Basi di Dati

Angelo Passarelli

October 28, 2023

Appunti basati sulle lezioni e dispense della professoressa Giovanna Rosone [1]

[1] https://pages.di.unipi.it/rosone/index.html

Contenuti

0 Introduzione

Definizione (Base di Dati)**.** Una base di dati è un insieme organizzato di dati utilizzati per il supporto allo svolgimento di attività.

Struttura dei Dati I dati sono organizzati in insiemi strutturati che possono presentare fra loro delle relazioni. Tuple che rappresentano dati nello stesso insieme devono essere omogenee ed univoche.

Definizione (Sistema Informativo)**.** Un sistema informativo è una combinazione di risorse umane e/o materiali e procedure organizzate per la raccolta, l'archiviazione, l'elaborazione e lo scambio di informazioni necessarie ad un'attività, le quali possono essere classificate in:

- Informazioni di servizio (operative).
- Informazioni di controllo (pianificazione e gestione).
- Informazioni di governo (pianificazione strategica).

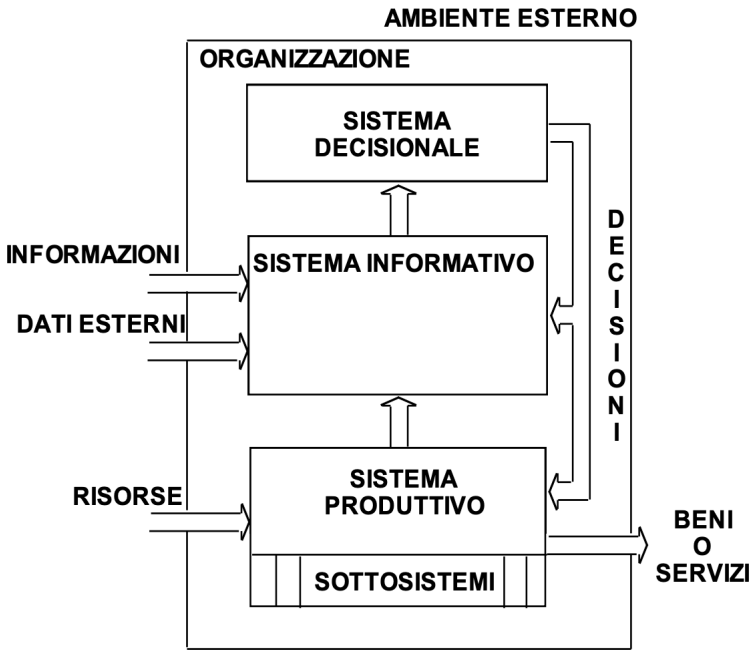


Figura 1: Esempio di sistema informativo

Definizione (Sistema Informativo Automatizzato)**.** Un sistema informativo automatizzato è una parte del sistema informativo che permette di implementare le procedure che si occupano della gestione delle informazioni usando un sistema informatico.

Definizione (Sistema Informatico)**.** Un sistema informatico è l'insieme delle tecnologie a supporto per le attività di un'organizzazione. Si possono classificare in:

- **Sistemi informatici operativi**: questi sistemi si utilizzano per svolgere le normali attività dell'azienda per la fruizione del suo bene o servizio, e per la gestione interna dei singoli reparti dell'azienda. Le operazioni sui dati in questo sistema sono di tipo OLTP (*On-Line Transaction Processing*) e prevedono elaborazioni semplici che coinvolgono pochi dati che vengono aggiornati molto frequentemente.
- **Sistemi informatici direzionali**: i dati sono organizzati in *Data Warehouse* che consentono di aiutare l'azienda nei processi di controllo delle prestazioni e di decisione manageriale. Le elaborazioni su questo tipo di sistema si chiamano OLAP (*On-Line Analytical Processing*) e prevedono l'utilizzo di una grande mole di dati che sono per lo più storici. In questo caso i dati vengono aggiornati molto raramente, ma su di essi vengono svolte molte operazioni, anche da un punto di vista multidimensionale, ovvero vengono incrociati più dati per analizzare le informazioni ottenute sotto molteplici punti di vista.

Definizione (DBMS)**.** Un *Database Management System* è un sistema che garantisce il controllo e la gestione di dati per renderli accessibili agli utenti opportuni in base ai loro privilegi. Il DBMS fornisce anche dei linguaggi che permettono di definire lo schema di un database, di scegliere le strutture dati opportune per la memorizzazione dei dati, di rispettare i vincoli per ogni tipo di dato e di poter modificare e interrogare il database.

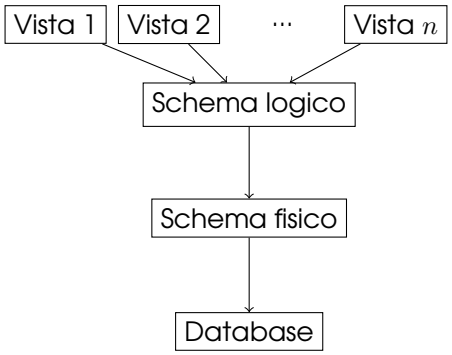
Metadati All'interno del database sono anche memorizzati dei metadati che si riferiscono agli utenti e allo schema utilizzato dal database stesso. Anche i metadati possono essere interrogati e modificati.

1 DBMS e Linguaggi

Si distinguono tre diversi livelli di descrizione dei dati:

- A livello di *vista* logica: descrive come deve apparire il database a seconda dell'utente che lo usa, in base ai suoi permessi.

- A livello logico: descrive la struttura degli insiemi dei dati e le relazioni fra essi, senza doversi occupare della loro organizzazione nella memoria.
- A livello fisico: viene descritto come sono organizzati fisicamente i dati nella memoria e vengono riportate quali strutture dati ausiliare vengono utilizzate.



Quest approccio permette di garantire le proprietà di indipendenza logica e fisica:

- **Indipendenza Logica**: gli applicativi non necessitano modifiche in seguite a variazioni dello schema logico.
- **Indipendenza Fisica**: gli applicativi non necessitano modifiche in seguito a cambiamenti dell'organizzazione fisica dei dati.

Per quanto riguarda i linguaggi di interrogazione, questi possono essere distinti in:

- **DML** (Data Manipulation Language): per l'interrogazione e l'aggiornamento dei dati.
- **DDL** (Data Definition Language): per la definizione di schemi, sia logici che fisici, ed altre operazioni.

1.1 Funzionalità del DBMS

Un DBMS deve prevedere più modalità d'uso per soddisfare le esigenze di più categorie di utenti che accedono al database. Deve poter offrire:

- Un'interfaccia grafica per accedere ai dati.
- Un linguaggio di interrogazione per gli utenti inesperti (non programmatori).

