

# Appunti di Basi di Dati

Luca Facchini

Matricola: 245965

Corso tenuto dal prof. Bouquet Paolo

Università degli Studi di Trento

A.A. 2024/2025

## **Sommario**

Questo documento contiene gli appunti del corso di Basi di Dati tenuto dal prof. Bouquet Paolo presso l'Università degli Studi di Trento nell'anno accademico 2024/2025.

# Indice

<b>1</b>	<b>Il Modello Relazionale</b>	<b>3</b>
1.1	Concetti Generali . . . . .	3
1.2	Relazione . . . . .	3
1.2.1	Definizione intensionale . . . . .	3
1.2.2	Definizione estensionale . . . . .	3
1.2.3	Stato di una relazione . . . . .	4
1.2.4	Chiave di una relazione . . . . .	4
1.3	Vincoli . . . . .	4
1.3.1	Definizioni . . . . .	4
1.3.2	Vincolo di chiave . . . . .	5
1.3.3	Vincolo di integrità delle entità . . . . .	5
1.3.4	Vincoli di integrità referenziale . . . . .	5
1.3.5	Altri vincoli . . . . .	5
1.4	Stato e Operazione di una base di dati . . . . .	5
1.4.1	Stato di una base di dati . . . . .	5
1.4.2	Operazioni di base . . . . .	6
<b>2</b>	<b>Algebra relazionale</b>	<b>7</b>
2.1	Introduzione . . . . .	7
2.1.1	Perché algebra relazionale . . . . .	7
2.1.2	Concetti generali . . . . .	7
2.2	Operazioni dell'AR . . . . .	7
2.2.1	Operazione Unarie . . . . .	8
2.2.2	Operazioni Insiemistiche . . . . .	9
2.2.3	Operazioni Binarie . . . . .	9
2.3	Query Tree . . . . .	10
<b>3</b>	<b>SQL - <i>Structured Query Language</i></b>	<b>12</b>
3.1	Definizioni . . . . .	12
3.2	Struttura Generale delle <i>query</i> SQL . . . . .	12
3.2.1	Comando <b>SELECT</b> - interrogazione dei dati . . . . .	12

# Capitolo 1

## Il Modello Relazionale

### 1.1 Concetti Generali

**La Storia** Il modello fu proposto da Edgar F. Codd nel 1970 nel suo articolo: "A Relational Model of Data for Large Shared Data Banks".

**Il Modello** Il modello relazionale è un modello di dati che si basa sul concetto matematico di relazione.

### 1.2 Relazione

**Definizione** *Informalmente*, una **relazione** può essere vista come una **tabella** con un insieme di valori su ogni riga.

Ci sono due livelli che definiscono una relazione:

- Lo **schema** della relazione (livello *intensionale*)
- **Istanze** della relazione (livello *estensionale*)

#### 1.2.1 Definizione intensionale

Lo schema di una relazione definisce:

- Il nome della relazione
- il nome di ogni attributo
- il dominio di ogni attributo

Si nota come l'ordine degli attributi non sia rilevante. Uno dei modi "standard" di rappresentare una relazione è il seguente: `Students(sid: string, name: string, login: string, age: integer, gpa: real)`.

**Grado** Il **grado** di una relazione è il numero di attributi che la compongono (nel caso dell'esempio, il grado è 5).

#### 1.2.2 Definizione estensionale

Un'istanza di uno schema di relazione è un insieme di **tuple** (o **record**), ognuna delle quali ha lo stesso numero di campi dello schema della relazione. Questo comporta:

- Essendo un insieme **non** possono esserci duplicati
- L'ordine delle tuple non è rilevante

**Cardinalità** La **cardinalità** di una relazione è il numero di tuple che contiene.

### 1.2.3 Stato di una relazione

Lo **stato di una relazione** è un sottoinsieme del prodotto cartesiano dei domini dei suoi attributi:

Data una relazione:  $R(A_1, A_2, \dots, A_n)$ , lo stato di  $R$  è definito come:

$$r(R) \subset \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$$

ovvero:

$$\begin{aligned} r(R) &= \{t_1, t_2, \dots, t_n\} && \text{dove ogni } t_i \text{ è una tupla} \\ t_i &= \langle v_1, v_2, \dots, v_n \rangle && \text{dove } v_i \text{ è un elemento dell'attributo } \text{dom}(A_j) \end{aligned}$$

### 1.2.4 Chiave di una relazione

Ogni riga di una relazione ha un campo (o un insieme di campi) il cui valore (o i cui valori) identifica univocamente quella la riga in quella tabella. Questo campo (o insieme di campi) è detto **chiave** della relazione.

