

# Appunti su Algebra Relazionale e Calcolo Relazionale

Basato sulle slide del Prof. Danilo Montesi

15 maggio 2025

## Indice

<b>1</b>	<b>Introduzione ai Linguaggi per Database</b>	<b>2</b>
<b>2</b>	<b>Algebra Relazionale</b>	<b>2</b>
2.1	Operatori dell'Algebra Relazionale . . . . .	2
2.2	Operatori Insiemistici . . . . .	2
2.3	Ridenominazione ( $\rho_{nuovo \leftarrow vecchio}(R)$ ) . . . . .	3
2.4	Selezione ( $\sigma_{predicato}(R)$ ) . . . . .	3
2.5	Proiezione ( $\pi_{lista\_attributi}(R)$ ) . . . . .	3
2.6	Combinazione di Selezione e Proiezione . . . . .	3
2.7	Join . . . . .	3
2.8	Outer Join (Join Esterni) . . . . .	4
2.9	Espressioni Equivalenti e Ottimizzazione . . . . .	4
2.10	Selezione con Valori NULL . . . . .	4
<b>3</b>	<b>Views (Viste)</b>	<b>5</b>
<b>4</b>	<b>Calcolo Relazionale</b>	<b>5</b>
4.1	Domain Relational Calculus (DRC) . . . . .	5
4.2	Tuple Relational Calculus (TRC) with Range Declarations . . . . .	6
4.3	Equivalenza tra Algebra e Calcolo . . . . .	6
<b>5</b>	<b>Limiti dell'Algebra e del Calcolo Relazionale Standard</b>	<b>7</b>
<b>6</b>	<b>Datalog</b>	<b>7</b>
<b>7</b>	<b>Conclusioni</b>	<b>8</b>

# 1 Introduzione ai Linguaggi per Database

I linguaggi per database si dividono principalmente in due categorie:

- **DDL (Data Definition Language):** Utilizzato per definire e modificare lo schema del database (es. creare tabelle, definire attributi e tipi).
- **DML (Data Manipulation Language):** Utilizzato per operare sui dati. Si suddivide ulteriormente in:
  - **Istruzioni di Query:** Per estrarre dati di interesse.
  - **Istruzioni di Aggiornamento:** Per inserire nuovi dati o modificare quelli esistenti.

I linguaggi di query possono essere:

- **Dichiarativi:** Specificano **cosa** si vuole ottenere, le proprietà del risultato. L'utente non si preoccupa di come il database recupererà i dati. SQL è prevalentemente dichiarativo.
- **Imperativi/Procedurali:** Specificano **come** il risultato deve essere ottenuto, descrivendo una sequenza di operazioni.

Panoramica dei linguaggi trattati:

- **Algebra Relazionale:** Procedurale (fondamento teorico).
- **Calcolo Relazionale:** Dichiarativo (fondamento teorico).
- **SQL (Structured Query Language):** Parzialmente dichiarativo (ampiamente implementato).
- **QBE (Query by Example):** Dichiarativo (implementato in alcuni sistemi).

## 2 Algebra Relazionale

L'algebra relazionale è un linguaggio di query formale, procedurale, che definisce un insieme di operatori che agiscono su relazioni (tabelle) per produrre nuove relazioni come risultato. Gli operatori possono essere composti.

### 2.1 Operatori dell'Algebra Relazionale

- Operatori insiemistici: **Unione** ( $\cup$ ), **Intersezione** ( $\cap$ ), **Differenza** ( $-$ ).
- **Ridenominazione** ( $\rho()$ ).
- **Selezione** ( $\sigma()$ ).
- **Proiezione** ( $\pi()$ ).
- **Join** ( $\bowtie$ ): natural join, prodotto cartesiano ( $\times$ ), theta-join ( $\bowtie_\theta$ ).

