Domande Orale

Indice

Transazioni

Transazione: complesso di operazioni (di *lettura* e *scrittura*) che portano il DB da uno stato corretto ad un altro stato corretto.
Un sistema transazionale è in grado di eseguire transazioni per un certo numero di applicazioni concorrenti.
(In SQL è possibile definire una transazione mediante uno specifico comando `start`, non è invece necessario definire l'`end` ⇒ nel blocco va poi utilizzato almeno un comando tra il `commit work` (permette terminazione corretta) e il `rollback work` (permette di abortire la tranisizione)).

ACID

La transazione gode di 4 proprietà:

- **Atomicità**: le operazioni che costituiscono la transazione compongono un blocco unico che deve essere eseguito nella sua interezza (Il DB non può essere lasciato in uno stato intermedio). In caso di guasti o errori prima del termine della transazione, le azioni sui dati già svolte devono essere annullate (operazione detta `UNDO`) e poi successivamente ripetute (`REDO`). ⇒Tali operazioni sono realizzate dal sistema di recovery (Ha il compito di riportare il DB in uno stato corretto in caso di guasti).
  Se la transazione ha successo, le modifiche vengono riporate in modo

permanente dal DBMS nella memoria fisica (stato corretto: stato in cui il DB non presenta modifiche dovute a transazioni incomplete).

> ⚠ NOTA: >
>
> Mediante rollback work è la transazione che richiede l'annullamento delle operazioni per qualche motivo. Talvolta però, questa richiesta può essere inoltrata direttamente dal sistema che non riesce ad eseguire le operazioni.

- **Consistenza**: la transazione rispetta i vincoli di integrità del DB. In questo modo si garantisce che lo stato di arrivo sia corretto.
- **Isolamento**: la transazione è isolata rispetto alle altre transazioni concorrenti.
- **Durabilità**: le transazioni che terminano con una operazione di commit vengono mantenute in memoria fisica in modo permanente.

Lock e Concorrenza

Per garantire tali proprietà occorre effettuare controlli su *affidabilità* (per atomicità e durabilità), *concorrenza* (per isolamento) e sull'*esecuzione* delle operazioni (consistenza), che vengono realizzati dal gestore delle transazioni.
Il controllo sull'affidabilità è realizzato mediante un particolare file detto `log` che al suo interno mantiene l'insieme delle operazioni svolte sul DB mediante transazioni. (All'interno del file troviamo anche **checkpoint**, che indicano quali sono i dati già riportati in memoria dal DBMS dopo operazioni di commit)

> ⚠ NOTA: >
>
> Le modifiche spesso vengono riportate prima sul log e solo dopo nella memoria secondaria contenente il DB. (modalità differita)

Il log è memorizzato all'interno della memoria stabile (memoria che non si può danneggiare, astrazione).
Per facilitare il recupero delle informazioni il DBMS mantiene in memoria stabile anche il dump del DB, ovvero una copia completa di riserva dei dati e delle informazioni.

Gestione dei guasti

Le operazioni di risoluzione vengono svolte dal **recovery manager**.

- *Guasti "soft"*: errori di programma, crash di sistema, perdita di tensione. In questo caso si perde il contenuto della memoria principale, ma rimane preservato il contenuto della memoria secondaria (gestite tramite ripresa a caldo);
- *Guasti "hard"*: si perde anche il contenuto della memoria secondaria. Rimane perciò preservato solo il contenuto della memoria stabile (`log` e `dump`).
  Ripresa a caldo: 4 fasi
- Individuazione ultimo *checkpoint* nel log
- Costruire insiemi `UNDO` (operazioni da disfare) e `REDO` (operazioni da rifare)
- Scandire il log a ritroso, cancellando tutte le operazioni inserite nell'insieme UNDO
- Avanzare nuovamente nel log implementando le operazioni dell'insieme REDO
  Ripresa a freddo: i dati vengono ripristinati mediante il dump. A questo punto vengono effettuate di nuovo tutte le operazioni registrate nel log fino al momento del guasto seguendo lo schema di una ripresa a caldo.

