

Basi di dati

VR443470

marzo 2023

Indice

1 Introduzione	5
1.1 Sistemi informativi, informazioni e dati	5
1.2 Basi di dati e sistemi di gestione di basi di dati	6
1.3 Linguaggi per basi di dati	7
1.4 Modelli dei dati	8
1.5 Astrazione (architettura) dei DBMS	9
1.6 Indipendenza dei dati	10
2 Metodologie e modelli per il progetto	11
2.1 Ciclo di vita dei sistemi informativi	11
2.2 Metodologie di progettazione e basi di dati	13
2.3 Il modello Entità-Relazione (E-R)	14
2.4 I costrutti principali del modello	15
2.4.1 Entità	15
2.4.2 Relazioni (o associazioni)	16
2.4.3 Attributi	17
2.5 Altri costrutti del modello	18
2.5.1 Cardinalità delle relazioni	18
2.5.2 Cardinalità degli attributi	19
2.5.3 Identificatori	20
2.5.4 Generalizzazioni	22
3 Progettazione concettuale	24
3.1 Strategie di progetto	24
3.1.1 Strategia top-down	24
3.1.2 Strategia bottom-up	25
3.1.3 Strategia inside-out	26
3.2 Qualità di uno schema concettuale	27
3.2.1 Correttezza	27
3.2.2 Completezza	27
3.2.3 Leggibilità	27
3.2.4 Minimalità	28
4 Progettazione logica	29
4.1 Fasi della progettazione logica	29
4.2 Traduzione verso il modello logico	30
4.2.1 Entità e attributo opzionale	30
4.2.2 Relazione uno a molti	31
4.2.3 Relazione uno a uno	32
4.2.4 Relazione molti a molti	33
4.2.5 Relazione uno a molti (identificatore esterno)	34
4.2.6 Relazione uno a molti (attributo sulla relazione)	35
4.2.7 Relazione uno a uno (una cardinalità minima a zero)	36
4.2.8 Relazione uno a uno (entrambe cardinalità minima a zero)	37
4.2.9 Relazione molti a molti (attributo sulla relazione)	38
4.2.10 Relazione molti a molti (identificatori con più attributi)	39
4.2.11 Relazione molti a molti (relazione ternaria)	40

4.2.12 Relazione molti a molti (relazione ternaria e cardinalità uno a uno)	41
5 Algebra relazionale	42
5.1 Insiemistici	43
5.1.1 Unione	43
5.1.2 Intersezione	44
5.1.3 Differenza	44
5.2 Specifici	45
5.2.1 Ridenominazione	45
5.2.2 Selezione	46
5.2.3 Proiezione	48
5.3 Join	50
5.3.1 Join naturale	50
5.3.2 Join completi e incompleti	52
5.3.3 Theta-join ed equi-join	55
5.4 Algebra con valori nulli	58
5.5 Ottimizzare ed equivalenza di espressioni algebriche	60
5.5.1 Definizioni	60
5.5.2 Trasformazioni di equivalenza	61
6 Calcolo relazionale	64
6.1 Calcolo su tuple con dichiarazioni di range	65
6.2 Unione, intersezione e differenza	66
6.3 Esempi	67
6.4 Esercizi	69
6.4.1 Aula, insegnamento, docente e lezione	69
7 Tecnologie per le basi di dati	73
7.1 Transazioni	73
7.2 Transazioni in SQL	73
7.3 Proprietà acide delle transazioni	74
7.3.1 Atomicità	74
7.3.2 Consistenza	75
7.3.3 Isolamento	75
7.3.4 Persistenza	75
7.4 Architettura di riferimento di un DBMS	76
7.4.1 Gestore (ottimizzatore) delle interrogazioni	77
7.4.2 Gestore dei metodi d'accesso e dell'esecuzione concorrente	77
7.4.3 Gestore dei metodi d'accesso e dell'affidabilità	78
7.4.4 Gestore dell'affidabilità e del buffer	78
8 PostgreSQL	79
8.1 Introduzione e installazione	79
8.2 Caratteri (character, character varying, text)	81
8.3 Booleani (boolean)	82
8.4 Tipi numerici esatti (numeric, decimal, integer, smallint, bigint)	82
8.5 Tipi numerici approssimati (float, real, double precision)	83
8.6 Istanti e intervalli temporali (date, time, timestamp, interval)	83
8.7 Oggetti di grandi dimensioni (blob, clob)	84

8.8	Definizione delle tabelle (<i>create table</i>)	84
8.9	Definizione dei domini (<i>create domain</i>)	85
8.10	Valori di default (<i>default</i>)	85
8.11	Vincoli intrarelazionali	86
8.11.1	Not null	86
8.11.2	Unique	86
8.11.3	Primary key	86

1 Introduzione

1.1 Sistemi informativi, informazioni e dati

Ogni organizzazione è dotata di un *sistema informativo*, che organizza e gestisce le informazioni necessarie per perseguire gli scopi dell'organizzazione stessa. Per indicare la **porzione automatizzata del sistema informativo** viene di solito utilizzato il termine *sistema informatico*.

Nei sistemi informatici le informazioni vengono rappresentate per mezzo di *dati*, che hanno bisogno di essere interpretati per fornire informazioni. Esiste una differenza sottile tra dato e informazioni. Solitamente i primi, se presi da soli, non hanno significato, ma, una volta interpretati e correlati opportunamente, essi forniscono informazioni, che consentono di arricchire la conoscenza:

Informazione: notizia, dato o elemento che consente di avere conoscenza più o meno esatta di fatti, situazioni, modi di essere;

Dato: ciò che è immediatamente presente alla conoscenza, prima di ogni elaborazione. In informatica, sono elementi di informazione costituiti da simboli che devono essere elaborati.

[ESAME] Definizione base di dati: Una *base di dati* è una collezione di dati, utilizzati per rappresentare con tecnologia informatica le informazioni di interesse per un sistema informativo.

1.2 Basi di dati e sistemi di gestione di basi di dati

Inizialmente, venne adottato un “approccio convenzionale” alla gestione dei dati. Esso **sfruttava** la presenza di archivi o **file per memorizzare e per ricercare dati**. Tuttavia, i metodi di accesso e condivisione erano semplici e banali. Infatti, erano presenti numerosi **problemi**:

- ✗ **Accesso sequenziale:** la scarsa efficienza nell’accesso ai dati su file rendeva lento l’accesso a tali informazioni;
- ✗ **Ridondanza:** i dati di interesse per più programmi sono replicati tante volte quanti sono i programmi che li utilizzano, con evidente ridondanza e possibilità di incoerenza;
- ✗ **Inconsistenza:** una diretta conseguenza della ridondanza. Con la presenza di più copie di un determinato dato, l’eventuale cambiamento di uno solo potrebbe portare a questo effetto;
- ✗ **Progettazione duplicata:** per ogni programma viene replicata la progettazione.

La **soluzione** è arrivata negli anni ’80 con l’avvento delle **basi di dati**. Quest’ultime gestiscono in modo integrato e flessibile le informazioni di interesse per diversi soggetti.

[ESAME] Definizione DBMS: Un *sistema di gestione di basi di dati* (in inglese *Data Base Management System*, **DBMS**) è un sistema software in grado di gestire collezioni di dati che siano:

- ✓ **Grandi;**
- ✓ **Condivise;**
- ✓ **Persistenti.**

assicurando allo stesso tempo:

- ★ **Affidabilità;**
- ★ **Privatezza;**
- ★ **Accesso efficiente.**

Il **vantaggio** di utilizzare un DBMS è stato evidenziato nella definizione. Quindi:

- ✓ **Maggiore astrazione** poiché le sue funzioni estendono il *file system*, fornendo la possibilità di accesso condiviso agli stessi dati da parte di più utenti e applicazioni;
- ✓ **Maggiore efficacia** poiché le operazioni di accesso ai dati si basano su un linguaggio di interrogazione.

1.3 Linguaggi per basi di dati

Su un DBMS è possibile specificare operazioni di vario tipo, ma principalmente si distinguono in due categorie:

- **Linguaggi di definizione dei dati** (*Data Definition Language*, abbreviato con **DDL**) utilizzati per definire gli schemi logici, esterni e fisici e le autorizzazioni per l'accesso;
- **Linguaggi di manipolazione dei dati** (*Data Manipulation Language*, abbreviato con **DML**) utilizzati per l'interrogazione e l'aggiornamento delle istanze di basi di dati:
 - *Linguaggio di interrogazione*, estrae informazioni da una base di dati (SQL, algebra relazionale);
 - *Linguaggio di manipolazione*, popola la base di dati, modifica il suo contenuto con aggiunte, cancellazioni e variazioni sui dati (SQL).

1.4 Modelli dei dati

Definizione modello dei dati: Un *modello dei dati* è un insieme di concetti utilizzati per organizzare i dati di interesse e descriverne la struttura in modo che essa risulti comprensibile ad un elaboratore.

Ogni modello dei dati fornisce **meccanismi di strutturazione**, analoghi ai **costruttori** di tipo dei linguaggi di programmazione (es: Java), che permettono di definire nuovi tipi sulla base di tipi predefiniti (elementari) e costruttori di tipo. Quindi, i *costruttori* consentono di:

- ☞ **Definire** le strutture dati che conterranno le informazioni della base di dati;
- ☞ **Specificare** le proprietà che dovranno soddisfare le istanze di informazione che saranno contenute nelle strutture dati.

Definizione schemi e istanze: È molto importante distinguere gli **schemi** e le **istanze** dal concetto di modello dei dati:

- **Schema:** parte invariante nel tempo, è costituita dalle caratteristiche dei dati. In altre parole, è la descrizione della struttura e delle proprietà di una specifica base di dati fatta utilizzando i costruttori del modello dei dati;
- **Istanza o stato:** parte variabile nel tempo, è costituita dai valori effettivi. Quest'ultimi, in un certo istante, popolano le strutture dati della base di dati.

Modello dei dati	Schema	Istanza						
Basi di dati Tabella (o relazione)	P(cognome: VARCHAR(40), nome: VARCHAR(30))	<table border="1"> <thead> <tr> <th>cognome</th><th>nome</th></tr> </thead> <tbody> <tr> <td>Rossi</td><td>Mario</td></tr> <tr> <td>Bianchi</td><td>Lisa</td></tr> </tbody> </table>	cognome	nome	Rossi	Mario	Bianchi	Lisa
cognome	nome							
Rossi	Mario							
Bianchi	Lisa							
Linguaggi di progr. Array	<pre>Class Persona { String cognome; String nome; } Class X { ... Persona[] p; p = new Persona[100]; }</pre>							

Figura 1: Esempio di modello di dati, schema e istanza.

1.5 Astrazione (architettura) dei DBMS

Esiste un'architettura standardizzata per i DMBS, la quale si caratterizza su tre livelli: **esterno**, **logico** e **interno**:

- ★ **Schema logico.** È la rappresentazione della **struttura** e delle **proprietà** della **base di dati** definita attraverso i costrutti del modello dei dati del DBMS. In altre parole, descrive l'intera base di dati per mezzo del modello logico adottato dal DBMS (quindi relazione o ad oggetti).
- ★ **Schema interno.** È la rappresentazione della base di dati per mezzo delle **strutture fisiche di memorizzazione** (e.g. file sequenziale, file hash, ecc.).
- ★ **Schema esterno.** Descrive una **porzione dello schema logico** di interesse per uno **specifico** utente o applicazione. Possono esistere più schemi esterni che consentono di avere punti di vista differenti senza cambiare la logica di base.

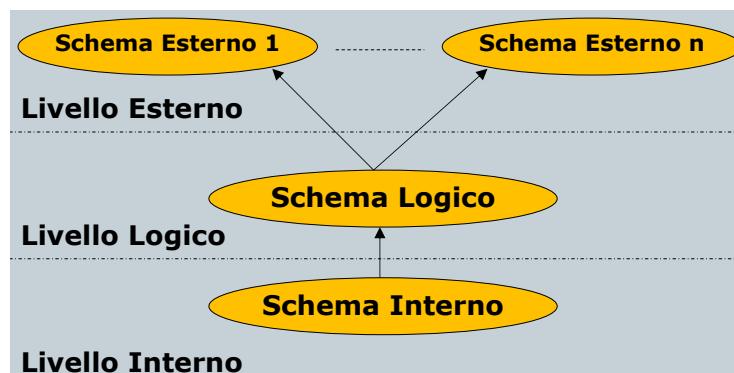


Figura 2: Architettura generale di un DBMS.

1.6 Indipendenza dei dati

L'architettura a livelli definita nel paragrafo 1.5 garantisce l'**indipendenza dei dati**, la **proprietà più importante** dei DBMS. L'**obbiettivo** è quello di poter fornire all'utente una basi di dati in grado di interagire con un elevato livello di astrazione. Esistono due tipi di indipendenza:

- ☛ **Indipendenza fisica.** Lo schema logico della basi di dati è completamente indipendente dallo schema interno. Quindi, l'interazione con il DBMS può essere effettuato in modo indipendente dalla struttura fisica dei dati.
Vantaggio: le modifiche non influiscono sullo schema logico, cioè sulle applicazioni che lo utilizzano.
- ☛ **Indipendenza logica.** Gli schemi esterni (definizione nel paragrafo: 1.5) della base di dati sono indipendenti dallo schema logico. Quindi, è possibile interagire con il livello esterno in modo indipendente dal livello logico.
Vantaggio:
 - I **Aggiunta/Modifica** di uno schema esterno in base alle esigenze di un nuovo utente, senza modificare lo schema logico;
 - II **Modifica** di uno schema logico mantenendo inalterate le strutture esterne.

2 Metodologie e modelli per il progetto

2.1 Ciclo di vita dei sistemi informativi

La progettazione di una base di dati costituisce solo una delle componenti del processo di sviluppo di un sistema informativo complesso e va quindi inquadrata in un contesto più ampio quello del **ciclo di vita** dei sistemi informativi:

- ☛ **Studio di fattibilità.** Definisce i costi delle varie alternative possibili e stabilisce le priorità di realizzazione delle varie componenti del sistema.
- ☛ **Raccolta e analisi dei requisiti.** Individua le proprietà e le funzionalità che il sistema informativo deve avere producendo una descrizione completa, ma generalmente informale.
- ☛ **Progettazione.** Si divide in due fasi:
 - **Progettazione dei dati.** Individua la struttura e l'organizzazione che i dati devono avere.
 - **Progettazione delle applicazioni.** Definizione delle caratteristiche dei programmi applicativi.
- ☛ **Implementazione (su un DBMS).** È la realizzazione del sistema informativo secondo la struttura e le caratteristiche fornite durante la fase di progettazione. In questa fase viene costruita e popola la base di dati.
- ☛ **Validazione e collaudo.** Verifica il corretto funzionamento e la qualità del sistema informativo.
- ☛ **Funzionamento.** Il sistema informativo diventa operativo ed esegue i compiti per i quali è stato progettato.

Spesso il processo **non** è strettamente sequenziale. Infatti, come si vede dalla seguente figura, durante l'esecuzione di una delle attività sopraelencate, è necessario rivedere decisioni prese nell'attività precedente.

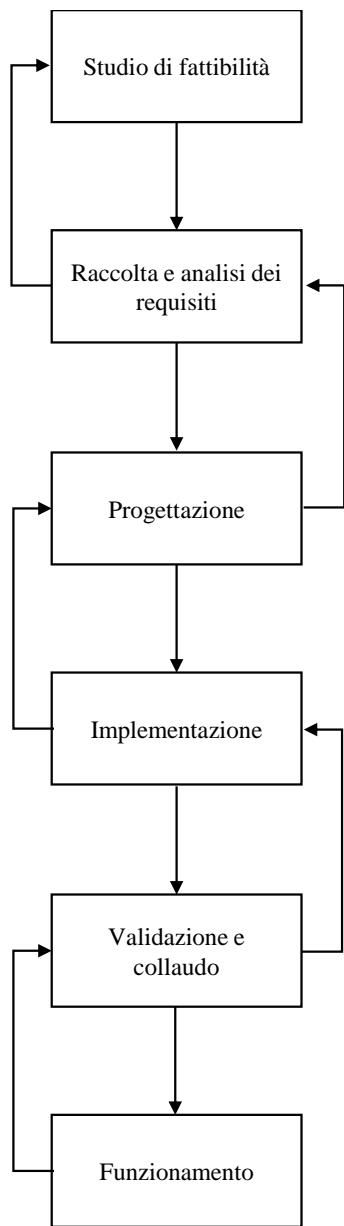


Figura 3: Ciclo di vita di un sistema informativo.

2.2 Metodologie di progettazione e basi di dati

Una **metodologia di progettazione** consiste in:

- ✓ **Decomposizione** dell'intera attività di progetto in passi successivi indipendenti tra loro.
- ✓ **Strategie** da seguire nei vari passi e **criteri** nel caso di alternative.
- ✓ **Modelli di riferimento** per descrivere i dati in ingresso e uscita delle varie fasi.