### 2.2 Operatori Insiemistici

Le relazioni sono insiemi di tuple. Questi operatori funzionano come le loro controparti nella teoria degli insiemi, ma richiedono che le relazioni coinvolte abbiano lo **stesso schema** (stessi nomi di attributi, nello stesso ordine e con tipi compatibili).

- **Unione** ( $R1 \cup R2$ ): Restituisce una relazione contenente tutte le tuple che sono in  $R1$ , in  $R2$ , o in entrambe. Le tuple duplicate vengono eliminate.
- **Intersezione** ( $R1 \cap R2$ ): Restituisce una relazione contenente solo le tuple che sono presenti sia in  $R1$  sia in  $R2$ .
- **Differenza** ( $R1 - R2$ ): Restituisce una relazione contenente le tuple che sono in  $R1$  ma non in  $R2$ .

**Esempio pratico:** Se hai due tabelle, *StudentiMagistrale* e *StudentiDottorato*, entrambe con colonne IDStudente e Nome, puoi fare l'unione per ottenere una lista unica di tutti gli studenti post-laurea. Se le colonne avessero nomi diversi (es. Matricola vs IDStudente), dovresti prima usare l'operatore di ridenominazione.

## 2.3 Ridenominazione ( $\rho_{nuovo \leftarrow vecchio}(R)$ )

Operatore unario che cambia i nomi degli attributi o della relazione stessa, senza alterare i dati.

- $\rho_{NuovoNomeAttr \leftarrow VecchioNomeAttr}((R))$ : Rinomina l'attributo *VecchioNomeAttr* in *NuovoNomeAttr* nella relazione *R*.
- $\rho_{NuovoNomeRel}((R))$ : Rinomina la relazione *R* in *NuovoNomeRel*.
- $\rho_{NuovoNomeRel(A1,A2,...)}((R))$ : Rinomina la relazione e i suoi attributi.

**Esempio pratico:** Se hai una tabella *Impiegati* con una colonna *stip* e vuoi renderla più chiara come *StipendioAnnuale*, useresti  $\rho_{StipendioAnnuale \leftarrow stip}((Impiegati))$ .

## 2.4 Selezione ( $\sigma_{predicato}(R)$ )

Operatore unario che restituisce un sottoinsieme delle tuple di una relazione *R* che soddisfano un dato *predicato* (condizione). Lo schema del risultato è identico a quello di *R*. **Esempio pratico (SQL):**  $\sigma_{Eta > 30 \wedge Dipartimento = 'IT'}((Impiegati))$  è equivalente a:

---

```
SELECT *
FROM Impiegati
WHERE Eta > 30 AND Dipartimento = 'IT';
```

---

## 2.5 Proiezione ( $\pi_{lista\_attributi}(R)$ )

Operatore unario che restituisce una nuova relazione contenente solo gli attributi specificati nella *lista\_attributi* dalla relazione *R*. Le tuple duplicate nel risultato vengono eliminate (poiché le relazioni sono insiemi).

**Esempio pratico (SQL):**  $\pi_{Nome,Cognome}((Impiegati))$  è equivalente a:

---

```
SELECT DISTINCT Nome, Cognome
FROM Impiegati;
```

---

Nota l'uso di *DISTINCT* in SQL per replicare il comportamento insiemistico della proiezione.

## 2.6 Combinazione di Selezione e Proiezione

Questi operatori sono spesso usati insieme per estrarre dati specifici. **Esempio pratico (SQL):** Trovare nome e cognome degli impiegati nel dipartimento 'Vendite' con stipendio superiore a 50000. Algebra:

$\pi_{Nome,Cognome}((\sigma_{Dipartimento = 'Vendite' \wedge Stipendio > 50000}((Impiegati))))$  SQL:

---

```
SELECT DISTINCT Nome, Cognome
FROM Impiegati
WHERE Dipartimento = 'Vendite' AND Stipendio > 50000;
```

---

## 2.7 Join

Il join è un operatore fondamentale per combinare informazioni da due o più relazioni.