- Un linguaggio di programmazione per chi sviluppa applicazioni, nello specifico deve prevedere l'integrazione del *DDL* e del *DML* nel linguaggio ospite.
- Un linguaggio per lo sviluppo di interfacce per le applicazioni.
- Predisporre per l'amministratore strumenti per stabilire i diritti d'accesso ai dati, per il ripristino del sistema e per la modifica e la definizione degli schemi logici (sia interno che esterno).

1.2 Proprietà dei Database

Il DBMS permette di garantire al database le seguenti proprietà:

- Integrità: mantenimento dei vincoli d'integrità dichiarati in fase di definizione dello schema.
- Affidabilità: protezione dei dati da parte di malfunzionamenti sia software che hardware e da anomalie indesiderate come l'accesso concorrente al database da parte di più utenti.
- Sicurezza: protezione dei dati da parte di utenti non autorizzati.

Inoltre un DBMS deve essere in grado di gestire collezioni di dati che siano:

- Grandi
- Persistenti: il periodo di vita dei dati è indipendente dai programmi che li utilizzano.
- Condivise: possono essere usati da programmi diversi.

Il DMBS deve essere anche efficiente (utilizzando al meglio le risorse in termini di *spazio* e *tempo*) ed efficace.

1.3 Transazioni

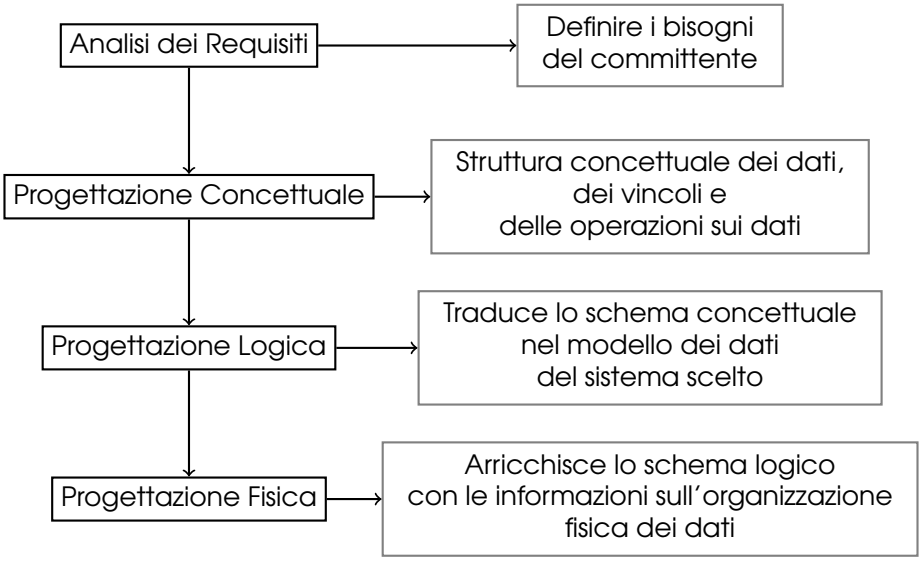
Definizione (Transazione)**.** Una transazione è una serie di azioni di lettura e scrittura sulla memoria permanente o di elaborazione dati in memoria temporanea. Presenta le seguenti proprietà:

- **Atomicità**: le transizioni che non vanno a buon fine o che vengono abortite sono trattate come se non fossero mai state eseguite.
- **Persistenza**: le modifiche effettuate da una transizione andata a buon fine sono permanenti, ovvero non possono essere alterate da malfunzionamenti.
- **Serializzabilità**: nel caso di esecuzioni concorrenti di più transazioni, l'effetto ottenuto è quello di un'esecuzione seriale.

2 Modellazione

Definizione (Modello Astratto)**.** Un modello astratto è la rappresentazione formale di idee e conoscenze relative ad un fenomeno.

La modellazione è centrale nella progettazione del database che comprende le fasi presenti in figura.



2.1 Fasi della Modellazione

2.2 Analizzare il Dominio

Il dominio può presentare più aspetti da dover analizzare:

- Aspetto *ontologico*: conoscere ciò che si suppone esista nell'universo del contesto e quindi ciò che è da modellare. Occorre analizzare tre tipi di conoscenze:
  - Conoscenza *concreta*: le entità del contesto e le associazioni fra di esse.

    Definizione (Entità)**.** Sono oggetti di cui occorre definire le proprietà.

    Definizione (Proprietà)**.** Descrivono le caratteristiche di determinate entità e sono formata da una coppia `(Attributo, Valore)`. Ogni proprietà ha ad essa associato un dominio, quindi un insieme di valori che può assumere. Inoltre le proprietà si possono classificare in:

    - Atomiche o Strutturate: atomiche se il loro valore non è ulteriormente scomponibile.
    - Totali o Parziali: se è obbligatoria oppure opzionale.
    - Univoche o Multivalore: univoca se per ogni entità la scelta del valore è unico (*es. codice fiscale*).
    - Costanti o Variabili
    - Calcolate o Non Calcolate: calcolata se è possibile derivarla da altre proprietà.

  Definizione (Tipo di un'entità)**.** Ogni entità appartiene ad un tipo che ne indica la propria natura.

  Definizione (Collezione)**.** Insieme di entità dello stesso tipo.

  – Conoscenza *astratta*: la struttura e i vincoli sulle entità.
  – Conoscenza *procedurale*: le operazioni di base, sia dei singoli utenti e sia come avviene la comunicazione con il sistema informatico.

- Aspetto *logico*: meccanismi di astrazione (*modello di dati, per es. diagrammi E-R* [2]) con cui descrivere la struttura della conoscenza concreta.
- Aspetto *linguistico*: linguaggio formale con cui definire il modello.
- Aspetto *pragmatico*: insieme di regole da seguire in fase di modellazione.

2.3 Oggetti e Classi

Definizione (Oggetto)**.** Un oggetto è un'entità software che presenta uno *stato*, un *comportamento* e un'*identità*. Lo stato è rappresentato da un insieme di costanti o variabili, mentre il comportamento è un insieme di procedure locali chiamate *metodi*. Un oggetto può rispondere a dei messaggi di input, con dei valori memorizzati nello stato o calcolandoli con un metodo.

Definizione (Classe)**.** Una classe è un insieme di oggetti dello stesso tipo, e presenta delle operazioni per l'inserimento e la rimozione di elementi.

Definizione (Tipo Oggetto)**.** Un tipo oggetto definisce l'insieme degli attributi a cui può combaciare un insieme di possibili oggetti. I tipi oggetto non sono presenti nei diagrammi E-R, però dagli attributi di una collezione è possibile dedurre il tipo oggetto associato.

[2] `https://it.wikipedia.org/wiki/Modello_E-R`

2.4 Associazioni

Definizione (Istanza di un'associazione)**.** Un'istanza di un'associazione determina un legame logico tra due o più istanze.