Le **proprietà** che una metodologia deve garantire sono:

- ★ **Generalità** rispetto alle applicazioni e ai sistemi in gioco;
- ★ **Qualità del prodotto** in termini di correttezza, completezza ed efficienza rispetto alle risorse impiegate;
- ★ **Facilità d'uso** delle strategie e dei modelli di riferimento.

Negli anni si è *consolidata una metodologia* di progetto che ha dato prova di soddisfare pienamente le proprietà descritte. Si basa sull'idea di separare le decisioni relative a "cosa" rappresentare in una base di dati (prima fase), da quelle relative a "come" farlo (seconda e terza fase):

● Progettazione concettuale.

Obiettivo: rappresentare le specifiche informali della realtà di interesse in termini di una descrizione formale e completa. La **rappresentazione deve essere indipendente** dai criteri di rappresentazione utilizzati nei sistemi di gestione di basi di dati.

Prodotto di questa fase: schema concettuale. È un documento formale che rappresenta il contenuto della base di dati in modo indipendente dall'implementazione (DBMS).

Applicazione: cercare di rappresentare il **contenuto informativo** della base di dati, senza preoccuparsi né della modalità con le quali queste informazioni verranno codificate in un sistema reale, né dell'efficienza dei programmi che faranno uso di queste informazioni.

● Progettazione logica.

Obiettivo: traduzione dello schema concettuale prodotto nella fase precedente, in termini del modello di rappresentazione dei dati adottato dal sistema di gestione di base di dati a disposizione.

Prodotto di questa fase: schema logico.

Applicazione: durante la traduzione, le scelte progettuali si devono basare anche su criteri di ottimizzazione delle operazioni da effettuare sui dati.

● Progettazione fisica.

Obiettivo: lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione dei dati.

Prodotto di questa fase: schema fisico.

2.3 Il modello Entità-Relazione (E-R)

Il **modello Entità-Relazione** è un modello **concettuale** di dati, quindi utilizzato nella **progettazione concettuale**, e fornisce una serie di strutture, chiamati **costrutti**, atte a descrivere la realtà di interesse in una maniera facile da comprendere e che prescinde dai criteri di organizzazione dei dati nei calcolatori.

I costrutti vengono utilizzati per **definire schemi** che descrivono l'**organizzazione e la struttura delle occorrenze** (o **istanze**) dei **dati**, ovvero, dei valori assunti dai dati al variare del tempo.

Si possono riassumere le caratteristiche del modello Entità-Relazione:

- ☛ **Modello concettuale.** Utilizzato durante la progettazione concettuale (definizione al paragrafo 2.2) di una base di dati.
- ☛ **Strumenti formali.** Vengono messi a disposizione diversi strumenti per definire la **struttura** e le **proprietà** di una base di dati (*esempio i costrutti*).
- ☛ **Indipendente dalla tecnologia.** Essendo un modello astratto, l'obiettivo è quello di definire la struttura e le proprietà della base di dati (non di implementarla!).
- ☛ **Formale.** È facile da utilizzare nonostante non ammetta ambiguità.
- ☛ **Grafico.** La sintassi è prettamente grafica e questo aumenta anche la leggibilità.

2.4 I costrutti principali del modello

Si analizzano i principali costrutti di questo modello: entità (pagina 15), relazioni (pagina 16) e attributi (pagina 17).

2.4.1 Entità

Definizione. Rappresentano **classi di oggetti** (per esempio, fatti, cose, persone) che hanno proprietà comuni ed esistenza “autonoma” ai fini dell’applicazione di interesse. Per esempio, “città, dipartimento, impiegato, acquisto e vendita” sono entità di un’applicazione aziendale. Inoltre, **ogni entità ha un nome identificativo**, il quale deve essere **univoco**. In sintesi:

- Hanno proprietà comuni;
- Hanno esistenza autonoma;
- Hanno identificazione univoca.

Sintassi grafica.



Figura 4: Sintassi grafica dell’entità.

Istanza (o occorrenza). Un’istanza (o occorrenza) di un’entità è un **oggetto della classe che l’entità rappresenta**. Le città di Roma, Milano e Palermo sono esempi di occorrenze dell’entità “Città”.

Attenzione! L’istanza di un’entità *non è un valore* che identifica un oggetto (per esempio, il cognome dell’impiegato o il suo codice fiscale), *ma è l’oggetto stesso* (l’impiegato in “carne e ossa”). Quindi, un’istanza ha un’esistenza indipendente dalle proprietà a esso associate.

Un’istanza dell’entità E è un oggetto appartenente alla classe rappresentata da E . Si indica con $I(E)$ l’insieme delle istanze di E che esistono nella base di dati in un certo istante.

2.4.2 Relazioni (o associazioni)

Definizione. Rappresentano **legami logici**, significativi per l'applicazione di interesse, **tra due o più entità**. Per esempio, “Residenza” è una relazione che sussiste tra le entità “Città” e “Impiegato”. Nello schema E-R, **ogni relazione ha un nome identificativo univoco**.

Le relazioni possono essere di **tipo**:

- **Ricorsive**, ovvero **relazioni tra un'entità e se stessa**. Per esempio, la relazione “Collega” sull'entità “Impiegato” connette coppie di impiegati che lavorano insieme.
- **n-arie**, ovvero **relazioni che coinvolgono più di due entità**. Per esempio, la relazione “Fornitura” tra le tre entità “Fornitore, Prodotto e Dipartimento” descrive il fatto che un fornitore rifornisce un dipartimento di un certo prodotto.

Nota fondamentale: per eseguire una relazione, le entità devono essere tutte piene o con almeno un dato all'interno.

Sintassi grafica. Una relazione R si rappresenta nello schema con un **rombo** a cui si collegano attraverso linee spezzate le entità coinvolte nella relazione. Il nome della relazione viene scritto a fianco del rombo.

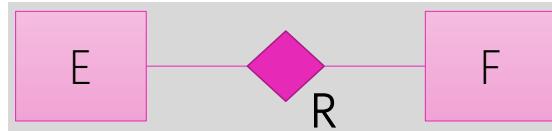


Figura 5: Sintassi grafica della relazione.

Istanza (o occorrenza). Un'istanza di relazione è un'**ennupla costituita da istanze di entità, una per ciascuna delle entità coinvolte**.

Data una relazione R tra n entità E_1, \dots, E_n , un'istanza della relazione R è una ennupla di istanze di entità:

$$(e_1, \dots, e_n) \text{ dove } e_i \in I(E_i), 1 \leq i \leq n$$

Infine, esiste una relazione importante. Data una relazione R tra n entità, vale sempre la seguente proprietà sull'insieme delle istanze di $R(I(R))$:

$$I(R) \subseteq I(E_1) \times \dots \times I(E_n)$$

2.4.3 Attributi

Definizione. Descrivono le **proprietà elementari** di entità o relazioni che sono di interesse ai fini dell'applicazione. Per esempio, "Cognome, Stipendio ed Età" sono possibili attributi dell'entità "Impiegato".

Un attributo associa a ciascun istanza di entità (o relazione) **uno e un solo** valore appartenente a un insieme, chiamato **dominio**, che contiene i valori ammissibili per l'attributo. Per esempio, l'attributo "Cognome" dell'entità "Impiegato" può avere come dominio l'insieme delle stringhe di 20 caratteri.

L'attributo può essere visto come una **funzione che ha come dominio le istanze dell'entità** (o relazione) e come **codominio l'insieme dei valori ammissibili**:

$$f_a : I(E) \rightarrow D$$

- a è un attributo dell'entità E ;
- $I(E)$ è l'insieme delle istanze di E ;
- D è l'insieme dei valori ammissibili.

Sintassi grafica.

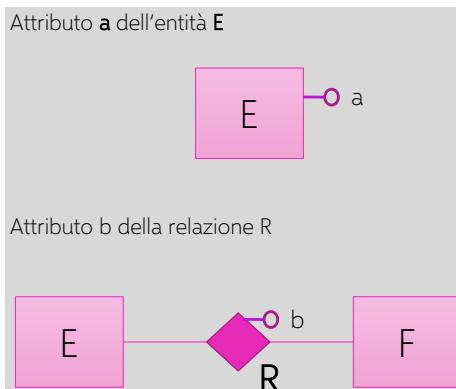


Figura 6: Sintassi grafica dell'attributo.

Istanza (o occorrenza). Dato un attributo a di un'entità E (o relazione R), un'istanza di a è il valore v che esso assume su un'istanza di E (o istanza di R).

Quindi, data un'istanza e dell'entità E (o relazione R), l'istanza di un suo attributo a si ottiene dalla funzione f_a applicata a e :

$$\text{valore di } a \text{ su } e = f_a(e)$$

Attributi composti.

Questo tipo di attributo viene introdotto solo a fini didattici, ma l'obiettivo è quello di usare unicamente gli attributi normali. Talvolta potrebbe tornare comodo raggruppare **attributi di una medesima entità o relazione che presentano affinità nel loro significato o uso**: tale insieme prende il nome di **attributo composto**. Per esempio, gli attributi "Via, Numero civico e CAP" dell'entità "Persona" per formare l'attributo composto "Indirizzo".

2.5 Altri costrutti del modello

I rimanenti costrutti del modello E-R sono le cardinalità delle relazioni e degli attributi e gli identificatori.

2.5.1 Cardinalità delle relazioni

Definizione. Le **cardinalità** vengono specificate per ciascuna entità collegata ad una relazione e descrivono il **numero minimo e massimo di occorrenze** di relazione a cui una occorrenza dell'entità può partecipare.

Più formalmente, data una relazione R i vincoli di cardinalità vengono specificati per ogni entità E_i coinvolta nella relazione R e specificano: il numero massimo e il numero minimo di istanze di R a cui un'istanza di E_i deve/può partecipare.

In parole povere, dicono quante volte, in una relazione tra entità, un'istanza di una di queste entità può essere legata a istanze delle altre entità coinvolte. **Per esempio**, in una relazione “Assegnamento” tra le entità “Impiegato” e “Incarnico” si specifica per la prima entità una cardinalità minima pari a uno e una cardinalità massima pari a cinque. Quindi, un impiegato può partecipare a un minimo di una occorrenza e a un massimo di cinque occorrenze della relazione “Assegnamento”.

N.B. Specificando **zero come cardinalità minima**, si impone che un'occocrenza può apparire oppure no.

È possibile **assegnare un qualunque valore intero non negativo a una cardinalità di una relazione con l'unico vincolo che la cardinalità minima deve essere minore o uguale della cardinalità massima**.

Tuttavia, nella maggior parte dei casi, è sufficiente utilizzare solamente tre simboli: 0, 1 e N (molti).

★ Cardinalità minima.

- **Zero.** La partecipazione dell'entità relativa è *opzionale*;
- **Uno.** La partecipazione dell'entità relativa è *obbligatoria*.

★ Cardinalità massima.

- **Uno.** La partecipazione dell'entità relativa è come una funzione che associa a una occorrenza dell'entità una sola occorrenza (o nessuna) dell'altra entità che partecipa alla relazione;
- **N (molti).** Esiste un'associazione con un numero arbitrario di occorrenze dell'altra entità.

Sintassi grafica.

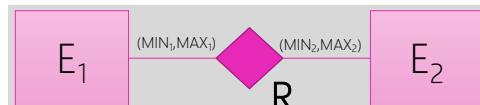


Figura 7: Sintassi grafica della cardinalità.

2.5.2 Cardinalità degli attributi

Definizione. Le **cardinalità degli attributi** è specificata per gli attributi di entità o relazione e hanno l'obiettivo di **descrivere il numero minimo e massimo di valori dell'attributo associati a ogni occorrenza di entità o relazione**.

Solitamente, il valore di cardinalità pari $(1, 1)$, ma si possono avere vari casi:

- $(1, 1)$ —> L'attributo rappresenta una funzione che associa ad ogni occorrenza di entità un solo valore dell'attributo. Solitamente viene omesso e, come si vede nell'immagine 8, la “Persona” ha uno e un solo “Cognome”;
- $(0, 1)$ —> L'attributo con cardinalità minima pari a zero vuol dire che è **opzionale** e la cardinalità massima pari a uno indica che nel **caso in cui esista**, questo valore è **unico**. Nell'esempio in figura 8, la persona può avere solo un numero di patente, ma potrebbe anche non avercela;
- $(1, N)$ —> L'attributo deve esistere, ma contiene più valori, quindi si dice che è **multivaleore**;
- $(0, N)$ —> L'attributo è opzionale, ma se esiste può essere multivaleore.

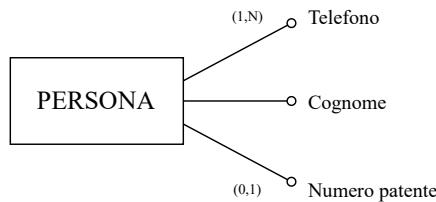


Figura 8: Esempio di cardinalità degli attributi.

2.5.3 Identificatori

Definizione. Vengono specificati per ciascuna entità di uno schema e descrivono i concetti (attributi e/o entità) dello schema che permettono di identificare in maniera univoca le occorrenze delle entità.

È assolutamente vietato inserire uno o più identificatori all'interno di una relazione. Quindi, quest'ultima non può avere identificatori interni!

Per esempio, un identificato interno per l'entità “Automobile” con attributi “Modello, Targa e Colore” è l'attributo “Targa”, in quanto non possono esistere due automobili con la stessa targa e quindi due occorrenze dell'entità “Automobile” con gli stessi valori sull'attributo “Targa”.

Un'entità E può essere identificata da altre entità solo se tali entità sono coinvolte in una relazione a cui E partecipa con cardinalità (1, 1). Nei casi in cui l'identificazione di un'entità è ottenuta utilizzando altre entità si parla di **identificatore esterno**.

Per comprendere meglio si espone un **esempio**. Per identificare univocamente uno studente serve, oltre al numero di matricola, anche la relativa università. Quindi, un identificatore corretto per l'entità “Studente” in questo schema è costituito dall'attributo “Matricola” e dall'entità “Università”.

Quindi, in generale:

- Un identificatore può **coinvolgere uno o più attributi**, ognuno dei quali deve avere cardinalità (1, 1);
- Un identificatore esterno può **coinvolgere una o più entità**, ognuna delle quali deve essere membro di una relazione alla quale l'entità da identificare partecipa con cardinalità (1, 1);
- Un identificatore esterno può **coinvolgere un'entità che è a sua volta identificata esternamente**, purché non vengano generati, in questa maniera, cicli di identificazione esterna;
- Ogni entità deve avere almeno un identificatore (interno o esterno), ma ne può avere in generale più di uno.

Sintassi grafica.

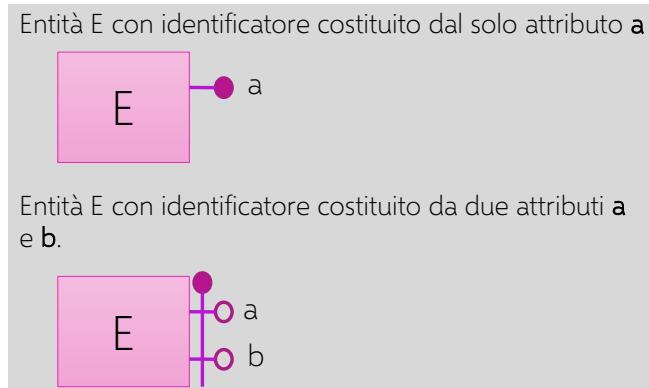


Figura 9: Sintassi grafica dell'identificatore.

2.5.4 Generalizzazioni

Definizione. Sono i legami logici tra un'entità E , chiamata **entità genitore**, e una o più entità E_1, \dots, E_n , dette **entità figlie**, di cui E è più generale, nel senso che le comprende come caso particolare. Quindi, si dice che E è **generalizzazione** di E_1, \dots, E_n e che le entità E_1, \dots, E_n sono **specializzazioni** dell'entità E .

Per esempio, l'entità “Persona” è una generalizzazione delle entità “Uomo e Donna”. Invece, “Professionista” è una generalizzazione delle entità “Ingegnere, Medico e Avvocato”.

Proprietà.

- **Ogni occorrenza di un'entità figlia è anche un'occorrenza dell'entità genitore.** Per esempio, una occorrenza dell'entità “Avvocato” è anche una occorrenza dell'entità “Professionista”.
- **Ogni proprietà dell'entità genitore** (come attributi, identificatori, relazioni e altre generalizzazioni) **è anche una proprietà delle entità figlie.** Per esempio, se l'entità “Persona” ha attributi “Cognome ed Età”, anche le entità “Uomo” e “Donna” possiedono questi attributi.

Classificazioni. Le generalizzazioni possono essere classificate:

- **Totale.** Ogni occorrenza dell'entità genitore è una occorrenza di almeno una dell'entità figlie. Se non è così, la generalizzazione è **parziale**;
- **Esclusiva.** Ogni occorrenza dell'entità genitore è al massimo un'occorrenza di una delle entità figlie. Se non è così, la generalizzazione è **sovraposta**.