- **Prodotto Cartesiano ( $R1 \times R2$ ):** Combina ogni tupla di *R1* con ogni tupla di *R2*. Il numero di tuple risultanti è  $|R1| \times |R2|$ . Lo schema è la concatenazione degli schemi di *R1* e *R2*. In SQL, è spesso scritto come *FROM R1, R2* (sintassi più vecchia) o *FROM R1 CROSS JOIN R2*. Di solito è seguito da una selezione per filtrare le combinazioni significative.
- **Theta-Join ( $R1 \bowtie_{condizione} R2$ ):** È un prodotto cartesiano seguito da una selezione. La *condizione* è un predicato che coinvolge attributi di *R1* e *R2*. Sintassi formale:  $\sigma_{condizione}((R1 \times R2))$ . **Esempio pratico (SQL):**  $Impiegati \bowtie_{Impiegati.IDDip = Dipartimenti.ID} Dipartimenti$  è equivalente a:

---

```
SELECT *
FROM Impiegati, Dipartimenti -- o Impiegati JOIN Dipartimenti
WHERE Impiegati.IDDip = Dipartimenti.ID;
```

---

- **Equi-Join:** Un Theta-Join in cui la condizione contiene solo confronti di uguaglianza ( $=$ ). L'esempio precedente è un equi-join.
- **Natural Join ( $R1 \bowtie R2$ ):** Un tipo speciale di equi-join. Le relazioni vengono combinate basandosi sull'uguaglianza dei valori degli attributi che hanno lo **stesso nome** in entrambe le relazioni. Gli attributi comuni appaiono una sola volta nel risultato. **Esempio pratico (SQL):** Se *Impiegati* e *Assegnazioni* hanno entrambe una colonna IDProgetto. *Impiegati*  $\bowtie$  *Assegnazioni* è equivalente a:

---

```
SELECT *
FROM Impiegati NATURAL JOIN Assegnazioni;
```

---

*Attenzione:* Il Natural Join può essere pericoloso se ci sono attributi con lo stesso nome ma significato diverso, o se si aggiungono/rimuovono colonne. È spesso preferibile usare join espliciti con clausola *ON*.

## 2.8 Outer Join (Join Esterni)

I join visti finora (inner join) scartano le tuple che non trovano una corrispondenza nell'altra relazione. Gli outer join includono queste tuple, riempiendo con *NULL* gli attributi mancanti.

- **Left Outer Join ( $R1 \ltimes R2$ ):** Mantiene tutte le tuple di *R1*. Se una tupla di *R1* non ha corrispondenze in *R2*, viene inclusa nel risultato con valori *NULL* per gli attributi di *R2*. SQL: *R1* *LEFT OUTER JOIN* *R2* *ON* *condizione*
- **Right Outer Join ( $R1 \rtimes R2$ ):** Mantiene tutte le tuple di *R2*. Simmetrico al left outer join. SQL: *R1* *RIGHT OUTER JOIN* *R2* *ON* *condizione*
- **Full Outer Join ( $R1 \Join R2$ ):** Mantiene tutte le tuple di entrambe le relazioni. Se non c'è corrispondenza, i campi dell'altra relazione sono *NULL*. SQL: *R1* *FULL OUTER JOIN* *R2* *ON* *condizione*

**Esempio pratico:** Trovare tutti gli impiegati e, se assegnati a un dipartimento, il nome del dipartimento. Se un impiegato non ha dipartimento, vogliamo comunque vederlo. *Impiegati*  $\Join$  *Dipartimenti* (assumendo un join su *IDDip*) SQL:

---

```
SELECT Impiegati.Nome, Dipartimenti.NomeDip
FROM Impiegati
LEFT OUTER JOIN Dipartimenti ON Impiegati.IDDip = Dipartimenti.ID;
```