Definizione (Associazione)**.** Un'associazione $R(X, Y)$ fra due collezioni di entità chiamate X e Y è un insieme, che varia nel tempo, di istanze di associazione tra gli elementi delle due collezioni. Il prodotto cartesiano $(X \cdot Y)$ è chiamato dominio dell'associazione.

Un'associazione è caratterizzata da due proprietà: molteplicità e totalità.

Definizione (Vincolo di Unicità)**.** Un'associazione $R(X, Y)$ è detta univoca rispetto ad X se per ogni elemento di $x \in X$ esiste al più un elemento $y \in Y$ che è associato ad x. Se questo vincolo non vale, si dice che l'associazione è multivalore rispetto ad X.

Cardinalità dell'Associazione

- $R(X, Y)$ è **(1:N)** se è multivalore sy X ed univoca su Y.
- $R(X, Y)$ è **(N:1)** se è univoca sy X e multivalore su Y.
- $R(X, Y)$ è **(N:M)** se è multivalore sy X e multivalore su Y.
- $R(X, Y)$ è **(1:1)** se è univoca sy X ed univoca su Y.

Definizione (Vincolo di Totalità)**.** Un'associazione $R(X, Y)$ è detta totale su X se per ogni elemento $x \in X$ esiste almeno un elemento $y \in Y$ associato ad x. Se questo vincolo non vale, si dice che l'associazione è parziale rispetto ad X.

Rappresentazione delle Associazioni Un'associazione fra due collezioni C_1 e C_2 si rappresenta con una linea che collega le due classi. La linea si etichetta con il nome dell'associazione. L'univocità di una classe C_1 si rappresenta disegnando una freccia singola sulla linea che esce và da C_1 a C_2. Se invece l'associazione è multivalore si indica con una doppia freccia. La parzialità invece è rappresentata con un taglio sulla linea vicino alla freccia, mentre la totalità è rappresentata dall'assenza del taglio.

Figura 2: In questo caso abbiamo un'associazione *multivalore* da entrambe la parti, ma *parziale* per C_2 e *totale* per C_1

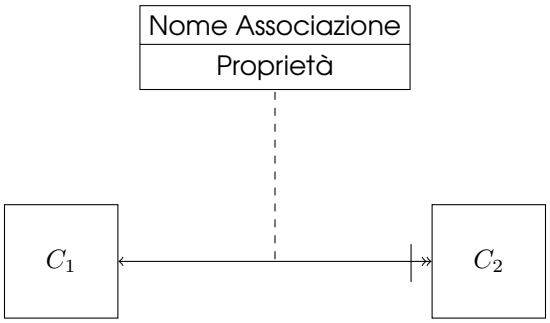


Figura 3: Le associazioni possono presentare proprietà

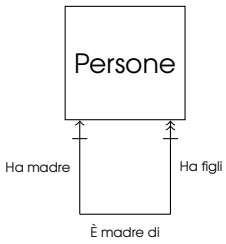


Figura 4: O possono anche essere ricorsive. In questo caso occorre etichettare l'associazione non solo con il proprio nome, ma anche con i nomi dei ruoli che hanno le due entità nell'associazione.

Associazione Non Binarie Le associazioni non binarie, per semplicità, non vengono rappresentate graficamente, ma ad esempio, per quanto

riguarda quelle *ternarie*, queste vengono trasformate in tre associazioni *binarie* aggiungendo un'altra collezione al posto dell'associazione *ternaria*.

2.5 Gerarchie

Le classi di entità possono essere organizzate in una gerarchia di *specializzazione*. Una classe della gerarchia minore viene chiamata **sottoclasse**, mentre le altre si chiamano **superclassi**. Gli elementi di una sottoclasse sono un sottoinsieme degli elementi della superclasse.

2.5.1 Tipi Oggetto

Fra i *tipi oggetto* viene definita una relazione di sottotipo, che comprende le seguenti proprietà:

- È una relazione asimmetrica, riflessiva e transitiva.
- Inoltre, se un tipo T è *sottotipo* di T', allora tutti gli elementi di T possono essere usati in tutti i contesti in cui appaiono elementi di tipo T'. Questa proprietà è chiamata sostitutività ed è data dal fatto che gli elementi di T hanno tutte le proprietà degli elementi di T', e per ogni proprietà di T', il suo tipo in T è un sottotipo di quello che ha in T'.

Ereditarietà L'*ereditarietà* è una proprietà delle gerarchie che permette di definire un *tipo oggetto* a partire da un altro. In quanto, nel nostro contesto, a partire da un tipo, si vuole solo definire un sottotipo, si parla di *ereditarietà stretta*, che permette solo:

- L'aggiunta di altri *attributi*.
- La ridefinizione di attributi del *supertipo*, però solo specializzando ulteriormente il tipo dell'attributo.

2.5.2 Classi

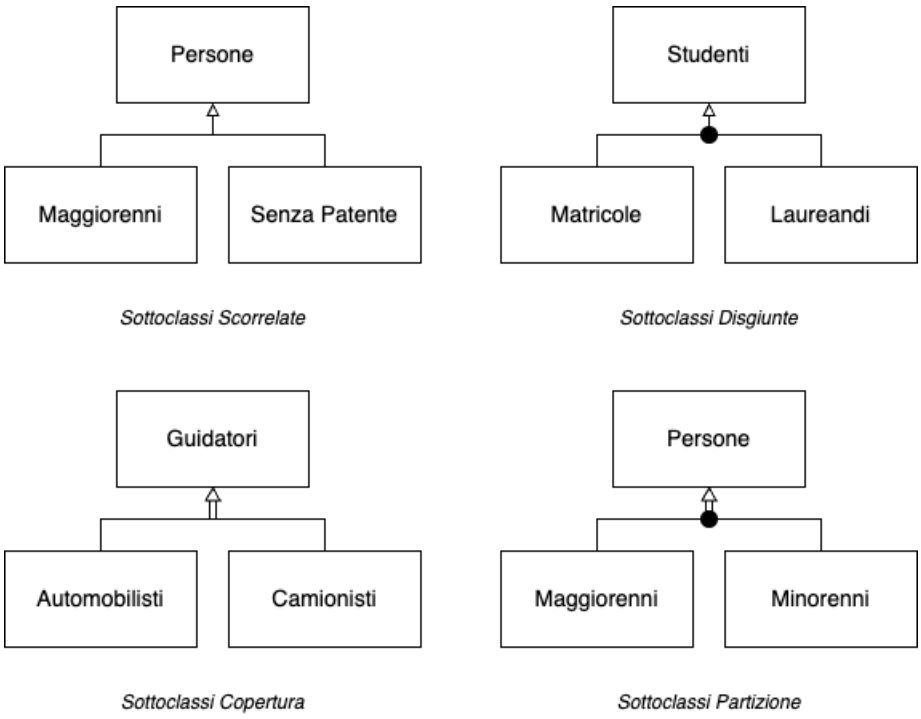
Fra le *classi*, invece, viene definita una relazione di sottoclasse, con le seguenti proprietà:

- Sempre asimmetrica, riflessiva e transitiva, come per i *tipi oggetto*.
- Se una classe C è *sottoclasse* di C', allora il tipo degli elementi di C è sottotipo del tipo degli elementi di C' (**vincolo intensionale**).
- Se C è *sottoclasse* di C', allora gli elementi di C sono un sottoinsieme degli elementi di C' (**vincolo estensionale**).

