Databases: Algebra Relazionale e Calcolo Relazionale

Based on slides from Danilo Montesi

15 maggio 2025

1 Introduzione: Linguaggi per Basi di Dati Relazionali

I linguaggi di interrogazione (query languages) per basi di dati relazionali possono essere classificati in base a come specificano il risultato:

- **Declarative:** Specificano **cosa** si vuole ottenere (le proprietà del risultato desiderato).
- **Imperative/Procedural:** Specificano **come** il risultato deve essere ottenuto (la sequenza di operazioni da eseguire).

Esempi noti:

- **Relational Algebra (RA):** Procedurale (teorico).
- **Relational Calculus (RC):** Dichiarativo (teorico, non implementato direttamente).
- **SQL:** Parzialmente dichiarativo (il linguaggio più usato, implementato).
- **QBE (Query by Example):** Dichiarativo (implementato).

L'Algebra e il Calcolo Relazionale sono fondamentali per capire le basi teoriche di SQL.

2 Algebra Relazionale (RA)

L'Algebra Relazionale è un linguaggio di interrogazione procedurale definito da un insieme di operatori che:

- Operano **su relazioni** (tabelle).
- Producono **relazioni** come risultato.
- Sono **componibili** (l'output di un operatore può essere l'input di un altro).

Questo la rende una "algebra" nel senso matematico.

2.1 Operatori Base dell'Algebra Relazionale

Gli operatori si dividono in operatori insiemistici e operatori relazionali specifici:

- **Set Operators** (richiedono che le relazioni abbiano schemi compatibili, cioè stesso numero di attributi con domini corrispondenti):
  - **Union ($\cup$):** Restituisce l'insieme di tutte le tuple presenti in almeno una delle due relazioni. Rimuove i duplicati (per definizione di relazione come insieme).
  - **Intersection ($\cap$):** Restituisce l'insieme di tutte le tuple presenti in entrambe le relazioni.
  - **Difference ($-$):** Restituisce l'insieme di tutte le tuple presenti nella prima relazione ma non nella seconda.
- **Relational Operators:**
  - **Renaming (ρ):** Operatore unario che cambia il nome di una relazione o di uno o più attributi. Importante per rendere compatibili gli schemi per gli operatori insiemistici o per distinguere attributi con lo stesso nome dopo un prodotto cartesiano o un join.

- **Selection (σ):** Operatore unario che filtra le **righe** (tuple) di una relazione in base a un **predicato** (condizione). Lo schema del risultato è identico a quello della relazione di input.
- **Projection (π):** Operatore unario che filtra le **colonne** (attributi) di una relazione. Lo schema del risultato contiene solo gli attributi specificati. Le tuple duplicate nel risultato vengono rimosse.
- **Join ($\bowtie$):** Operatore binario fondamentale per combinare tuple da **due relazioni diverse** in base a un qualche criterio di correlazione.

2.2 Dettagli sugli Operatori Relazionali

2.2.1 Renaming (ρ)

- **Scopo:** Cambiare nome a una relazione o a suoi attributi. Non altera i dati.
- **Sintassi:**
  - Ridenominazione di una relazione: $\rho_{NewR}(R)$
  - Ridenominazione di attributi: $\rho_{A_1 \leftarrow B_1,...,A_n \leftarrow B_n}(R)$ (rinomina gli attributi B_i di R in A_i)
  - Combinata: $\rho_{NewR(A_1,...,A_n)}(R)$ (rinomina la relazione e i suoi attributi)
- **Importanza:** Rende possibile eseguire operatori insiemistici su relazioni con schemi nominalmente diversi ma strutturalmente compatibili. È anche utile per distinguere attributi con lo stesso nome originari da relazioni diverse dopo un Join.

2.2.2 Selection (σ)

- **Scopo:** Estrarre un **sottoinsieme orizzontale** (righe) di una relazione.
- **Sintassi:** $\sigma_P(R)$, dove P è il predicato (condizione).
- **Predicato:** Un'espressione booleana sugli attributi delle tuple (es. 'Age ¿ 30 AND Office = 'Milan''').
- **Risultato:** Contiene solo le tuple di R per cui P è vero. Stesso schema di R.