Concorrenza

I sistemi odierni devono essere in grado di governare centinaia di transazioni al secondo che possono essere eseguite anche nello stesso arco temporale.
Se la concorrenza non venisse gestita correttamente, potrebbero verificarsi gravi problematiche legate alla consistenza dei dati.

Consideriamo la seguente situazione: una transizione T_1 prova a modificare il valore di una certa variabile X, però, prima che questo venga trascritto in memoria fisica, viene schedulata una seconda transizione T_2 che legge X e la modifica. In questo caso il risultato della transizione T_1 viene perso, perché T_2 leggerà il valore iniziale di X e sarà questo ad essere modificato.
Un'altra situazione critica è quando la transizione T_1 prova a leggere due volte il valore di una certa X. Se tra queste due operazioni viene schedulata una T_2 che modifica X, T_1 leggerà per X due valori distinti senza averla modificata.

Si possono infine generale anche altre anomalie come gli inserimenti fantasma (T_1 legge stipendi impiegati, poi viene schedulata T_2 che inserisce un nuovo impiegato.

T_1 a questo punto viene caricata di nuovo in CPU per essere completata e calcola la media degli stipendi $\Rightarrow$ non teniamo conto di inserimento fatto da T_2).

Il gestore delle transizioni si affida per risolvere tali problematiche a specifici *scheduler*, come quelli seriali. In questo caso le operazioni delle transizioni vengono eseguite una alla volta in maniera successiva. (Una volta terminata T_1 si passa poi a T_2) $\Rightarrow$ se T presenta un elevato numero di operazioni, avremo occupazione non efficiente della CPU.
Gli scheduler, in generale, possono provocare le problematiche precedentemente citate. Una parte di essi, però, produrrà il risultato corretto, in questo caso sono detti serializzabili (ovvero producono sul DB gli stessi effetti di uno scheduler seriale).
Abbiamo bisogno di un metodo per stabilire tale equivalenza. (Nella seguente trattazione supponiamo di dover gestire solo transazioni andate in commit)

Scheduler View-Serializzabili (VSR)

Relazione "`legge da`": dato un'ordinamento di scheduler S, si dice che una operazione di lettura $ri(x)$ `LEGGE_DA` un'operazione di scrittura $wj(x)$ se $wj(x)$ *precede* $ri(x)$ in S e non sono presenti altre $wk(x)$ tra le due operazioni citate.
Scrittura finale: $wi(x)$ è scrittura finale, se è l'ultima operazione di scrittura su x che appare in S. Due scheduler sono *view-equivalenti* se hanno la stessa relazione "`legge da`" e le stesse scritture finali.
Uno scheduler è **view-serializzabile** se è *view-equivalente* ad uno scheduler seriale.
PROBLEMA: stabilire view-serializzabilità di uno scheduler è un problema NP-completo.

Scheduler Conflict-Serializzabili

In uno scheduler S due azioni distinte a_i ed a_j sono in conflitto tra loro se operano sullo stesso oggetto x ed almeno una è di **write**.
Due scheduler si dicono **conflict-equivalenti** se contengono le *stesse operazioni* ed i conflitti appaiono nello stesso ordine.
Uno scheduler è **conflict-serializzabile** se è *conflict-equivalente* ad uno scheduler seriale.
Per verificare la *conflict-serializzabilità* basta utilizzare un algoritmo su grafi $(grafo \Rightarrow (nodo = transazione)$, (arco orientato da T_i a T_j se c'è almeno un conflitto

tra una azione a_i ed una a_j)). Se il grafo ottenuto in questo modo è *aciclico*, lo scheduler è **conflict-serializzabile**.

