la struttura e il significato dei dati, come le *intestazioni delle tabelle*
    - **Schema logico**: descrive la base di dati nel modello logico, come la struttura delle tabelle

    - **Schema interno/fisico**: rappresenta lo schema logico attraverso strutture fisiche di memorizzazione
    - **Schema esterno**: descrive parti della base di dati in un modello logico, come le *viste parziali o derivate*
- **Istanza**: i valori attuali dei dati che possono cambiare rapidamente, rappresentati dal *corpo delle tabelle*

Indipendenza dei dati

Il livello logico è indipendente da quello fisico, permettendo alle tabelle di essere utilizzate nello stesso modo **indipendentemente dalla loro realizzazione fisica**, che può cambiare nel tempo.

L'indipendenza dei dati è una conseguenza dell'articolazione in livelli di un DBMS, che permette l'accesso ai dati solo tramite il livello esterno, il quale può coincidere con il livello logico. Esistono quindi due forme:

- **Indipendenza fisica**: il livello logico e quello esterno sono indipendenti dal livello fisico; ciò significa che una relazione può essere utilizzata nello stesso modo indipendentemente dalla sua realizzazione fisica, che può cambiare senza necessità di modificare i programmi
- **Indipendenza logica**: il livello esterno è indipendente dal livello logico. Aggiunte o modifiche alle viste non richiedono modifiche al livello logico, e modifiche allo schema logico invece che lasciano inalterato lo schema esterno sono trasparenti agli utenti

Interfacce e linguaggi

Per le basi di dati abbiamo a disposizione due tipi di linguaggi:

- **DDL** (Data Definition Languages): utilizzati per la definizione di schemi logici, fisici e autorizzazioni di accesso
  DML (Data Manipulation Languages): permettono l'interrogazione e l'aggiornamento delle basi di dati. Alcuni linguaggi come SQL hanno funzioni di entrambi le categorie.

La disponibilità di vari linguaggi e interfacce contribuisce all'efficacia dei DBMS: *linguaggi testuali interattivi come SQL*, *comandi SQL immersi in linguaggi ospiti*, *comandi SQL immersi in linguaggi ad hoc con funzionalità aggiuntive*, *interfacce utente amichevoli che non richiedono l'uso di linguaggio testuale*.

Dobbiamo avere inoltre qualche caratteristica fondamentale per il database:

- **Astrazione dei dati**: si usa un modello di dati per nascondere dettagli e presentare all'utente una vista concettuale del database
- **Supporto di viste multiple dei dati**: ogni utente può usare una vista differente del database, contenente solo i dati di interesse per quell'utente.

Database Management System

Un DBMS è un **insieme di programmi** che permettono di creare, usare e **gestire** una base di dati. È quindi un software general purpose che facilita il processo di definizione, costruzione e manipolazione del database per varie applicazioni.

È quindi un sistema che gestisce collezioni di dati grandi, persistenti e condivise, garantendo privatezza, affidabilità, efficienza ed efficacia.

I DBMS garantiscono:

- **Privatezza**, poiché si possono definire meccanismi di autorizzazione
- **Affidabilità**, poiché è resistente a malfunzionamenti hardware e software. Come tecnica fondamentale per questo abbiamo la *gestione delle transazioni*
- **Efficacia**: cercano di utilizzare al meglio le risorse di spazio di memoria (principale e secondaria) e tempo (esecuzione e risposta). Rischiano l'inefficienza, ed è inoltre anche il risultato della qualità delle applicazioni
    - Cercano di rendere produttive le attività dei loro utilizzatori, offrendo funzionalità *articolate*, *potenti* e *flessibili*

Creazione di un database

La creazione di un database avviene in tre step:

- **Definizione**: in questa fase si specifica la *struttura logica del database*. Si definiscono gli schemi per le tabelle, le relazioni, gli indici e le viste. Questo include anche la definizione di vincoli di integrità e regole di sicurezza
- **Creazione/popolazione**: dopo aver definito lo schema del database, occorre *creare fisicamente le tabelle e le altre strutture***;** successivamente, bisognerà inserire i dati all'interno di esso
- **Manipolazione**: questa fase permette agli utenti di *interagire* col database attraversamento operazioni come l'*inserimento*, l'*aggiornamento*, la *cancellazione* e la *ricerca di dati*. Questi sono eseguite tramite linguaggi di interrogazione quali SQL.

Interrogazione di un database

L' interrogazione di un database è un aspetto fondamentale per l'accesso e l'analisi dei dati. Utilizzando un linguaggio di interrogazione, come **SQL**, gli utenti possono formulare domande specifiche al database per recuperare informazioni pertinenti.

Guardiamo un esempio:

```
SELECT [Nome], [Cognome], [Indirizzo], [Città] FROM Studenti
WHERE [Cognome] = “Rossi”;
```

Questa istruzione seleziona i *nomi, cognomi, indirizzi e città degli studenti con cognome "Rossi" dalla tabella "Studenti"*.

L'efficacia di queste interrogazioni dipende non solo dalla conoscenza del contenuto del database, ma anche dall'esperienza con il linguaggio di interrogazione stesso. Inoltre, un'interfaccia di interrogazione semplice ed efficace può facilitare gli utenti nell'eseguire query complesse, rendendo l'interazione con il database più intuitiva e meno soggetta ad errori.

Transazioni

Una transazione in un DBMS è un gruppo di operazioni che vengono trattate come un'unità **singola** e **indivisibile**, nota come *atomicità*. Questo significa che *tutte le operazioni all'interno della transazione devono essere completate con successo*; in caso contrario, la transazione viene annullata, e il database ritorna allo stato precedente, come se la transazione non fosse mai stata eseguita.

Sono anche progettate per essere **corrette in ambienti concorrenti**, dove molteplici transazioni possono avvenire simultaneamente. Questo è gestito in modo che i risultati siano coerenti, come se le transazioni fossero state eseguite una dopo l'altra, garantendo l'integrità dei dati.

Una volta che una transazione è stata completata con successo, i suoi effetti sono **permanenti**, anche in caso di guasti del sistema: ciò è essenziale per la durabilità e affidabilità del database.

Sono programmi che realizzano attività frequenti e predefinite, con poche eccezioni previste a priori. Sono solitamente realizzate con programmi in linguaggio ospite, ed è anche una **sequenza indivisibile di operazioni che vengono eseguite tutte o nessuna**.

Descrizioni dei dati

I DBMS utilizzano **descrizioni e rappresentazioni dei dati a vari livelli** per garantire l'indipendenza dei dati dalla loro rappresentazione fisica. Questo significa che i programmi interagiscono con i dati a un livello astratto, permettendo *modifiche alle rappresentazioni sottostanti senza influenzare i programmi stessi*. Questa indipendenza è resa possibile attraverso il **modello dei dati**.

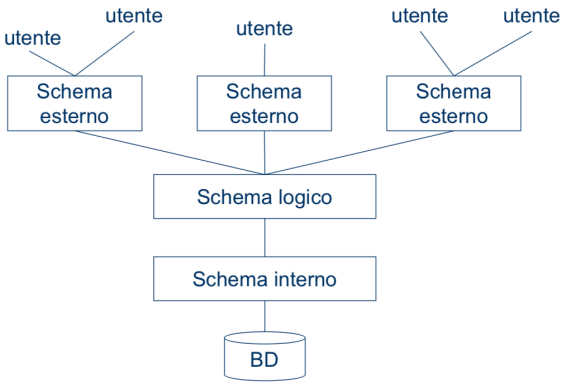
Modello dei dati

Il modello dei dati è un insieme di costrutti utilizzati per organizzare i dati di interesse e descriverne la dinamica.

Un componente chiave è il **meccanismo di strutturazione**, simile ai costruttori di tipo nei linguaggi di programmazione, che definisce nuovi tipi di dati. Per esempio, il **modello relazionale** utilizza il costruttore 'relazione' per definire insiemi di record omogenei.

Abbiamo però due tipi principali:

- **Modelli logici**: utilizzati nei DBMS esistenti per l'organizzazione dei dati, sono indipendenti dalle strutture fisiche e includono tipi come *relazionale*, *reticolare*, *gerarchico*, *a oggetti*, *XML*, etc.
- **Modelli concettuali**: rappresentano i dati in modo indipendente da qualsiasi sistema, descrivendo i concetti del mondo reale e sono utilizzati nelle fasi preliminari di progettazione.
    - Il modello più diffuso è il modello **Entity-Relationship**



I ruoli

- **Progettisti e realizzatori di DBMS**: sviluppano e mantengono il sistema di gestione del database
- **DBA (Database Administrator)**: responsabile del controllo entralizzato e della gestione del sistema, delle prestazioni, affidabilità e autorizzazioni
- **Progettisti e programmatori di applicazioni**: creano applicazioni che utilizzano il database
- **Utenti finali e casuali**: gli utenti finali eseguono applicazioni predefinite, mentre gli utenti casuali eseguono operazioni non previste a priori, usando linguaggi interattivi.

Vantaggi e svantaggi

I vantaggi sono:

- I dati sono considerati una **risorsa comune** dell'organizzazione, disponibile per molteplici applicazioni e utenti
- Modello unificato e preciso della parte di mondo di interesse, utilizzabile in applicazioni attuali e future
- Controllo centralizzato dei dati che riduce ridondanze e inconsistenze
- Indipendenza dei dati che favorisce lo sviluppo di applicazioni flessibili e facilmente modificabili

Gli svantaggi invece riguardano:

- Costosi e complessi, con specifici requisiti di software e hardware
- Difficoltà nel separare i servizi effettivamente utilizzati da quelli inutili

- Inadatti per la gestione di applicazioni con pochi utenti, a meno che i costi di creazione e manutenzione non siano giustificati.