In generale, una stessa entità può essere coinvolta in più generalizzazioni diverse. Possono esserci **generalizzazioni su più livelli**: in questo caso si parla di **gerarchia** di generalizzazioni. Infine, una **generalizzazione può avere una sola entità figlia**: in questo caso si parla di **sottoinsieme**.

Sintassi grafica. Non è difficile da comprendere, ma si presti attenzione a (x, y) . Esse indicano il **tipo di generalizzazione**:

- $[t, e] \rightarrow$ totale ed esclusiva;
- $[t, s] \rightarrow$ totale e sovrapposta;
- $[p, e] \rightarrow$ parziale ed esclusiva;
- $[p, s] \rightarrow$ parziale e sovrapposta.

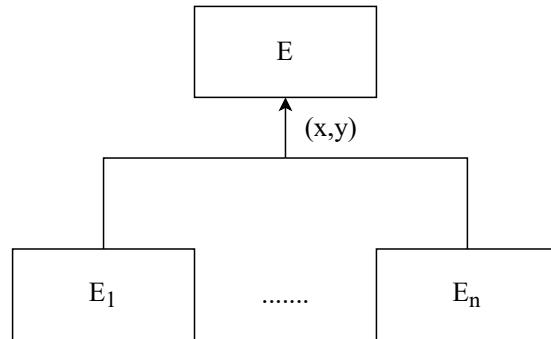


Figura 10: Sintassi grafica della generalizzazione.

3 Progettazione concettuale

3.1 Strategie di progetto

Lo sviluppo di uno schema concettuale a partire dalle sue specifiche può essere considerato un processo di ingegnerizzazione.

3.1.1 Strategia top-down

Lo schema concettuale viene prodotto mediante una serie di raffinamenti successivi a partire da uno schema iniziale che descrive tutte le specifiche con pochi concetti molto astratti. Lo schema viene poi via via raffinato mediante opportune trasformazioni che aumentano il dettaglio dei vari concetti presenti.

In sintesi:

1. **Fase 1**, si considerano le *specifiche globalmente* e si produce uno schema iniziale completo ma con *pochi concetti molto astratti*;
2. **Fase 2**, si esegue un *raffinamento* dei concetti astratti fino ad arrivare allo schema concettuale *completo* in ogni *dettaglio*.

Vantaggio: il progettista può **descrivere inizialmente tutte le specifiche dei dati trascurandone i dettagli**, per **poi entrare nel merito** di un concetto alla volta.

Svantaggio: si deve **possedere una visione globale e astratta** di *tutte* le componenti del sistema, ma solitamente è difficile.

3.1.2 Strategia bottom-up

Le specifiche iniziali sono suddivise in componenti via via sempre più piccole, fino a quando queste componenti descrivono un frammento elementare della realtà di interesse. A questo punto, le varie componenti vengono rappresentate da semplici schemi concettuali che possono consistere anche in singoli concetti. I vari schemi così ottenuti vengono poi fusi fino a giungere attraverso una completa integrazione di tutte le componenti, allo schema concettuale finale.

La **differenza rispetto alla strategia top-down** è che i vari concetti presenti nello schema finale vengono via via introdotti durante le varie fasi.

In sintesi:

1. **Fase 1**, si *decompongono* le specifiche iniziali in *parti elementari*, ovvero in frasi che descrivono lo stesso concetto;
2. **Fase 2**, si *generano* gli schemi per tutte le parti elementari individuate;
3. **Fase 3**, si *fondono* gli schemi (introducendo altri costrutti del modello E-R) in modo da *integrare* tutti gli schemi componenti e generare lo *schema finale*.

Vantaggio: questa strategia si adatta a una **decomposizione del problema in componenti più semplici**, facilmente individuabili, il cui progetto può essere affrontato anche da progettisti diversi.

Svantaggio: vengono richieste delle operazioni di **integrazione di schemi concettuali diversi** che, nel caso di schemi complessi, presentano **quasi sempre grosse difficoltà**.

3.1.3 Strategia inside-out

Si individuano inizialmente solo alcuni concetti importanti e poi si procede, a partire da questi, a “macchia d’olio”. Si rappresentano cioè prima i concetti in relazione con i concetti iniziali, per poi muoversi verso quelli più lontani attraverso una “navigazione” tra le specifiche.

In sintesi:

1. **Fase 1**, si *individua* nelle specifiche alcuni *concetti importanti*, chiamati concetti guida;
2. **Fase 2**, si *generano* gli schemi per i concetti guida;
3. **Fase 3**, si *fondono* gli schemi precedenti e si genera lo *schemma finale*.

Vantaggio: non sono richiesti passi di integrazione.

Svantaggio: è necessario, di volta in volta, esaminare tutte le specifiche per individuare concetti non ancora rappresentati e descrivere i nuovi concetti nel dettaglio.

3.2 Qualità di uno schema concettuale

L'analisi della qualità dello schema concettuale prodotto può essere suddivisa in diverse fasi:

- Correttezza
- Completezza
- Leggibilità
- Minimalità

3.2.1 Correttezza

Definizione. Uno schema concettuale è **corretto** quando **utilizza propriamente i costrutti** messi a disposizione dal modello concettuale di riferimento.

Gli errori che si possono commettere nello schema concettuale sono principalmente due:

- **Sintattici.** Vengono utilizzati costrutti non ammessi. Per esempio, una generalizzazione tra relazioni invece che tra entità.
- **Semantici.** Vengono usati costrutti senza rispettare la loro definizione. Per esempio, l'uso di una relazione per descrivere il fatto che una entità è specializzazione di un'altra.

3.2.2 Completezza

Definizione. Uno schema concettuale è **completo** quando **rappresenta tutti i dati** di interesse e quando **tutte le operazioni possono essere eseguite** a partire dai concetti descritti nello schema.

La completezza è **possibile verificarsi** controllando che tutte le specifiche sui dati siano rappresentate da qualche concetto presente nello schema che stiamo costruendo, e che tutti i concetti coinvolti in un'operazione presente nelle specifiche siano raggiungibili "navigando" attraverso lo schema.

3.2.3 Leggibilità

Definizione. Uno schema concettuale è **leggibile** quando rappresenta i requisiti in maniera naturale e facilmente comprensibile.

Per garantire questa proprietà è necessario rendere lo schema autoesplicativo, per esempio, mediante una scelta opportuna dei nomi da dare ai concetti. La leggibilità dipende anche da criteri puramente estetici.

3.2.4 Minimalità

Definizione. Uno schema concettuale è **minimale** quando tutte le specifiche sui dati sono rappresentate una sola volta nello schema.

Uno schema quindi non è minimale quando esistono delle **ridondanze**, ovvero concetti che possono essere derivati da altri. La minimalità di uno schema si può **verificare** per ispezione, controllando se esistono concetti che possono essere eliminati dallo schema che stiamo costruendo senza inficiare la sua completezza.

4 Progettazione logica

4.1 Fasi della progettazione logica

L'**obiettivo** della progettazione logica è **produrre uno schema logico che descriva in modo corretto ed efficace tutte le informazioni contenute nello schema concettuale.**

Le attività principali della progettazione logica sono la riorganizzazione dello schema concettuale e la traduzione in un modello logico:

- **Ristrutturazione dello schema Entità-Relazione:** è una fase indipendente dal modello logico scelto e si basa su **criteri di ottimizzazione** dello schema e di **semplificazione** della fase successiva. In particolare, le fasi sono:
 - Analisi delle ridondanze dovute alla presenza di dati derivabili;
 - Eliminazione delle generalizzazioni;
 - Accorpamento/partizionamento di entità e relazioni;
 - Scelta degli identificatori principali.
- **Traduzione verso il modello logico:** fa riferimento a uno specifico modello logico (nel nostro caso modello relazionale) e può includere una ulteriore ottimizzazione che si basa sulle caratteristiche del modello logico stesso.

4.2 Traduzione verso il modello logico

La seconda fase della progettazione logica corrisponde ad una traduzione tra modelli di dati diversi. Si parte da uno schema E-R ristrutturato e si costruisce uno schema logico equivalente, in grado cioè di rappresentare le medesime informazioni.

Si affronta il problema della traduzione caso per caso, iniziando dal caso più generale che ci suggerisce l'idea generale su cui si basa la metodologia di traduzione.

4.2.1 Entità e attributo opzionale

L'**entità** si rappresenta una relazione (lettera maiuscola) con lo stesso nome avente per attributi (lettere tra parentesi) i medesimi attributi dell'entità e per chiave il suo identificatore.

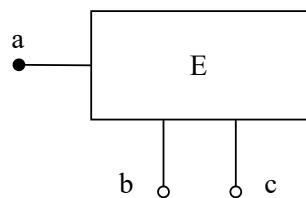


Figura 11: Modello E-R di un'entità.

Relativo modello relazionale:

$$\mathbf{E}(a, b, c)$$

Invece, un possibile **attributo nullo** si rappresenta inserendo un asterisco.

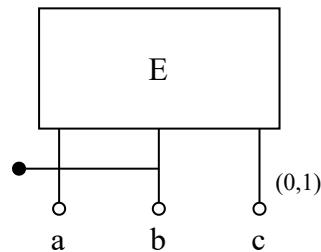


Figura 12: Modello E-R di un possibile attributo nullo.

Relativo modello relazionale:

$$\mathbf{E}(\underline{a}, \underline{b}, \underline{c}^*)$$

4.2.2 Relazione uno a molti

La **relazione uno a molti** è caratterizzata dal fatto che un'entità è in relazione con un'altra con cardinalità $(1, 1)$ e la corrispondente entità ha cardinalità (x, N) (x può essere qualsiasi valore minimo).

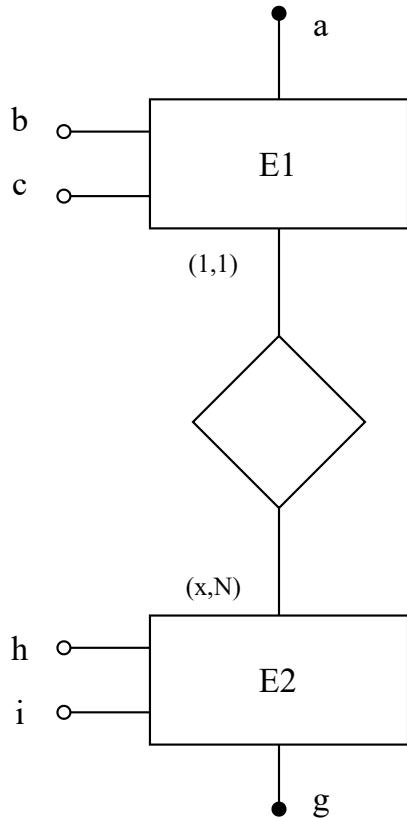
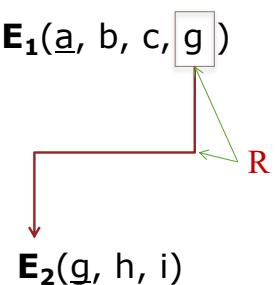


Figura 13: Modello E-R di una relazione uno a molti.

Relativo modello relazionale:



4.2.3 Relazione uno a uno

La relazione uno a uno è caratterizzata dal fatto che entrambe le entità hanno cardinalità (1, 1).

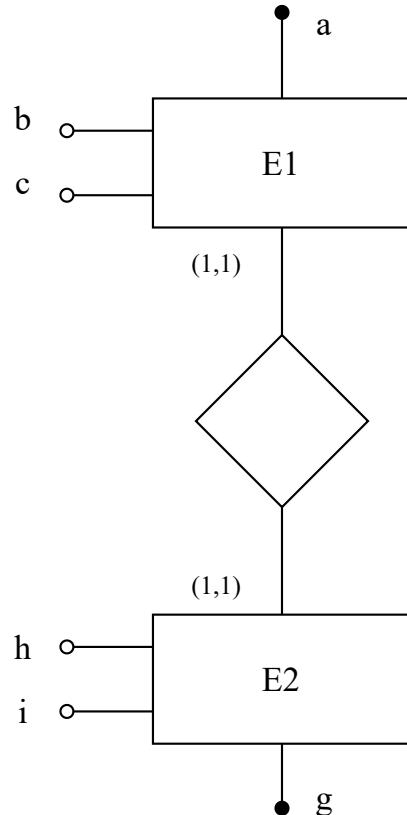
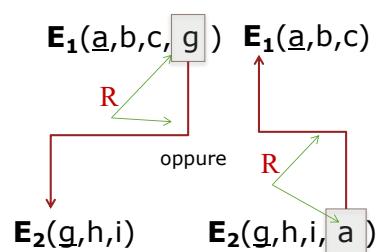


Figura 14: Modello E-R di una relazione uno a uno.

Relativo modello relazionale:



4.2.4 Relazione molti a molti

La **relazione molti a molti** è caratterizzata dal fatto che entrambe le entità hanno cardinalità (x, N) (x può essere qualsiasi valore minimo).

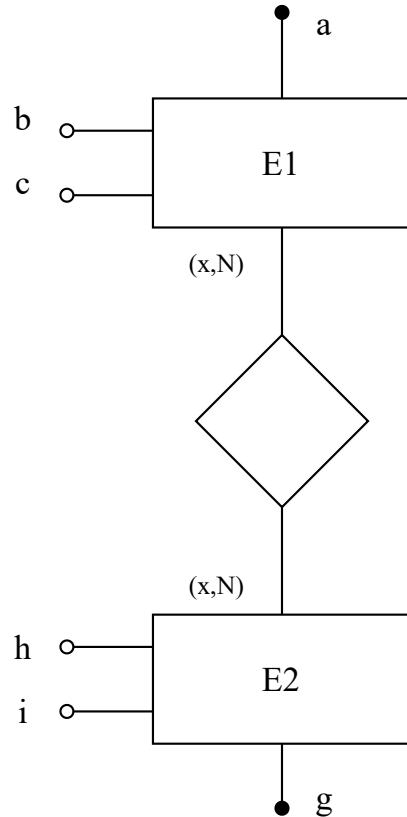
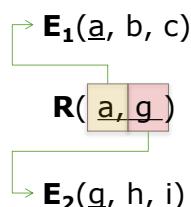


Figura 15: Modello E-R di una relazione molti a molti.

Relativo modello relazionale:



4.2.5 Relazione uno a molti (identificatore esterno)

La relazione uno a molti con identificatore esterno è caratterizzata dal fatto che un'entità ha cardinalità $(1, 1)$ e l'altra entità ha (x, N) (x può essere qualsiasi valore minimo).

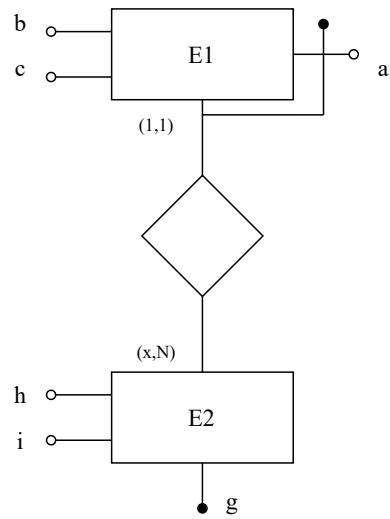
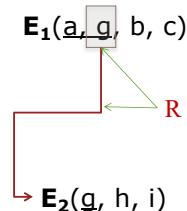


Figura 16: Modello E-R di una relazione molti a molti.

Relativo modello relazionale:



4.2.6 Relazione uno a molti (attributo sulla relazione)

La relazione uno a molti con un attributo sulla relazione è caratterizzata dal fatto che un'entità ha cardinalità $(1, 1)$, l'altra entità ha (x, N) (x può essere qualsiasi valore minimo) e un attributo è presente nella relazione.

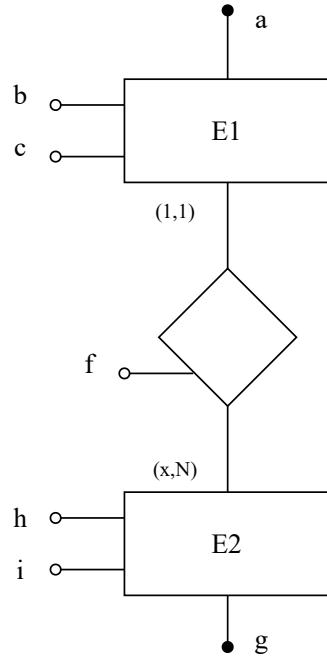
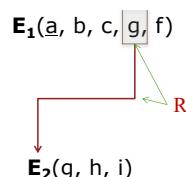


Figura 17: Modello E-R di una relazione uno a molti con un attributo sulla relazione esterno.