(Se S è conflict-serializzabile esiste uno scheduler S' seriale che presenta operazioni in conflitto nello stesso ordine di S. Poiché S' avrà un ordinamento delle transizioni del tipo $T_1, T_2, \ldots, T_n$, il grafo associato ad S non presenterà cicli).
Nonostante una diminuzione a livello di costi computazionali, il modello basato su **conflict-serializzabilità** funziona *solo* nel caso in cui *tutte le transazioni terminano con una commit e non risulta quindi implementabile in realtà*. Come si procede allora?

Nei DBMS moderni vengono utilizzati protocolli di **locking** sulle transazioni.

MECCANISMO: i lock non sono nient'altro che *variabili* associate a singoli oggetti in memoria del DB che descrivono lo stato dell'oggetto stesso rispetto alle operazioni che possono essere effettuate su di esso.

Una transazione richiede un lock mediante una operazione di locking, e lo rilascia mediante una unlocking.

In questo modo implementiamo un meccanismo di mutua esclusione sulle informazioni mantenute in memoria. (Se una transizione prova a richiedere il lock di un certo oggetto, ma questo non risulta disponibile, la transizione rimarrà in attesa finchè questo non viene rilasciato $\Rightarrow$ il tutto è gestito mediante un'apposita tabella dei conflitti).

I protocolli di locking più utilizzati nei sistemi moderni sono:

- **Locking a due fasi**: una transazione non può acquisire più di una volta il lock di un dato oggetto
- Una sua variante, **locking a due fasi stretto**: i lock possono essere rilasciati da una transazione solo dopo una `commit/abort`. (Garantiamo isolamento completo delle transizioni, anche in caso di abort)

Nei sistemi basati su lock possono però verificarsi due **problematiche** fondamentali, che vanno risolte:

- *Livelock*: transazione può rimanere in attesa di un lock per un tempo indefinito. Possiamo risolvere utilizzando coda di priorità dove la priorità assegnata ad ogni transizione coincide con l'istante di richiesta del lock (in questo modo ogni transizione ad un certo avrà priorità massima per ottenere il lock)
- *Deadlock*: si ha quando due transizioni detengono ciascuna una risorsa e aspettano quella detenuta dell'altra. Si può risolvere facendo abortire le

transazioni.

Normalizzazione

Le forme normali nelle basi di dati (o normalizzazione dei database) sono una serie di linee guida progettate per organizzare i dati in una base di dati relazionale per minimizzare la ridondanza e migliorare l'integrità dei dati. La normalizzazione comporta la suddivisione delle tabelle grandi in tabelle più piccole e il collegamento tra di esse attraverso relazioni ben definite. Ecco una panoramica delle forme normali principali:

Prima Forma Normale (1NF)

Requisiti:

- Ogni colonna contiene solo valori atomici (indivisibili).
- Ogni colonna contiene solo valori di un singolo tipo.
- Ogni riga è unica e identificata da una chiave primaria.
  Obiettivo: Eliminare i gruppi ripetuti all'interno di una tabella.
  Esempio:

ID	Nome	Telefono
1	Anna	123-456, 789-012
2	Bob	345-678

Dopo la normalizzazione in 1NF:

ID	Nome	Telefono
1	Anna	123-456
1	Anna	789-012
2	Bob	345-678

Seconda Forma Normale (2NF)

Requisiti:

- Deve essere già in 1NF.

- Tutte le colonne non chiave devono dipendere interamente dalla chiave primaria.
  Obiettivo: Eliminare la dipendenza parziale delle colonne non chiave dalla chiave primaria.
  Esempio: Supponiamo una tabella che contiene informazioni sui corsi e sugli studenti iscritti.

StudenteID	CorsoID	NomeStudente	NomeCorso
1	101	Anna	Matematica
2	102	Bob	Fisica

Dopo la normalizzazione in 2NF:

Tavola Studenti:

StudenteID	NomeStudente
1	Anna
2	Bob

Tavola Corsi:

CorsoID	NomeCorso
101	Matematica
102	Fisica

Tavola Iscrizioni:

StudenteID	CorsoID
1	101
2	102

Terza Forma Normale (3NF)

Requisiti:

- Deve essere già in 2NF.
- Tutte le colonne non chiave devono essere indipendenti tra loro (nessuna dipendenza transitiva).
  Obiettivo: Eliminare la dipendenza transitiva delle colonne non chiave dalla chiave primaria.