Progettazione basi di dati

La progettazione di basi di dati rappresenta un'attività cruciale all'interno del processo di sviluppo dei sistemi informativi. Queste attività sono svolte da analisti, progettisti e utenti, e sono essenziali per lo sviluppo e l'uso efficace dei sistemi informativi. Il processo è **iterativo**, formando un ciclo che si ripete nel tempo.

Fasi del ciclo di vita

Le fasi del ciclo di vita dei sistemi informativi includono:

- **Studio di fattibilità**: si definiscono i costi e le priorità
- **Raccolta e analisi dei requisiti**: si studiano le proprietà del sistema
- **Progettazione**: si occupa dell'organizzazione e della struttura dei dati e delle funzioni
- **Implementazione**: realizzazione pratica del sistema
- **Validazione e collaudo**: consistono nella sperimentazione e verifica del sistema
- **Funzionamento**: quando il sistema diventa operativo

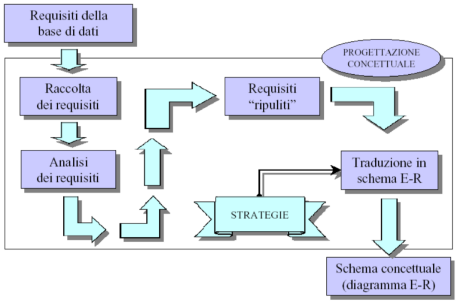
Il **modello a spirale** è un approccio che enfatizza l'importanza dell'analisi, e della raccolta dei requisiti come base per lo sviluppo di versioni successive del programma. Questo modello prevede una continua alternanza tra progettazione, realizzazione, verifica e manutenzione.

Per garantire prodotti di buona qualità, è consigliabile seguire una metodologia di progetto ben articolata in fasi e passi. Questa metodologia dovrebbe permettere di suddividere la progettazione in fasi successive indipendenti, fornire strategie e criteri di scelta in caso di alternative, e offrire modelli di riferimento per descrivere la realtà che si sta progettando. Dovrebbe garantire inoltre generalità rispetto ai problemi da affrontare, qualità in termini di correttezza, completezza ed efficienza, e facilità d0uso.

Una metodologia efficace di progettazione di basi di dati si basa sulla **separazione netta tra le decisioni relative** a cosa rappresentare (i requisiti della base di dati e lo schema concettuale), e come farlo (progettazione fisica e logica. Prevede tre fasi principali:

Progettazione concettuale

È il primo passo nel processo di sviluppo di un sistema informativo. In questa fase, i **requisiti** del sistema vengono **tradotti in una descrizione formale** e integrata delle esigenze aziendali. Questa descrizione è espressa in modo indipendente dalle scelte implementati e come il DBMS, il software e l'hardware. L'obiettivo è creare una rappresentazione che sia:



- **Formale**: chiara e non ambigua, per descrivere in modo completo il sistema analizzato
- **Integrata**: capace di catturare l'intero ambiente analizzato in una visione unificata
- **Indipendente dall'ambiente tecnologico**: concentrata sui dati e sulle loro relazioni, piuttosto che sulle scelte implementative

Abbiamo vari vantaggi:

- Fornisce una descrizione dei dati che è indipendentemente dagli aspetti tecnologici, facilitando la comprensione e la comunicazione tra progettisti e utenti
- Utilizza una rappresentazione **prevalentemente grafica**, migliorando la comunicazione e la documentazione del progetto

Analisi dei requisiti

L'analisi dei requisiti e la progettazione concettuale sono processi fondamentali nella creazione di basi di dati. Queste fasi comprendono diverse attività interconnesse:

- **Acquisizione dei requisiti**: questo passaggio iniziale coinvolge la raccolta di informazioni dagli utenti tramite *interviste* e l'*esame della documentazione esistente*, come normative, regolamenti interni e modulistica
    - È essenziale effettuare **verifiche di comprensione e coerenza**, usare esempi concreti e richiedere definizioni precise. È importante inoltre distinguere gli aspetti essenziali da quelli marginali.
    - Per documentare efficacemente i requisiti, è necessario:
        - Scegliere il giusto livello di **astrazione**
        - Standardizzare la struttura delle frasi
        - Separate le informazioni sui dati da quelle sulle funzioni
- **Analisi dei requisiti**: una volta raccolti i primi requisiti, l'analisi inizia e *può portare alla scoperta di ulteriori necessità*. Questa fase è cruciale per comprendere a fondo le esigenze degli utenti, e per indirizzare le successive acquisizioni
- **Costruzione dello schema concettuale**: lo schema concettuale è il risultato dell'analisi dei requisiti, e serve a organizzare e strutturare le informazioni in modo logico e coerente

- **Costruzione del glossario**: un glossario ben definito aiuta a chiarire i termini utilizzati, identificando *omonimi e sinonimi*, e *stabilendo un linguaggio comune tra gli stakeholder*. Dobbiamo inoltre rendere espliciti i riferimenti tra termini (*es. Se si parla di posti dove hanno lavorato, è importante specificare se l'indirizzo e il numero di telefono si riferiscono ai partecipanti o ai luoghi di lavoro*

Scrittura dei requisiti

È fondamentale **identificare e unificare i termini** per evitare ambiguità. Ad esempio, si dovrebbe preferire un termine unico per i sinonimi, e usare termini distinti per gli omonimi (es. "Posto" può riferirsi sia a un impiego che a una città).

La creazione di un glossario è un passo opzionale, ma utile, che facilita la comprensione e la precisione dei termini. Ogni termine dovrebbe avere una **breve descrizione**, i sinonimi, e un riferimento ai termini correlati. Ovviamente, occorre utilizzare la stessa terminologia utilizzata per le specifiche dei dati nelle **operazioni**. Qua sotto un breve esempio di come un glossario potrebbe presentarsi:

Termine	**Descrizione**	**Sinonimi**	**Collegamenti**
Partecipante	Persona che partecipa ai corsi	Studente	Corso, società
Docente	Docente dei corsi, può essere esterno	Insegnante	Corso

Dalle specifiche al modello ER

Si utilizzano le definizioni dei costrutti del modello ER:

- Se un concetto ha proprietà significative, e descrive classi di oggetti con esistenza autonoma conviene rappresentarlo con una **entità**
- Se un concetto ha una struttura semplice, e non possiede proprietà rilevanti associate conviene rappresentarlo come un **attributo** di un altro concetto a cui si riferisce
- Se sono state individuate due o più entità, e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato con una **relazione**
- Se uno o più concetti risultano essere il caso particolare di un altro è opportuno rappresentarli facendo uso della **generalizzazione**

Design pattern

I design pattern offrono soluzioni progettuali standard a problemi comuni nell'ingegneria del software, e sono applicabili anche nella progettazione concettuale di basi di dati.

Reificazione

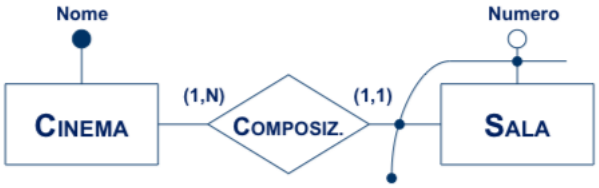
Questo processo trasforma **un concetto astratto in un modello di dati concreto**. Ad esempio, la *reificazione di attributo di entità* potrebbe coinvolgere la trasformazione di un

attributo in un'entità separata per enfatizzare la sua importanza, o per gestire meglio le sue relazioni. Abbiamo vari tipi:

- **Reificazione di relazione binaria**: si verifica quando una relazione diretta tra due entità viene trasformata in un'entità separata.
- **Reificazione di relazione ricorsiva**: avviene quando una relazione che ha un'entità ha con se stessa viene trasformata in un'entità separata
- **Reificazione di attributo di relazione**: quando un attributo di una relazione è sufficientemente complesso o importante da necessitare di una propria identità, si procede con questo. Comporta la trasformazione dell'attributo in un'**entità separata**, permettendo una gestione più dettagliata, e una migliore rappresentazione delle informazioni.

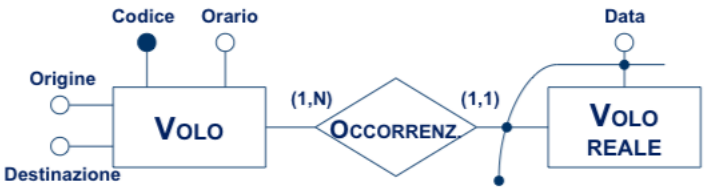
Part-of

Questa relazione indica che un'entità è parte di un'altra, spesso rappresentata come una relazione di tipo **1,N**.



Instance-of

Questo pattern si verifica quando si crea un'entità astratta che diventa concreta in un'entità istanza.



Progettazione logica

Segue la concettuale, e si occupa della traduzione dello schema concettuale nel modello di dati specifico del DBMS scelto. Il risultato è uno schema logico, definito nel linguaggio DDL del DBMS. Durante questa fase, si prendono in considerazione anche i vincoli e l'efficienza del sistema. La progettazione logica si articola in due sotto-fasi:

- **Ristrutturazione dello schema concettuale**: ottimizzazione dello schema per il modello di dati del DBMS
- **Traduzione verso il modello logico**: definizione dello schema in termini del modello di dati

Il cuore della progettazione logica è la **traduzione** dello schema concettuale in uno schema logico; questo non solo deve rappresentare gli stessi dati in maniera corretta, ma deve anche essere ottimizzato per garantire l'efficienza delle interrogazioni. È essenziale quindi semplificare lo schema ER originale, eliminando o modificando quegli elementi che non si adattano facilmente al modello relazionale, come gli attributi multivalore e le generalizzazioni.