2.2.3 Projection (π)

- **Scopo:** Estrarre un **sottoinsieme verticale** (colonne) di una relazione.
- **Sintassi:** $\pi_A(R)$, dove A è la lista degli attributi desiderati.
- **Risultato:** Contiene le tuple di R ristrette agli attributi in A. Le **tuple duplicate vengono eliminate** per garantire che il risultato sia un insieme.
- **Cardinalità:** La cardinalità del risultato è $\leq$ la cardinalità di R. È strettamente minore se la proiezione rimuove gli attributi che rendevano le tuple distinte. Se A è una superchiave di R, la cardinalità è uguale.

2.2.4 Combinare Selection e Projection

- σ e π sono operatori **ortogonali**: σ opera per righe, π per colonne.
- Possono essere combinati per estrarre informazioni specifiche (righe e colonne) da una **singola relazione**. Esempio: $\pi_{Number,Surname}(\sigma_{Salary>50}(EMPLOYEE))$.
- **Limite:** Con solo σ e π non è possibile correlare informazioni tra tuple diverse (né nella stessa relazione né tra relazioni diverse). Per questo serve il Join.

2.3 Join Operators ($\bowtie$)

Il Join è cruciale per combinare dati da relazioni diverse.

2.3.1 Cartesian Product ($\times$)

- **Scopo:** Combinare ogni tupla di R_1 con ogni tupla di R_2.
- **Sintassi:** $R_1 \times R_2$.
- **Risultato:** Schema è l'unione degli attributi (con eventuali ridenominazioni se ci sono nomi in comune). Cardinalità: $|R_1| \times |R_2|$.
- **Utilizzo Pratico:** Raramente usato da solo, quasi sempre seguito da una selezione per filtrare le combinazioni indesiderate.

2.3.2 Theta-Join ($\bowtie_C (R_1, R_2)$)

- **Scopo:** Un Join basato su una **condizione generale** C.
- **Definizione:** $\bowtie_C (R_1, R_2) \equiv \sigma_C((R_1 \times R_2))$.
- **Sintassi:** $R_1 \bowtie_C R_2$.
- **Condizione C:** Un predicato che può confrontare attributi di R_1 e R_2 usando vari operatori ($=, <, >, \leq, \geq, \neq$).

2.3.3 Equi-Join

- **Scopo:** Un caso speciale di Theta-Join dove la condizione C è una **congiunzione di uguaglianze** ($=$).
- **Vantaggio:** Spesso più efficiente del calcolo esplicito del prodotto cartesiano seguito dalla selezione. Permette di joinare attributi con nomi diversi specificando l'uguaglianza.

2.3.4 Natural Join ($\bowtie$)

- **Scopo:** Join basato sull'**uguaglianza automatica degli attributi con lo stesso nome** nelle due relazioni.
- **Sintassi:** $R_1 \bowtie R_2$.
- **Processo:** Trova gli attributi comuni. Combina le tuple di R_1 e R_2 che hanno valori uguali su tutti gli attributi comuni. Gli attributi comuni compaiono una sola volta nello schema del risultato.
- **Relazione con Equi-Join e Proiezione:** Un Natural Join può essere espresso tramite Ridenominazione (se necessario), Prodotto Cartesiano, Selezione sull'uguaglianza degli attributi comuni e Proiezione per eliminare gli attributi duplicati. Spesso, $R_1 \bowtie R_2$ (su attributi comuni A) è equivalente a $\pi_{\text{schema}}(()\sigma_{R_1.A=R_2.A}((R_1 \times R_2)))$.