Esempio: Supponiamo una tabella con informazioni sugli studenti e sulle città in cui vivono:

StudenteID	NomeStudente	Città	Stato
1	Anna	Milano	Lombardia
2	Bob	Roma	Lazio

Dopo la normalizzazione in 3NF:

Tavola Studenti:

StudenteID	NomeStudente	CittàID
1	Anna	1
2	Bob	2

Tavola Città:

CittàID	Città	Stato
1	Milano	Lombardia
2	Roma	Lazio

Forma Normale di Boyce-Codd (BCNF)

La Forma Normale di Boyce-Codd (BCNF) è una versione più rigorosa della Terza Forma Normale (3NF). È stata introdotta per risolvere alcuni problemi che possono rimanere nelle tabelle anche dopo la normalizzazione in 3NF. BCNF garantisce una migliore eliminazione delle anomalie di aggiornamento e delle ridondanze rispetto alla 3NF.

Requisiti della BCNF

Una tabella è in BCNF se e solo se per ogni dipendenza funzionale non banale $A \rightarrow B$, A è una superchiave.

Definizione:

- Una dipendenza funzionale $A \rightarrow B$ è non banale se B non è un sottoinsieme di A.
- Un attributo o un insieme di attributi A è una superchiave se A determina univocamente ogni attributo nella tabella.

Esempio di BCNF

Consideriamo una tabella di un'università con le seguenti colonne:

StudenteID	CorsoID	Professore	Aula
1	101	Rossi	10
2	102	Bianchi	20
3	101	Rossi	10

In questo esempio, potremmo avere le seguenti dipendenze funzionali:

1. (StudenteID,CorsoID)→Professore
2. Professore→Aula

Per controllare se la tabella è in BCNF, dobbiamo verificare se per ogni dipendenza funzionale non banale, l'insieme di attributi a sinistra è una superchiave.

Verifica della BCNF

1. (StudenteID,CorsoID)→Professore(StudenteID, CorsoID) \rightarrow Professore(StudenteID,CorsoID)→Professore
   - (StudenteID,CorsoID)(StudenteID, CorsoID)(StudenteID,CorsoID) è una superchiave perché determina univocamente ProfessoreProfessoreProfessore (e in effetti tutti gli altri attributi della tabella).
2. Professore→AulaProfessore \rightarrow AulaProfessore→Aula
   - ProfessoreProfessoreProfessore non è una superchiave, perché non determina univocamente tutti gli attributi della tabella (ad esempio, non determina StudenteIDStudenteIDStudenteID o CorsoIDCorsoIDCorsoID).

Quindi, la tabella non è in BCNF a causa della dipendenza funzionale Professore→AulaProfessore \rightarrow AulaProfessore→Aula.

Come rendere una tabella in BCNF

Per portare una tabella in BCNF, dobbiamo decomporre la tabella in modo da eliminare le dipendenze funzionali che violano la BCNF.

Decomposizione:

1. Creiamo una nuova tabella per la dipendenza Professore→AulaProfessore \rightarrow AulaProfessore→Aula:

Tabella Professori:
Professore	Aula
Rossi	10
Bianchi	20

2. Modifichiamo la tabella originale per rimuovere l'attributo AulaAulaAula:

   Tabella Iscrizioni:

StudenteID	CorsoID	Professore
1	101	Rossi
2	102	Bianchi
3	101	Rossi

Ora entrambe le tabelle sono in BCNF:

- Nella tabella **Professori**, ProfessoreProfessoreProfessore è una superchiave e determina univocamente AulaAulaAula.
- Nella tabella **Iscrizioni**, (StudenteID,CorsoID)(StudenteID, CorsoID) (StudenteID,CorsoID) è una superchiave e determina univocamente ProfessoreProfessoreProfessore.