Modello ER	Modello relazionale
Entità	Relazione (tabella)
Relazione	Riferimento (chiave esterna)
Generalizzazione	*Non presente*
Attributo semplice	Attributo (campo)
Attributo multivalore	*Non presente*

Dati di ingresso e uscita

I dati di **ingresso** comprendono lo *schema concettuale*, le *informazioni sul carico applicativo* e il *modello logico*.

Come dati di **uscita** otteniamo quindi uno *schema logico* e la relativa *documentazione associata*.

Ristrutturazione dello schema

La ristrutturazione implica l'**eliminazione** di tutti i costrutti **non direttamente rappresentabili** nel modello logico target; questo include la rimozione degli *attributi multivalore* e le *generalizzazioni*. Poi, si procede con il partizionamento o l'accorpamento di entità e associazioni, la scelta degli identificatori primari, e l'analisi delle ridondanze.

Per ottimizzare le prestazioni, è necessario analizzarle, ma non sono valutabili con precisione su uno schema concettuale, in quanto dipendono dalle caratteristiche del DBMS, dal volume dei dati, e dalle caratteristiche delle operazioni. Il carico applicativo fornisce indicatori dei parametri che regolano le prestazioni, come il *tempo di esecuzione delle operazioni* e lo *spazio di memoria* necessario per memorizzare i dati.

Analisi delle ridondanze

Nel contesto della progettazione di basi di dati, l'analisi delle ridondanze in uno schema ER è fondamentale. Una ridondanza si verifica quando **un'informazione significativa è derivabile da altre informazioni** presenti. Si decide quindi se mantenere o eliminare le ridondanze, bilanciando i vantaggi (*semplificazione delle interrogazioni*) con gli svantaggi (*appesantimento degli aggiornamenti, aumento di occupazione di spazio*).

Le ridondanze possono manifestarsi sotto forma di attributi deriva bili da altri attributi della stessa entità o relazione, o da attributi di altre entità o relazioni. Possono anche presentarsi come *associazioni derivabili* dalla composizione di altre relazioni, specialmente in presenta di cicli. È importante valutare attentamente la semantica delle relazioni coinvolte prima di decidere di eliminare tali ridondanze.

Il modello relazionale non rappresenta direttamente le generalizzazioni, quindi è necessario trasformare le gerarchie in entità e relazioni, con tre approcci principali:

- **Accorpamento delle entità figlie nel genitore**: comporta l'eliminazione delle entità figlie e l'aggiunta dei loro attributi all'entità genitore. Si introduce un attributo discriminante per distinguere le istanze. Gli attributi derivanti da una figlia possono essere nulli, e le relazioni che provengono da una sola figlia avranno cardinalità minima pari a zero.
    - *Vantaggi*: accesso contestuale agli attributi del genitore e della figlia
    - *Svantaggi*: spreco di memoria per i valori nulli
- **Accorpamento del genitore nelle entità figlie**: l'entità genitore viene eliminata e i suoi attributi, associazioni e identificatori vengono aggiunti alle entità figlie. Ogni associazione definita sul genitore genera una associazione distinta per ogni figlia. Questo approccio è vantaggioso se **le operazioni sono prevalentemente su istanze di una classe figlia** e non introduce valori nulli. È applicabile solo se la generalizzazione è totale.
- **Sostituzione della generalizzazione con relazioni**: si introduce una relazione uno-a-uno tra l'entità genitore e ciascuna entità figlia, con il vincolo che *ogni istanza del genitore può partecipare solo a una relazione di legame con le figlie*. Questo approccio è utile quando la generalizzazione non è totale, e si desidera mantenere separate le operazioni tra entità genitore e figlie senza introdurre valori nulli.

Schema di operazione e tavola degli accessi

Lo **schema di operazione** descrive i dati coinvolti in un'operazione, e corrisponde al frammento dello schema ER interessato. Utilizzando lo schema di operazione, possiamo fare una *stima del costo di un'operazione* contando il numero di accessi alle istanze di entità e relazioni. Il risultato, quindi, può essere riassunto in una **tavola degli accessi**, che distingue gli accessi in **scrittura (S)** e in **lettura (L)**, notando che le operazioni di scrittura sono generalmente più onerose.

Tabella dei volumi

La tabella dei volumi riflette il numero di istanze di entità e relazioni previste, influenzato dalle caridnalità nello schema e dal numero medio di volte che le istanze delle entità partecipano alle relazioni. La valutazione delle prestazioni sarà necessariamente approssimata, poiché dipende anche da parametri fisici difficilmente prevedibili in questa fase.

Progettazione fisica

Completa lo schema logico con le specifiche tecniche dell'hardware e del software scelti. Il risultato è lo **schema fisico**, che descrive le strutture di memorizzazione e accesso ai dati.

Modello Entità-Relazione

Il modello È-R è un linguaggio grafico semi-formale utilizzato per la rappresentazione di schemi concettuali. Si è affermato come standard nelle metodologie di progetto e nei software di supporto alla progettazione. Il modello include **diversi costrutti** come entità, relazioni, attributi semplici e composti, cardinalità e identificatori.

Entità

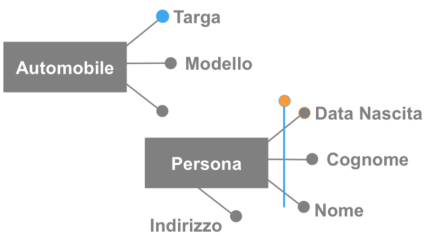
Un'entità può essere definita come una classe di oggetti che hanno proprietà comuni e che esistono autonomamente all'interno dell'applicazione di interesse. Questi oggetti possono essere *fatti*, *persone* o *cose*, e l'obiettivo è registrare informazioni specifiche su di essi. Ogni entità è identificata univocamente all'interno dello schema da un **nome**, che dovrebbe essere espressivo e seguire convenzioni appropriate, come l'uso del singolare.

Un'entità è composta da un **insieme di oggetti** noti come **istanze**. Queste istanze non sono semplici valori identificativi, ma rappresentano gli oggetti stessi. La rappresentazione delle entità nello schema concettuale è *astratta*; non si rappresentano le singole istanze, ma piuttosto la classe di oggetti nel suo complesso.

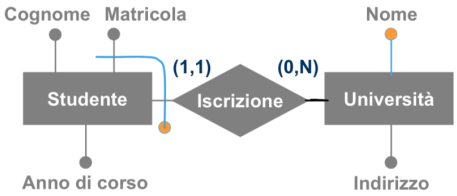
Identificatore

L'identificatore è lo strumento che permette l'identificazione univoca delle occorrenze di un'entità. Ogni entità deve avere almeno un identificatore, ma può averne più di uno. Può essere costituito da:

- *attributi dell'entità* (identificatore **interno**)
    - Se l'identificatore è unico, si annerisce il cerchio corrispondente; se è **composto**, si collegano gli attributi con una linea che termina con un cerchio annerito
    - Se è composto, vuol dire che *tale entità è rappresentata univocamente da quell'insieme di attributi*

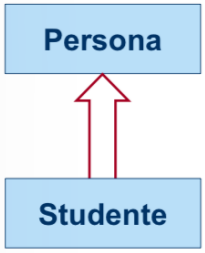


- *combinazione di attributi dell'entità e relazioni con altre entità* (identificatore **esterno**)
    - Si collegano attributi e relazioni con una linea che termina con un cerchio annerito
    - L'identificazione esterna è possibile solo se l'entità partecipa a una relazione con cardinalità (1,1)



Relazione IS-A

La relazione IS-A, o di sottoinsieme, si verifica quando le **istanze di un'entità sono anche le istanze di un'altra**; questa relazione si definisce tra un'entità padre e un'entità figlia, dove la figlia rappresenta un sottoinsieme del padre.

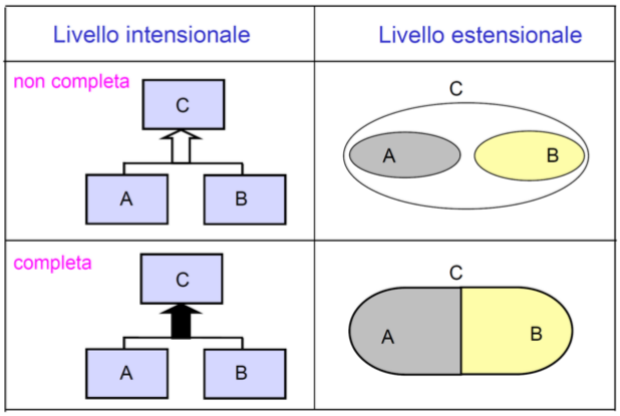


Ereditarietà

Secondo il principio di ereditarietà, **ogni proprietà dell'entità padre è anche una proprietà della sottoentità**; quindi, vengono ereditate, e non sono rappresentate esplicitamente nel diagramma.

Generalizzazione

La generalizzazione si verifica quando **un'entità padre è più generale rispetto a diverse sottoentità**; le sottoentità possono avere insiemi di istanze disgiunti, e la generalizzazione può essere *completa* o *non completa*, a seconda che l'unione delle istanze dellesottoentità corrisponda o meno all'insieme delle istanze dell'entità padre . **Una sottoentità può avere una e una sola entità padre**; quindi, l'ereditarietà multipla non è permessa. Ogni attributo o relazione dell'entità padre è implicitamente un attributo o relazione delle sottoentità. Un'entità può partecipare a **più gerarchie**, sia come entità padre che come sottoentità; se una generalizzazione include solo una sottoentità, questa viene considerata un sottoinsieme dell'entità padre.