---

## 2.9 Espressioni Equivalenti e Ottimizzazione

Esistono diverse espressioni algebriche che producono lo stesso risultato. I DBMS (Database Management Systems) utilizzano regole di equivalenza per trasformare una query in una forma equivalente ma più efficiente da eseguire. Ad esempio, "spingere" le selezioni il più presto possibile (*pushdownselection*) riduce la dimensione delle relazioni intermedie, velocizzando i join successivi. **Esempio:**  $\sigma_{\text{Stipendio} > 100K}((\text{Impiegati} \Join \text{Dipartimenti}))$  potrebbe essere più efficiente se riscritta come:  $(\sigma_{\text{Stipendio} > 100K}(\text{Impiegati})) \Join \text{Dipartimenti}$  (se *Stipendio* è solo in *Impiegati*). Il DBMS fa queste ottimizzazioni automaticamente.

## 2.10 Selezione con Valori NULL

I valori *NULL* rappresentano l'assenza di un valore o un valore sconosciuto. Nei predicati di selezione:

- Un confronto con *NULL* (es. *Eta*  $>$  *NULL* o *Stipendio*  $=$  *NULL*) restituisce *UNKNOWN*.
- L'operatore  $\sigma$  seleziona solo le tuple per cui il predicato è *TRUE*.

- Per testare esplicitamente i *NULL*, si usano i predicati Attributo IS NULL e Attributo IS NOT NULL.

**Esempio pratico (SQL):** Trovare gli impiegati senza un numero di telefono assegnato.  $\sigma_{\text{Telefono IS NULL}}((\text{Impiegati}))$   
SQL:

---

```
SELECT * FROM Impiegati WHERE Telefono IS NULL;
```

---

### 3 Views (Viste)

Una vista è una tabella virtuale il cui contenuto è definito da una query sull'algebra relazionale (o SQL). Non memorizza dati propriamente, ma li deriva dalle tabelle base al momento della query sulla vista.

- **Tabelle Base:** Tabelle che contengono fisicamente i dati.
- **Tabelle Derivate (Viste):** Definite da query.

Le viste sono utili per:

- **Schema Esterno:** Fornire diverse rappresentazioni dei dati a utenti diversi, semplificando la complessità e implementando la sicurezza (mostrando solo dati pertinenti).
- **Strumento di Programmazione:** Semplificare query complesse riutilizzando sotto-espressioni comuni, o per mantenere la compatibilità con applicazioni esistenti quando lo schema delle tabelle base cambia.

**Esempio pratico (SQL):** Creare una vista che mostra solo gli impiegati del dipartimento IT. Algebra:  $\text{ImpiegatiIT} := \sigma_{\text{Dipartimento}='IT'}((\text{Impiegati}))$  SQL:

---

```
CREATE VIEW ImpiegatiIT AS
SELECT *
FROM Impiegati
WHERE Dipartimento = 'IT';

-- Successivamente si può interrogare la vista:
SELECT Nome, Cognome FROM ImpiegatiIT WHERE Stipendio > 60000;
```

---

Il DBMS traduce la query sulla vista in una query sulle tabelle base (es.  $\sigma_{\text{Stipendio}>60000}((\sigma_{\text{Dipartimento}='IT'}((\text{Impiegati}))))$ ).

## 4 Calcolo Relazionale

Il calcolo relazionale è un linguaggio di query formale, **dichiarativo**, basato sulla logica dei predicati del primo ordine. Specifica *cosa* si vuole ottenere, non *come*. Esistono due forme principali:

- **Domain Relational Calculus (DRC):** Le variabili assumono valori dai domini degli attributi.
- **Tuple Relational Calculus (TRC):** Le variabili rappresentano tuple di relazioni.