Vantaggi della BCNF

- **Elimina Anomalie:** Riduce al minimo le anomalie di inserimento, aggiornamento e cancellazione.
- **Integrità dei Dati:** Mantiene l'integrità dei dati e riduce le ridondanze.
- **Consistenza:** Garantisce una maggiore consistenza dei dati attraverso la rigorosa eliminazione delle dipendenze funzionali non necessarie.

In sintesi, la Forma Normale di Boyce-Codd è uno step avanzato nella normalizzazione dei database che garantisce che tutte le dipendenze funzionali siano gestite correttamente, migliorando l'integrità e la coerenza dei dati.

Dipendenze funzionali

Le dipendenze funzionali sono un concetto chiave nella teoria delle basi di dati relazionali, utilizzato per esprimere le relazioni tra attributi in una tabella. Comprendere le dipendenze funzionali è essenziale per la normalizzazione del database e per garantire l'integrità dei dati. Ecco una spiegazione dettagliata delle dipendenze funzionali:

Definizione di Dipendenza Funzionale

Una dipendenza funzionale tra due insiemi di attributi di una relazione è una relazione in cui un insieme di attributi determina un altro insieme di attributi. Formalmente, si dice che un insieme di attributi X determina un insieme di attributi Y (notato come $X \rightarrow Y$) se, per ogni coppia di tuple in una relazione, se le tuple concordano su tutti gli attributi di X, allora devono concordare anche su tutti gli attributi di Y.

Esempio di Dipendenza Funzionale

Consideriamo una tabella di studenti con le seguenti colonne:

Matricola	Nome	Corso	Professore
1	Anna	Matematica	Rossi
2	Bob	Fisica	Bianchi
3	Carla	Matematica	Rossi

In questo esempio:

- Matricola $\rightarrow$ Nome significa che ogni matricola identifica univocamente il nome di uno studente.
- Corso $\rightarrow$ Professore significa che ogni corso è tenuto da un unico professore.

Tipi di Dipendenze Funzionali

Dipendenza Funzionale Completa

Una dipendenza funzionale $X \rightarrow Y$ è completa se la rimozione di qualsiasi attributo da X fa sì che la dipendenza non sia più valida. In altre parole, tutti gli attributi di X sono necessari per determinare Y.

Esempio:

(Matricola, Corso) → Voto	
1, Matematica → 30	
2, Fisica → 28	

In questo caso, solo conoscendo sia la matricola che il corso possiamo determinare univocamente il voto di uno studente in quel corso.

Dipendenza Funzionale Parziale

Una dipendenza funzionale $X \rightarrow Y$ è parziale se esiste un sottoinsieme proprio di X che determina ancora Y.

Esempio:

(Matricola, Nome) → Corso	
1, Anna → Matematica	
2, Bob → Fisica	

Qui, la dipendenza è parziale perché il nome da solo può determinare il corso.

Dipendenza Transitiva

Una dipendenza funzionale $X \rightarrow Z$ è transitiva se esistono attributi Y tali che $X \rightarrow Y$ e $Y \rightarrow Z$.

Esempio:

| Matricola → Corso |

Corso → Professore	
1 → Matematica	
Matematica → Rossi	

In questo caso, $Matricola \rightarrow Corso \rightarrow Professore$ rappresenta una dipendenza transitiva.

Importanza delle Dipendenze Funzionali

Le dipendenze funzionali sono utilizzate per:

1. **Normalizzazione:** Aiutano a identificare le anomalie di aggiornamento e a eliminare le ridondanze nei dati.
2. **Progettazione del Database:** Garantiscono che la struttura del database sia ottimale per mantenere l'integrità e l'efficienza dei dati.
3. **Integrità dei Dati:** Assicurano che le relazioni tra i dati siano mantenute correttamente, prevenendo inconsistenze.