Relativo modello relazionale:



4.2.7 Relazione uno a uno (una cardinalità minima a zero)

La relazione uno a uno con una cardinalità minima uguale a zero è caratterizzata dal fatto che un'entità ha cardinalità $(0, 1)$, l'altra entità ha $(1, 1)$.

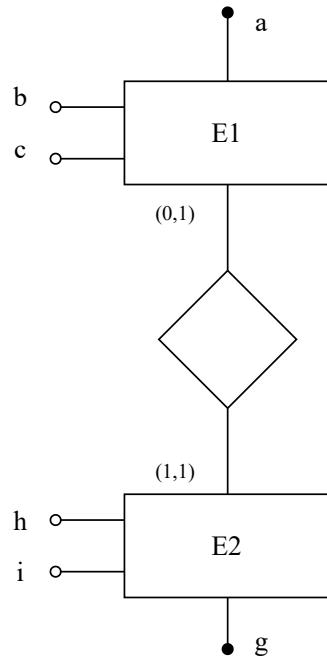
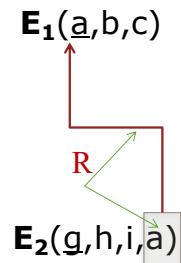


Figura 18: Modello E-R di una relazione uno a uno con una cardinalità minima uguale a zero.

Relativo modello relazionale:



4.2.8 Relazione uno a uno (entrambe cardinalità minima a zero)

La **relazione uno a uno con entrambe le cardinalità minima uguale a zero** è caratterizzata dal fatto che un'entità ha cardinalità $(0, 1)$, l'altra entità ha $(0, 1)$.

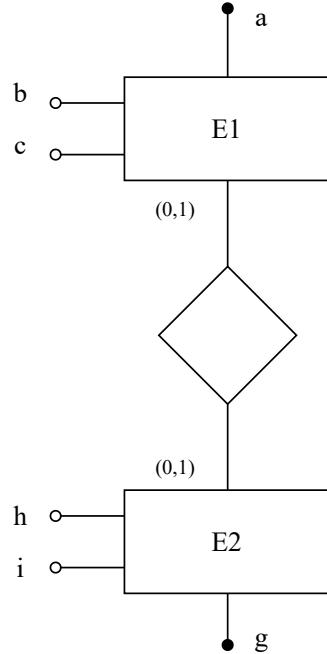
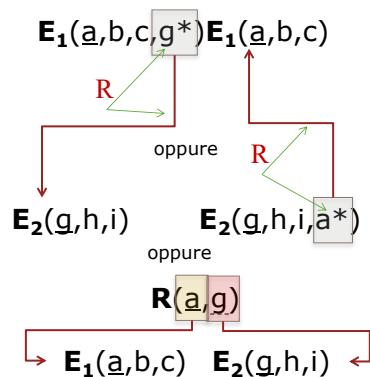


Figura 19: Modello E-R di una relazione uno a uno con entrambe le cardinalità minima uguale a zero.

Relativo modello relazionale:



4.2.9 Relazione molti a molti (attributo sulla relazione)

La **relazione molti a molti con un attributo sulla relazione** è caratterizzata dal fatto che un'entità ha cardinalità (x, N) , l'altra entità ha (x, N) (x può essere qualsiasi valore minimo) e un attributo è presente nella relazione.

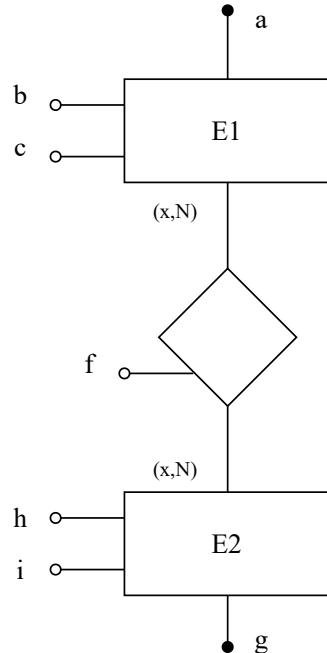
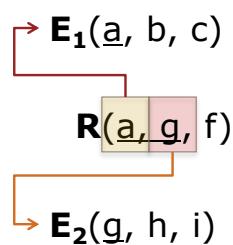


Figura 20: Modello E-R di una relazione molti a molti con un attributo sulla relazione esterno.

Relativo modello relazionale:



4.2.10 Relazione molti a molti (identificatori con più attributi)

La **relazione molti a molti con identificatori con più attributi** è caratterizzata dal fatto che un'entità ha cardinalità (x, N) , l'altra entità ha (x, N) (x può essere qualsiasi valore minimo) e un attributo è presente nella relazione.

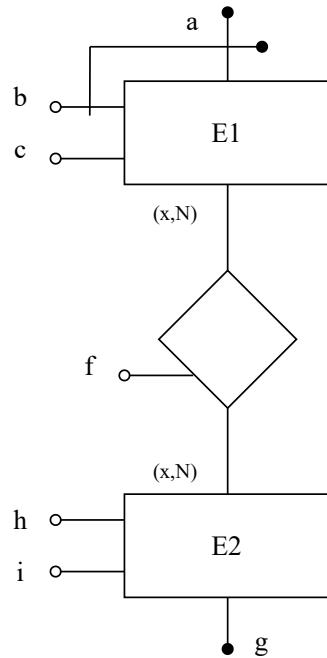
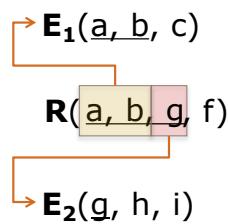


Figura 21: Modello E-R di una relazione molti a molti con identificatori con più attributi.

Relativo modello relazionale:



4.2.11 Relazione molti a molti (relazione ternaria)

La relazione molti a molti con relazione ternaria è caratterizzata dal fatto che un'entità ha cardinalità (x, N) , l'altra entità ha (x, N) , l'altra entità ancora ha cardinalità (x, N) (x può essere qualsiasi valore minimo) e un attributo è presente nella relazione.

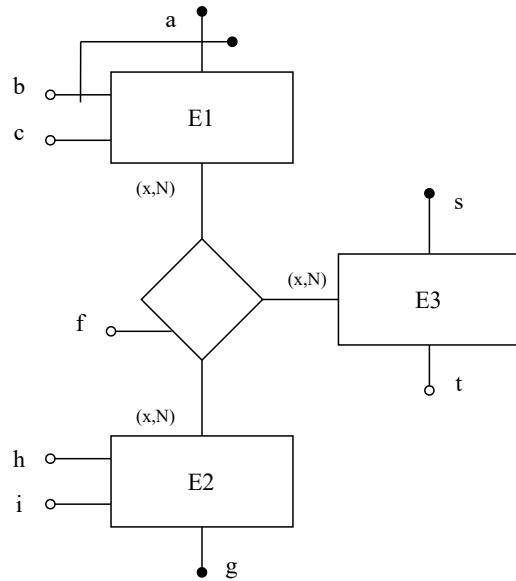
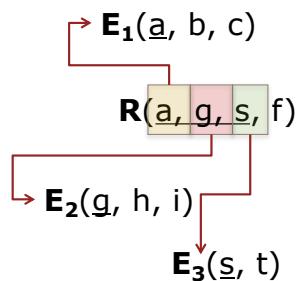


Figura 22: Modello E-R di una relazione molti a molti con relazione ternaria.

Relativo modello relazionale:



4.2.12 Relazione molti a molti (relazione ternaria e cardinalità uno a uno)

La relazione molti a molti con relazione ternaria e cardinalità uno a uno è caratterizzata dal fatto che un'entità ha cardinalità $(1, N)$, l'altra entità ha (x, N) , l'altra entità ancora ha cardinalità (x, N) (x può essere qualsiasi valore minimo) e un attributo è presente nella relazione.

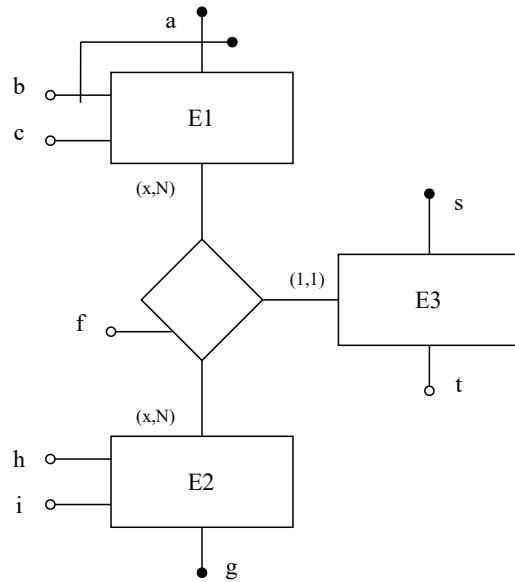
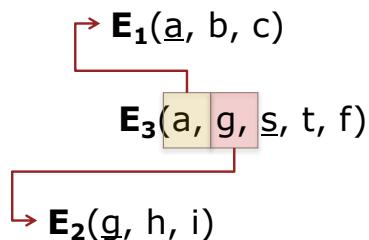


Figura 23: Modello E-R di una relazione molti a molti con relazione ternaria.

Relativo modello relazionale:



5 Algebra relazionale

L'**algebra relazionale** è un linguaggio procedurale, basato su concetti di tipo algebrico. Esso è costituito da un insieme di operatori, definiti su relazioni e che producono ancora relazioni come risultati. I vari operatori sono:

- **Insiemistici**

- *Unione*
- *Intersezione*
- *Differenza*

- **Specifici**

- *Ridenominazione*
- *Selezione*
- *Proiezione*

- **Più importante join**

- *Join naturale*
- *Prodotto*
- *Cartesiano*
- *Semijoin*
- *Theta-join*

In altre parole, l'algebra relazione è un insieme di **operatori su relazioni**. Dato che le relazioni sono insiemi, ha senso definire su di esse gli operatori insiemistici come l'unione, la differenza e l'intersezione

5.1 Insiemistici

5.1.1 Unione

L'**unione** di due relazioni r_1 e r_2 definite sullo stesso insieme di attributi X è indicata con $r_1 \cup r_2$ ed è una relazione ancora su X contenente le tuple che appartengono a r_1 oppure a r_2 , oppure a entrambe.

La **cardinalità**, ovvero il numero di tuple contenute nella relazione del risultato:

$$\max(|r_1|, |r_2|) \leq |r_1 \cup r_2| \leq |r_1| + |r_2|$$

Esempio

Laureati

Matricola	Cognome	Età
7274	Rossi	37
7432	Neri	39
9824	Verdi	38

Dirigenti

Matricola	Cognome	Età
9297	Neri	56
7432	Neri	39
9824	Verdi	38

Laureati \cup Dirigenti

Matricola	Cognome	Età
7274	Rossi	37
7432	Neri	39
9824	Verdi	38
9297	Neri	56

5.1.2 Intersezione

L'intersezione di $r_1(X)$ e $r_2(X)$ è indicata con $r_1 \cap r_2$ ed è una relazione su X contenente le tuple che appartengono sia a r_1 sia a r_2 .

La **cardinalità**, ovvero il numero di tuple contenute nella relazione del risultato:

$$0 \leq |r_1 \cap r_2| \leq \min(|r_1|, |r_2|)$$

Esempio

Laureati

Matricola	Cognome	Età
7274	Rossi	37
7432	Neri	39
9824	Verdi	38

Dirigenti

Matricola	Cognome	Età
9297	Neri	56
7432	Neri	39
9824	Verdi	38

Laureati \cap Dirigenti

Matricola	Cognome	Età
7432	Neri	39
9824	Verdi	38

5.1.3 Differenza

La differenza di $r_1(X)$ e $r_2(X)$ è indicata con $r_1 - r_2$ ed è una relazione su X contenente le tuple che appartengono a r_1 e non appartengono a r_2 .

La **cardinalità**, ovvero il numero di tuple contenute nella relazione del risultato:

$$0 \leq |r_1 - r_2| \leq |r_1|$$

Esempio

Laureati

Matricola	Cognome	Età
7274	Rossi	37
7432	Neri	39
9824	Verdi	38

Dirigenti

Matricola	Cognome	Età
9297	Neri	56
7432	Neri	39
9824	Verdi	38

Laureati – Dirigenti

Matricola	Cognome	Età
7274	Rossi	37

5.2 Specifici

5.2.1 Ridenominazione

L'**obiettivo** di questo operatore è risolvere le limitazioni degli operatori insiemistici. Infatti, esso **adegua i nomi degli attributi**, a seconda delle necessità, in particolare alla fine di facilitare le operazioni insiemistiche. Ovviamente, la ridenominazione avviene solamente sugli attributi, il **contenuto** rimane **inalterato**.

Il simbolo che lo rappresenta è la lettera greca rho ρ . **Al pedice viene inserita la ridenominazione, a destra il nome dell'attributo da rinominare e a sinistra il nuovo nome dell'attributo.** In generale, sia r una relazione definita sull'insieme di attributi X e sia Y un (altro) insieme di attributi con la stessa cardinalità. Inoltre, siano $A_1 A_2 \dots A_k$ e $B_1 B_2 \dots B_k$ rispettivamente un ordinamento per gli attributi in X e un ordinamento per quelli in Y . Allora la ridenominazione:

$$\rho_{B_1 B_2 \dots B_k \leftarrow A_1 A_2 \dots A_k} (r)$$

Contiene una tupla t' per ciascuna tupla t in r , definita come segue: t' è una tupla su Y e $t'[B_i] = t[A_i]$, per $i = 1, \dots, k$.

La definizione conferma che ciò che cambia sono i nomi degli attributi, mentre i valori rimangono inalterati e vengono associati ai nuovi attributi.

Esempio

Impiegati

Cognome	Agenzia	Stipendio
Rossi	Roma	45
Neri	Milano	53

Operai

Cognome	Fabbrica	Salario
Verdi	Latina	33
Bruni	Monza	32

Risultato (vedi sotto)

Cognome	Sede	Retribuzione
Rossi	Roma	45
Neri	Milano	53
Verdi	Latina	33
Bruni	Monza	32

Il risultato è ottenuto con la seguente operazione:

$$\rho_{\text{Sede}, \text{Retribuzione} \leftarrow \text{Agenzia}, \text{Stipendio}} (\text{Impiegati}) \cup \rho_{\text{Sede}, \text{Retribuzione} \leftarrow \text{Fabbrica}, \text{Salario}} (\text{Operai})$$

5.2.2 Selezione

La selezione produce una porzione dell'operando. Più precisamente, la **selezione** produce un sottoinsieme delle tuple su tutti gli attributi. Il **risultato** contiene le tuple dell'operando che soddisfano la condizione di selezione. Quest'ultima viene indicata nel pedice della notazione della selezione, ovvero sigma σ . Inoltre, le condizioni possono prevedere confronti fra attributi e confronti fra attributi e costanti, e possono essere complesse, ottenute combinando condizioni semplici con i connettivi logici \vee (or), \wedge (and) e \neg (not).

In generale, data una relazione $r(X)$, una *forma proposizionale* F su X è una formula ottenuta combinando, con i connettivi \wedge , \vee , \neg , condizioni atomiche del tipo $A\theta B$ o $A\theta c$, dove:

- θ è un **operatore di confronto**, il quale può essere:

$- =$
 $- \neq$
 $- >$
 $- <$
 $- \geq$
 $- \leq$

- A e B sono **attributi** in X sui cui valori il confronto θ abbia senso;
- c è una **costante** “compatibile” con il dominio di A (cioè tale che il confronto θ sia definito).

Data una formula F e una tupla t , è definito un valore di verità (cioè “vero” o “falso”) per F su t :

- $A\theta B$ è vera su t se $t[A]$ è in relazione θ con $t[B]$, altrimenti è falsa (per esempio, $A = B$ è vera su t se e solo se $t[A] = t[B]$);
- $A\theta c$ è vera su t se $t[A]$ è in relazione θ con c , altrimenti è falsa;
- $F_1 \vee F_2, F_1 \wedge F_2$ e $\neg F_1$ hanno l'usuale significato.

La **definizione**, in altre parole, è: la **selezione** $\sigma_F(r)$, in cui r è una relazione e F una formula proposizionale, produce una relazione sugli stessi attributi di r che contiene le tuple di r su cui F è vera.

Esempio 1

Impiegati

Cognome	Nome	Età	Stipendio
Rossi	Mario	25	2.000,00
Neri	Luca	40	3.000,00
Verdi	Nico	36	4.500,00
Rossi	Marco	40	3.900,00

Risultato (vedi sotto)

Cognome	Nome	Età	Stipendio
Verdi	Nico	36	4.500,00

Il risultato è ottenuto con la seguente operazione:

$$\sigma_{\text{Eta} > 30 \wedge \text{Stipendio} > 4.000,00} (\text{Impiegati})$$

Esempio 2

Cittadini

Cognome	Nome	CittàDiNascita	Residenza
Rossi	Mario	Roma	Milano
Neri	Luca	Roma	Roma
Verdi	Nico	Firenze	Firenze
Rossi	Marco	Napoli	Firenze

Risultato (vedi sotto)

Cognome	Nome	CittàDiNascita	Residenza
Neri	Luca	Roma	Roma
Verdi	Nico	Firenze	Firenze

Il risultato è ottenuto con la seguente operazione:

$$\sigma_{\text{CittàDiNascita} = \text{Residenza}} (\text{Cittadini})$$

5.2.3 Proiezione

La **proiezione** dà un risultato cui contribuiscono tutte le tuple, ma su un sottoinsieme degli attributi.