Talvolta si usano valori convenzionali per identificare una riga in una tabella. Si parla di *chiave artificiale* (*artificial key*) o *chiavi surrogate* (*surrogate key*).

## 1.3 Vincoli

**Definizione** I vincoli determinano quali stati di una relazione in una base di dati relazionale sono ammissibili e quali non lo sono.

Esistono tre tipi di vincoli:

1. **Vincoli Impliciti:** dipendono dal *data model* stesso.
2. **Vincoli basati sullo schema** (o **espliciti**): sono definiti nello schema usando gli strumenti forniti dal modello **ER**.
3. **Vincoli applicativi o semantici:** si trattano di vincoli che vanno al di là del modello e devono essere imposti a livello di programma

Un vincolo è definito come una **condizione** che DEVE valere affinché lo **stato** di una relazione sia **valido**. I principali tipi di vincoli espliciti che possono essere espressi nel modello relazionale sono:

- Vincolo di **dominio**
- Vincolo di **chiave**
- Vincolo di **integrità delle entità**
- Vincolo di **integrità referenziale**

### 1.3.1 Definizioni

**Superchiave** di una relazione  $R$ : è un insieme di attributi  $S_K$  di  $R$  tale che:

- Non esistano due tuple di  $r(R)$  in cui gli attributi di  $S_K$  abbiano lo stesso valore
- Questa condizione deve essere rispettata in *ogni stato valido* di  $R$

**Chiave** di una relazione  $R$ : è una superchiave minimale, ovvero una superchiave tale che la rimozione di qualsiasi attributo produrrebbe un insieme di attributi che non è più una superchiave di  $R$ .

Ogni chiave minimale è detta anche **chiave candidata**

**Una chiave è sempre una superchiave ma non viceversa**

### 1.3.2 Vincolo di chiave

Se una relazione ha più di una **chiave candidata** allora ne viene scelta una come **chiave primaria** (in generale quella più piccola per numero di attributi)

I valori della chiave primaria sono usati per *identificare* in modo *univoco* le tuple di una relazione.

### 1.3.3 Vincolo di integrità delle entità

Nessuno degli attributi che compongono la chiave primaria  $P_K$  di una relazione  $R$  può avere valore NULL in una tupla di  $r(R)$ .

### 1.3.4 Vincoli di integrità referenziale

Il vincolo di integrità referenziale coinvolge più relazioni:

- una **relazione referenziante**  $R_1$  che contiene un riferimento alla chiave primaria di un'altra relazione
- una **relazione referenziata**  $R_2$  che contiene la chiave primaria referenziata

Inoltre una tupla  $t_1$  in  $R_1$  si dice che **referenzia** una tupla  $t_2$  in  $R_2$  se  $t_1[FK] = t_2[PK]$ .

I valori degli attributi della chiave esterna  $FK$  della **relazione referenziale**  $R_1$  possono essere:

1. Uno dei valori del corrispondente attributo della chiave primaria  $PK$  in  $R_2$
2. Assumere il valore NULL

Se NULL la  $FK$  in  $R_1$  non deve far parte degli attributi della chiave primaria di  $R_2$ .

**Osservazione 1** La  $FK$  può fare riferimento alla stessa relazione di appartenenza della  $PK$ .

**Osservazione 2** Gli attributi alla  $FK$  non necessariamente devono avere lo stesso nome degli attributi della  $PK$ .