2.4 Cardinalità del Risultato del Join

Per $R_1 \bowtie R_2$:

- $0 \leq |R_1 \bowtie R_2| \leq |R_1| \times |R_2|$.
- Se l'attributo/i di join è una **chiave candidata in** R_2, allora ogni tupla di R_2 può matchare al massimo una tupla di R_1 (sull'attributo di join), quindi $|R_1 \bowtie R_2| \leq |R_1|$.
- Se l'attributo/i di join è una **chiave primaria in** R_2 e c'è un **vincolo di integrità referenziale** (Foreign Key in R_1 che referenzia Primary Key in R_2), allora ogni tupla di R_1 referenzia una tupla esistente in R_2, quindi $|R_1 \bowtie R_2| = |R_1|$.

2.5 Outer Join

Il Join standard (Inner Join) scarta le tuple che non trovano corrispondenza. Gli Outer Join mantengono queste tuple, riempiendo con valori **NULL** gli attributi mancanti.

- **Left Outer Join ($⟕$):** Mantiene tutte le tuple della relazione **sinistra**. Se una tupla sinistra non ha corrispondenze nella relazione destra, viene comunque inclusa, e gli attributi della relazione destra vengono riempiti con NULL.
- **Right Outer Join ($⟖$):** Mantiene tutte le tuple della relazione **destra**. Simmetrico al Left Outer Join.
- **Full Outer Join ($⟗$):** Mantiene tutte le tuple da **entrambe** le relazioni. Le tuple senza corrispondenza in una relazione vengono incluse, con gli attributi mancanti riempiti con NULL.

2.6 Espressioni Equivalenti in RA (Ottimizzazione)

Due espressioni RA sono equivalenti se producono lo stesso risultato per qualsiasi stato del database. I DBMS utilizzano queste equivalenze per riscrivere le query in forme più efficienti.

- **Push Down Selection:** Spostare le selezioni il più possibile verso il basso nell'albero di valutazione (applicarle il prima possibile).
  - $\sigma_{C_1 \wedge C_2}(R) = \sigma_{C_1}(\sigma_{C_2}(R))$ (Ordine delle selezioni non importa).
  - $\sigma_C((R_1 \times R_2)) = R_1 \times \sigma_C(R_2)$ (se C coinvolge solo attributi di R_2). Molto efficiente perché riduce la cardinalità prima del prodotto cartesiano.
  - $\sigma_C((R_1 \bowtie_D R_2)) = R_1 \bowtie_D \sigma_C(R_2)$ (se C coinvolge solo attributi di R_2). Similmente, riduce la cardinalità prima del join.
- **Push Down Projection:** Spostare le proiezioni il più possibile verso il basso, ma con attenzione.
  - $\pi_{A_1}(R) = \pi_{A_1}(\pi_{A_1 \cup A_2}(R))$ (Proiezione idempotente se la lista attributi è un superinsieme).
  - $\pi_A((R_1 \bowtie_C R_2)) = \pi_A((\pi_{A'}(R_1) \bowtie_C \pi_{A''}(R_2)))$, dove A' sono gli attributi di R_1 in A o necessari per la condizione C, e A'' sono gli attributi di R_2 in A o necessari per C. Efficiente perché riduce la dimensione delle tuple prima del join.
- Altre proprietà: Distributività di σ su $\cup$ e $-$. Distributività di π su $\cup$. Associatività e Commutatività per $\cup, \cap, \bowtie, \times$.

2.7 Gestione dei Valori NULL nella Selezione

- I confronti con NULL (es. 'Age ¿ 40' se Age è NULL) risultano in **UNKNOWN**.
- La Selection $\sigma_P(R)$ include solo le tuple per cui P è valutato **TRUE**. Le tuple con risultato FALSE o UNKNOWN vengono scartate.
- Per selezionare tuple con valori NULL specifici si usano i predicati **'IS NULL'** e **'IS NOT NULL'**. (Es. 'selectopAge IS NULLPEOPLE').
- Il "problema" di un risultato 'UNKNOWN' è la base per la logica a tre valori (TRUE, FALSE, UNKNOWN) in SQL.