Formalmente, dati una relazione $r(X)$ e un sottoinsieme Y di X , la **proiezione** di r su Y (indicata con $\pi_Y(r)$) è l'insieme di tuple su Y ottenute dalle tuple di r considerando solo i valori su Y :

$$\pi_Y(r) = \{t[Y] \mid t \in r\}$$

Dagli esempi è chiaro che la proiezione permette di decomporre verticalmente le relazioni.

Esempio 1

In questo caso, il risultato della proiezione contiene tante tuple quante l'operando, definite però solo su parte degli attributi.

Impiegati

Cognome	Nome	Reparto	Capo
Rossi	Mario	Vendite	Gatti
Neri	Luca	Vendite	Gatti
Verdi	Mario	Personale	Lupi
Rossi	Marco	Personale	Lupi

Risultato (vedi sotto)

Cognome	Nome
Rossi	Mario
Neri	Luca
Verdi	Mario
Rossi	Marco

Il risultato è ottenuto con la seguente operazione:

$$\pi_{\text{Cognome}, \text{Nome}}(\text{Impiegati})$$

Esempio 2

In questo caso, il risultato contiene un numero di tuple inferiore rispetto a quelle dell'operando, perché alcune tuple, avendo uguali valori su tutti gli attributi della proiezione, danno lo stesso contributo alla proiezione stessa. Essendo le relazioni definite come insieme, non possono, per definizione, in esse comparire più tuple uguale fra loro: i contributi “collassano” in una sola tupla.

Impiegati

Cognome	Nome	Reparto	Capo
Rossi	Mario	Vendite	Gatti
Neri	Luca	Vendite	Gatti
Verdi	Mario	Personale	Lupi
Rossi	Marco	Personale	Lupi

Risultato (vedi sotto)

Reparto	Capo
Vendite	Gatti
Personale	Lupi

Il risultato è ottenuto con la seguente operazione:

$$\pi_{\text{Reparto}, \text{Capo}} (\text{Impiegati})$$

5.3 Join

L'operatore di *join* consente di correlare dati contenuti in relazioni diverse, confrontando i valori contenuti in esse e utilizzando quindi la caratteristica fondamentale del modello, ovvero quella di essere basta su valori.

Formalmente, due relazione r_1 e r_2 di schema X_1 e X_2 rispettivamente, gli operatori di *join* generano tuple t nella relazione risultato a partire dalle coppie di tuple $(t_1, t_2) \in r_1 \times r_2$ che soddisfano una certa condizione (chiamata **predicato di join**).

5.3.1 Join naturale

Il **join naturale** è un operatore che correla dati in relazioni diverse, sulla base di valori uguali in attributi con lo stesso nome (si veda l'esempio come chiarificatore). Questa operazione viene denotata con il simbolo \bowtie .

Il **risultato** dell'operatore è una relazione sull'unione degli insiemi di attributi degli operandi e le sue tuple sono ottenute combinando le tuple degli operandi con valori uguali sugli attributi comuni. Nel primo esempio sotto, la prima tupla del *join* deriva dalla combinazione della prima tupla della relazione r_1 e dalla seconda tupla della relazione r_2 .

Formalmente, il **join naturale** $r_1 \bowtie r_2$ di $r_1(X_1)$ e $r_2(X_2)$ è una relazione definita su $X_1 X_2$ (cioè sull'unione degli insiemi X_1 e X_2), come segue:

$$r_1 \bowtie r_2 = \{t \text{ su } X_1 X_2 \mid t[X_1] \in r_1, t[X_2] \in r_2\}$$

Si noti che è molto frequente eseguire *join* sulla base di valori della chiave di una delle relazioni coinvolte, esplicitando i riferimenti fra tuple che sono realizzati per mezzo di valori soprattutto valori di chiavi. Osservando l'esempio 2, si vede che ciascuna delle tuple di **Infrazioni** è stata combinata con una e una sola delle tuple di **Auto**:

- Una sola perché **Prov** e **Numero** formano una chiave di **Auto**;
- Almeno una perché è definito il vincolo di integrità referenziale fra **Prov** e **Numero** in **Infrazioni** e (la chiave primaria di) **Auto**

Dunque, il *join* ha esattamente tante tuple quante la relazione **Infrazioni**.

Esempio 1

r_1		r_2	
Impiegato	Reparto	Reparto	Capo
Rossi	vendite	produzione	Mori
Neri	produzione	vendite	Bruni
Bianchi	produzione		

Risultato (vedi sotto)

Impiegato	Reparto
Rossi	vendite
Neri	produzione
Bianchi	produzione

Il risultato è ottenuto con la seguente operazione:

$$r_1 \bowtie r_2$$

Esempio 2

Infrazioni

Codice	Data	Agente	Articolo	Stato	Numero
143256	25/10/2017	567	44	I	AB 234 ZK
987554	26/10/2017	456	34	I	AB 234 ZK
987557	26/10/2017	456	34	I	CB 123 AA
630876	15/10/2017	456	53	F	CB 123 AA
539856	12/10/2017	567	44	F	CB 123 AA

Auto

Stato	Numero	Proprietario	Indirizzo
I	CB 123 AA	Verdi Piero	Via Tigli
I	DE 834 ZZ	Verdi Piero	Via Tigli
I	AB 234 ZK	Bini Luca	Via Aceri
F	CB 123 AA	Beau Marcel	Rue Louis

Risultato (vedi sotto)

Codice	Data	Agente	Articolo	Stato	Numero	Proprietario	Indirizzo
143256	25/10/2017	567	44	I	AB 234 ZK	Bini Luca	Via Aceri
987554	26/10/2017	456	34	I	AB 234 ZK	Bini Luca	Via Aceri
987557	26/10/2017	456	34	I	CB 123 AA	Verdi Piero	Via Tigli
630876	15/10/2017	456	53	F	CB 123 AA	Beau Marcel	Rue Louis
539856	12/10/2017	567	44	F	CB 123 AA	Beau Marcel	Rue Louis

Il risultato è ottenuto con la seguente operazione:

$$\text{Infrazioni} \bowtie \text{Auto}$$

5.3.2 Join completi e incompleti

Nell'esempio 1 nel paragrafo del *join naturale*, si può dire che ciascuna tupla di ciascuno degli operandi contribuisce almeno una tupla del risultato (in questo caso il *join* viene detto **completo**): per ogni tupla t_1 di r_1 , esiste una tupla t in $r_1 \bowtie r_2$ tale che $t[X_1] = t_1$ (e analogamente per r_2).

L'esempio 1 a fine paragrafo, mostra un *join* in cui alcune tuple degli operandi, in particolare la prima di r_1 e la seconda di r_2 , non contribuiscono al risultato, perché l'altra relazione non contiene tuple con gli stessi valori sull'attributo comune. In questo caso il *join* viene detto **dangling**, ovvero **dondolante**.

Infine, come caso limite, è ovviamente possibile che nessuna delle tuple degli operandi sia combinabile, e allora il risultato del *join* è la **relazione vuota** (esempio 2 a fine paragrafo). All'estremo opposto, è possibile che ciascuna delle tuple di ciascuno degli operandi sia combinabile con tutte dell'altro, come mostrato nell'ultimo esempio del paragrafo, e in questo caso il risultato ha un numero di tuple pari al prodotto delle cardinalità degli operandi e cioè $|r_1| \times |r_2|$ (dove $|r|$ indica la cardinalità della relazione r).

Alcune osservazioni finali:

- Se il *join* di r_1 e r_2 è completo, allora contiene almeno un numero di tuple pari al massimo fra $|r_1|$ e $|r_2|$;
- Se $X_1 \cap X_2$ contiene una chiave per r_2 , allora il *join* di $r_1(X_1)$ e $r_2(X_2)$ contiene al più $|r_1|$ tuple;
- Se $X_1 \cap X_2$ coincide con una chiave per r_2 e sussiste il vincolo di riferimento fra $X_1 \cap X_2$ in r_1 e la chiave di r_2 , allora il *join* di $r_1(X_1)$ e $r_2(X_2)$ contiene esattamente $|r_1|$ tuple.

Esempio 1

r₁			
Impiegato	Reparto	Reparto	Capo
Rossi	vendite	produzione	Mori
Neri	produzione	acquisti	Bruni
Bianchi	produzione		

Risultato (vedi sotto)

Impiegato	Reparto	Capo
Neri	produzione	Mori
Bianchi	produzione	Mori

Il risultato è ottenuto con la seguente operazione:

$$r_1 \bowtie r_2$$

Esempio 2

r₁			
Impiegato	Reparto	Reparto	Capo
Rossi	vendite	concorsi	Mori
Neri	produzione	acquisti	Bruni
Bianchi	produzione		

Risultato (vedi sotto)

Impiegato	Reparto	Capo

Il risultato è ottenuto con la seguente operazione:

$$r_1 \bowtie r_2$$

Esempio 3

r₁

Impiegato	Progetto
Rossi	A
Neri	A
Bianchi	A

r₂

Progetto	Capo
A	Mori
A	Bruni

Risultato (vedi sotto)

Impiegato	Reparto	Capo
Rossi	A	Mori
Neri	A	Mori
Bianchi	A	Mori
Rossi	A	Bruni
Neri	A	Bruni
Bianchi	A	Bruni

Il risultato è ottenuto con la seguente operazione:

$$r_1 \bowtie r_2$$

5.3.3 Theta-join ed equi-join

Osservando l'esempio 1 a fine paragrafo, è possibile notare come un prodotto cartesiano ha di solito ben poca utilità, in quanto concatena tuple non necessariamente correlate dal punto di vista semantico. Infatti, il **prodotto cartesiano viene spesso seguito da una selezione**, la quale centra l'attenzione su tuple correlate secondo le esigenze.

Per esempio, nell'esempio 2 a fine paragrafo, sulle relazioni *Impiegati* e *Progetti* ha senso definire un prodotto cartesiano seguito dalla selezione che mantiene solo le tuple con valori uguali sull'attributo *Progetto* di *Impiegati* e su *Codice* di *Progetti*.

Per questo motivo, viene definito un operatore derivato (cioè per mezzo di altri operatori), chiamato **theta-join**. Esso è considerato come un **prodotto cartesiano seguito da una selezione**, nel modo seguente:

$$r_1 \underset{F}{\bowtie} r_2 = \sigma_F(r_1 \bowtie r_2)$$

Attenzione! Gli schemi r_1 e r_2 devono essere **disgiunti**, cioè:

$$r_1 \cap r_2 = \emptyset$$

Per esempio, nell'esempio 2 a fine paragrafo, la relazione può essere ottenuta per mezzo del *theta-join*:

$$\begin{array}{c} \text{Impiegati} \quad \bowtie \quad \text{Progetti} \\ \text{Progetto} = \text{Codice} \end{array}$$

Un *theta-join* in cui la condizione di selezione F sia una congiunzione di atomi di uguaglianza, con un attributo della prima relazione e uno della seconda, viene chiamato **equi-join**. Quindi, la relazione scritta qua sopra è un *equi-join*.

Infine, il **join naturale è possibile simularlo** tramite la ridenominazione, l'*equi-join* e la proiezione. Date due relazioni $r_1(ABC)$ e $r_2(BCD)$, il join naturale di r_1 e r_2 può essere espresso per mezzo degli altri operatori, in tre passi:

1. **Ridenominando** gli attributi così da ottenere relazioni su schemi disgiunti:

$$\rho_{B'C' \leftarrow BC}(r_2)$$

2. Effettuando l'**equi-join**, con condizioni di uguaglianza sugli attributi corrispondenti:

$$r_1 \underset{B=B' \wedge C=C'}{\bowtie} \rho_{B'C' \leftarrow BC}(r_2)$$

3. Concludendo con una **proiezione** che elimina gli attributi “doppioni”, che presentano valori identici a quelli di altri attributi:

$$\pi_{ABCD} \left(r_1 \underset{B=B' \wedge C=C'}{\bowtie} \rho_{B'C' \leftarrow BC}(r_2) \right)$$

Esempio 1

Impiegati

Impiegato	Progetto
Rossi	A
Neri	A
Neri	B

Progetti

Codice	Nome
A	Venere
B	Marte

Risultato (vedi sotto)

Impiegato	Progetto	Codice	Nome
Rossi	A	A	Venere
Neri	A	A	Venere
Neri	B	A	Venere
Rossi	A	B	Marte
Neri	A	B	Marte
Neri	B	B	Marte

Il risultato è ottenuto con la seguente operazione:

Impiegati \bowtie Progetti

Esempio 2

Impiegati

Impiegato	Progetto
Rossi	A
Neri	A
Neri	B

Progetti

Codice	Nome
A	Venere
B	Marte

Risultato (vedi sotto)

Impiegato	Progetto	Codice	Nome
Rossi	A	A	Venere
Neri	A	A	Venere
Neri	B	B	Marte

Il risultato è ottenuto con la seguente operazione:

$$\sigma_{\text{Progetto}=\text{Codice}} (\text{Impiegati} \bowtie \text{Progetti})$$

5.4 Algebra con valori nulli

È necessario introdurre la possibilità di avere dei valori nulli nell'algebra relazionale. Un **valore nullo** (*unknown*) viene rappresentato con il simbolo U e un **predicato prende tale valore quando almeno uno dei termini del confronto è sconosciuto**. Considerando l'esempio a fine paragrafo, data la seguente selezione:

$$\sigma_{\text{Eta} > 30} (\text{Persone})$$

La prima tupla certamente appartiene al risultato (appartenenza **vera**), la seconda certamente non appartiene (appartenenza **falsa**), la terza forse appartiene e forse no (appartenenza **sconosciuta**).

Le **tavola di verità** dei connettivi *not*, *and*, *or* tenendo conto del nuovo valore logico, si estendono nel seguente modo: Il valore nullo rappresenta un valore di

<i>not</i>	<i>and</i>	<i>or</i>
V	V U F	V V V V
U	U U F	V U U
F	F F F	V U F

verità intermedio tra vero e falso, e il significato dei tre connettivi in questo contesto diventa il seguente:

- Il ***not*** è vero solo se il valore di partenza è falso;
- L'***and*** è vero solo se tutti i termini sono veri;
- L'***or*** è vero se almeno uno dei termini è vero.

Considerando ancora l'esempio a fine paragrafo, data la seguente espressione:

$$\sigma_{\text{Eta} > 30} (\text{Persone}) \cup \sigma_{\text{Eta} \leq 30} (\text{Persone})$$

Logicamente si potrebbe pensare che il risultato sia esattamente la relazione Persone. Questo è **sbagliato** poiché la terza tupla, quella con il valore NULL, ha un'appartenenza sconosciuta a ciascuna sottoespressione e dunque anche all'unione. Questo discorso vale anche per la seguente espressione:

$$\sigma_{\text{Eta} > 30 \vee \text{Eta} \leq 30} (\text{Persone})$$

In cui la disgiunzione viene valutata secondo la logica a tre valori.

Da questo problema nasce l'esigenza di esplicitare due nuove condizioni: **IS NULL** e **IS NOT NULL**. Il loro significato:

- **A IS NULL** assume un valore vero su una tupla *t* se il valore di *t* su *A* è nullo e falso (altrimenti) se esso è specificato;
- **A IS NOT NULL** assume un valore vero su una tupla *t* se il valore di *t* su *A* è specificato e falso (altrimenti) se esso è nullo;

Considerando l'esempio a fine paragrafo, data l'espressione:

$$\sigma_{\text{Eta} > 30 \vee \text{Eta IS NULL}} (\text{Persone})$$

Si ottiene le persone che hanno o potrebbero avere più di trent'anni (quindi età nota e maggiore di 30 oppure non nota). Invece, per ottenere tutte le tuple:

$$\sigma_{\text{Eta} > 30 \vee \text{Eta} \leq 30 \vee \text{Eta IS NULL}} (\text{Persone})$$

Esempio

Persone

Nome	Età	Reddito
Aldo	35	15
Andrea	27	21
Maria	NULL	42

5.5 Ottimizzare ed equivalenza di espressioni algebriche

5.5.1 Definizioni

Ogni espressione (*query*) ricevuta dal DBMS è soggetta ad un processo di elaborazione. Esiste una parte software ad un livello più basso che implementa l'**ottimizzatore**. Esso ha l'obiettivo di generare un'espressione equivalente all'originale ma con un costo minore. Per **costo** si intende la dimensione dei risultati intermedi.