### 1.3.5 Altri vincoli

#### Vincoli di integrità semantica

I vincoli di integrità semantica sono vincoli che vanno al di là del modello relazionale e devono essere imposti a livello di programma.

## 1.4 Stato e Operazione di una base di dati

### 1.4.1 Stato di una base di dati

#### Base di dati relazionale

Ogni *relazione* è tipicamente popolata da un insieme di tuple. Uno **stato di una base di dati relazionale** con schema  $S$  è un sottoinsieme di stati delle relazioni  $\{r_1, r_2, \dots, r_n\}$  tali che ogni  $r_i$  è uno stato di  $R_i$  e che  $r_i$  soddisfa i vincoli di integrità relazionale in  $IC$ .

Uno **stato** che **non** soddisfa i vincoli di integrità è detto **stato non valido**.

#### Base di dati popolata

Lo *stato della base di dati* è l'unione di tutti i singoli stati delle relazioni che compongono la base di dati, ogni volta che questa è modificata si passa ad un nuovo stato.

### 1.4.2 Operazioni di base

Le operazioni di base che possono essere eseguite su una base di dati relazionale sono:

**INSERT** Inserisce una nuova tupla in una relazione

**DELETE** Rimuove una tupla da una relazione

**MODIFY** Modifica il valore di uno o più attributi di una tupla

**Importante:** queste operazioni non devono portare alla violazione di alcun vincolo di integrità, per garantire questa condizione può essere necessario **propagare automaticamente** gli aggiornamenti

#### Operazioni con vincoli di integrità

**RESTRICT** o anche (NO ACTION, REJECT) impedisce l'operazione se viola un vincolo di integrità

**CASCADE, SET NULL, SET DEFAULT** Modificano automaticamente i valori delle chiavi esterne in modo da mantenere l'integrità referenziale

**routine** Esegue una procedura specifica per gestire l'errore

#### Violazioni per l'operazioni INSERT

- Violazione del vincolo di dominio - il valore inserito non è nel dominio dell'attributo
- Violazione del vincolo di chiave - il valore inserito è duplicato rispetto alla chiave
- Violazione del vincolo di integrità referenziale - il valore inserito non è presente nella chiave referenziata
- Violazione del vincolo di integrità delle entità - il valore della chiave primaria è NULL

#### Violazioni per l'operazioni DELETE

- Violazione del vincolo di integrità referenziale - esistono tuple in altre relazioni che fanno riferimento alla tupla che si vuole eliminare

#### Violazioni per l'operazioni UPDATE

- UPDATE della chiave primaria  $\Rightarrow$  possibile violazione del vincolo di integrità referenziale
- UPDATE di una chiave esterna  $\Rightarrow$  possibile violazione del vincolo di integrità referenziale
- UPDATE di un attributo che fa parte di una chiave  $\Rightarrow$  possibile violazione del vincolo di dominio o vincoli UNIQUE e NOT NULL

#### Come preservare l'integrità referenziale

- **CASCADE** - cancella tutte le tuple che referenziavano la chiave primaria della tupla cancellata o modificata
- **SET NULL** - Imposta a NULL i valori delle chiavi esterne che referenziano la chiave primaria della tupla cancellata o modificata - NON può essere fatto se la chiave esterna è parte della chiave primaria
- **SET DEFAULT** - assegnare un valore di default alle chiavi esterne che referenziano la chiave primaria della tupla cancellata o modificata - NON può essere fatto se la chiave esterna è parte della chiave primaria
- **RESTRICT** - Impedisce l'operazione se viola un vincolo di integrità

# Capitolo 2

## Algebra relazionale

### 2.1 Introduzione

**Linguaggio di interrogazione** Un **linguaggio di interrogazione** (*query language*) per il modello relazionale specializzato per manipolare (tipicamente estrarre) dati di una base di dati relazionali. I linguaggi possono essere suddivisi in:

**Procedurali** Il programmatore descrive passo passo come ottenere il risultato.

**Dichiarativi** Il programmatore descrive il risultato che vuole ottenere, senza specificare come ottenerlo.

L'Algebra relazionale è un linguaggio dichiarativo.