Verifica delle Dipendenze Funzionali

Per verificare se una dipendenza funzionale $X \rightarrow Y$ è valida in una relazione, si esaminano tutte le tuple della relazione per vedere se per ogni coppia di tuple, se le tuple hanno gli stessi valori per X, allora devono avere gli stessi valori per Y.

In conclusione, le dipendenze funzionali sono fondamentali per capire come gli attributi di una base di dati sono correlati tra loro e per garantire che i dati siano organizzati in modo da ridurre le ridondanze e mantenere l'integrità.

Algebra Relazionale

L'algebra relazionale è un insieme di operazioni utilizzate per manipolare e interrogare i dati nelle basi di dati relazionali. Queste operazioni sono formali e forniscono un modo per esprimere le query su una base di dati relazionale. L'algebra relazionale è fondamentale per la teoria dei database relazionali e costituisce la base per SQL, il linguaggio di query più comunemente utilizzato.

Operazioni di Base dell'Algebra Relazionale

Le operazioni dell'algebra relazionale si suddividono in due categorie principali: operazioni di base (primitive) e operazioni derivate.

1. Selezione (σ)

La selezione è un'operazione unaria che restituisce un sottoinsieme delle tuple di una relazione che soddisfano una certa condizione.

Notazione: $\sigma_{\text{condizione}}(R)$

Esempio:

```SQL
σ_{salario > 50000}(Dipendenti)
```

Questa operazione restituisce tutte le tuple della relazione Dipendenti dove il salario è maggiore di 50000.

2. Proiezione (π)

La proiezione è un'operazione unaria che restituisce una nuova relazione contenente solo le colonne specificate.

Notazione: $\pi_{\text{attributi}}(R)$
Esempio:

```SQL
π_{nome, salario}(Dipendenti)
```

Questa operazione restituisce una nuova relazione con solo le colonne nome e salario dalla relazione Dipendenti.

3. Unione (U)

L'unione è un'operazione binaria che restituisce una relazione contenente tutte le tuple di due relazioni, eliminando le duplicazioni.

Notazione: $R \cup S$
Esempio:

```SQL
Dipendenti_{Italia} ∪ Dipendenti_{Francia}
```

Questa operazione restituisce una relazione contenente tutte le tuple dei dipendenti in Italia e Francia.

4. Differenza (−)

La differenza è un'operazione binaria che restituisce una relazione contenente tutte le tuple che sono in una relazione ma non nell'altra.

Notazione: $R - S$
Esempio:

```SQL
Dipendenti_{Italia} - Dipendenti_{Francia}
```

Questa operazione restituisce una relazione contenente tutte le tuple dei dipendenti in Italia che non sono in Francia.

5. Prodotto Cartesiano (×)

Il prodotto cartesiano è un'operazione binaria che restituisce una nuova relazione combinando tutte le tuple di due relazioni.

Notazione: $R \times S$
Esempio:

```SQL
Dipendenti × Progetti
```

Questa operazione restituisce una relazione contenente tutte le possibili combinazioni di tuple da Dipendenti e Progetti.

6. Rinominazione (ρ)

La rinominazione è un'operazione unaria che cambia il nome degli attributi di una relazione.

Notazione: $\rho_{S(A_1,A_2,\ldots,A_n)}(R)$
Esempio:

```SQL
ρ_{D(Nome, Salario)}(Dipendenti)
```

Questa operazione rinomina la relazione Dipendenti in D e i suoi attributi in Nome e Salario.

Operazioni Derivate dell'Algebra Relazionale

Oltre alle operazioni di base, ci sono diverse operazioni derivate che possono essere definite usando le operazioni di base.

1. Intersezione ($\cap$)

L'intersezione restituisce una relazione contenente tutte le tuple che sono comuni a due relazioni.

Notazione: $R \cap S$
Esempio:

```SQL
Dipendenti_{Italia} ∩ Dipendenti_{Francia}
```

Questa operazione restituisce una relazione contenente tutte le tuple dei dipendenti che sono sia in Italia che in Francia.