Vincoli Possiamo distinguere due tipi di vincoli su insiemi di sottoclassi:

- Disgiunzione: ogni coppia di sottoclassi nell'insieme è *disgiunta*, quindi è priva di elementi comuni.
- Copertura: l'unione degli elementi delle sottoclassi coincide con l'insieme degli elementi della *superclasse*.

Se possiedono entrambi i vincoli, allora l'insieme delle *sottoclassi* forma una partizione della *superclasse*. Altrimenti se nessuno dei due vincoli è rispettato, si dice che le sottoclassi sono scorrelate.



2.5.3 Ereditarietà Multipla

Un tipo può anche essere definito per *ereditarietà* anche a partire da più di un *supertipo*. Questo però può creare alcuni problemi quando lo stesso *attributo* viene ereditato da più di un supertipo, ma i tipi degli attributi fra loro sono diversi.

3 Progettazione Logica

L'obbiettivo della **progettazione logica** è quello di ridefinire lo *schema concettuale* in uno *schema logico relazionale* che rappresenti gli stessi dati ma in una maniere più efficiente (minimizzando la ridondanza) e corretta, per esempio tenendo conto della dimensione dei dati e del tipo di operazioni che si effettueranno sul database. Anche perchè alcuni costrutti dello *schema relazionale* non sono rappresentabili concretamente.

3.1 Relazioni

Definizione (Relazione Matematica)**.** Una *relazione matematica* è un insieme di *n*-uple ordinate $(d_1, \ldots, d_n)$ tali che $d_1 \in D_1, \ldots, d_n \in D_n$.

Essendo un insieme, non c'è ordinamento fra le *n*-uple, che inoltre devono essere tutte distinte. Però l'ordinamento all'interno della *n*-upla conta, infatti l'*i*-esimo valore deve provenire dall'*i*-esimo dominio. Per questo si dice che la struttura della relazione è *posizionale*.

Definizione (Attributo)**.** A ciascun dominio della relazione si associa un nome, chiamata **attributo**.

Definizione (Tupla)**.** Una **tupla** su un insieme di *attributi* chiamato X, è una funzione t che associa a ciascun *attributo* un valore del suo *dominio*.

Una *relazione su* X è un insieme di *tuple* su X.

3.2 Tabelle

Una **tabella** rappresenta una relazione se:

- I valori di ogni *colonna* sono fra loro *omogenei*.
- Le *righe* sono tutte *diverse* fra loro.
- Le *intestazioni* delle colonne sono tutte *diverse* fra loro.

In una *tabella* l'ordinamento tra le righe e le colonne è irrilevante.
In ogni tabella è possibile distinguere due parti:

- Lo **schema** è rappresentato dalle intestazioni della tabella, che sono invarianti nel tempo e descrivono la struttura della tabella (**aspetto intensionale**).
- L'**istanza** sono i valori attuali presenti nella tabella, che possono cambiare nel tempo (**aspetto estensionale**).

Definizione (Tipo Ennupla)**.** Un tipo ennupla chiamato T è un insieme finito di coppie (*Attributo*, *Tipo Elementare*).

Definizione (Schema di Relazione)**.** Se T è un *tipo ennupla*, allora $R(T)$ è lo schema della relazione R.

Definizione (Schema di Database)**.** Lo schema di un database è un insieme di *schemi di relazioni* $R_i(T_i)$.

Definizione (Istanza di Relazione)**.** Un'istanza di relazione, anche chiamata **relazione** su uno schema $R(T)$ è l'insieme r di tuple di tipo T.

Definizione (Istanza di Database)**.** Un'istanza di database su uno schema $R = \{R_1(T_1), \ldots, R_n(T_n)\}$ è l'insieme delle *relazioni* $r = \{r_1, \ldots, r_n\}$, dove r_i è un'*istanza di relazione* su $R_i(T_i)$.

Valore Nullo Si indica con `NULL`, e indica l'assenza di un valore del dominio.

3.3 Vincoli di Integrità

Un **vincolo d'integrità** è una proprietà che deve essere soddisfatta da ogni singola *istanza* della relazione, in modo tale che rappresenti informazioni corrette per l'applicazione.

Il vincolo viene espresso mediante un predicato, che associa ad ogni *istanza* il valore *vero* o *falso*.

I vincoli si possono classificare in:

- Vincoli intrarelazionali: ovvero quelli che devono essere rispettati da valori della relazioni presa in considerazione, e possono essere:
  - Vincoli sul dominio, che coinvolgono un solo *attributo*.
  - Vincoli di ennupla, che esprimono condizioni sui valori di ogni ennupla, indipendentemente dalle altre ennuple.
- Vincoli interrelazionali: sono quei vincoli che devono essere rispettati da valori presenti in relazioni diverse.

3.4 Chiavi

Definizione (Superchiave)**.** Un insieme K di attributi viene chiamato **superchiave** per una relazione r, se r non contiene due ennuple distinte t_1, t_2 tali che $t_1[K] = t_2[K]$.

Definizione (Chiave)**.** K invece viene definito **chiave** per r se è una *superchiave minimale* per r, ovvero non deve contenere altre *superchiavi*.

Nota Bene Dato che una *relazione* non può contenere ennuple con valori uguali, allora per ogni *relazione* esiste sempre una superchiave rappresentata dall'insieme di tutti gli attributi su cui è definita.

Definizione (Chiave Primaria)**.** Una **chiave primaria** è una *chiave* sulla quale non sono ammessi valori nulli.

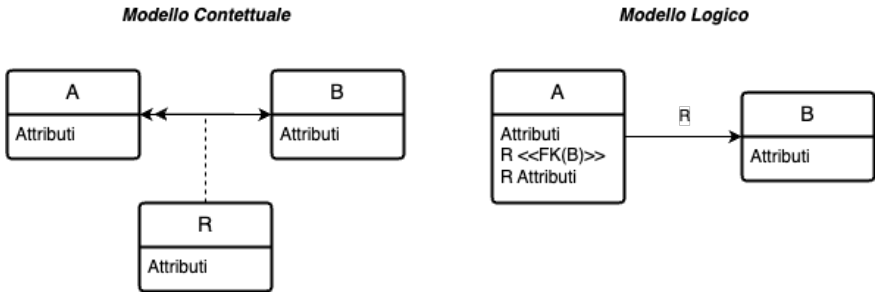
Nel modello relazionale per correlare due relazioni diversi si usano i valori delle *chiavi primarie*.