#### 2.1.1 Perché algebra relazionale

- **Fondamenta teoriche per le operazioni sui dati**
- **Ottimizzazione delle query**
- **Portabilità tra DBMS**
- **Comprensione degli algoritmi di esecuzione delle query**

#### 2.1.2 Concetti generali

**Input Output** Nella AR input e output sono *relazioni* (nel senso del modello relazionale).

**Risultato** il risultato di un'operazione è una *nuova relazione*, che viene generata a partire da una o più relazioni di *input*.

**Algebra Chiusa** Effetto diretto della definizione di input e output come relazioni è che l'algebra relazionale è *chiusa*, ovvero il risultato di un'operazione è sempre una relazione.

### 2.2 Operazioni dell'AR

**Operazioni unarie**

- **SELECT** (simbolo  $\sigma$ )
- **PROJECT** (simbolo  $\pi$ )
- **RENAME** (simbolo  $\rho$ )

**Operazioni insiemistiche di AR**

- **UNIONE** (simbolo  $\cup$ )
- **INTERSEZIONE** (simbolo  $\cap$ )
- **DIFFERENZA** (simbolo  $\circ$ )
- **PRODOTTO CARTESIANO** (simbolo  $\times$ )

**Operazioni binarie**

- **JOIN** (simbolo  $\bowtie$ )
- **DIVISIONE** (simbolo  $\div$ )

**Altre Operazioni**

- **OUTER JOINS; OUTER UNION**
- **AGGREGAZIONE**

**2.2.1 Operazione Unarie****Selezione**

**Proprietà** La forma generale dell'operazione *select* è:

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

- $\sigma$  è l'operatore di selezione
- $\theta$  è il predicato di selezione o condizione
- $R$  è la relazione di input

**Commutativo** L'operazione di selezione è commutativa, ovvero:

$$\sigma_{\theta_1}(\sigma_{\theta_2}(R)) = \sigma_{\theta_2}(\sigma_{\theta_1}(R))$$

Questa però può essere semplificata in:

$$\sigma_{\theta_1 \wedge \theta_2}(R)$$

**Proiezione**

**Proprietà** La forma generale dell'operazione *project* è:

$$\pi_{A_1, A_2, \dots, A_n}(R)$$

- $\pi$  è l'operatore di proiezione
- $A_1, A_2, \dots, A_n$  è la lista degli attributi di  $R$  che si vogliono mantenere

**Duplicati** L'operazione di proiezione rimuove le tuple duplicate in quanto il risultato è un insieme di tuple.

**Risultato** Il numero di tuple in  $\pi_{A_1, A_2, \dots, A_n}(R)$  è minore o uguale al numero di tuple in  $R$ . Inoltre se  $A_1, A_2, \dots, A_n$  include una *chiave* di  $R$ , allora il numero di tuple restituite da PROJECT sarà sempre *uguale* al numero di tuple in  $R$ .



**non commutativo** L'operazione di proiezione non è commutativa, ovvero:

$$\pi_{A_1, A_2, \dots, A_n}(\pi_{B_1, B_2, \dots, B_m}(R)) \neq \pi_{B_1, B_2, \dots, B_m}(\pi_{A_1, A_2, \dots, A_n}(R))$$

sono uguali  $\Leftrightarrow A_1, A_2, \dots, A_n$  e  $B_1, B_2, \dots, B_m$  sono uguali.

### Rename

L'operatori di ridenominazione ha la seguente forma:  $\rho(R(A_1, \dots, A_n), E)$  dove:

- $E$  è una qualunque espressione algebrica
- $R$  è una nuova relazione che ha le stesse tuple di  $E$  ma con alcuni attributi rinominati
- $A_1, \dots, A_n$  è la lista di ridenominazione e contiene espressioni nella forma *vecchioNome*  $\rightarrow$  *nuovoNome*

Se il nome degli attributi non viene modificato si può omettere la lista di ridenominazione.