2. Join Naturale ($\bowtie$)

Il join naturale è un'operazione binaria che combina due relazioni basate su attributi comuni.

Notazione: $R \bowtie S$
Esempio:

```SQL
Dipendenti ⋈ Progetti
```

Questa operazione combina le tuple di Dipendenti e Progetti dove i valori degli attributi comuni corrispondono.

3. Divisione ($\div$)

La divisione è un'operazione binaria che restituisce una relazione contenente le tuple di una relazione che sono associate a tutte le tuple di un'altra relazione.

Notazione: $R \div S$
Esempio:

```SQL
Dipendenti \div Progetti
```

Questa operazione restituisce i dipendenti che lavorano su tutti i progetti.

Calcolo Relazionale

Il calcolo relazionale è un altro approccio formale per esprimere query su basi di dati relazionali, complementare all'algebra relazionale. Mentre l'algebra relazionale usa un insieme di operazioni per costruire query, il calcolo relazionale si basa sulla logica del primo ordine, usando formule per specificare le condizioni che le tuple devono soddisfare. Ci sono due varianti principali del calcolo relazionale: il calcolo relazionale sui tuple (TRC) e il calcolo relazionale sui domini (DRC).

Calcolo Relazionale sui Tuple (TRC)

Il calcolo relazionale sui tuple (TRC) utilizza variabili che rappresentano tuple in una relazione. Le query in TRC specificano le proprietà che le tuple di risultato devono soddisfare.

Sintassi Generale: $\{t \mid \varphi(t)\}$
Dove:

- t è una variabile che rappresenta una tuple.
- $\varphi(t)$ è una formula logica che specifica le condizioni che t deve soddisfare.

Esempio: Consideriamo una relazione Dipendenti con attributi (DID,Nome,Salario).

Trova i nomi e i salari dei dipendenti che guadagnano più di 50000.

$\{t.Nome, t.Salario \mid t \in Dipendenti \land t.Salario > 50000\}$

Calcolo Relazionale sui Domini (DRC)

Il calcolo relazionale sui domini (DRC) utilizza variabili che rappresentano valori singoli, piuttosto che tuple. Le query in DRC specificano le proprietà che i valori di dominio devono soddisfare.

Sintassi Generale: $\{(x_1, x_2, \ldots, x_n) \mid \varphi(x_1, x_2, \ldots, x_n)\}$
Dove:

- $x_1, x_2, \ldots, x_n$ sono variabili che rappresentano valori singoli.
- $\varphi(x_1, x_2, \ldots, x_n)$ è una formula logica che specifica le condizioni che questi valori devono soddisfare.

Esempio: Consideriamo la stessa relazione Dipendenti(DID,Nome,Salario).

Trova i nomi e i salari dei dipendenti che guadagnano più di 50000.

$\{(N, S) \mid \exists DID\,(Dipendenti(DID, N, S) \land S > 50000)\}$

Differenze tra TRC e DRC

- **Rappresentazione delle Variabili:** Nel TRC, le variabili rappresentano tuple, mentre nel DRC, le variabili rappresentano valori singoli.
- **Sintassi:** TRC utilizza variabili di tuple e formule che descrivono condizioni sulle tuple, mentre DRC utilizza variabili di dominio e formule che descrivono condizioni sui valori di dominio.

Sicurezza delle Query

In entrambi i tipi di calcolo relazionale, è importante garantire che le query siano sicure, ovvero che restituiscano un insieme finito di risultati e che il calcolo possa essere eseguito in un tempo finito. Le query non sicure possono causare problemi di performance e di gestione delle risorse.

Potenza espressiva

Il calcolo relazionale e l'algebra relazionale hanno la stessa potenza espressiva, il che significa che qualsiasi query esprimibile in uno può essere espressa anche nell'altro. Tuttavia, il calcolo relazionale tende a essere più dichiarativo,

specificando cosa si desidera ottenere piuttosto che come ottenerlo, mentre l'algebra relazionale è più procedurale.