Definizione (Integrità Referenziale)**.** Un vincolo di **integrità referenziale**, anche chiamato *foreign key*, fra alcuni attributi X di una relazione R_1 e un'altra relazione R_2 impone ai valori di X di comparire come valori della *chiave primaria* di R_2.

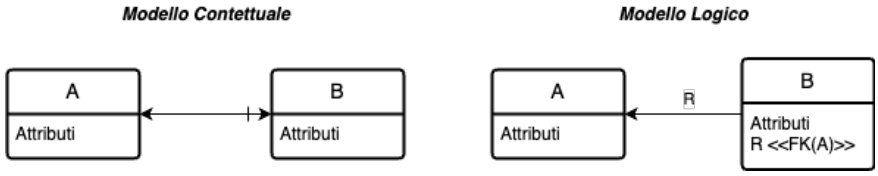
Violazioni Se per esempio si provasse ad eliminare una ennupla dalla tabella che viene riferita si verificherebbe un rifiuto dell'operazione, nel caso in cui la *chiave primaria* di quella ennupla viene riferita da altre ennuple di altre tabelle. Quindi in questo caso occorre procedere con un'*eliminazione a cascata*, ovvero si impostano prima a `NULL` i valori degli attributi delle ennuple delle tabelle che contengono riferimenti a quella *chiave primaria*; successivamente si può procedere con l'eliminazione della ennupla che adesso non verrà più riferita.

3.5 Rappresentazione delle Associazioni

Uno a Molti Le associazioni *uno a molti* si rappresentano aggiungendo, agli attributi della relazione rispetto alla quale l'associazione è univoca, una *chiave esterna* che si riferisce all'altra relazione. Nel caso in cui l'associazione ha degli attributi si aggiungono anch'essi alla relazione.

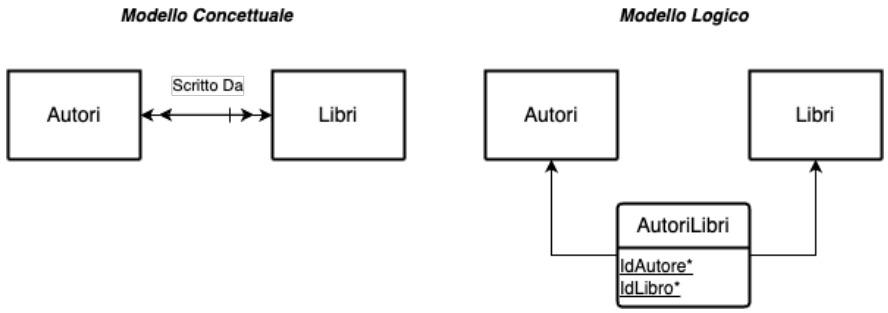


Uno a Uno In questo caso si aggiunge la *chiave esterna* scegliendo arbitrariamente una delle due relazioni ma, in caso in cui esiste un vincolo di totalità, si preferisce la relazione rispetto alla quale l'associazione è *totale*.

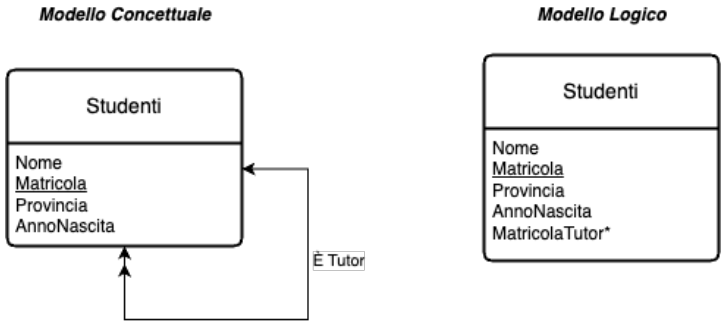


Vincoli sulle Cardinalità La direzione dell'associazione rappresentata dalla *chiave esterna* è chiamata la *diretta* dell'associazione. Per imporre un vincolo di univocità della diretta occorre definire un vincolo di chiave sulla *chiave esterna*; mentre per descrivere un vincolo di totalità della diretta si impone un vincolo `NOT NULL` sempre sulla *chiave esterna*.

Molti a Molti Un'associazione *molti a molti* si traduce aggiungendo tra le due relazioni, una terza che ha come attributi (e come *chiave primaria*) le *chiavi primarie* delle due relazioni.

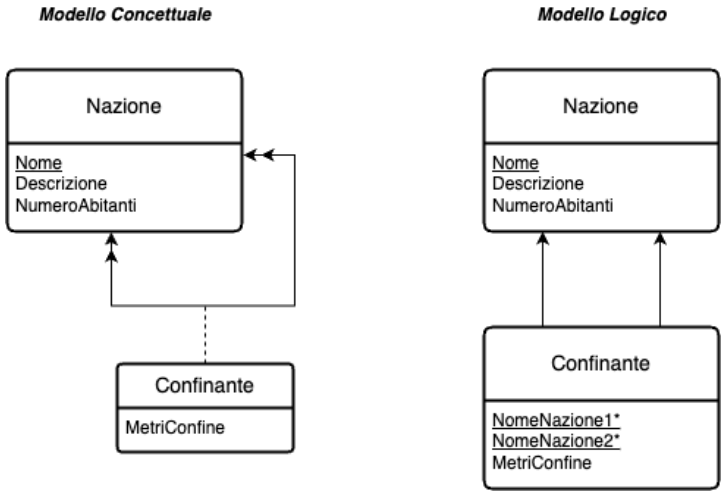


Ricorsione In questo caso la *chiave esterna* si aggiunge alla stessa e sola relazione.



Ricorsione Molti a Molti Anche qui si costruisce una seconda relazione che ha come *chiave primaria* le due *chiavi primarie* delle due istanze

coinvolte.



3.6 Rappresentazione delle Gerarchie

Le *gerarchie* non possono essere rappresentate direttamente, quindi vanno eliminate sostituendole con altre classi e relazioni.

Ci sono tre metodi:

- **Relazione Unica**: se A_0 è la classe genitore di A_1 e A_2, queste vengono eliminate e accorpate ad A_0. Ad A_0 viene aggiunto un attributo chiamato discriminatore, che indica per ogni istanza da quale classe figlia deriva. Ovviamente anche gli attributi delle classi figlie vengono assorbite dal genitore e assumono valore `NULL` sulle istanze di una classe figlia che non possedeva quegli attributi. Per quanto riguarda le relazioni, invece, anche queste vengono assorbite dalla classe genitore ma avranno comunque cardinalità minima uguale a 0, anche qui per le istanze di una classe figlia che non aveva quella relazione.