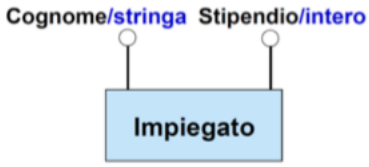
La generalizzazione si rappresenta graficamente con un **arco** che collega le sottoentità all'entità padre. Se la freccia che connette l'arco all'entità padre è *annerita*, significa che la

generazione è completa, ovvero ogni istanza delle sottoentità è anche un'istanza dell'entità padre.

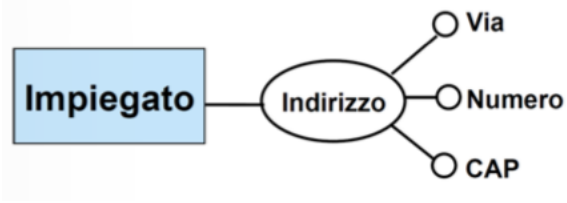
Quando un'entità E è la generalizzazione di altre entità, *tutte le proprietà di E sono ereditate dalle entità figlie***.** Questo significa che ogni istanza delle entità figlie è anche un'istanza dell'entità padre.

Attributi

Gli attributi sono proprietà locali delle entità che sono rilevanti per l'applicazione. Ogni attributo **associa un valore ad ogni istanza di entità**, valore che appartiene a un insieme definito come il dominio dell'attributo. Questi valori possono essere interi, caratteri, stringhe, ecc.... Una proprietà si considera **locale** quando il suo valore dipende esclusivamente dall'oggetto stesso, e non da altri elementi dello schema.



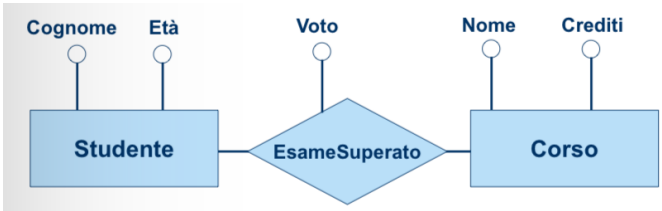
Possono essere anche **composti**, raggruppando attributi di una stessa entità o relazione che hanno significati o usi simili. Ad esempio, *via*, *numero civico* e *CAP* possono essere combinati in un unico attributo composto chiamato *indirizzo*.



Attributi di relazione

Un attributo di relazione è una caratteristica specifica che descrive una proprietà della relazione stesa, piuttosto che delle singole entità. Questo attributo è associato a ciascuna istanza di relazione, e assume un valore all'interno di un insieme definito come dominio dell'attributo.

Graficamente, ogni attributo di relazione è identificato da un nome unico, e rappresentato da un cerchio collegato alla relazione di appartenenza.

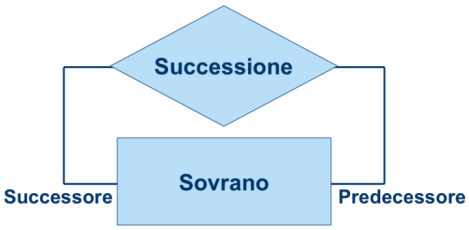


Relazioni

Descrivono azioni o situazioni che stabiliscono **legami logici tra le istanze delle entità**. Questi legami possono coinvolgere più di due entità, e il numero di entità coinvolte determina

il grado della relazione. Ogni relazione ha un *nome univoco* all'interno dello schema, che dovrebbe essere espressivo, e seguire convenzioni appropriate, come l'uso di sostantivi al singolare.

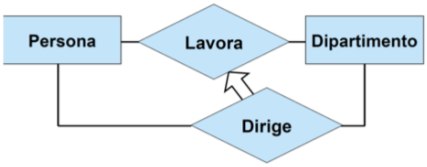
A livello estensivo ale, una relazione tra le entità E e F è costituita da un **insieme di coppie di istanze**; se nello schema S è definita una relazione R tra le entità E e F, allora in ogni istanza I dello schema S, alla relazione R è associato un insieme di coppie, c he rappresentano l'estensione di R in quella istanza. Quindi, è una *relazione matematica* tra le istanze delle entità coinvolte.



$$Istanze(I,R) \subseteq istanze(I,E) \times istanze(I,F)$$

Relazioni ISA fra relazioni

La semantica non cambia rispetto al caso della relazione ISA tra entità: se in uno schema S è definita la relazione ISA tra R e Q, allora in ogni istanza I dello schema S si ha che:



$$Istanze(I,R) \subseteq istanze(I,Q)$$

Cardinalità

Nel modello ER, utilizziamo tre simboli principali per esprimere la cardinalità delle relazioni:

- **0 e 1** per la cardinalità minima, dove 0 indica una partecipazione opzionale, e 1 indica una partecipazione obbligatoria
- **1 e "N"** per la cardinalità massima, con "N" che indica l'assenza di limiti superiori

Classificazione

Le relazioni si classificano in base alle cardinalità massime in:

- **uno a uno**: entrambe le entità hanno una cardinalità massima di uno
- **Uno a molti**: un'entità ha cardinalità massima uno, l'altra N
- **Molti a molti**: entrambe le entità hanno cardinalità massima N

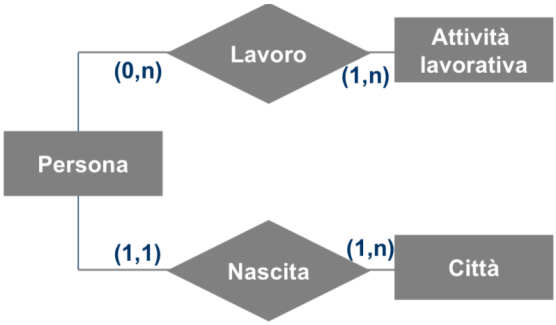
Vincolo di cardinalità

Il vincolo tra un'entità E, e una relazione R, stabilisce il numero minimo e massimo di istanze di R a cui ogni istanza d i E può partecipare. Questo vincolo serve a definire più precisamente il significato di una relazione.

Cardinalità degli attributi

Anche agli attributi si possono associare delle cardinalità per indicare:

- **Opzionalità**: l' attributo può essere presente o meno
- **Multivalore**: l' attributo può aver epiù valori contemporaneamente



Stanza di associazione

L'associazione è un concetto chiave nel modello entità-relazione, che si riferisce alla combinazione di istanze di entità diverse che partecipano insieme in una relazione. È importante notare che, seguendo la semantica delle relazioni, non è possibile avere due istanze identiche della stessa relazione che includono le stesse entità. Questo perché ogni istanza di associazione è unica, e rappresenta un legame specifico tra le entità coinvolte.

Istanze di associazione

Le entità possono partecipare **a più relazioni contemporaneamente**, e una relazione può coinvolgere più di due entità. In termini pratici, all'interno di un'istanza dello schema, una relazione tra le entità è rappresentata da un insieme di tuple, dove ogni x è **un'istanza dell'entità E_i in i**. Ogni tupla è considerata un'istanza della relazione R all'interno dell'istanza dello schema S.

Abbiamo vari tipi di relazioni:

- **Relazioni uno a uno**: ogni istanza di una entità è associata a non più di un'istanza dell'altra entità, e viceversa
- **Relazioni uno a molti**: un'istanza di un'entità è associata a molteplici istanze dell'altra entità, ma non viceversa
- **Relazioni molti a molti**: le istanze di entrambe le entità possono essere associate a molteplici istanze dell'altra entità,

Analisi dei casi dubbi

Quando si incontrano nomi di entità o associazioni che possono trarre in inganno, è utile analizzare la situazione pensando in termini di istanze concrete.

Unìassociazione può includere la stessa entità più volte, come nel caso di un'**associazione ricorsiva**. Questo può creare ambiguità, come nel caso di una "*Successione*", dove non è chiaro immediatamente chi sia il predecessore e chi il successore. Per risolvere questo problema, è necessario **specificare i ruoli all'interno dell'associazione**. Inoltre, un'associazione può essere simmetrica, riflessiva o transitiva, a seconda delle proprietà che possiede.

Diverse scelte

Un concetto può essere modellato come entità se le sue istanze hanno *significato concettuale autonomo*, *possiedono proprietà indipendenti*, o *sono importanti per l'applicazione*. Al contrario, sarà modellato come **attributo** se *rappresenta solo una proprietà locale di un altro concetto*, e le sue istanze non hanno significato autonomo.

Similmente, un concetto sarà modellato come entità se è *concettualmente significativo*, e può avere *proprietà o relazioni indipendenti*. Sarà invece modellato come **relazione** se le sue istanze non sono significative da sole, e rappresentano combinazioni di altre istanze.

Modello di dati

Il modello di dati è un insieme di concetti che **ci aiutano a organizzare e descrivere la struttura dei dati**. Al cuore di questo modello ci sono i meccanismi di strutturazione, noti anche come **costruttori di tipo**, che sono fondamentali per definire i tipi di dati attraverso il costruttore di relazione.

Questo ci permette di organizzare i dati in *insiemi di record* con una struttura fissa, rappresentati visivamente da una **tabella** composta da righe, che corrispondono a specifici *record*, e colonne, che rappresentano *i campi dei record*.

Campi dei record (schema)
Istanza/record

Esistono tre modelli logici principali: il **modello gerarchico**, il **modello reticolare**, e il **modello relazionale**.