### 4.1 Domain Relational Calculus (DRC)

Una query DRC ha la forma:  $\{A_1 : x_1, \dots, A_k : x_k \mid \text{Formula}(x_1, \dots, x_k)\}$  dove:

- $A_1 : x_1, \dots, A_k : x_k$  è la **target list**: specifica gli attributi del risultato e le variabili che ne conterranno i valori.
- $x_1, \dots, x_k$  sono variabili che variano sui domini dei rispettivi attributi.
- $\text{Formula}(x_1, \dots, x_k)$  è una formula della logica del primo ordine che usa:

- Predicati corrispondenti alle relazioni nel database (es. *IMPIEGATO*(Num :  $m$ , Nome :  $n$ , ...)).
- Operatori di confronto ( $=, >, <, \dots$ ).
- Operatori logici ( $\wedge$  AND,  $\vee$  OR,  $\neg$  NOT).
- Quantificatori ( $\forall$  per tutti,  $\exists$  esiste).

Il risultato è l'insieme di tuple  $(A_1 : v_1, \dots, A_k : v_k)$  tali che, quando le variabili  $x_i$  assumono i valori  $v_i$ , la Formula è vera.

**Esempio DRC:** Trovare numero, nome, età e salario degli impiegati che guadagnano più di 40.

```
{ Numero:m, Nome:n, Eta:a, Salario:w |
  IMPIEGATO(Numero:m, Nome:n, Eta:a, Salario:w)  w > 40 }
```

Se vogliamo solo nome ed età:

```
{ Nome:n, Eta:a |
  existsop w (IMPIEGATO(Numero:m, Nome:n, Eta:a, Salario:w)  w > 40) }
```

(Il *Numero : m* nella seconda query è una variabile libera nella formula *IMPIEGATO*, ma non nella target list, quindi la sua esistenza è implicitamente richiesta. Le slide mostrano *EMPLOYEE*(*Number : m, Name : n, Age : a, Wage : w*) anche quando *m* non è nella target list; questo significa che deve esistere una tupla con qualche *m* che soddisfi il resto. Per essere più precisi, si quantificherebbero le variabili non nella target list).

## 4.2 Tuple Relational Calculus (TRC) with Range Declarations

Una query TRC ha la forma: {TargetList | RangeList | Formula} dove:

- TargetList: Specifica gli attributi da restituire, spesso nella forma  $t.A$  (attributo  $A$  della tupla  $t$ ).
- RangeList: Dichiarare le variabili di tupla e le relazioni a cui appartengono (es.  $e(IMPIEGATO), s(SUPERVISORE)$ ).
- Formula: Una condizione sulle variabili di tupla dichiarate.

Le variabili variano sull'insieme delle tuple della relazione specificata.

**Esempio TRC:** Trovare tutte le informazioni sugli impiegati che guadagnano più di 40.

```
{ e.* | e(IMPIEGATO) | e.Salario > 40 }
```

Trovare nome ed età degli impiegati che guadagnano più di 40:

```
{ e.Nome, e.Eta | e(IMPIEGATO) | e.Salario > 40 }
```

Il TRC è spesso considerato più vicino a come si pensa in SQL, poiché si ragiona in termini di "tuple che soddisfano certe condizioni".

## 4.3 Equivalenza tra Algebra e Calcolo

Per le query "safe" (che non producono risultati infiniti e dipendono solo dai dati nel database), l'Algebra Relazionale, il Domain Relational Calculus (safe) e il Tuple Relational Calculus (safe) sono **espressivamente equivalenti**. Ciò significa che qualsiasi query esprimibile in uno di questi linguaggi può essere espressa anche negli altri. Questo è un risultato teorico importante (Teorema di Codd).

## 5 Limiti dell'Algebra e del Calcolo Relazionale Standard

Nonostante la loro potenza, l'algebra e il calcolo relazionale standard hanno dei limiti:

- **No Calcoli Aritmetici/Nuovi Valori:** Non possono calcolare nuovi valori (es. stipendio + bonus) o eseguire aggregazioni (somma, media, conteggio). SQL estende queste capacità con funzioni aritmetiche e di aggregazione (*SUM()*, *AVG()*, *COUNT()*, *GROUPBY*).
- **No Chiusura Transitiva (Recursion):** Non possono esprimere query ricorsive, come trovare tutti i superiori di un impiegato (il capo, il capo del capo, ecc.) o tutte le tratte aeree possibili tra due città (dirette e indirette). Questo richiederebbe un numero potenzialmente illimitato di join.