- **Partizionamento Orizzontale**: la classe genitore A_0 viene eliminata e le classi figlie A_1 e A_2 ereditano gli attributi e le relazioni del genitore. L'unico caso in cui non si può adoperare questo metodo è quando è presente un *vincolo referenziale* verso la classe genitore, in quanto non è possibile spezzare il vincolo in più relazioni diverse.

- **Partizionamento Verticale**: la gerarchia si trasforma in tante associazioni *uno a uno* che legano ogni classe figlia con la classe genitore. In questo caso vanno aggiunti dei vincoli, ovvero ogni istanza di A_0 non può partecipare a tutte le associazioni, ma ad al più una; e nel caso in cui la gerarchia è totale deve essere esattamente 1.

Nota Bene In quest'ultimo metodo la *chiave primaria* della classe genitore è sia *chiave esterna* che *chiave primaria* per le figlie.

3.7 Rappresentazione Campi Multivalore

Per la gestione dei *campi multivalore* viene creata una nuova relazione che ha come attributi il nome del campo e una *chiave esterna* che si riferisce alla relazione in cui si trovava. La *chiave primaria* di questa nuova relazione è l'intera ennupla.

4 Algebra Relazionale

Definizione (Algebra Relazionale)**.** Con **algebra relazionale** si intende un insieme di *operatori* su relazioni che danno come risultato altre relazioni. Non viene usato come linguaggio di interrogazione dei *DBMS,* ma come rappresentazione interna delle interrogazioni.

4.1 Operatori Insiemistici

Le relazioni vengono viste come degli *insiemi*. Gli operatori di *unione*, *differenza* ed *intersezione* sono applicabili solo a relazioni definite sugli stessi *attributi*.

- **Unione**: l'unione di due relazioni R e S definite sullo stesso insieme di attributi X è indicata con $R \cup S$ ed è una relazione su X che contiene le tuple che appartengono a R o a S senza ripetizioni.
- **Differenza**: la differenza è indicata con $R - S$ ed è una relazione sempre su X che contiene tutte le tuple che appartengono ad R ma non a S.
- **Ridenominazione**: è un operatore *monadico*, ovvero su una sola relazione, che permette di modificare lo schema, ridenominando il nome di uno o più attributi, ma lasciando inalterate le istanze.

$$\rho_{Nuovo\ Nome\ Attributo \leftarrow Vecchio\ Nome\ Attributo}(Relazione)$$

- **Proiezione**: è un altro operatore *monadico* che produce un risultato su un sottoinsieme degli attributi della relazione e contiene le ennuple della relazione ristrette solo agli attributi del sottoinsieme.

$$\pi_{Lista\ Attributi}(Relazione)$$

La cardinalità di una *proiezione* è basata sul sottoinsieme X degli attributi:

  - Se X è una *superchiave* di R, allora $\pi_X(R)$ contiene esattamente tante ennuple quante ne contiene R.
  - Altrimenti se X non è *superchiave*, potrebbero essere presenti dei valori che su quegli attributi sono ripetuti e che quindi vengono rappresentati una sola volta.

  Nella *proiezione* è possibile ridenominare un attributo nella $Lista\ Attributi$ scrivendo `Nome Vecchio as Nome Nuovo`.

- **Selezione** o **Restrizione**: è sempre un operatore *monadico* che produce una relazione con lo stesso schema dell'operando, che contiene un sottoinsieme delle sue ennuple che soddisfano una data condizione.

  $$\sigma_{\ Cond}\ (R)$$

  Con $Cond$ generata dalla seguente grammatica:

  ```
  Cond ::= Exp Theta Exp | Cond And Cond | Cond Or Cond | Not Cond
  Exp ::= Attributo | Costante | Exp Op Exp
  Theta ::= = | < | > | != | <= | >=
  Op ::= + | - | * | StringConcat
  ```

  Le condizioni atomiche si riferiscono solo ai valori non nulli, per riferirsi anche ai valori nulli si usano forme di condizioni come `IS NULL` e `IS NOT NULL`.

- **Prodotto**: operatore fra più relazioni che devono avere fra loro tutti i nomi degli attributi distinti, e che esegue il prodotto cartesiano fra gli insiemi di tuple.

- **Intersezione**: l'intersezione è indicata con $R \cap S$ ed è una relazione definita sullo stesso insieme di attributi X che contiene le tuple che appartengono sia ad R che ad S. L'*intersezione* è chiamato operatore *derivato*, dato che è possibile derivarlo usando altri operatori:

  $$R \cap S\ =\ \{x \mid x \in R \wedge \exists\ y \in S\ t.c.\ x = y\}$$

  allora se per esempio R ed S sono definiti con $X = \{A, B\}$.

  $$\pi_{A,B}(\sigma_{A=S.A\ AND\ B=S.B}(R \times \rho_S(S)))$$

  Partendo dall'interno, si ridenominano tutti gli attributi di S ponendogli davanti il prefisso S, successivamente viene fatto il *prodotto* con R e tramite la *selezione* selezioniamo solo le tuple che hanno gli attributi semanticamente uguali con gli stessi valori. Infine si fà una *proiezione* per eliminare gli attributi ridondanti.

4.1.1 Join

Il **join** o **giunzione** permette di correlare dati in relazioni diverse. Anch'esso è un operatore derivato in quanto si può ottenere per composizione degli operatori visti precedentemente. Si indica con il simbolo $\bowtie$.

Il **join naturale** produce un'unione degli attributi dei due operandi, combinando le ennuple degli operandi che hanno valori uguali sugli attributi in comune. Ad esempio $R_1 \bowtie R_2$ è una relazione su $X_1 \cup X_2$ definita come segue:

$$R_1 \bowtie R_2 = \{t \; su \; X_1 \cup X_2 \mid esistono \; t_1 \in R_1 \; e \; t_2 \in R_2 \\ con \; t[X_1] = t_1 \; e \; t[X_2] = t_2\}$$

Un *join* viene chiamato *completo* se ogni ennupla delle due relazioni contribuisce al risultato. Al contrario si parla di *join non completo*. Generalmente possiamo definire un limite inferiore e superiore alla cardinalità di una join:

$$0 \leq |R_1 \bowtie R_2| \leq |R_1| \times |R_2|$$

Nel caso il join è *completo* possiamo restringere il limite inferiore dicendo che $|R_1 \bowtie R_2| \geq \max\{|R_1|, |R_2|\}$.