Modello relazionale

Il modello relazionale è stato sviluppato per favorire l'*indipendenza dei dati*. Si basa sul concetto formale di relazione matematica, arricchito da varianti che ne aumentano la flessibilità.

In matematica, una relazione è un **sottoinsieme del prodotto cartesiano di domini**, rappresentato come insieme di n-uple ordinate.

$$D_1 \times \ldots \times D_n \approx \text{domini}$$

Nel modello relazionale dei dati, una **tabella** rappresenta una relazione se:

- i valori di ogni colonna sono omogenei
- le righe sono uniche senza ripetizioni
- le intestazioni delle colonne sono distinte

L'ordinamento tra le righe e colonne è irrilevante.

Il **dominio** di un attributo si riferisce all'insieme di tutti i valori possibili che l'attributo può assumere. Questi valori devono essere dello stesso tipo, e devono rispettare le regole definite per quel particolare dominio.

Possiamo riassumere tutti i termini del modello relazionale in questo:

Termini informali	Termini formali
Tabella	Relazione
Colonna	Attributo/dominio
Riga	Tupla
Valori nella colonna	Dominio
Definizione della tabella	Schema della relazione
Tabella popolata	Istanza della relazione

Modello basato su valori

I dati sono organizzati in tabelle, dove ogni dato è **identificato univocamente dai valori contenuti nelle sue colonne**. I riferimenti tra dati in relazioni diverse sono rappresentati attraverso valori uguali nei campi corrispondenti.

È tipico dei **database relazionali**, dove le relazioni tra le tabelle sono stabilite tramite chiavi primarie ed esterne, che corrispondono ai valori dei dati.

Questo modello, inoltre, presenta una serie di vantaggi:

- Indipendenza dalle strutture fisiche che possono cambiare dinamicamente
- Si rappresenta solo ciò che è rilevante dal punto di vista dell'applicazione
- I dati sono portabili più facilmente da un sistema ad un altro
- Per accedere ai dati non serve sapere come sono memorizzati fisicamente

Modello basato su puntatori

Questo modello utilizza puntatori o riferimenti per collegare record in strutture dati complesse come alberi o grafi. È caratteristico dei **modelli gerarchici o reticolari** di database, dove i dati sono direttamente collegati attraverso puntatori.

Schema e istanze

Lo **schema** di una base di dati è la parte in variante nel tempo che descrive la **struttura**, o l'aspetto intensionale, come le intestazioni delle tabelle. Di solito è espresso in maiuscole, con gli attributi tra parentesi.

SCHEMA(Attributi...)

Una collezione di schemi di relazione forma lo **schema di una base di dati**, indicato come

$$R = \{R_1(X_1), \ldots, R_k(X_k)\}$$

L'**istanza** di una base di dati rappresenta i *valori attuali*, che possono cambiare rapidamente, come il corpo della tabella o l'aspetto estensionale.

Chiavi e identificazione unica

Una tupla su un insieme di attributi X è una *funzione* che associa ciascun attributo A in X a una linea della tabella.

La **chiave** è un insieme di attributi che identifica univocamente le tuple di una relazione.

Un insieme k di attributi è una **superchiave** per una relazione r, se non esistono due tuple distinte t_1 e t_2 in r, tali che $\mathbf{t_1[K]=t_2[K]}$. Vediamo un esempio:

In questo caso, abbiamo due casi che potrebbero essere definiti come chiave:

- **Matricola** è una superchiave ed è minimale (contiene un solo attributo)
- **Cognome, Nome, Nascita** è anch'essa una superchiave ed è minimale

Matricola	Cognome	Nome	Corso	Nascita
27655	Rossi	Mario	Ing Inf	5/12/03
78763	Rossi	Mario	Ing Inf	3/11/02
65432	Neri	Piero	Ing Mecc	10/7/01
87654	Neri	Mario	Ing Inf	3/11/03
67653	Rossi	Piero	Ing Mecc	5/12/03

Questi due sono superchiavi poiché non esistono alcune istanze che sono uguali che possono essere distinte dalla chiave, e sono chiavi perché ci si possono identificare univocamente le tuple.

Informazione incompleta

I dati sono organizzati in una struttura rigida composta da tuple, che devono conformarsi agli schemi di relazione predefiniti; questo significa che ogni tuple è una sequenza di valori che rappresentano un'entità o una relazione. Può captare che i dati a nostra disposizione **non si adattino perfettamente a questi schemi**: potrebbero avere attributi aggiuntivi non previsti, o mancare di alcuni attributi necessari.

Per gestire l'informazione incompleta, il modello relazionale usa un approccio semplice ma efficace: **il valore NULL**, che rappresenta l'assenza di un valore, e non è considerato parte del dominio dell'attributo.

Vincoli di integrità

Sono regole che definiscono quali configurazioni di valori sono accettabili per le tuple in una relazione. Questi vincoli possono essere applicati a livello di dominio, di tuple, di chiave o tra relazioni diverse. Ad esempio, un *vincolo di chiave assicura che non ci siano due tuple con lo stesso valore per un attributo chiave*.

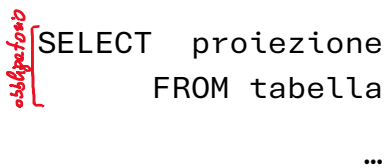
SQL

SQL (Structured Query Language) è un linguaggio utilizzato per **gestire e manipolare dati** in sistemi di gestione di database relazionali. Si compone di due parti principali: **DDL** (Data Definition Language) per la definizione e la modifica dello schema, e **DML** (Data Manipulation Language), per le operazioni di interrogazione e aggiornamento.

SQL è **dichiarativo**, quindi non possiamo scegliere l'ordine con cui avvengono le operazioni, ma dobbiamo attenerci alla struttura sintattica delle istruzioni.

La sintassi è la seguente:

- **I termini in maiuscolo** sono i termini del linguaggio
- **I termini in minuscolo** sono i termini variabili
- **I termini tra [e]** sono quelli opzionali
- **I termini tra { e }** compaiono zero o più volte
- **Il termine |** serve per dividere due opzioni.



La differenza tra algebra relazionale

Abbiamo queste differenze tra le due:

Algebra relazionale	SQL
Linguaggio prettamente **formale** che forma la base per linguaggi 'reali'	Linguaggio più usato per basi di dati relazionali
Linguaggio **procedurale**: si specifica l'algoritmo con cui ottenere il risultato	Linguaggio **dichiarativo**: si specifica il risultato da ottenere senza preoccuparsi di specificare l'algoritmo
Istruzioni equivalenti possono differire in termini di efficienza	Istruzioni equivalenti differiscono solo per **leggibilità**
Relazioni intese come insiemi di tuple definite su attributi	Relazioni intese come **tabelle**
Negli insiemi **non** ci possono essere **elementi uguali**	Possono esserci **righe uguali**
Gli operatori insiemistici si applicano solo a relazioni definite sugli stessi attributi; **l'ordine** degli attributi è **irrilevante**	Gli operatori insiemistici si applicano a relazioni definite sullo **stesso numero di attributi**; l'ordine degli attributi è rilevante
Il theta join ha senso solo per relazioni che non hanno attributi in comune, perché si combinano sempre le tuple che hanno gli stessi valori per gli attributi comuni	**Il theta join ha sempre senso**, perché si combinano solo le tuple che hanno lo stesso valore per gli attributi specificati nella condizione di join; occorre disambiguare attributi omonimi con la **clausola punto**
Si possono **ridenominare** solo **attributi**	Si possono **ridenominare attributi e tabelle**

Select

Quando lavoriamo con i database, l'operazione di selezione è fondamentale. Abbiamo due diverse selezioni:

- **Istanza di una tabella**: quando si seleziona un'intera tabella
  - Il risultato di una select è sempre una tabella
  - Non basta scrivere il nome come in AR, ma bisogna sempre usare l'istruzione di select
  - Per ottenere tutti gli attributi di una tabella si usa *****
- **Proiezione**: coincide con la semantica del corrispondente operatore in AR
  - **I duplicati vengono mantenuti**, quindi la cardinalità di una proiezione coincide sempre con la cardinalità dell'operando (a meno che non diversamente specificato)

Istruzioni principali

- **CREATE**: utilizzata per creare oggetti nel database

    - `CREATE TABLE`: crea una tabella, definendo uno schema di relazione e creando un'istanza vuota, senza tuple)
    - `CREATE DOMAIN`: definisce un dominio elementare o definito dall'utente
- **ALTER**: modifica oggetti esistenti nello schema
    - `ALTER DOMAIN`: modifica domini
    - `ALTER TABLE`: modifica tabelle
    - *`ADD COMLUMN…AFTER`, `DROP COLUMN`, `CHANGE COLUMN`, `MODIFY COLUMN`*
- **DROP**: cancella oggetti dal database
    - `DROP DOMAIN`: rimuove un dominio
    - `DROP TABLE`: elimina una tabella
    - Opzione **RESTRICT** (un oggetto non è rimosso se non è vuoto), **CASCADE** (viene rimosso l'oggetto e tutto ciò che è coinvolto nella definizione dell'oggetto)
- **SELECT**: utilizzata per estrarre informazioni da una o più tabelle
    - `SELECT proiezione FROM tabelle WHERE condizioni`
    - **LIKE**: seleziona stringhe di caratteri con due caratteri speciali:
        - _ indica un singolo carattere arbitrario
        - **%** indica una stringa con un numero arbitrario di caratteri
    - **IS_NULL**: gestisce i valori nulli
- **INSERT**: utilizzata per inserire nuove tuple in una tabella
    - `INSERT INTO tabella (colonna1, colonna2, …) VALUES (valore1, valore2, …)`
    - Aggiunge una nuova tupla con i valori specificati nelle colonne specificate
- **UPDATE**: utilizzata per modificare i dati esistenti in una tabella
    - `UPDATE tabella SET colonna1 = valore1, colonna2 = valore2 WHERE condizioni`
    - Modifica i valori delle colonne specificate per le tuple che soddisfano le condizioni specificate
- **DELETE**: utilizzata per rimuovere tuple da una tabella
    - `DELETE FROM tabella WHERE condizioni`
    - Rimuove le tuple che soddisfano le condizioni specificate