Da questa necessità, si crea l'**equivalenza delle espressioni algebriche**, cioè espressioni fra loro equivalenti che producono quindi lo stesso risultato.

Esistono **due tipi** di equivalenza:

- **Equivalenza assoluta.** Non viene esplicitato nessuno schema, quindi è completamente indipendente (assoluta).

Più formalmente:

$$E_1 \equiv E_2 \text{ se } E_1 \equiv_{\mathbf{R}} E_2 \text{ per ogni schema } \mathbf{R}$$

Un esempio:

$$\pi_{AB}(\sigma_{A>0}(R)) \equiv \sigma_{A>0}(\pi_{AB}(R))$$

- **Equivalenza dipendente dallo schema.** Viene esplicitato lo schema e dunque l'equivalenza è vera se e solo se l'intersezione degli insiemi è uguale.

Più formalmente:

$$E_1 \equiv_{\mathbf{R}} E_2 \text{ se } E_1(\mathbf{r}) = E_2(\mathbf{r}) \text{ per ogni istanza } \mathbf{r} \text{ di } \mathbf{R}$$

Un esempio:

$$\pi_{AB}(R_1) \bowtie \pi_{AC}(R_2) \equiv_{\mathbf{R}} \pi_{ABC}(R_1 \bowtie R_2)$$

L'equivalenza è valida se e solo se nello schema **R** l'intersezione fra gli insiemi di attributi di R_1 e R_2 è pari ad A .

5.5.2 Trasformazioni di equivalenza

Nel contesto delle ottimizzazioni, vengono spesso utilizzate le **trasformazioni di equivalenza**, ovvero operazioni che **sostituiscono un'espressione con un'altra a essa equivalente**. In particolare, risultano interessanti le trasformazioni che riducono le dimensioni dei risultati intermedi e quelle che preparano un'espressione all'applicazione di una trasformazione che riduce le dimensioni dei risultati intermedi.

- I. **Atomizzazione delle selezioni**: una selezione σ congiuntiva¹ può essere sostituita da una cascata di selezioni atomiche:

$$\sigma_{F_1 \wedge F_2}(E) \equiv \sigma_{F_1}(\sigma_{F_2}(E))$$

In cui E è una qualunque espressione.

- II. **Idempotenza delle proiezioni**: una proiezione π può essere trasformata in una cascata di proiezioni che “eliminano” i vari attributi in fasi successive:

$$\pi_X(E) \equiv \pi_X(\pi_{XY}(E))$$

Se E è definita su un insieme di attributi che contiene Y (oltre a X).

- III. **Pushing selections down (Anticipazione della selezione rispetto al join)**:

$$\sigma_F(E_1 \bowtie E_2) \equiv E_1 \bowtie \sigma_F(E_2)$$

Se la condizione F fa riferimento solo ad attributi nella sottoespressione E_2 .

- IV. **Pushing projections down (Anticipazione della proiezione rispetto al join)**: siano E_1 ed E_2 definite rispettivamente su X_1 e X_2 . Allora, se $Y_2 \subseteq X_2$ e $Y_2 \supseteq (X_1 \cap X_2)$, ovvero se gli attributi in $X_2 - Y_2$ non sono coinvolti nel join, vale:

$$\pi_{X_1 Y_2}(E_1 \bowtie E_2) \equiv E_1 \bowtie \pi_{Y_2}(E_2)$$

Combinando questa regola con quella della idempotenza delle proiezioni, si può ottenere:

$$\pi_Y\left(E_1 \underset{F}{\bowtie} E_2\right) \equiv \pi_Y\left(\pi_{Y_1}(E_1) \underset{F}{\bowtie} \pi_{Y_2}(E_2)\right)$$

Si indica con X_1 e X_2 gli attributi di E_1 ed E_2 rispettivamente, con J_1 e J_2 i rispettivi sottoinsiemi coinvolti nella condizione F di join:

$$\begin{aligned} Y_1 &= (X_1 \cap Y) \cup J_1 \\ Y_2 &= (X_2 \cap Y) \cup J_2 \end{aligned}$$

In altre parole, è possibile **eliminare subito da ciascuna relazione gli attributi che non compaiono nel risultato finale e non sono coinvolti nel join**.

¹Ovvero due espressioni connesse da un and, or, not logico

V. **Inglobamento di una selezione in un prodotto cartesiano a formare un join:**

$$\sigma_F(E_1 \bowtie E_2) \equiv E_1 \underset{F}{\bowtie} E_2$$

Indicando X_1 e X_2 i rispettivi attributi di E_1, E_2 , allora $X_1 \cap X_2 = \emptyset$.

Queste trasformate sono quelle più importanti.

Esistono altre trasformate non meno importanti ma di minore rilievo rispetto alle prime che sono considerate fondamentali.²

VI. **Distributività della selezione rispetto all'unione:**

$$\sigma_F(E_1 \cup E_2) \equiv \sigma_F(E_1) \cup \sigma_F(E_2)$$

VII. **Distributività della selezione rispetto alla differenza:**

$$\sigma_F(E_1 - E_2) \equiv \sigma_F(E_1) - \sigma_F(E_2)$$

VIII. **Distributività della proiezione rispetto all'unione:**

$$\pi_X(E_1 \cup E_2) \equiv \pi_X(E_1) \cup \pi_X(E_2)$$

Un altro gruppo minore di trasformate è quello basato sull'interazione fra gli operatori insiemistici e le selezioni complesse:

IX. $\sigma_{F_1 \vee F_2}(R) \equiv \sigma_{F_1}(R) \cup \sigma_{F_2}(R)$

X. $\sigma_{F_1 \wedge F_2}(R) \equiv \sigma_{F_1}(R) \cap \sigma_{F_2}(R) \equiv \sigma_{F_1}(R) \bowtie \sigma_{F_2}(R)$

XI. $\sigma_{F_1 \wedge \neg F_2}(R) \equiv \sigma_{F_1}(R) - \sigma_{F_2}(R)$

Infine, la **proprietà distributiva del join rispetto all'unione:**

XII. $E \bowtie (E_1 \cup E_2) \equiv (E \bowtie E_1) \cup (E \bowtie E_2)$

²All'esame non verranno chieste le trasformate minori!

Esempio

Si vede un esempio per chiarire le trasformazioni più importanti. Data la seguente base di dati:

Impiegati				Supervisione	
Matricola	Nome	Età	Stipendio	Capo	Impiegato
101	Mario Rossi	34	40	210	101
103	Mario Bianchi	23	35	210	103
104	Luigi Neri	38	61	210	104
105	Nico Bini	44	38	231	105
210	Marco Celli	49	60	301	210
231	Siro Bisi	50	60	301	231
252	Nico Bini	44	70	375	252
301	Sergio Rossi	34	70		
375	Mario Rossi	50	65		

Si vuole trovare il numero di matricola dei capi di impiegati con meno di trenta anni. Immediatamente si potrebbe pensare di eseguire un join sulle due tabelle, specificare la condizione di selezione e infine proiettare l'unica colonna d'interesse:

$$\pi_{\text{Capo}} (\sigma_{\text{Impiegato}=\text{Matricola} \wedge \text{Età} < 30} (\text{Impiegati} \bowtie \text{Supervisione}))$$

È evidente che l'espressione risulta qualitativamente bassa poiché per calcolare pochi valori, in questo caso uno, viene effettuato un prodotto cartesiano che produce un risultato notevole. Quindi, si esegue l'ottimizzazione:

1. (Regola 1) Atomizzazione delle selezioni. Si elimina la coniugazione logica and:

$$\pi_{\text{Capo}} (\sigma_{\text{Impiegato}=\text{Matricola}} (\sigma_{\text{Età} < 30} (\text{Impiegati} \bowtie \text{Supervisione})))$$

2. (Regola 3) Anticipazione della selezione rispetto al join e (Regola 5) inglobamento di una selezione in un prodotto cartesiano a formare un join. Si fonde la prima selezione ($\sigma_{\text{Impiegato}=\text{Matricola}}$) con il prodotto cartesiano formando un join con condizione e successivamente si anticipa la seconda selezione ($\sigma_{\text{Età} < 30}$) rispetto al join:

$$\pi_{\text{Capo}} \left(\sigma_{\text{Età} < 30} (\text{Impiegati}) \underset{\text{Impiegato}=\text{Matricola}}{\bowtie} \text{Supervisione} \right)$$

3. (Regola 4) Anticipazione della proiezione rispetto al join. Si elimina dal primo argomento del join anche gli attributi non necessari:

$$\pi_{\text{Capo}} \left(\pi_{\text{Matricola}} (\sigma_{\text{Età} < 30} (\text{Impiegati})) \underset{\text{Impiegato}=\text{Matricola}}{\bowtie} \text{Supervisione} \right)$$

6 Calcolo relazionale

Il **calcolo relazionale** fa riferimento ad una famiglia di linguaggi di interrogazione, basati sul calcolo dei predicati del primo ordine, che hanno la caratteristica di essere **dichiarativi**, cioè di **specificare le proprietà del risultato delle interrogazioni, anziché la procedura seguita per generarlo**.

Al contrario l'algebra relazionale è un **linguaggio procedurale** poiché le sue espressioni specificano passo passo la costruzione del risultato.

Esistono diversi **tipi** di calcolo relazionale:

- **Calcolo relazione su domini.** Presenta in modo naturale le caratteristiche originali dei linguaggi del calcolo relazionale.
- **Calcolo su tuple con dichiarazioni di range.** Metodo adottato durante questo corso, costituisce la base per molti costrutti disponibili per le interrogazioni nel linguaggio SQL.

Prima di presentare le varie caratteristiche, si tiene a precisare che rispetto al modello relazionale (paragrafo 2.3), il calcolo su tuple con dichiarazioni di range utilizza una **notazione non posizionale**.

6.1 Calcolo su tuple con dichiarazioni di range

Il **calcolo su tuple con dichiarazioni di range** presenta la seguente forma:

$$\{T \mid L \mid f\}$$

In cui le variabili hanno diversi significati:

- La variabile ***T*** indica la **target list**, ovvero la lista degli obiettivi dell’interrogazione. I suoi elementi hanno la seguente forma $Y : x.Z$ con x variabile e Y e Z sequenze di attributi (di pari lunghezza). Chiaramente, gli attributi in Z devono comparire nello schema della relazione che costituisce il *range* di x .

Una notazione alternativa per abbreviare $X : x.X$ è $x.*$;

- La variabile ***L*** indica la **range list**, ovvero la lista contenente le variabili libere della formula f con i relativi *range*. I suoi elementi hanno la seguente forma $x(R)$ con x variabile e R nome di relazione;
- La variabile ***f*** indica una **formula**, la quale può essere di tre tipi:

– Formula con atomi del tipo:

- * $x.A\theta c$, si confronta il valore di x sull’attributo A con la costante c ;
- * $x_1.A_1\theta x_2.A_2$, si confronta il valore di x_1 su A_1 con quello di x_2 su A_2 .

– Formula con connettivi (\wedge, \vee, \neg);

– Formula con quantificatori che associano i range alle relative variabili:

$$\exists x(R)(f) \quad \forall x(R)(f)$$

Intuitivamente, $\exists x(R)(f)$ significa “esiste nella relazione R una tupla x che soddisfa la formula f ”.

6.2 Unione, intersezione e differenza

Purtroppo, il calcolo su tuple con dichiarazioni di range non permette di esprimere tutte le interrogazioni che possono essere formulate in algebra relazionale.

In particolare, le interrogazioni i cui risultati possono provenire indifferentemente da due o più relazioni (in algebra si realizza con un'unione) non possono essere espresse in questa versione del calcolo.

Infatti, i risultati sono costituiti a partire da tutte le variabili libere, i cui range sono definiti nella target list, e ogni variabile ha come range una sola relazione.

Per esempio, si consideri un'unione di due relazioni sugli stessi attributi: $R_1(AB)$ e $R_2(AB)$. Se l'espressione avesse due variabili libere, allora ogni tupla del risultato dovrebbe corrispondere a una tupla di ciascuna delle relazioni, il che non è necessario, poiché l'unione richiede alle tuple nel risultato di comparire in almeno uno degli operandi, non necessariamente in entrambi. Viceversa, se l'espressione avesse una sola variabile libera, questa dovrebbe far riferimento a una sola delle relazioni, senza acquisire tuple dell'altra per il risultato.

Nonostante l'**operatore di unione non sia rappresentabile**, gli operatori di intersezione e differenza risultano esprimibili:

- L'**intersezione** richiede che le tuple siano in entrambi gli operandi, ovvero richiede l'esistenza di una tupla uguale nell'altra relazione. Per esempio:

$$\pi_{BC}(R_1) \cap \pi_{BC}(R_2)$$

Si esprime come:

$$\{x_1.BC \mid x_1(R_1) \mid \exists x_2(R_2)(x_1.B = x_2.B \wedge x_1.C = x_2.C)\}$$

- La **differenza** produce le tuple di un operando non contenute nell'altro e può essere specificata richiedendo le tuple del minuendo che non compaiono nel sottraendo. Per esempio:

$$\pi_{BC}(R_1) - \pi_{BC}(R_2)$$

Si esprime come:

$$\{x_1.BC \mid x_1(R_1) \mid \neg \exists x_2(R_2)(x_1.B = x_2.B \wedge x_1.C = x_2.C)\}$$

6.3 Esempi

Tutte le interrogazioni che verranno affrontate riguardano la seguente basi di dati (già vista nei precedenti paragrafi 5.5.2):

Impiegati				Supervisione	
Matricola	Nome	Età	Stipendio	Capo	Impiegato
101	Mario Rossi	34	40	210	101
103	Mario Bianchi	23	35	210	103
104	Luigi Neri	38	61	210	104
105	Nico Bini	44	38	231	105
210	Marco Celli	49	60	301	210
231	Siro Bisi	50	60	301	231
252	Nico Bini	44	70	375	252
301	Sergio Rossi	34	70		
375	Mario Rossi	50	65		

La prima interrogazione è la seguente:

1. Si richiede la matricola, il nome, l'età e lo stipendio degli impiegati che guadagnano più di 40 mila euro:

$$\{i.* \mid i(\text{Impiegati}) \mid i.\text{Stipendio} > 40\}$$

Per produrre meno risultati è sufficiente modificare la *target list*:

2. Si richiede la matricola, il nome e l'età degli impiegati che guadagnano più di 40 mila euro:

$$\{i.(\text{Matricola}, \text{Nome}, \text{Età}) \mid i(\text{Impiegati}) \mid i.\text{Stipendio} > 40\}$$

Per eseguire interrogazioni che coinvolgono più relazioni, si modifica la *range list* usando più variabili. Si noti la prima condizione atomica che corrisponde a quella del join (e l'altra per la selezione):

3. Trovare le matricole dei capi degli impiegati che guadagnano più di 40 mila euro:

$$\{s.(\text{Capo}) \mid i(\text{Impiegati}), s(\text{Supervisione}) \mid i.\text{Matricola} = s.\text{Impiegato} \wedge i.\text{Stipendio} > 40\}$$

Nel caso di join di una relazione con se stessa si ha più variabili aventi la stessa relazione come range:

4. Trovare il nome e stipendio dei capi degli impiegati che guadagnano più di 40 mila euro:

$$\{\text{NomeC}, \text{StipC} : i'.(\text{Nome}, \text{Stipendio}) \mid i'(\text{Impiegati}), s(\text{Supervisione}), i(\text{Impiegati}) \mid i'.\text{Matricola} = s.\text{Capo} \wedge s.\text{Impiegato} = i.\text{Matricola} \wedge i.\text{Stipendio} > 40\}$$

5. Trovare gli impiegati che guadagnano più del rispettivo capo, mostrando matricola, nome e stipendio di ciascuno di essi e del capo:

$$\{i.(\text{Nome}, \text{Matr}, \text{Stip}), \text{NomeC}, \text{MatrC}, \text{StipC} : i'.(\text{Nome}, \text{Matr}, \text{Stip}) \mid i'(\text{Impiegati}), s(\text{Supervisione}), i'(\text{Impiegati}) \mid i.\text{Matr} = s.\text{Impiegato} \wedge s.\text{Capo} = i'.\text{Matr} \wedge i.\text{Stipendio} > i'.\text{Stipendio}\}$$

Le interrogazioni con i quantificatori (\exists, \forall) mostrano appieno la maggiore sinteticità e praticità del calcolo su tuple con dichiarazioni di range:

6. Trovare la matricola e il nome dei capi i cui impiegati guadagnano tutti più di 40 mila:

- Quantificatore universale:

$$\{i.(\text{Matricola}, \text{Nome}) \mid i(\text{Impiegati}), s(\text{Supervisione}) \mid i.\text{Matr} = s.\text{Capo} \wedge \forall i'(\text{Impiegati}) (\forall s'(\text{Supervisione}) (\neg(s.\text{Capo} = s'.\text{Capo} \wedge s'.\text{Impiegato} = i'.\text{Matr}) \vee i'.\text{Stipendio} > 40)))\}$$

- Quantificatore esistenziale:

$$\{i.(\text{Matricola}, \text{Nome}) \mid i(\text{Impiegati}), s(\text{Supervisione}) \mid i.\text{Matr} = s.\text{Capo} \wedge \neg(\exists i'(\text{Impiegati}) (\exists s'(\text{Supervisione}) (s.\text{Capo} = s'.\text{Capo} \wedge s'.\text{Impiegato} = i'.\text{Matr} \wedge i'.\text{Stipendio} \leq 40))))\}$$

6.4 Esercizi

6.4.1 Aula, insegnamento, docente e lezione

Testo

Dato il seguente schema relazionale:

- **AULA**(NomeAula, Capienza, Edificio);
- **INSEGNAMENTO**(NomeIns, AnnoAcc, Docente);
- **DOCENTE**(Matricola, Nome, Cognome, Età, Ruolo: {ordinario, associato, ricercatore, esterno});
- **LEZIONE**(NomeIns, AnnoAcc, NomeAula, Giorno, Semestre, OraInizio, OraFine).