Nel caso in cui il join coinvolge come attributo comune una *chiave* di R_2, allora:

$$0 \leq |R_1 \bowtie R_2| \leq |R_1|$$

Se oltre a coinvolgere una *chiave* di R_2, vi è un vincolo di *integrità referenziale* tra un attributo di R_1 e la chiave di R_2 allora la cardinalità della join è esattamente $|R_1|$.

Un **join esterno** estende con valori nulli le ennuple che verrebbero tagliate fuori da un *join naturale*. Esiste in tre versioni:

- Join esterno sinistro $R \overset{\leftarrow}{\bowtie} S$: mantiene tutte le ennuple del primo operando, estendendole con valori nulli se necessario.
- Join esterno destro $R \overset{\rightarrow}{\bowtie} S$: mantiene tutte le ennuple del secondo operando, estendendole con valori nulli se necessario.
- Join esterno completo $R \overset{\leftrightarrow}{\bowtie} S$: mantiene tutte le ennuple di entrambi gli operandi, estendendole con valori nulli se necessario.

Nota Bene Il *prodotto cartesiano* si può vedere come un *join naturale* su relazioni che non hanno attributi in comune.

Dato che il *prodotto cartesiano* non ha senso se non lo facciamo seguire da una selezione (che permette di specificare la condizione sui campi che semanticamente rappresentano lo stesso attributo), si usa l'operazione di **theta-join** definito come segue:

$$R_1 \bowtie_{Condizione} R_2$$

che è equivalente a:

$$\sigma_{Condizione} (R_1 \bowtie R_2)$$

N.B.: dato che R_1 ed R_2 non hanno attributi comuni, in questo caso $\bowtie = \times$.

Se nel *theta-join* l'operatore di confronto nella $Condizione$ è sempre l'uguaglianza, allora si parla di **equi-join**.

Self Join Supponiamo di avere questa relazione `Genitori`:

Genitore	Figlio
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

e di voler ottenere una relazione `Nonno-Nipoti`. Per far ciò occorre riutilizzare la stessa tabella ma facendo $Genitori \bowtie Genitori$ si otterrebbe di nuovo la tabella `Genitori`. In questo caso occorre ridenominare la tabella due volte:

Nonno	Genitore
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

$\rho_{Nonno,\, Genitore \leftarrow Genitore,\, Figlio}(Genitore)$

Genitore	Nipote
Luca	Anna
Maria	Anna
Giorgio	Luca
Silvia	Maria
Enzo	Maria

$\rho_{Nipote \leftarrow Figlio}(Genitore)$

Infine, effettuando una *join* seguita da una *proiezione* otteniamo:

Nonno	Nipote
Giorgio	Anna
Silvia	Anna
Enzo	Anna

$$\pi_{Nonno,\, Nipote}(\rho_{Nonno,\, Genitore \leftarrow Genitore,\, Figlio}(Genitore) \bowtie \rho_{Nipote \leftarrow Figlio}(Genitore))$$

4.2 Raggruppamento

Il **raggruppamento** è definito dall'espressione ${}_{\{A_i\}}\gamma_{\{f_i\}}(R)$, dove gli A_i sono gli attributi di R mentre le f_i sono funzioni di aggregazione. Il *raggruppamento* prima partiziona le ennuple di R mettendo nello stesso gruppo le ennuple con valori uguali solo sui campi degli A_i, successivamente si calcolano le funzioni f_i per ogni partizione. Infine per ogni partizione verrà restituita una solo ennupla che avrà come attributi i valori degli A_i e i valori delle funzioni f_i.

4.3 Trasformazioni Algebriche

Permettono di scegliere diversi ordini di *join* e di anticipare *proiezioni* e *selezioni* per lavorare su tabelle più piccole.

Idempotenza delle Proiezioni

$$\pi_A(\pi_{A,B}(R)) \equiv \pi_A(R) \tag{1}$$

Atomizzazione delle Selezioni

$$\sigma_{Cond_1}(\sigma_{Cond_2}(R)) \equiv \sigma_{Cond_1 \wedge Cond_2}(R) \tag{2}$$

Atomizzazione della Selezione rispetto al Join
Pushing Selection Down

$$\sigma_{Cond}(R \bowtie S) = R \bowtie \sigma_{Cond}(S) \tag{3}$$

Se $Cond$ fà riferimento solo agli attributi di S.

Anticipazione della Proiezione rispetto al Join
Pushing Projections Down

$$\pi_{X_1 Y_2}(R \bowtie S) = R \bowtie \pi_{Y_2}(S) \tag{4}$$

Se R è definito su X_1 e S su X_2, $Y_2 \subseteq X_2$ e gli attributi in $X_2 - Y_2$ non sono coinvolti nel join.

Distributività della Selezione

$$\sigma(R \cup S) = \sigma(R) \cup \sigma(S) \tag{5}$$

$$\sigma(R - S) = \sigma(R) - \sigma(S) \tag{6}$$

Distributività della Proiezione

$$\pi_X(R \cup S) = \pi_X(R) \cup \pi_X(S) \tag{7}$$

Nota Bene La *proiezione* sulla *differenza* non gode della proprietà di *distributività*:

$$\pi_X(R - S) <> \pi_X(R) - \pi_X(S) \tag{8}$$

Inglobamento della Selezione nella Join

$$\sigma_{Cond}(R \bowtie S) \equiv R \bowtie_{Cond} S \tag{9}$$

$$\sigma_{Cond_1 \wedge Cond_2}(R \times S) \equiv \sigma_{Cond_1}(R) \times \sigma_{Cond_2}(S) \tag{10}$$

$$\sigma_{Cond_1 \vee Cond_2}(R) \equiv \sigma_{Cond_1}(R) \cup \sigma_{Cond_2}(R) \tag{11}$$

$$\sigma_{Cond_1 \wedge Cond_2}(R \times S) \equiv \sigma_{Cond_1}(R) \cap \sigma_{Cond_2}(R) \tag{12}$$

$$\sigma_{Cond_1 \wedge \neg Cond_2}(R \times S) \equiv \sigma_{Cond_1}(R) - \sigma_{Cond_2}(R) \tag{13}$$

$$R \times (S \times T) \equiv (R \times S) \times T \tag{14}$$

$$(R \times S) \equiv (S \times R) \tag{15}$$

$$\sigma_{Cond}({}_X\gamma_F(R)) \equiv {}_X\gamma_F(\sigma_{Cond}(R)) \tag{16}$$

4.4 Operatori Non Insiemistici

Proiezione MultiInsiemistica $\pi^b_{A_i}(R)$, la b sta ad indicare che le tuple duplicate non vanno eliminate.

Ordinamento $\tau_{A_i}(R)$, ordina i valori degli attributi di ogni tupla seguendo l'ordine degli A_i.