## 2.2.2 Operazioni Insiemistiche

**Criteri** Per tutte le operazioni insiemistiche valgono i seguenti criteri:

1.  $R$  e  $S$  devono avere lo stesso numero di attributi (se diverso non rispetta il significato di insieme)
2. Gli attributi di  $R$  e  $S$  devono avere lo stesso dominio o dominio compatibile
3. Se i nomi degli attributi sono diversi, bisognerà rinominare in output il nome degli attributi

In sostanza le operazioni insiemistiche sono definite solo per relazioni "compatibili".

### Unione

$(R \cup S)$ : è la relazione che include tutte le tuple che sono in  $R$  o in  $S$  o in entrambe.

### Intersezione

$(R \cap S)$ : è la relazione che include tutte le tuple che sono in  $R$  e in  $S$ .

### Differenza

$(R - S)$ : è la relazione che include tutte le tuple che sono in  $R$  ma non in  $S$ . Questo

### Prodotto Cartesiano

$(R \times S)$ : è la relazione che ha come schema l'unione degli attributi di  $R$  e  $S$  e una tupla  $\langle r, s \rangle$  (la concatenazione di  $r$  e  $s$ ) per ogni coppia di tuple  $r \in R$  e  $s \in S$ . Il grado della relazione risultante è la somma dei gradi delle relazioni di input, mentre la cardinalità è il prodotto delle cardinalità delle relazioni di input.

$$R(A_1, A_2, \dots, A_n) \times S(B_1, B_2, \dots, B_m) = Q(A_1, A_2, \dots, A_n, B_1, B_2, \dots, B_m)$$

Notiamo come il grado di  $Q$  sia  $n + m$  e la cardinalità sia  $|R| \times |S|$ .

## 2.2.3 Operazioni Binarie

### Join

L'operazione di JOIN di due relazioni  $R$  e  $S$  ( $R \bowtie_c S$ ) è definita come **prodotto cartesiano** seguito da una **selezione**:

$$R \bowtie_c S = \sigma_\theta(R \times S)$$

Il risultato è l'insieme delle combinazioni di tuple di  $R$  e  $S$  che soddisfano il predicato  $c$ .

**JOIN commutativo e associativo** L'operazione di JOIN è commutativa e associativa nei casi di EQUI JOIN (join con condizione di uguaglianza).

**Cardinalità** La cardinalità del risultato di un JOIN è al massimo il prodotto delle cardinalità delle relazioni di input (caso nel quale tutte le tuple soddisfano il predicato), e al minimo 0 (caso in cui nessuna tupla soddisfa il predicato).

**Equi Join** Un EQUI JOIN è un JOIN in cui il predicato è una condizione di uguaglianza tra attributi delle relazioni di input.

**Natural Join** Un NATURAL JOIN è un JOIN in cui il predicato è l'uguaglianza di **tutti** gli attributi con lo stesso nome. In caso di attributi con nome diverso ma vogliamo comunque fare un NATURAL JOIN dobbiamo prima eseguire un RENAME (almeno su una delle due relazioni).

**Outer Join** Un OUTER JOIN è un JOIN nel quale vengono mantenute le tuple che **non** soddisfano il predicato. Si distinguono tre tipi di OUTER JOIN:

**Left Outer Join** ( $R \bowtie_c S$ ): mantiene tutte le tuple di  $R$  sia che soddisfino il predicato che no. Se una tupla di  $R$  non soddisfa il predicato, i valori degli attributi di  $S$  saranno NULL.

**Right Outer Join** ( $R \bowtie_c S$ ): mantiene tutte le tuple di  $S$  sia che soddisfino il predicato che no. Se una tupla di  $S$  non soddisfa il predicato, i valori degli attributi di  $R$  saranno NULL.

**Full Outer Join** ( $R \bowtie_c S$ ): mantiene tutte le tuple di  $R$  e  $S$  sia che soddisfino il predicato che no. Se una tupla di  $R$  o  $S$  non soddisfa il predicato, i valori degli attributi dell'altra relazione saranno NULL.