## 6 Datalog

Datalog è un linguaggio di query e programmazione logica orientato ai database, che supera alcuni limiti di RA/RC, in particolare per le query ricorsive. È un sottoinsieme di Prolog. Concetti chiave:

- **Predicati Estensionali (EDB - Extensional Database):** Corrispondono alle relazioni base del database (fatti).
- **Predicati Intensionali (IDB - Intensional Database):** Corrispondono a viste o relazioni derivate, definite tramite **regole**.
- **Regole:** Hanno la forma *testa* : *-corpo*. (o *testa* < *-corpo* nelle slide). Significa: "la *testa* è vera se il *corpo* è vero". La *testa* è un singolo predicato intensionale. Il *corpo* è una congiunzione (AND) di predicati (estensionali o intensionali) e condizioni.
- **Query:** Indicate con un prefisso **?** davanti a un predicato.

**Esempio Datalog (non ricorsivo):** Trovare i capi degli impiegati che guadagnano più di 40. Relazioni EDB: *IMPIEGATO*(Num, Nome, Eta, Salario), *SUPERVISORE*(Capo, Impiegato)

---

```
% Predicato intensionale: IMPIEGATO_RICCO
IMPIEGATO_RICCO(Num, Nome, Eta, Salario) :-
IMPIEGATO(Num, Nome, Eta, Salario), Salario > 40.

% Predicato intensionale: CAPO_DI_IMPIEGATO_RICCO
CAPO_DI_IMPIEGATO_RICCO(NumCapo) :-
IMPIEGATO_RICCO(NumImp, _, _, _),
SUPERVISORE(NumCapo, NumImp).

% Query
? CAPO_DI_IMPIEGATO_RICCO(X).
```

---

**Esempio Datalog (Ricorsione - Chiusura Transitiva):** Trovare tutti i superiori (diretti e indiretti) di un impiegato. Relazione EDB: *CAPO\_DIRETTO*(Superiore, Subordinato)

---

```
% Caso base: un capo diretto e' un superiore
SUPERIORE(X, Y) :- CAPO_DIRETTO(X, Y).

% Caso ricorsivo: il superiore di un mio superiore e' anche mio superiore
SUPERIORE(X, Y) :- CAPO_DIRETTO(X, Z), SUPERIORE(Z, Y).

% Query: trovare tutti i superiori di 'Rossi' (assumendo che 'Rossi' sia un ID)
? SUPERIORE(Capo, 'Rossi').
```

---

Datalog, grazie alla sua capacità di esprimere la ricorsione, è più potente dell'algebra e del calcolo relazionale standard. Le estensioni ricorsive di SQL (come *WITHRECURSIVE*) sono ispirate a Datalog.

## 7 Conclusioni

L'Algebra Relazionale e il Calcolo Relazionale forniscono le fondamenta teoriche per i linguaggi di query dei database relazionali come SQL.

- L'**Algebra Relazionale** è procedurale e definisce come costruire il risultato passo dopo passo. È cruciale per l'implementazione interna e l'ottimizzazione delle query nei DBMS.
- Il **Calcolo Relazionale** è dichiarativo, permettendo di specificare le proprietà del risultato desiderato senza dettagliarne il processo di ottenimento.
- Entrambi (nelle loro forme "safe") hanno lo stesso potere espressivo ma non possono gestire calcoli complessi o ricorsione.
- **Datalog** estende questi concetti introducendo la ricorsione, aumentando il potere espressivo.

Comprendere questi modelli teorici aiuta a capire meglio il funzionamento e le potenzialità di SQL e dei sistemi di gestione di database moderni.