Creazione di uno schema

La dichiarazione **CREATE TABLE** viene utilizzata per creare una nuova tabella nel database. Ecco un esempio:

```
CREATE TABLE Impiegato(
     Matricola CHAR(6) PRIMARY KEY,
     Nome CHAR(20) NOT NULL,
     Cognome CHAR(20) NOT NULL,
     Dipart CHAR(15),
     Stipendio NUMERIC(9) DEFAULT 0,
```

```
    FOREIGN KEY(Dipart) REFERENCES Dipartimento(NomeDip),
    UNIQUE(Cognome, Nome));
```

In questo caso, si creano questi:

- **Matricola**: è un attributo i tipo CHAR(6) e viene utilizzato come chiave primaria
- **Nome** e **COGNOME**: sono attributi di tipo CHAR(20) e non possono essere nulli
- **Dipart**: è un attributo che fa riferimento al nome del dipartimento e si collega alla tabella "Dipartimento"
- **Stipendio**: è un attributo di tipo NUMERIC(9) con un valore predefinito di 0
- **UNIQUE(Cognome, Nome)**: garantisce che la combinazione di cognome e nome sia unica

Creazione di un dominio

Un dominio rappresenta un tipo di dato. Possiamo utilizzare i **domini elementari** (VARCHAR, INTEGER, etc.), o definire domini personalizzati. Vediamo un esempio:

```
CREATE DOMAIN Email AS VARCHAR(50);
```

Questo crea un dominio chiamato "E-mail" con lunghezza massima di 50 caratteri.

Questi sono i domini elementari predefiniti:

- **Testuali**: *VARCHAR(n)*, stringa di lunghezza variabile tra 0 le n
    - **Tipi stringa**: lunghezza fissa con *CHAR(N)*, lunghezza variabile con *VARCHAR(N)*, *TEXT*, *MEDIUMTEXT*, *LONGTEXT*
- **Numerici**: *INTEGER, SMALLINT, DECIMAL(i,j), NUMERIC(i,j)*
    - *TINYINT* (8 bit), *SMALLINT* (16 bit), *MEDIUMINT* (24 bit), *INT* (32 bit), *BIGINT* (64 bit)
    - **Decimali**: Virgola mobile con *FLOAT* (32 bit), *DOUBLE* (64 bit), virgola fissa con *DECIMAL(M,D)*
- **Data e ora:** *TIMESTAMP*
    - *DATE*, *DATETIME*, *TIMESTAMP* (1/1/1970 – 19/01/2038), *YEAR*
- **Booleani**: con dominio 0,1
- **BLOB, CLOB**: *oggetti di grandi dimensioni* (Blob: binari, Clob: caratteri)
- **Binari**: *BIT* (1-64 bit), *BINARY(N)* (0-255 byte), *VARBINARY(N)* (0-65536 byte), *BLOB*, *LONGBLOG*
- **Enumerazioni:** *ENUM("val1", "val2", "val3"; …)*
- **Booleani, solo di MySQL:** BOOLEAN

Vincoli di integrità

I vincoli di integrità garantiscono la coerenza e l'integrità dei dati. Alcuni esempi includono:

- **Intrarelazionali**: proprietà sempre valida all'interno di una relazione
  - NOT NULL: valore nullo non ammesso per il dominio
  - UNIQUE: definisce chiavi (fa eccezione per il valore nullo)
    - Impone che i valori dell'attributo A o insieme I siano chiave
    - Fa eccezione il valore nullo, che si assume sempre diverso nelle diverse occorrenze
    - Può essere *nella definizione di un attributo*, o *come elemento separato se definito su più attributi*
  - PRIMARY KEY: una sola chiave primaria
    - Può essere usata una sola volta in una tabella
  - CHECK: condizioni specifiche sui dati
  - CONSTRAINT: consente di assegnare un nome specifico a un vincolo. Possiamo poi usare *NOT NULL* and *UNIQUE*
- **Interrelazionali**: proprietà sempre valida relativa a più relazioni
  - FOREIGN KEY: riferimento a un'altra tabella
  - REFERENCES: specifica la tabella di riferimento

Reazione alle violazioni

Se viene eseguita una operazione di aggiornamento che porta alla violazione di un vincolo di integrità referenziale, lo stato della base di dati diventa non valido. Sono perciò dichiarate nella dichiarazione DDL un **insieme di politiche** per evitare che questo accada. Queste politiche sono eseguite usando **ON UPDATE** e **ON DELETE**. Abbiamo un esempio:

Tipo di violazione	Politica di reazione
Inserimento o modifica di tupla che viola il vincolo sulla tabella interna	In entrambi i casi il comando non viene eseguito
Inserimento o modifica di tupla che viola il vincolo sulla tabella esterna	Quattro possibili operazioni: - `cascade`: operazione propagata alla tabella interna - `set null`: null nella TI - `set default` - `no action`: rifiutata

Operatori insiemistici

Gli operatori insiemistici permettono di combinare i risultati di diverse query in modi che riflettono le operazioni insiemistiche classiche di unione, intersezione e differenza.

Nell'**algebra relazionale**, gli operatori insiemistici si applicano solo a relazioni con gli stessi attributi, mentre in **SQL** si applicano a relazioni con lo stesso numero di attributi. L'ordine degli attributi nell'algebra relazionale è irrilevante, ma in SQL è rilevante.

Abbiamo i seguenti operatori insiemistici comuni:

- **Unione**: combina le righe di due tabelle eliminando le righe duplicate
- **Intersezione**: trova le righe comuni tra due tabelle, eliminando le duplicazioni
- **Minus**: calcola la differenza tra le righe di due tabelle, escludendo le righe duplicate

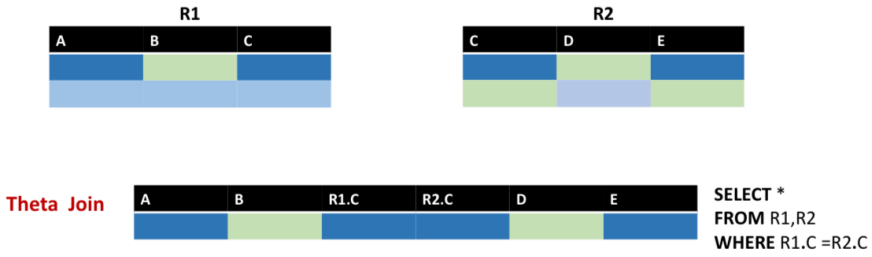
Ridenominazione

Utilizzando la clausola **AS**, è possibile ridenominare attributi e relazioni. In SQL, è spesso utile assegnare alias brevi alle tabelle per semplificare le query, e sono **temporanei** e **validi solo per la durata della query**.

Prodotto cartesiano e join

Elencare più tabelle nella clausola **FROM** produce un prodotto cartesiano tra di esse. Per ottenere un join significativo, si applica una condizione di selezione nella clausola **WHERE**, ottenendo così un theta join.

Un **theta join** ha senso solo quando le relazioni non hanno attributi in comune, combinando tutte con valori identici per gli attributi comuni. In SQL, il theta join è **sempre valido** poiché combina tulle basate sui valori specificati nella condizione di join.

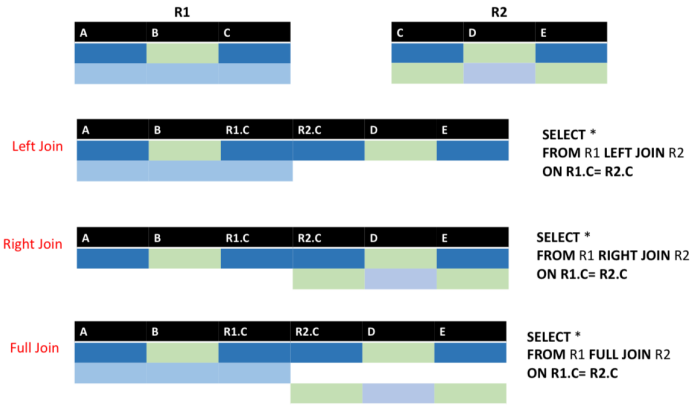


La clausola **.** specifica a quale relazione appartiene un attributo o a quale database appartiene una tabella.

Join

Il predicato di join può essere specificato direttamente nella clausola **FROM** usando il costrutto **JOIN ON**. Di default, JOIN esegue un inner join, ma è possibile specificare anche un join.

Un join sensato si ottiene applicando una selezione sul prodotto cartesiano, simile all'algebra relazionale. La condizione di selezione viene inserita nella clausola WHERE per ottenere il theta join. Il costrutto JOIN permette di specificare il tipo di join, come *inner join*, *join esterno* o *naturale*.



Inoltre, abbiamo per esempio varianti del join, come **INNER JOIN**, che viene utilizzato per combinare le righe di due o più tabelle basandosi su una condizione di corrispondenza. Questo join restituisce solo le righe che hanno valori corrispondenti in entrambe le righe.