## Divisione

La DIVISIONE è una operazione che non è primitiva nell'algebra relazionale e raramente implementata dai DBMS. Risponde alla domanda: "Quali sono i valori di  $A$  per i quali valgono tutti i valori di  $B$ ?". È definita come:

$$R \div S = \{ \langle x \rangle \mid \exists \langle x, y \rangle \in R, \forall \langle y \rangle \in S \}$$

dove  $R$  e  $S$  sono relazioni e  $x$  e  $y$  sono gli attributi di  $R$  e  $S$  rispettivamente.

## 2.3 Query Tree

**Definizione** La *query tree* è una struttura dati usata per rappresentare i passi di esecuzione di una query.

**Proprietà** Le *query tree* sono standard quando si parla di stimare: il lavoro necessario per eseguire una query, la generazione dei risultati intermedi e se è possibile ottimizzare la query. In una query tree:

- Ogni nodo rappresenta un'operazione (selezione, proiezione, join, ecc...)
- Le foglie rappresentano le relazioni di partenza

**Cosa fare con il *query tree*?** Una volta che una query è stata scritta è utile costruirsi il suo *query tree* per verificare se esiste un altro piano di esecuzione che:

- produca lo stesso risultato
- abbia un costo inferiore (in termini di tempo e risorse)

**Esempi di ottimizzazioni comuni**

**Anticipazione delle selezioni** Eseguire le selezioni il prima possibile in modo da ridurre il numero di tuple da processare, questo applicando eventuali filtri prima di eseguire altre operazioni.

**Anticipazione delle proiezioni** Eseguire le proiezioni il prima possibile in modo da ridurre il numero di attributi da processare.

**Riordino dei join** Cambiare l'ordine dei join in modo da ridurre il numero di tuple intermedie tra un join e l'altro, o per ridurre il numero di tuple da processare successivamente.

## Capitolo 3

# SQL - *Structured Query Language*

### 3.1 Definizioni

SQL o *Structured Query Language* è un linguaggio per la gestione di basi di dati relazionali. Questo è usato per la creazione, modifica e interrogazione dei dati. Oggi questo è standardizzato ma ne esistono diversi "dialetti" a seconda del DBMS utilizzato.

**DDL - *Data Definition Language*** Per DDL o *Data Definition Language* è il sottoinsieme di comandi (o più precisamente *query*) che permettono di definire la struttura di un database

**DML - *Data Manipulation Language*** Per DML o *Data Manipulation Language* intendiamo il sottoinsieme *query* che permettono di:

- Inserire dati
- Aggiornare dati
- Cancellare dati
- Ottenere dati

### 3.2 Struttura Generale delle *query* SQL

#### 3.2.1 Comando SELECT - interrogazione dei dati

Il comando **SELECT** è il comando principale per l'interrogazione dei dati. La sua struttura generale è la seguente:

```
SELECT [DISTINCT] <lista di attributi>  
FROM <lista di relazioni>  
WHERE <condizione>
```

<**lista di relazioni**> è la lista dei nomi delle relazioni da cui si vogliono estrarre i dati. Queste possono anche definire solo una porzione di una tabella (tramite variabili di range)

<**lista di attributi**> è la lista degli attributi che si vogliono estrarre. Questi possono essere anche espressioni aritmetiche o funzioni

<**condizione**> è un insieme di condizioni con predicati di confronto: ( $<$ ,  $>$ ,  $=$ ,  $\leq$ ,  $\geq$ ,  $\neq$ ). Questi predicati possono essere combinati con gli operatori logici **AND**, **OR**, **NOT**

**DISTINCT** è un modificatore opzionale che permette di eliminare i duplicati, di default NON vengono eliminati

### Esecuzione di una *query* SELECT

Questo genere di *query* viene eseguito seguendo i seguenti passaggi:

1. Si calcola il prodotto cartesiano delle relazioni coinvolte in <lista di relazioni>
2. Si prendono ora solo le tuple che soddisfano la o le condizioni specificate in <condizione> (se presenti)
3. Si proiettano ora solo gli attributi specificati in <lista di attributi> rimuovendo gli attributi non specificati
4. Se presente il modificatore **DISTINCT** si eliminano i duplicati