2.8 Views

Una View è una "tabella virtuale" definita da una query. Permette rappresentazioni diverse dei dati sottostanti (Base Tables).

- **Scopo:**
  - **Data Hiding / Security (Schema Esterno):** Presentare solo i dati e gli attributi pertinenti a un utente o applicazione, nascondendo la complessità o le parti riservate dello schema base.
  - **Programming Tool:** Semplificare la scrittura di query complesse, riusando definizioni comuni o adattando l'interfaccia dati a software esistente.

- **Tipi:**
  - **Materialized Views:** Il risultato della query che definisce la view è effettivamente memorizzato fisicamente. Pro: Accesso molto veloce. Contro: Costo di memorizzazione, aggiornamenti delle base table richiedono aggiornamento della view (costoso), aggiornamenti *sulla view materializzata* sono complessi e raramente supportati.
  - **Virtual Views:** Solo la definizione della query è memorizzata. Quando la view viene interrogata, il DBMS **riscrive la query originale** sostituendo il nome della view con la sua definizione (query sottostante) e poi valuta la query riscritta sulle base tables. Pro: Nessuna ridondanza di memorizzazione, sempre aggiornata con le base tables. Contro: Può essere meno performante di una materialized view. Gli aggiornamenti *sulla view virtuale* sono possibili solo in casi molto semplici e non ambigui (problema dell'Incremental Update: come trasformare un update sulla view in uno o più update sulle base table?).
- Le Virtual Views non cambiano l'efficienza delle query; l'ottimizzazione avviene sulla query espansa sulle base tables.

2.9 Notazione Alternativa per il Join (SQL-like)

Per rendere più chiara la distinzione tra attributi con lo stesso nome provenienti da relazioni diverse (soprattutto dopo Join), si usa spesso la notazione **'Relation.Attribute'** (come in SQL). Adottando questa convenzione, il Natural Join "automatico" su nomi comuni è meno usato esplicitamente, a favore di join con condizioni esplicite che usano questa notazione.

3 Calcolo Relazionale (RC)

Il Calcolo Relazionale è una famiglia di linguaggi **dichiarativi** basati sulla Logica del Primo Ordine (First Order Logic - FOL). L'utente specifica le proprietà delle tuple che vuole nel risultato, non la procedura per ottenerle.

3.1 Domain Relational Calculus (DRC)

- **Variabili:** Rangeano sui **domini dei valori** (numeri, stringhe, ecc.).
- **Sintassi:** $\{A_1 : x_1, \ldots, A_k : x_k \mid F\}$, dove A_i sono gli attributi di output, x_i le variabili corrispondenti, e F è una formula FOL.
  - $\{$Target List $\mid$ Formula$\}$
  - La Formula F contiene predicati:
    - Predicati che rappresentano le **relazioni base** (es. 'EMPLOYEE(m, n, a, w)' significa che la tupla di valori (m, n, a, w) esiste nella relazione EMPLOYEE).
    - Predicati **built-in** per confronti ('x ¿ y', 'z = 'Rome'').
  - La Formula F può usare operatori logici ($\wedge, \vee, \neg$) e **quantificatori** ($\forall, \exists$) sulle variabili del dominio.
- **Semantica:** L'insieme di tuple $(v_1, \ldots, v_k)$ tali che, sostituendo x_i con v_i, la Formula F è vera.
- **Note:** L'approccio è "non-positional" nella Target List (si associa variabile al nome dell'attributo).
- **Contro:** Può essere **prolisso** per query semplici. Permette la scrittura di formule "domain dependent" (es. 'x — ¬R(x)') che non sono esprimibili in RA e si riferiscono all'intero universo dei valori, non solo quelli presenti nel database. Queste formule sono da evitare nelle basi di dati.

3.2 Tuple Relational Calculus with Range Declarations (TRC-RD)

- **Scopo:** Superare la prolissità del DRC e garantire "domain independence".
- **Variabili:** Rangeano sulle **tuple di specifiche relazioni**.
- **Sintassi:** $\{T \mid R \mid F\}$, dove T è la Target List, R è la Range List, e F è la Formula.