Sintassi del join esplicito

Quindi, la condizione di join non compare nella clausola WHERE, ed è spostata nella FROM; nella WHERE restano le eventuali altre condizioni.

```
SELECT … FROM Table { … JOIN Table On Condition }, … [WHERE …]
```

Self Join

Nell'ambito dell' algebra relazionale, il self join è un'operazione che **permette di unire una tabella con una copia di se stessa**. È particolarmente utile per *confrontare righe all'interno della stessa tabella*, *identificare elementi duplicati,* e *gestire riferimenti incrociati tra dati contenuti nella stessa tabella*.

```
SELECT … FROM Table { … AS A JOIN … AS B ON Condition }, … [WHERE]
```

Operazioni di aggiornamento

Le operazioni di aggiornamento in SQL sono fondamentali per mantenere l'attualità dei dati; tra queste troviamo:

- **INSERT**: aggiunge nuove tuple alle tabelle
  - `INSERT INTO Table [(Attributes)] VALUES(Valori)` : singoli valori
  - `INSERT INTO Table [(Attributes)] SELECT…` : insiemi di tuple da BD
- **DELETE**: rimuove tuple esistenti

    - `DELETE FROM Table [WHERE Condition]`
- **UPDATE**: modifica i valori delle tuple
    - `UPDATE Table SET Attribute=<Expression | SELECT … | NULL | DEFAULT> [WHERE Condition]`
    - Il nuovo valore di un attributo può essere determinato da:
        - **Espressione**: valutazione di un'espressione basata sugli attributi della tabella
        - **SELECT**: risultato di una query
        - **NULL**: assegnazione del valore nullo
        - **DEFAULT**: assegnazione del valore di default
    - È importante considerare l'ordine di esecuzione delle query UPDATE, specialmente quando si modificano insiemi di righe che possono sovrapporsi

Regole per l'inserimento

Quando si inseriscono dati in una tabella con l'operazione INSERT, è importante seguire alcune regole:

- Se un attributo non è specificato, viene inserito il valore **NULL**
- Se si viola un vincolo NOT NULL, l'operazione viene rifiutata
- L'ordine degli attributi e dei valori inseriti è significativo
- Le liste degli attributi e dei valori devono avere lo stesso numero di elementi
- Se la lista di attributi è omessa, si fa riferimento a tutti gli attributi della relazione nell'ordine in cui sono definiti
- Se la lista degli attributi non include tutti gli attributi della relazione, per gli attributi mancanti viene inserito un valore di default o, in assenza di questo, il valore NULL

Regole per l'eliminazione

L'eliminazione può innescare eliminazioni in altre relazioni se **esistono vincoli di integrità** referenziale con politica di reazione a cascata.

Differenza tra DELETE e DROP

Con **DELETE** si rimuovono le tuple dalla tabella senza modificare lo schema del database. **DROP** invece rimuove una tabella *e tutti i suoi riferimenti dallo schema del database*; se si utilizza RESTRICT, il comando fallisce se ci sono tuple presenti nella tabella.

Interrogazioni

Abbiamo, come base, le **interrogazioni semplici**, dove abbiamo le condizioni atomiche; queste condizioni ci permettono di confrontare valori elementari, o al massimo aggregati di gruppi di valori. Per rendere queste interrogazioni più potenti, si possono estendere i confronti usando una sintassi con l'operatore WHERE; questo ci consente di confrontare attributi o liste di attributi con il risultato di **istruzioni SELECT annidate**.

```
WHERE Attributes/List OperatoreDiConfronto (SELECT ..)
```

Subquery e predicati

Le subquery possono essere utilizzate nelle clausole WHERE e HAVING, permettendoci di confrontare valori elementari con tabelle ottenute da altre query. I predicati possono essere semplici operatori di confronto come **=**, **<>**, **<**, **<=**, **>**, **>=**. **<>** vuol dire **diverso**.

Se si tratta di subquery che restituiscono più di un valore, dobbiamo ricorre a quantifica tori come ALL, ANY o predicati come [NOT] IN.

È importante che il dominio dell'attributo nella query esterna sia compatibile con quello della query interna; il numero di attributi confrontati deve corrispondere.

ALL e ANY

Questi due quantifica tori estendono i normali operatori di confronto.

ANY afferma che una condizione è vera se *il confronto è valido per almeno uno degli elementi di una lista*.

ALL richiede che *la condizione sia vera per tutti gli elementi***.**

IN e NOT IN

Questi operatori sono utilizzati per selezionare righe in base alla presenza o assenza di determinati valori in una lista.

IN è vero se *il valore è presente nell'insieme risultante della query*, mentre **NOT IN** è vero se *il valore non è presente*. Con questo possiamo anche confrontare tuple intere, non solo singoli attributi, utilizzando le parentesi.

Operatori aggregati

Gli operatori aggregati come **COUNT**, **MIN**, **MAX**, **AVG** e **SUM** sono valutati su insiemi di tuple e non sono meccanismi di selezione. Sono piuttosto funzioni che restituiscono un valore,

ignorando i valori nulli. Possono essere applicati a partizioni delle relazioni, utilizzando la clausola **GROUP BY**.

```
SELECT COUNT (*| [ Distinct ] List) FROM …
```

```
SELECT operatorAggregato ([DISTINCT] Attributo) FROM …
```

SQL permette di specificare un criterio di ordinamento delle righe, con un ordinamento **alfabetico** o **numerico**. La clausola **ORDER BY** è inserita in fondo alla query, e serve a questo.

```
ORDER BY attributo1 [asc|desc] {, attributo2 [asc|desc]}
```

Group By

Le funzioni di aggregazione possono essere applicate a partizioni delle relazioni, cioè a gruppi di tuple; in questo caso, si usa la clausola GROUP BY. L'operazione GROUP BY inizia *eseguendo l'interrogazione base* senza tener conto del GROUP BY, poi successivamente raggruppa le **righe che hanno stessi valori** per gli attributi che compaiono nella lista Attributi della GROUP BY. Infine, applica l'*operatore aggregato* a ciascun gruppo righe.

```
SELECT operatoreAggregato (* | [DISTINCT] Attributes)
FROM … WHERE …
GROUP BY listaAttributi
```

In una interrogazione che fa uso della GROUP BY, può comparire come argomento della SELECT, e quindi come **insieme di attributi** nella target list, solitamente un sottoinsieme degli attributi che compaiono nella clausola GROUP BY. Non è possibile però **esprimere condizioni su sottoinsiemi**.

Having

La clausola HAVING permette di esprimere **condizioni sui gruppi**, applicate a ogni insieme di n-ple risultato dell'applicazione del GROUP BY. È utile specificare HAVING BY, perché permette di specificare delle condizioni dopo aver specificato un GROUP BY.

```
SELECT operatoreAggregato (* | [DISTINCT] Attributes)
FROM … WHERE …
GROUP BY listaAttributi
HAVING CondizioniSelezioneGruppi
```

MySQL

MySQL include alcune istruzioni in più rispetto a SQL.

Autoincrement

Una colonna di tipo intero può essere dichiarata *ad incremento automatico* con la keyword AUTO_INCREMENT.

Il suo valore viene automaticamente generato all'inserimento dei record, tramite un **contatore incrementale interno** al DBMS associato alla tabella.

Comandi

Nome	Esempio
Elenco dei database	`SHOW DATABASES; SHOW SCHEMAS;`
Selezione di un database	`USE database;`
Elenco delle tabelle	`SHOW TABLES; SHOW TABLES FROM database;`
Ispezione della struttura di una tabella	`DESCRIBE database.table; SHOW COLUMNS FROM table FROM database;`
Creazione di un database	`CREATE DATABASE [IF NOT EXISTS] database; CREATE SCHEMA database;`
Eliminazione di un database	`DROP DATABASE database; DROP SCHEMA database;`
Aggiunta di colonne	`ALTER TABLE table; ADD COLUMN …;`
Rimozione di colonne	`ALTER TABLE table; DROP COLUMN …;`
Modifica di colonne	`ALTER TABLE table; CHANGE COLUMN …; MODIFY COLUMN …;`
Valore di default	`ALTER TABLE table; ALTER COLUMN … DROP DEFAULT; ALTER COLUMN … SET DEFAULT "default";`
Definizione chiave primaria	`ALTER TABLE table ADD PRIMARY KEY (key);`
Eliminazione chiave primaria	`ALTER TABLE table DROP PRIMARY KEY;`
Vincolo di unicità e di integrità referenziale	`ALTER TABLE table ADD CONSTRAINT … UNIQUE, ADD CONSTRAINT … FOREIGN KEY (..) REFERENCES (…);`
Foreign keys	`CONSTRAINT key FOREIGN KEY (attributo) REFERENCES (otherAttributo);`
Eliminazione vincolo di unicità	`ALTER TABLE table DROP …;`
Eliminazione vincolo di integrità referenziale	`ALTER TABLE table DROP FOREIGN KEY …;`
Ridenominazione di una tabella	`ALTER TABLE table RENAME TO …`
Eliminazione di una tabella	`DROP TABLE table;`