### Operatori di Confronto

Gli operatori di confronto usabili nella clausola **WHERE** del comando **SELECT** possono essere differenti per i diversi tipi di dati e DBMS. I principali operatori sono:

1. Operatori aritmetici ( $=, \neq, <, \leq, >, \geq$ )
2. **BETWEEN** - per verificare se un valore è compreso in un intervallo (chiuso)
3. **[IN]** - per verificare se un valore è contenuto in una lista di valori
4. **LIKE** - per verificare se un valore è simile ad una stringa contenente caratteri jolly (% e -)
5. **IS [NOT] NULL** - per verificare se un valore è **NULL** o meno. In quanto **NULL** è uno stato particolare che può assumere un attributo scrivere **WHERE attributo = NULL** non ha senso ed è sbagliato, si deve usare **IS NULL**

### Operatori di aggregazione

SQL supporta 5 operatori di aggregazione:

**count(\*)** - conta il numero di tuple risultanti dalla *query*

**count([DISTINCT] A)** - conta il numero di tuple risultanti dalla *query* che hanno un valore non **NULL** per l'attributo A, se **DISTINCT** è presente vengono conteggiate solo le tuple con valori distinti

**sum([DISTINCT] A)** - somma i valori dell'attributo A per le tuple risultanti dalla *query*, se **DISTINCT** è presente vengono sommati solo i valori distinti

**avg([DISTINCT] A)** - calcola la media dei valori dell'attributo A per le tuple risultanti dalla *query*, se **DISTINCT** è presente vengono considerati solo i valori distinti

**max(A)** - restituisce il valore massimo dell'attributo A per le tuple risultanti dalla *query*

**min(A)** - restituisce il valore minimo dell'attributo A per le tuple risultanti dalla *query*

### Operatori GROUP BY & HAVING

Gli operatori **GROUP BY** e **HAVING** permettono di eseguire operazioni di aggregazione solo su sottoinsiemi di tuple. In particolare **GROUP BY** permette di raggruppare le tuple in base ai valori di uno o più attributi, mentre **HAVING** permette di filtrare i risultati in base a condizioni di aggregazione.

```
SELECT <lista di attributi>, <funzione di aggregazione>
FROM <lista di relazioni>
WHERE <condizione>
GROUP BY <lista di attributi di raggruppamento>
HAVING <condizione di aggregazione>
```

<lista di attributi di raggruppamento> è la lista degli attributi su cui si vuole raggruppare le tuple

<condizione di aggregazione> è la condizione che deve essere soddisfatta dal risultato della funzione di aggregazione per essere incluso nel risultato

<funzione di aggregazione> è una o più funzioni di aggregazione da applicare ai gruppi di tuple

### Operatori insiemistici

Gli operatori insiemistici permettono di combinare i risultati di più *query* in un unico risultato. Gli operatori insiemistici sono:

UNION - restituisce l'unione dei risultati di due *query* (eliminando i duplicati)<sup>1</sup>

UNION ALL - restituisce l'unione dei risultati di due *query* (senza eliminare i duplicati e ordinare i risultati)

INTERSECT - restituisce l'intersezione dei risultati di due *query*

EXCEPT - restituisce la differenza tra i risultati di due *query*

### Operatore JOIN

L'operatore JOIN permette di combinare i risultati di due *query* in un unico risultato. Gli operatori JOIN sono:

INNER JOIN - restituisce le tuple che hanno un valore comune in entrambe le relazioni

LEFT JOIN - restituisce tutte le tuple della relazione di sinistra e le tuple della relazione di destra che hanno un valore comune

RIGHT JOIN - restituisce tutte le tuple della relazione di destra e le tuple della relazione di sinistra che hanno un valore comune

FULL JOIN - restituisce tutte le tuple delle relazioni coinvolte

---

<sup>1</sup>In quanto l'operatore UNION elimina i duplicati se si sta cercando di unire due query allora non è necessario usare il modificatore DISTINCT in quanto ordinamento e rimozione dei duplicati sono già garantiti dall'operatore