5 SQL

Le interrogazioni al database vengono fatte mediante il comando `SELECT`, nel quale occorre specificare la lista degli attributi interessati. Il comando presenta anche due clausole, `FROM` che indica in quali tabelle sono contenuti i dati, e `WHERE` (*opzionale*) per esprimere le condizioni che i dati devono soddisfare.

```
1 SELECT Lista Attributi
2 FROM Lista Tabelle
3 [WHERE Condizione]
```

Specificare la *Lista Attributi*, anche chiamata **target list** rappresenta l'operazione di *proiezione*. La clausola `WHERE`, invece permette di implementare la *selezione*.

Il **theta-join**, invece è implementato indicando nella clausola `FROM` le due tabelle, e nel `WHERE` la condizione che stabilisce quali righe accoppiare.

```
1 SELECT *
2 FROM Studenti, Esami
3 WHERE Studenti.Matricola = Esami.Studenti
```

Nel caso in cui la clausola `WHERE` non è presente, la query sopra indicata esegue il prodotto cartesiano anche chiamato *cross-join*.

DISTINCT *SQL* è un linguaggio che lavora su *multiinsiemi*, quindi di *default*, a seguito di una *query*, potrebbero essere presenti righe identiche. Per evitare ciò basta specificare questo facendo seguire il comando `SELECT` dalla clausola `DISTINCT`.

Ridenominazione Per ridenomiare un attributo, basta farlo seguire da `AS Nuovo Nome Attributo`. Mentre per ridenomiare una tabella, basta far seguire il nome originale nella clausola `FROM` dal nuovo nome.

JOIN L'operazione di *Join*, oltre che essere implementato come visto prima tramite la combinazione di `FROM` e `WHERE`, è possibile codificarla in modo esplicito, ecco alcuni esempi:

- `Studenti s JOIN Esami e ON s.Matricola = e.Matricola`
- `Studenti s JOIN Esami e USING Matricola`
- `Studenti s NATURAL JOIN Esami`
- `Studenti s LEFT [OUTER] JOIN Esami e ON s.Matricola = e.Matricola`

Self-Join Per implementare il *self-join* occorre prima ridenominare la tabella due volte:

```
1 SELECT T1.*, T2.*
2 FROM Tabella T1, Tabella T2
3 WHERE T1.codice = T2.codice
```

LIKE Questo operatore è usato per effettuare dei *pattern matching* con una stringa tramite l'uso di due *wildcard*:

- **%** : che denota la presenza di 0 o più caratteri.
- _ : presenza di esattamente 1 carattere.

Nel caso si voglione usare le *wildcards* nel *patter matching*, si imposta un carattere di *escape*:

```
1 SELECT *
2 FROM Modelli
3 WHERE nome_modello LIKE 'C#_F%' ESCAPE '#'
```

NULL Quando si confronta un attributo con `NULL` è sempre consigliabile farlo tramite gli operatori `IS [NOT] NULL`, dato che il confronto tramite l'uguaglianza, nel caso in cui il valore dell'attributo sia `NULL`, restituisce *valore sconosciuto*, ovvero `NULL`, e non un valore *booleano*. Nel calcolo di espressioni, inoltre, se si incontra un valore `NULL`, viene sempre restituito *valore sconosciuto*.

ORDER BY Permette di dare un ordinamento del risultato della `SELECT`, basandosi sui valori di uno o più attributi e opzionalmente indicando in maniera esplicita se l'ordine deve essere crescente (`ASC`, predefinito) o decrescente (`DESC`).

5.1 Operatori Aggregati

Nella *target list* possiamo avere anche espressioni che calcolano valori a partire da un insieme di ennuple, e che restituiscono un singolo valore scalare. *SQL* prevede 5 operatori aggregati:

- **Count**: restituisce il numero di righe del risultato della *query*. Mediante la specifica `(*)` si contano tutte le righe selezionate, con `ALL` (specifica di *default*) si contano tutte le righe selezionate che non sono `NULL`, mentre con `DISTINCT` si contano tutte le righe non nulle con valori distinti.
- **Max & Min**: calcolano rispettivamente il massimo e il minimo degli elementi presenti in un colonna dello schema.
- **Avg**: calcola la media dei valori non nulli su una colonna, si possono utilizzare le specifiche `ALL` e `DISTINCT`.
- **Sum**: somma tutti i valori non nulli su una colonna, anche qui si trovano le specifiche `ALL` e `DISTINCT`.

GROUP BY Se si vogliono applicare gli operatori aggregati non sull'intera tabella ma raggruppando i valori per un sottoinsieme di attributi si utilizza la clausola `GROUP BY`, che specifica gli attributi su cui fare i raggruppamenti. Le funzioni di aggregazione saranno applicate su ogni gruppo.

HAVING Tramite questa clausola si possono applicare condizioni sui valori aggregati per ogni gruppo.

SottoSelect o SubQuery È possibile innestare una `SELECT` in un'altra, tramite le parole chiave `ANY`, `ALL`, `[NOT] IN`, `[NOT] EXISTS`, più tutte gli altri operatori della clausola `WHERE`, seguita dalla *sottoselect*.

```
SELECT *
FROM Studenti
WHERE voto > (SELECT AVG(voto)
              FROM Studenti)
```

Oppure è possibile fare operazioni insiemistiche tra i risultati di due `SELECT` tramite gli operatori `UNION`, `INTERSECT`, `EXCEPT`.

Ci sono tre tipologie di *subquery*:

- Subquery Scalari: una `SELECT` che restituisce un solo valore.
- Subquery di Colonna: restituisce una colonna.
- Subquery di Tabella: restituisce una tabella.

Nelle query nidificate le *regole di visibilità* sono simili a quelle dei linguaggi di programmazione, ovvero all'esterno non è possibile riferirsi a variabili definite in *query* più interne o nello stesso livello, viceversa sì.

NOTA Nelle *subquery* non è possibile utilizzare le clausole `HAVING` e `GROUP BY`.

Quantificazione È importante ricordarsi due *"trasformazioni"* molto utili per la quantificazione:

- L'universale negata è uguale all'esistenziale: *non tutti i voti sono* $\leq$ *24* vale a dire *almeno un voto* $>$ *24*.
- L'esistenziale negata è uguale all'universale: *non esiste un voto diverso da 30* vale a dire *tutti i voti sono uguali a 30*.

Quando si utilizza nel `WHERE` un operatore scalare seguito da `ANY` più una *subquery*, il predicato sarà vero solo se almeno uno dei valori restituiti dalla *subquery* lo soddisfa. Mentre se si usa `ALL`, tutti i valori restituiti devono soddisfarlo.

La keyword `EXISTS` invece rappresenta un quantificatore esistenziale, ciò rende vero il predicato se la *subquery* che lo segue restituisce almeno una tupla.

È bene notare che utilizzare `IN` equivale alla forma `= ANY`.