Interrogazioni

Nome	Esempio
Select	SELECT … FROM table WHERE condition
Seleziona tutte le colonne	SELECT * FROM table WHERE condition
Between	WHERE thing BETWEEN min AND max
Like	Thing LIKE "%" (*%: carattere jolly*)
Rimozione di duplicati	SELECT DISTINCT column FROM table
Funzioni matematiche	ABS(v): valore assoluto CEIL (v): minor valore intero >= v FLOOR(v): maggior valore intero <= v SIN(v), COS(v), TAN(v), etc. LOG(v), LN(v), LOG10(v), LOG2(v) RAND(): valore casuale floating-point tra 0.0 e 1.0 ROUND(v,d): arrotonda v a d cifre decimali
Funzioni su stringhe	LENGTH(s): numero di byte della stringa CHAR_LENGTH(s): numero di caratteri della stringa CONCAT(s1, s2, …): concatenazione delle stringhe LCASE(s): converte in lowercase UCASE(s): converte in uppercase REPLACE(s, from, to): rimpiazza ogni occorrenza di *from* nella stringa *s* con la stringa *to* LEFT(s, n): restituisce i primi n caratteri della stringa s *SELECT column FROM table WHERE LEFT(s,n) = char;* RIGHT(s, n): restituisce gli ultimi n caratteri della stringa s LTRIM(s), RTRIM(s), TRIM(s): rimuovono eventuali blank SUBSTRING(s, p, l) o SUBSTRING(s FROM p FOR l): estrae dalla stringa s una sottostringa di lunghezza l partendo dal carattere in posizione p
Funzioni temporali	NOW(), CURRENT_TIMESTAMP(): data e ora attuale CURTIME(), CURRENT_TIME(): ora attuale CURDATE(), CURRENT_DATE(): data attuale DATE(dt): estrae la data dal valore dell'espressione temporale dt TIME(dt): estra l'ora dal valore dell'espressione temporale dt YEAR(dt), MONTH(dt), DAY(dt), HOUR(dt), MINUTE(dt), SECOND(dt), MICROSECOND(dt): estrae vari formati DAYNAME(dt): nome del giorno della settimana rappresentato in dt DATE_FORAMT(dt, format): formatta la data dt secondo il formato *format* ADDDATE(dt, INTERVAL expr unit), SUBDATE(dt, INTERVAL expr unit): aggiunge o sottrae dalla data dt il valore expr di unità temporali unit ADDTIME(dt1, dt2), SUBTIME(dt1, dt2): somma o sottrae le espressioni temporali dt1 e dt2 DATEDIFF(dt1, dt2): numero di giorni di differenza tra dt1 e dt2

Controllo di flusso	CASE v WHEN v1 THEN r1 WHEN v2 THEN r2 … ELSE r END; confronta v con i valori indicati nelle clausole WHEN, e ritorna il risultato indicato nella corrispondente clausola THEN CASE WHEN cond1 THEN r1 WHEN cond2 THEN r2 … ELSE r END; valuta le condizioni indicati nelle clausole WHEN IF(expr, exprTrue, exprElse): valuta l'espressione, se produce TRUE ritorna il valore exprTrue, altrimenti di exprElse IFNULL(expr, exprNull) NULLIF(expr1, expr2): ritorna NULL se expr1=expr2, altrimenti ritorna expr1
Selezione da più tabelle (join implicito)	SELECT columns… AS alias, FROM table1 AS alias, table2 AS alias
Join esplicito	FROM table JOIN tabellaInJoin ON condizioneDiJoin LEFT JOIN: mantiene solo le tuple della tabella alla sinistra dell'operatore di join RIGHT JOIN: mantiene solo le tuple della tabella alla destra FULL OUTER JOIN: mantiene le tuple di entrambe le tabelle
Ordinamento	ORDER BY column ASC (crescente), column DESC (decrescente)
Aggregazione	(All'interno di SELECT) COUNT(expr): conta il numero di valori non NULL SUM(expr): somma dei valori AVG(expr): calcola la media MIN(expr): calcola il minimo MAX(expr): calcola il massimo
Raggruppamento	GROUP BY column; HAVING condition; filtra il risultato finale, valutata dopo il raggruppamento
Quantificatory	ANY/SOME … : quantificatore **esistenziale**, confronto con almeno una ALL : quantificatore **universale**, confronto con tutte le tuple
Esistenza	EXISTS: true se la subquery produce almeno un risultato NOT EXISTS: true se la subquery non produce risultati
Insiemistiche	UNION: unione INTERSECT: intersezione EXCEPT: differenza Risultato privo di duplicati

Abbiamo anche **interrogazioni nidificate** con subquery:

- Se è nella clausola WHERE, serve per **filtrare i record** selezionati dalla query principale sulla base dell'esito della subquery
- Se è nella clausola FROM, serve per **arricchire l'insieme di tuple** su cui viene effettuata l' interrogazione principale con un set di tuple fornito dalla subquery.

Propagation constraints

Azione	In caso di modifica	In caso di eliminazione

CASCADE	Valore automaticamente aggiornato	Record automaticamente eliminato
RESTRICT	Se l'operazione viola l'integrità, viene negata	Se l'operazione viola l'integrità, viene negata
NO ACTION	Integrità verificata solo al termine dell'esecuzione	Della transazione
SET DEFAULT	Valore impostato al valore di default della colonna	Stesso di modifica
SET NULL	Valore impostato a NULL	Valore impostato a NULL

Algebra relazionale

I modelli **concettuali e logici** sono strumenti fondamentali per descrivere le informazioni; tuttavia, non sono immediatamente interpretabili da un computer. Per questo motivo, è necessario tradurre i modelli in **linguaggi di programmazione**, che sono dotati di **sintassi** (definisce le regole grammaticali del linguaggio, stabilendo quali frasi sono corrette) e di **semantica** (determina le operazioni che vengono eseguite quando degli operatori del linguaggio sono messi in atto). Esistono due *tipi di linguaggio principali:*

- **Linguaggi procedurali**: si specificano le modalità di generazione del risultato, ovvero il *come* ottenere ciò che si desidera
- **Linguaggi dichiarativi**: si concentrano sulle proprietà del risultato, descrivendo il *che cosa* si vuole ottenere, piuttosto che il come.

Un esempio di linguaggio formale è l'**algebra relazionale**, che è **procedurale**, e richiede di specificare l'algoritmo per ottenere un risultato. D'altra parte, invece, **SQL** è un linguaggio **parzialmente dichiarativo**, dove si specifica il risultato desiderato senza doversi preoccupare dell'algoritmo.

Algebra relazionale	SQL
Linguaggio prettamente formale che forma la base per linguaggi reali	Linguaggio più usato per basi di dati relazionali
Istruzioni equivalenti possono differire in termini di efficienza	Istruzioni equivalenti differiscono solo per leggibilità
Relazioni intese in senso matematico, come **insiemi di tuple** definite su attributi	Relazioni intese come **tabelle**
Negli insiemi **non ci possono essere elementi uguali**	Possono esserci **righe uguali**

Operatori

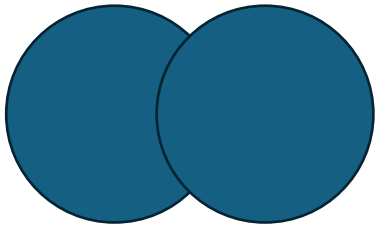
Gli operatori agiscono su relazioni per **produrre nuove** relazioni, e possono essere combinati per formare interrogazioni complesse. Poiché le relazioni sono insiemi, possiamo applicare gli

operatori insiemistici che producono a loro volta insiemi. Abbiamo però delle regole fondamentali:

- È possibile applicare operazioni di **unione**, **intersezione** e **differenza** solo tra relazioni definite sugli stessi attributi
- I risultati di queste operazioni devono essere relazioni, quindi tuple identiche nel risultato si traducono in una singola tuple nel set finale

Unione

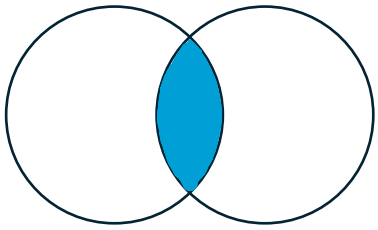
L'operazione di unione, indicata con $r1 \cup r2$,



combina le tuple di due relazioni che hanno lo stesso schema, eliminando eventuali duplicati. Corrisponde all'unione insiemistica in matematica, e viene utilizzata per aggregare i dati da due insiemi.

Intersezione

L'operazione di intersezione, rappresentata da $r1 \cap r2$, identifica le tuple che sono presenti in **entrambe le relazioni**. Le relazioni devono avere lo stesso schema, e seleziona solo gli elementi comuni.



Ridenominazione

La ridenominazione, rappresentata da ρy,z ¬ x,w (r) è un'operazione che consente di cambiare il nome di una relazione o di uno dei suoi attributi. È utile per evitare **ambiguità** quando si lavora con più relazioni, e si necessita di unire o confrontare dati con nomi di attributi simili.

Proiezione

La proiezione, indicata con $\prod_x(r)$, è un'operazione che estrae determinate colonne da una relazione, eliminando le colonne non specificate. Questo permette di focalizzarti su specifici dati all'interno di una relazione più ampia. Si scrive, inoltre **PROJ**.

La **cardinalità** di una relazione è il **numero delle sue tuple**, e si indica con $|r|$. Una relazione risultante da una proiezione contiene al massimo tante tuple quante sono nell'operando originale, ma può anche contenerne di meno.

Se x è una superchiave della relazione r, allora la proiezione su x, indicata con $\Pi_x(r)$, conterrà esattamente **tante tuple quante sono in r**, effettivamente:

$$|\Pi_x(r)|=|r|$$

Questo accade perché, per definizione, ogni superchiave compare una sola volta nella relazione, garantendo l'unicità delle tuple nella relazione.

Selezione

Selezione, indicato come SEL o σ. Seleziona le righe che soddisfano una condizione specifica, come:

$$SEL_{condition}(row)$$