- T: ‘y.A‘ (attributo A della variabile di tupla y), eventualmente ridenominato ‘NewA: y.A‘.
- R: Dichiara le variabili di tupla e la relazione su cui rangeano (es. $e \in EMPLOYEE$, o sintassi più compatta ‘e(EMPLOYEE)‘). Le variabili nella Formula *devono* essere dichiarate qui.
- F: Formula FOL su **attributi di variabili di tupla** (es. ‘e.Wage ¿ 40‘).

- **Semantica:** L’insieme di tuple formate valutando la Target List, dove le variabili di tupla rispettano i loro range e la Formula è vera.
- Spesso più vicino alla ”forma” delle query SQL.

3.3 Equivalenza tra Calcolo e Algebra

- L’Algebra Relazionale, il Domain Relational Calculus (limitato a espressioni domain-independent) e il Tuple Relational Calculus with Range Declarations sono **equivalenti** in termini di potere espressivo. Tutto ciò che si può esprimere in uno si può esprimere negli altri.
- Questo teorema di equivalenza è fondamentale nella teoria dei database.

3.4 Limiti del Potere Espressivo di RA e RC Standard

Nonostante la loro equivalenza, RA e RC standard non possono esprimere tutte le query ”ragionevoli” che potrebbero servire in pratica:

- Non possono **computare nuovi valori** non presenti esplicitamente nel database (es. calcolare la percentuale di sconto). Possono solo estrarre o combinare valori esistenti.
- Non possono esprimere **funzioni aggregate** (SUM, AVG, COUNT, MIN, MAX).
- Non possono esprimere **query ricorsive**, come la **Transitive Closure**.

3.4.1 Transitive Closure

- **Definizione:** Data una relazione binaria R su un insieme X, la transitive closure R^+ contiene tutte le coppie (x, y) tali che esiste un cammino da x a y in R (una sequenza di uno o più ”salti” in R).
- **Esempio:** In una relazione SUPERVISOR(Employee, Chief), la transitive closure includerebbe non solo i capi diretti, ma anche i capi dei capi, ecc.
- **Impossibilità in RA/RC standard:** Esprimere la transitive closure in RA richiederebbe l’unione di un numero potenzialmente illimitato di join (R ∪ R ⋈ R ∪ R ⋈ R ⋈ R ∪ ...). RA/RC sono limitati a esprimere query di profondità finita.

4 Datalog

Datalog è un linguaggio di programmazione logica basato su Prolog, specificamente pensato per i database. Supera alcuni limiti di RA/RC, in particolare permettendo la ricorsione.

- **Predicati:**
  - **Extensional:** Corrispondono alle **relazioni base** del database.
  - **Intensional:** Corrispondono a **viste** o concetti derivati, definiti da regole.
- **Regole:** Hanno la forma ‘head ¡- body‘.
  - ‘head‘: Un predicato intensionale (ciò che si vuole derivare).
  - ‘body‘: Una congiunzione (lista) di predicati estensionali o intensionali (le condizioni per la derivazione).
- **Query:** Iniziano con ‘¿, interrogando un predicato. (Es. ‘? richer(m, n, a, w).‘)

- **Ricorsione:** Datalog permette regole ricorsive, dove un predicato intensionale compare sia nella head che nel body (es. per definire la Transitive Closure).
- **Negazione:** Datalog supporta la negazione ($\neg$) nel body delle regole, ma il suo comportamento con la ricorsione richiede definizioni semantiche più complesse (”negation as failure”, stratificazione).
- **Potere Espressivo:**
  - Datalog non ricorsivo (con o senza negazione) è equivalente a RA e RC standard.
  - Datalog **ricorsivo (senza negazione)** può esprimere query che RA/RC standard non possono (come la Transitive Closure).
  - Datalog ricorsivo **con negazione** è ancora più espressivo, ma può avere problemi semantici.