Si calcoli:

1. Trovare il nome e la capienza delle aule dove nel 2° semestre 2015/2016 il venerdì non si sono svolte lezioni.
2. Trovare i docenti che nel 1° semestre 2016/2017 hanno svolto almeno una lezione di durata maggiore di 180 minuti, riportando nel risultato il nome e il cognome del docente insieme alla durata della lezione (durata in minuti = (OraFine – OraInizio)).
3. Trovare per ogni lezione del 1° semestre 2016/2017 che si svolge il martedì prima delle 17.00 in aula A, la lezione immediatamente successiva nella stessa aula, riportando nel risultato per la prima lezione il nome dell'insegnamento, l'ora di inizio e l'ora di fine e per la lezione successiva solo il nome dell'insegnamento.

Soluzione 1

Trovare il nome e la capienza delle aule dove nel 2° semestre 2015/2016 il venerdì non si sono svolte lezioni.

La soluzione è la seguente:

$$Q = \{\text{Nome, Capienza} : A(\text{NomeAula}, \text{Capienza}) \mid A(\text{Aula}) \mid \neg \exists L(\text{Lezione}) (A(\text{NomeAula}) = L(\text{NomeAula}) \wedge L(\text{AnnoAcc}) = "2015/2016" \wedge L(\text{Semestre}) = "2^o" \wedge L(\text{Giorno}) = "Venerdì")\}$$

Il nome e la capienza delle aule si trovano nella tabella Aula, per cui nella *target list* andranno i suoi due campi e basta.

Nella *range list* è sufficiente l'aula, nonostante nella tabella lezione ci siano alcuni dati importanti.

Nel campo formula ci sono alcune condizioni. Per manifestare la negazione è necessario l'operatore logico *not*. Esso deve operare su un campo presente nella tabella Lezione, ma allo stesso tempo deve avere le caratteristiche richieste dal risultato. Quindi, si scrive che non esiste una *L* appartenente alla tabella Lezione tale per cui essa abbia:

- Il nome dell'aula (chiave) uguale sia nella tabella Aula che Lezione;
- L'anno accademico (tabella Lezione) uguale al valore 2015/2016;
- Il semestre (tabella Lezione) uguale al valore 2^o;
- Il giorno (tabella Lezione) uguale al valore Venerdì.

Quindi, in questo caso si richiede una Lezione che non è presente nella tabella Aula, cioè che non si sia svolta. Tuttavia, oltre a non essersi svolta, deve in un anno, semestre e giorno preciso.

Soluzione 2

Trovare i docenti che nel 1° semestre 2016/2017 hanno svolto almeno una lezione di durata maggiore di 180 minuti, riportando nel risultato il nome e il cognome del docente insieme alla durata della lezione (durata in minuti = (OraFine – OraInizio)).

La soluzione è la seguente:

$$\begin{aligned} Q = \{ & \text{Nome, Cognome : } D(\text{Nome, Cognome}), \\ & \text{Durata Lezione : } L(\text{OraFine}) - L(\text{OraInizio}) \quad | \\ & D(\text{Docente}), I(\text{Insegnamento}), L(\text{Lezione}) \quad | \\ & D(\text{Matricola}) = I(\text{Docente}) \wedge I(\text{NomeIns}) = L(\text{NomeIns}) \wedge \\ & I(\text{AnnoAcc}) = L(\text{AnnoAcc}) \wedge L(\text{AnnoAcc}) = "2016/2017" \wedge L(\text{Semestre}) = 1^{\circ} \wedge \\ & (L(\text{OraFine}) - L(\text{OraInizio})) \geq 180 \} \end{aligned}$$

Il risultato richiede il nome e il cognome del docente, quindi la *target list* avrà tali campi e la durata. Quest'ultima sarà specificata eseguendo la differenza tra l'ora di fine e l'ora di inizio.

Nella *range list* si specifica ovviamente la tabella del docente, dell'insegnamento e della lezione.

Nel campo formula ci sono alcune condizioni. È necessario collegare tutte le chiavi esterne, quindi le matricole del docente, il nome e l'anno accademico tra l'Insegnamento e la Lezione. Inoltre, le condizioni sul numero di semestre, anno accademico e durata delle lezioni, cioè ora di inizio e fine, sono specificate nella tabella Lezione. Quindi, le condizioni vengono espresse su di essa.

Soluzione 3

Trovare per ogni lezione del 1° semestre 2016/2017 che si svolge il martedì prima delle 17.00 in aula A, la lezione immediatamente successiva nella stessa aula, riportando nel risultato per la prima lezione il nome dell'insegnamento, l'ora di inizio e l'ora di fine e per la lezione successiva solo il nome dell'insegnamento.

La soluzione è la seguente:

```

$$Q = \{ \text{Ins, Inizio, Fine : } L1.(\text{NomeIns, OraInizio, OraFine}), \\ \text{InsSuccessivo : } L2.(\text{NomeIns}) \mid L1(\text{LEZIONE}), L2(\text{LEZIONE}) \mid \\ L1.\text{Semestre} = "1" \wedge L2.\text{Semestre} = "1" \wedge L1.\text{AnnoAcc} = "2016/2017" \wedge \\ L2.\text{AnnoAcc} = "2016/2017" \wedge L1.\text{Giorno} = "\text{Martedì}" \wedge L2.\text{Giorno} = "\text{Martedì}" \wedge \\ L1.\text{NomeAula} = "A" \wedge L2.\text{NomeAula} = "A" \wedge L1.\text{OraInizio} < 17:00 \wedge \\ L1.\text{OraInizio} < L2.\text{OraInizio} \wedge \neg \exists L3(\text{LEZIONE}) (L3.\text{Semestre} = "1" \wedge \\ L3.\text{AnnoAcc} = "2016/2017" \wedge L3.\text{Giorno} = "\text{Martedì}" \wedge L3.\text{NomeAula} = "A" \wedge \\ L1.\text{OraInizio} < L3.\text{OraInizio} \wedge L3.\text{OraInizio} < L2.\text{OraInizio}) \}$$

```

Il risultato richiede il nome, l'ora di inizio e l'ora di fine dell'insegnamento della prima lezione. Inoltre, anche il nome dell'insegnamento della lezione successiva alla prima. Quindi, nella *target list* si aggiungono tali campi

Nella *range list* si specificano due riferimenti diversi per la tabella LEZIONE poiché sono necessari per indicare due lezioni diverse. Entrambi si riferiscono alla tabella LEZIONE dato che in essa ci sono tutti i dati richiesti.

Nel campo formula ci sono alcune condizioni. Le prime condizioni sono necessarie per indicare che la lezione prima delle ore 17 : 00 e quella successiva:

- Siano del primo semestre;
- Dell'anno accademico 2016/2017;
- Si svolgono di martedì;
- La prima lezione si svolga prima delle ore 17 : 00 (è sufficiente l'ora di inizio);
- La successiva (seconda) lezione si svolga dopo l'ora di inizio della prima.

Oltre a queste condizioni, è necessario specificare che la seconda lezione *L2* sia la successiva di *L1*. Per farlo, è possibile utilizzare il quantificatore esistenziale per negare l'esistenza di una lezione tra la prima e la seconda. In questo modo, quest'ultima diventa la successiva. Quindi, le condizioni del semestre, anno accademico, giorno e aula sono le stesse. Al contrario, l'ora di inizio dovrà essere successiva alla prima lezione *L1* e dovrà essere anche precedente alla seconda lezione *L2*. Negando l'esistenza di una lezione con queste condizioni, si ottiene la successione.

7 Tecnologie per le basi di dati

7.1 Transazioni

Una **transazione** identifica un'unità elementare di lavoro svolta da un'applicazione, su cui si vogliono associare particolari proprietà di **correttezza**, **robustezza** e **isolamento**.

Un sistema che mette a disposizione un meccanismo per la definizione e l'esecuzione di transazioni con le caratteristiche suddette viene detto **sistema transazionale**.

La **principale caratteristica** di una transazione è che essa, una volta eseguita, si è certi che non ci saranno esecuzioni parziali. Dunque, l'operazione andrà a buon fine, oppure fallirà senza modificare la base dati.

7.2 Transazioni in SQL

La **sintassi** di una transazione in SQL è la seguente:

```
1 begin transaction
2   <istruzione> | commit work | rollback work
3 end transaction
4
```

Dove al posto di:

- **Istruzione**, possono andare una serie di comandi da eseguire;
- **Commit work** (abbreviazione **commit**), la transazione va a buon fine al raggiungimento di tale operazione;
- **Rollback work** (abbreviazione **rollback**), la transazione non ha alcun effetto al raggiungimento di tale operazione.

Una **transazione** è **ben formattata** se rispetta le seguenti caratteristiche:

1. Inizia con un'istruzione **begin transaction**;
2. Termina con un'istruzione **end transaction**;
3. L'esecuzione incontra un **commit** o un **rollback** e successivamente non esegue altri accessi alla base di dati.

Esempio di transazione ben formata:

```
1 begin transaction;
2   update ContoCorrente
3     set Saldo = Saldo + 10
4     where NumConto = 12202;
5   update ContoCorrente
6     set Saldo = Saldo - 10
7     where NumConto = 42177;
8   select Saldo into A
9     from ContoCorrente
10    where NumConto = 42177;
11  if A >= 0
12    then commit work;
13  else rollback work;
14 end transaction;
15
```

7.3 Proprietà acide delle transazioni

Una transazione possiede quattro **proprietà acide** (Atomicity Consistency Isolation Durability, ACID) che sono:

- **Atomicità** (*Atomicity*)
- **Consistenza** (*Consistency*)
- **Isolamento** (*Isolation*)
- **Persistenza** (*Durability*)

7.3.1 Atomicità

L'**atomicità** rappresenta il fatto che una transazione è un'unità **indivisibile** di esecuzione. Quindi, o vengono resi visibili tutti gli effetti di una transazione, oppure essa non deve aver alcun effetto sulla base di dati (tutto o niente).

Per applicare questa proprietà, si implementano due implicazioni:

- **Transazione interrotta prima** del **commit**, il sistema deve ricostruire la situazione esistente prima dell'esecuzione della transazione eliminando il lavoro eseguito fino a quel momento.
- **Transazione interrotta dopo** il **commit**, il sistema deve assicurare che la transazione lasci la base di dati nel suo stato finale.

7.3.2 Consistenza

La **consistenza** richiede che l'esecuzione della transazione non violi i vincoli di integrità definiti sulla base di dati. Nel caso di una violazione, il sistema interviene per annullare la transazione o per correggere la violazione del vincolo. Il controllo della violazione può essere di due tipi:

- **Controllo della violazione immediata.** I controlli vengono eseguiti durante l'esecuzione della transazione. Così facendo, è possibile rimuovere gli effetti della specifica istruzione di manipolazione dei dati che causa la violazione del vincolo, senza imporre un aborto delle transazioni.
- **Controllo della violazione differita.** I controlli vengono eseguiti al termine dell'esecuzione della transazione, ovvero dopo il **commit**. Nel caso in cui ci sia una violazione, viene abortita l'intera transazione.

7.3.3 Isolamento

L'**isolamento** richiede che l'esecuzione di una transazione sia indipendente dalla contemporanea esecuzione di altre transazioni.

Per applicare questa proprietà, si implementano due implicazioni:

- **Esecuzione concorrente** di un insieme di transazioni deve essere analogo al risultato che le stesse transazioni otterrebbero qualora ciascuna di esse fosse eseguita da sola.
- **Esecuzione indipendente.** L'esecuzione indipendente di ogni transazione evita che un eventuale **rollback** provochi un effetto domino generando una **rollback** anche nelle altre.

7.3.4 Persistenza

La **persistenza** richiede che l'esecuzione di una transazione che ha eseguito il **commit** correttamente non venga più perso.

L'applicazione di questa proprietà garantisce gli effetti delle transazioni che, al momento di un eventuale guasto, abbiano già eseguito un **commit**.

7.4 Architettura di riferimento di un DBMS

Si lascia qui di seguito l'immagine di un'architettura di un DBMS. Non è necessario approfondire più di tanto tale argomento, ma è utile sapere della sua conoscenza. L'unica cosa **importante** da ricordare è **quali sono i moduli che contribuiscono a garantire le proprietà delle transizioni**:

- Gestore dei metodi d'accesso:
 - Consistenza
- Gestore dell'esecuzione concorrente:
 - Atomicità
 - Isolamento
- Gestore dell'affidabilità
 - Atomicità
 - Persistenza

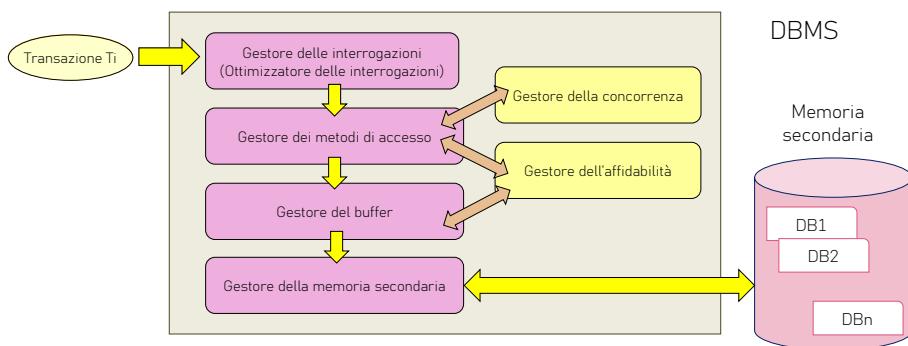


Figura 24: Architettura generale di un DBMS.

7.4.1 Gestore (ottimizzatore) delle interrogazioni

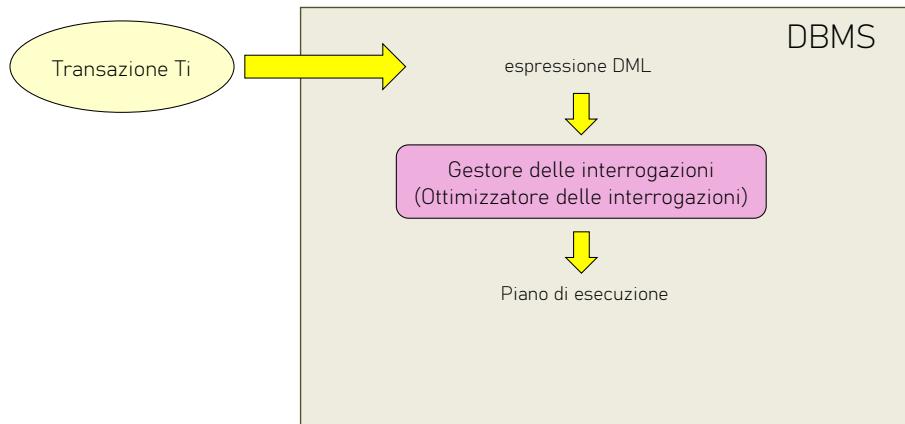


Figura 25: Gestore (ottimizzatore) delle interrogazioni.

7.4.2 Gestore dei metodi d'accesso e dell'esecuzione concorrente

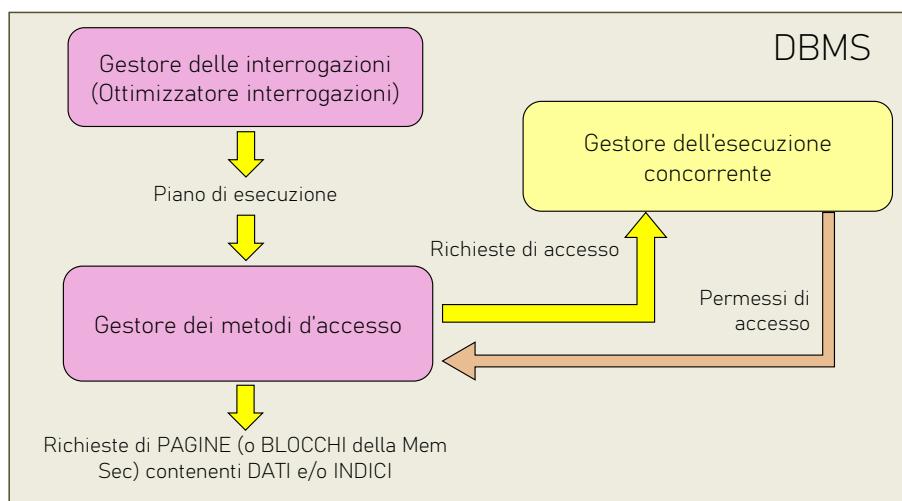


Figura 26: Gestore dei metodi d'accesso e dell'esecuzione concorrente.

7.4.3 Gestore dei metodi d'accesso e dell'affidabilità

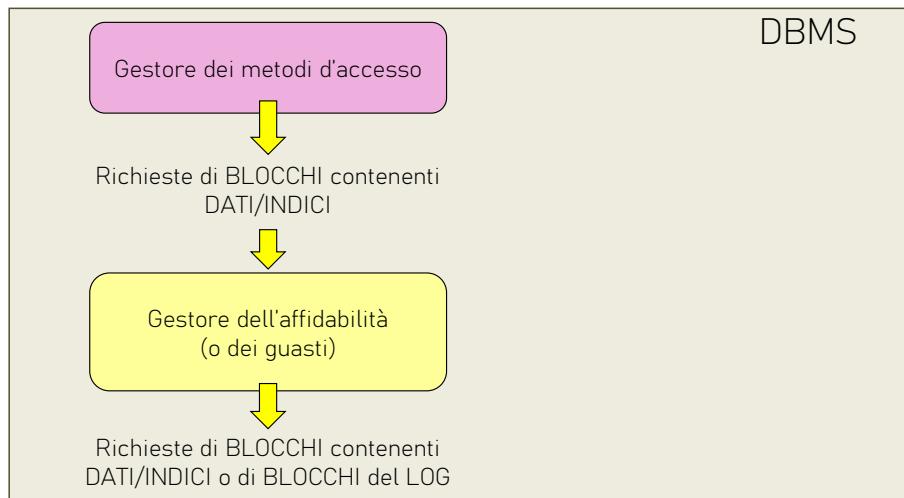


Figura 27: Gestore dei metodi d'accesso e dell'affidabilità.

7.4.4 Gestore dell'affidabilità e del buffer

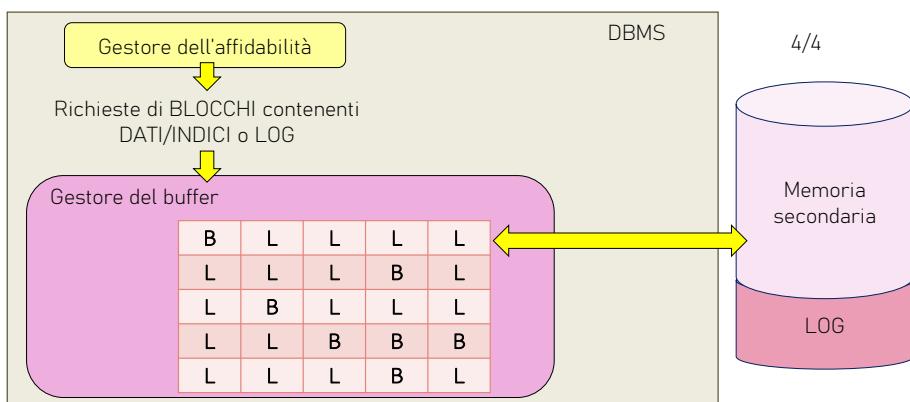


Figura 28: Gestore dell'affidabilità e del buffer.

8 PostgreSQL

8.1 Introduzione e installazione

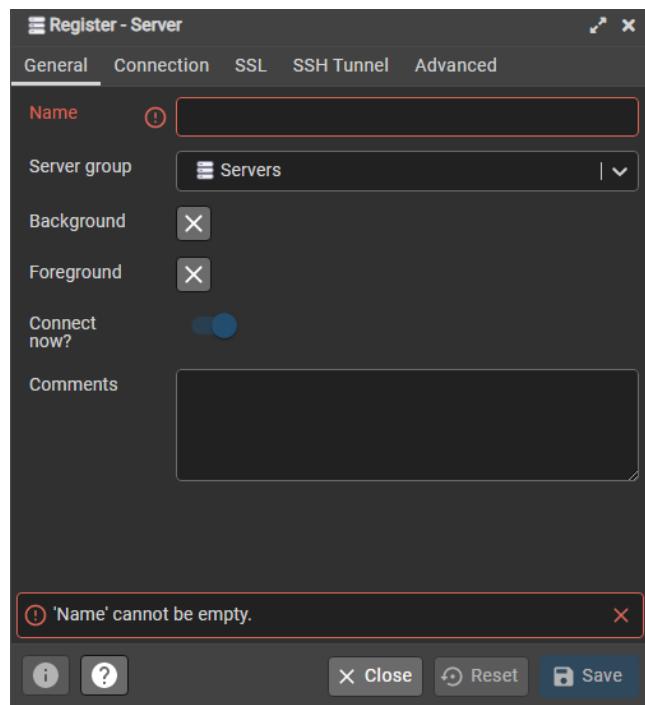
Il sistema PostgreSQL è un Relational Data Base Management System (RDBMS o DBMS) con alcune funzionalità orientate agli oggetti. È possibile installarlo sui sistemi più famosi: Windows, Mac OS, Linux, ecc. Viene gestito da un gruppo di volontari e la sua pagina ufficiale è la seguente: [PostgreSQL](#).

Per installarlo su windows si seguono i seguenti passi:

1. Si scarica il `setup.exe` dalla piattaforma ufficiale: [link download](#);
2. Si esegue il setup andando avanti e lasciando come porta quella predefinita suggerita dal programma;
3. Si rifiuta il download aggiuntivo di un altro programma al termine dell'installazione.

Al termine dell'installazione, è possibile aprire il programma pgAdmin 4, cercandolo banalmente dal menù start di Windows, e iniziare a configurarlo per collegarsi alla piattaforma. La configurazione del server dell'università è la seguente:

1. Sulla colonna di sinistra, si clicca con il destro sopra il nome Server → Register → Server barra della applicazioni si clicca su File → Preferences;
2. All'apertura della seguente finestra:



Si inserisce come nome un alias del server, per esempio “gimmy”.

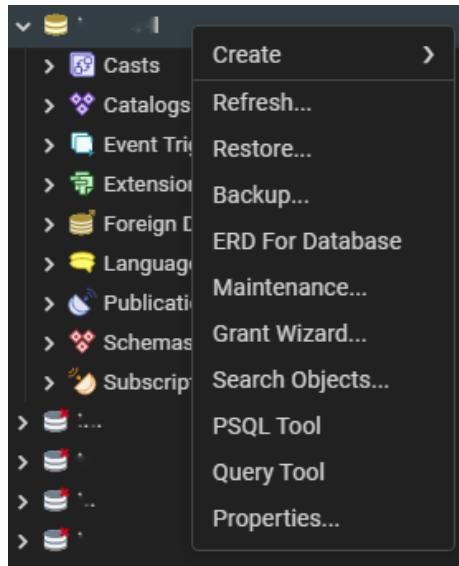
Nella scheda **Connection**:

- (a) Si inserisce come Host name l’indirizzo del server di base di dati, ovvero: dbserver.scienze.univr.it
- (b) Nel campo **port** la porta utilizzata durante l’installazione, se è stata lasciata quella di default è la 5432
- (c) Nei campi **Maintenace database** e **Username** si inserisce l’username utilizzato per identificarsi sulla piattaforma ESSE3, quindi id123abc
- (d) Il campo **password** è possibile riempirlo con la password utilizzata per identificarsi sulla piattaforma ESSE3, oppure lasciarlo vuoto per poi scrivere la password al primo accesso al database.

3. Cliccare su **Save**.

Una volta configurata la piattaforma, sarà possibile accedervi tramite la colonna di sinistra. Sotto la voce **Server** sarà possibile trovare l’alias indicato nella configurazione e tutti i database presenti all’interno della piattaforma.

Andando sul proprio id, si potrà accedere al database personale. Per eseguire le operazioni di query, si utilizza **Query Tool**. Per avviarlo basta andare sul proprio database personale → click destro → **Query Tool**:



Alla sua apertura sarà possibile eseguire query sulla propria base di dati inizialmente vuota. Dopo aver scritto un’espressione ben formattata, sarà possibile eseguirla con la scorciatoia da tastiera F5 o cliccando sul tasto play in alto.

ATTENZIONE! Al termine di ogni sessione, è necessario scollegarsi dal server cliccando con il destro sull’alias iniziale → **Disconnect from server.**

8.2 Caratteri (character, character varying, text)

Il dominio **character** consente di rappresentare singoli caratteri oppure stringhe di lunghezza fissa. Per indicare quest'ultima, viene scritto un valore all'interno delle parentesi tonde dopo il comando. Nel caso in cui non fosse presente, il dominio rappresentare un carattere singolo.

Sintassi:

```
1 character [ ( Lunghezza ) ]
2   [ character set NomeFamigliaCaratteri ]
```

Il campo lunghezza e l'insieme (*set*) sono opzionali. È possibile abbreviare il comando scrivendo **char**.

Esempio di stringa di 20 caratteri: **character(20)**.

Il dominio **character varying** consente di rappresentare stringhe di lunghezza variabile. Il suo funzionamento è identico a quello di **character** con la differenza che la lunghezza è variabile in questo caso.

Sintassi:

```
1 character varying [ ( Lunghezza ) ]
2   [ character set NomeFamigliaCaratteri ]
```

È possibile abbreviare il comando scrivendo **varchar**.

Esempio di stringa di caratteri dell'alfabeto greco a lunghezza variabile, di lunghezza massima 1000: **character varying (1000) character set Greek**.

Il dominio **text** è un'estensione di PostgreSQL e indica una stringa di lunghezza variabile **senza limite** fissato. La sua sintassi è banale: **text**.

8.3 Booleani (`boolean`)

Il dominio `boolean` consente di rappresentare singoli valori booleani: *true* e *false*.

8.4 Tipi numerici esatti (`numeric`, `decimal`, `integer`, `smallint`, `bigint`)

I tipi numerici esatti sono una famiglia contenenti i domini che consentono di rappresentare i valori esatti, interi o con una parte decimale di lunghezza prefissata.

Sintassi:

```
1 numeric [ ( Precisione [, Scala] ) ]
2 decimal [ ( Precisione [, Scala] ) ]
3 bigint
4 integer
5 smallint
```

- `numeric` e `decimal` rappresentano i numeri in base **decimale**. I loro parametri sono:
 - `Precisione`, specifica il numero di cifre significative;
 - `Scala`, specifica la scala di rappresentazione, ovvero si indica quante cifre devono comparire dopo la virgola

La loro **differenza** principale riguarda che la precisione nel dominio `numeric` rappresenta un valore esatto, mentre in `decimal` rappresenta un requisito minimo. Nel caso in cui non venga specificata la dimensione, il sistema usa un valore caratteristico dell'implementazione. Se la scala non si specifica, si assume che valga zero.

- `bigint`, `integer` e `smallint` utilizzano la rappresentazione binaria del calcolatore per rappresentare i valori interi. L'ordine di grandezza dei domini è quello scritto.

Esempi:

- Tutti i numeri decimali con tre valori dopo la virgola e un valore intero (range $-9,999$ e $+9,999$): `decimal(4)`
- I valori con 4 valori dopo la virgola e due interi (range $-99,9999$ e $+99,999$): `decimal(6, 4)`

8.5 Tipi numerici approssimati (`float`, `real`, `double precision`)

Per la rappresentazione di valori reali approssimati, utili nelle rappresentazioni di grandezze fisiche per esempio, si utilizzano principalmente tre comandi.

Sintassi:

```
1 float [ ( Precisione ) ]
2 real
3 double precision
```

Consentono di descrivere numeri approssimati mediante una rappresentazione in virgola mobile, in cui a ciascun numero corrisponde una coppia di valori: mantissa ed esponente. L'importante è sapere che a `float` è possibile assegnare una precisione, la quale rappresenta il numero di cifre dedicate alla rappresentazione della mantissa.

8.6 Istanti e intervalli temporali (`date`, `time`, `timestamp`, `interval`)

Per descrivere **intervalli temporali** o **istanti temporali**, si utilizzano diverse forme.

Sintassi:

```
1 date
2 time [ (Precisione) ] [ with time zone ]
3 timestamp [ (Precisione) ] [ with time zone ]
4 interval PrimaUnitaDiTempo [ to UltimaUnitaDiTempo ]
```

Il campo:

- `date` ammette i campi `year`, `month`, `day`
- `time` ammette i campi `hour`, `minute`, `second`
- `timestamp` ammette tutti i campi elencati

Nei due campi `time` e `timestamp` è possibile accedere al fuso orario tramite `with time zone`. Quindi, si accede ai due campi `timezone_hour` e `timezone_minute` che rappresentano la differenza di fuso orario tra l'ora locale a l'ora universale (UTC).

L’intervallo di tempo rappresenta la durata di un evento e può prendere i seguenti parametri:

- `interval year to month` per indicare che la durata dell’intervallo di tempo deve essere misurata in numero di anni e di mesi. Esempio di rappresentazione di intervalli con numero di anni a cinque cifre (range fino a 99'999 e 11 mesi): `interval year(5) to month`
- `interval day to second` per indicare che la durata dell’intervallo di tempo deve essere misurata in numero di giorni e di secondi. Esempio di rappresentazione di intervalli con numero di giorni a quattro cifre e secondi a sei cifre (range fino a 9'999 giorni, 23 ore, 59 minuti, 59, 999999 secondi): `interval day(4) to second(6)`

Attenzione! Non è possibile mischiare gli insiemi (e.g. `interval year to second`) poiché questo vorrebbe dire sovraccaricare il sistema di calcolo. Dunque, questa operazione non è ammessa.

8.7 Oggetti di grandi dimensioni (`blob`, `clob`)

I due domini `blob`, `clob` consentono di rappresentare oggetti molto grandi, come immagini o video, costituiti da una sequenza arbitraria di valori binari (`blob`, *binary large object*) o di caratteri (`clob`, *character large object*). Il sistema garantisce solo la loro memorizzazione, ma non consente di utilizzarli come criterio di selezione per le interrogazioni.

8.8 Definizione delle tabelle (`create table`)

Una tabella `table` SQL è costituita da una collezione ordinata di attributi e (optional) da un insieme di vincoli.

Sintassi:

```
1 create table NomeTabella
2 ( NomeAttributo Dominio [ ValoreDiDefault ] [ Vincoli ]
3   { , NomeAttributo Dominio [ ValoreDiDefault ] [ Vincoli ] }
4   AltriVincoli
5 )
```

Un esempio della creazione di una tabella:

```
1 create table Dipartimento
2 (
3   Nome      varchar(20) primary key,
4   Indirizzo varchar(50),
5   Citta     varchar(20)
6 )
```

Attenzione alla formattazione!

8.9 Definizione dei domini (create domain)

Nella definizione precedente di tabella, si può far riferimento ai domini predefiniti del linguaggio o a domini definiti dall'utente a partire dai domini predefiniti. Ovvero, è possibile creare insiemi di valori che formano un dominio.

Sintassi:

```
1 create domain NomeDominio as TipoDiDati  
2   [ ValoreDiDefault ]  
3   [ Vincolo ]
```

Il dominio è caratterizzato dal proprio nome, da un valore di default e da un insieme di vincoli eventuali.

Esempio:

```
1 create domain giorniSettimana as char (3)  
2   check(value in('LUN','MAR','MER','GIO','VEN','SAB','DOM'));
```

8.10 Valori di default (default)

I valori di default, come si è visto nei precedenti paragrafi, sono valori che una riga di una tabella assume nel momento in cui viene aggiunta all'interno della base di dati senza che tale campo abbia ricevuto una modifica. Quando il valore di default non viene specificato, si assume il valore nullo.

Sintassi:

```
1 default < GenericoValore | user | null >
```

Si deve scegliere solo uno dei tre valori:

- GenericoValore, rappresenta un valore compatibile con il dominio
- user, impone come valore di default l'identificativo dell'utente che esegue il comando di aggiornamento della tabella
- null, valore nullo

8.11 Vincoli intrarelazionali

8.11.1 Not null

Il valore nullo indica un attributo senza valore. Il suo contrario, ovvero **not null**, indica che il valore nullo non è ammesso come valore dell'attributo e quindi esso deve essere sempre specificato.

Sintassi con esempio:

```
1 Cognome varchar(20) not null
```

8.11.2 Unique

8.11.3 Primary key