

Basi di Dati

Riassunti

Matteo Ceccherini

2021

Indice

1 Introduzione	6
1.1 Sistemi informativi	6
1.2 DataBase Management System (DBMS)	6
1.3 Sistemi informatici	7
1.3.1 Sistemi informatici operativi	7
1.3.2 Elaborazioni su BD : OLTP	8
1.3.3 Sistemi informatici direzionali	8
1.3.4 Elaborazioni su DW: OLAP	8
1.3.5 Differenze tra OLTP e OLAP	9
1.4 Analisi dei dati	9
1.4.1 Big Data	9
1.4.2 Sistemi per basi di dati	9
1.4.3 Le basi di dati gestite dai DBMS	10
1.4.4 Caratteristiche dei dati gestiti dai DBMS	10
1.4.5 Funzionalità dei DBMS	10
1.5 Linguaggio per la definizione della base dei dati (DDL)	11
1.5.1 Livello vista logica	11
1.5.2 Livello logico	11
1.5.3 Livello fisico	11
1.5.4 Funzionalità dei DBMS : linguaggi per l'uso dei dati	12
1.5.5 Funzionalità dei DBMS : meccanismi per il controllo dei dati	12
1.5.6 Controllo dei dati : le transazioni	12
1.5.7 Riepilogo vantaggi e svantaggi dei DBMS	13
2 La progettazione di basi di dati	13
2.1 Modelli informatici	13
2.2 Fasi della progettazione	14
2.3 Aspetti del problema	14
2.3.1 Aspetto ontologico : cosa si modella	14
2.3.2 Aspetto logico : il modello dei dati a oggetti	14
2.4 Conoscenza concreta : cosa si modella	14
2.4.1 Entità e proprietà	15
2.5 Conoscenza concreta : collezione di entità	15
2.6 Conoscenza concreta : Le associazioni	16
2.6.1 Tipi di associazioni	17
2.6.2 Vincoli	17
2.6.3 Esempi	17
2.6.4 Questioni terminologiche	18
2.7 Conoscenza concreta : gerarchie di classi	18
2.7.1 Scelta delle sottoclassi	18
2.8 Modellazione a oggetti : gli oggetti	18
2.8.1 Tipo oggetto	18
2.8.2 Le classi	19
2.9 Modello a oggetti : le associazioni	19
2.10 Modello a oggetti : gerarchia tra tipi oggetto	19
2.11 Ereditarietà	20
2.12 Gerarchie fra le classi	20

2.12.1	Relazione tra sottoinsiemi	21
2.12.2	Gerarchia multipla	22
2.13	La conoscenza astratta	22
2.14	La costruzione di una base di dati	22
2.15	Fasi della progettazione	23
2.15.1	Analisi dei requisiti	23
2.15.2	Progettazione concettuale di schemi settoriali	23
2.15.3	Esempio della biblioteca	24
3	Il modello relazionale	25
3.1	Chiavi e associazioni	25
3.1.1	Trasformazione di schemi a oggetti in relazionali	26
3.1.2	Le sottoclassi	27
3.1.3	Gli attributi multivalore	27
3.2	Linguaggi relazionali	28
3.2.1	Proiezione	28
3.2.2	Restrizione	28
3.2.3	Unione, differenza e prodotto	29
3.2.4	Altri operatori	29
3.2.5	Raggruppamento	29
3.2.6	Esecuzione Raggruppamento	30
3.2.7	Trasformazioni algebriche	30
3.2.8	Operatori algebrici non insiemistici	30
3.2.9	Alberi logici e trasformazioni algebriche	31
4	SQL : Structured Query Language	31
4.1	SQL per interrogare : SELECT, FROM, WHERE	32
4.1.1	Attributi	32
4.1.2	Tabelle	33
4.1.3	Condizione	33
4.2	Sintassi della SELECT	33
4.3	Il valore NULL	35
4.4	Il raggruppamento	36
4.4.1	SQL -> algebra	37
4.4.2	Esecuzione di GROUP BY	37
4.4.3	La clausola HAVING	38
4.5	La quantificazione	38
4.5.1	La quantificazione esistenziale	38
4.5.2	La quantificazione esistenziale : EXISTS	39
4.5.3	La quantificazione esistenziale : GIUNZIONE	39
4.5.4	La quantificazione esistenziale : ANY	40
4.5.5	La quantificazione esistenziale : IN	40
4.5.6	Riassumendo	41
4.5.7	La quantificazione universale	41
4.5.8	La quantificazione universale con ALL	41
4.5.9	Gli insiemi vuoti	42
4.5.10	Ottimizzare il NOT EXISTS	43
4.5.11	NOT IN ED OUTER JOIN	43
4.5.12	SQL per modificare i dati	44

5 SQL per la definizione di basi di dati	45
5.1 Definizione di tabelle	45
5.2 Tabelle inizializzate e tabelle calcolate	46
5.3 Viste modificabili	46
5.3.1 Utilità delle viste	46
5.3.2 Esempi di viste e interrogazioni impossibili	47
5.4 Vincoli d'integrità : chiavi e generali	48
5.5 Create procedure/function	49
5.6 I trigger	49
5.7 Controllo degli accessi	50
5.7.1 Esempi di GRANT	50
5.8 Grafo delle autorizzazioni	51
5.9 Creazione di indici	51
5.10 Catalogo (dei metadati)	51
5.11 Riepilogo	52
6 Uso di SQL da programmi	52
6.1 Linguaggio integrato : PL/SQL di Oracle	53
6.1.1 Cursore	54
6.2 Linguaggio con interfaccia API	55
6.3 SQLJ : Java che ospita SQL	56
7 Teoria relazionale	57
7.1 Dipendenze funzionali	58
7.1.1 Esprimere le dipendenze funzionali	58
7.1.2 Manipolazione di clausole	59
7.1.3 Proprietà delle DF	59
7.2 Regole di inferenza	60
7.2.1 Correttezza e completezza degli assiomi di Armstrong	60
7.3 Derivazione	60
7.4 Chiusura di un insieme F	61
7.4.1 Chiusura lenta	61
7.5 Chiavi e attributi primi	61
7.6 Copertura di insiemi di DF	62
7.6.1 Esistenza della copertura canonica	62
7.7 Decomposizione di schemi	62
7.7.1 Decomposizioni binarie	63
7.8 Proiezione delle dipendenze	63
7.9 Preservazione delle dipendenze	63
7.10 Forme normali	64
7.10.1 FNBC	64
7.11 L'algoritmo di analisi	64
7.12 Terza forma normale	65
7.12.1 L'algoritmo di sintesi : versione base	65
8 Architettura Dei DBMS	66
8.1 Memorie a disco	66
8.2 Gestore memoria permanente e gestore del buffer	67
8.3 Struttura di una pagina	67
8.4 Gestore strutture di memorizzazione	68

8.4.1	Organizzazioni seriale e sequenziale	68
8.4.2	Organizzazione per chiave	69
8.4.3	Ordinamento di archivi	70
8.5	Realizzazione degli operatori relazionali	70
8.5.1	Proiezione	71
8.5.2	Restrizione con condizione semplice	71
8.5.3	Operazioni di aggregazione	71
8.5.4	Giunzione	72
8.6	Operatori logici e fisici	72
8.7	Piani di accesso	74
8.8	Ottimizzatore delle interrogazioni	74
8.8.1	Esempi di piani di accesso	75
8.9	Gestione delle transizioni	78
8.9.1	Le transizioni	78
8.9.2	La transizione per il DBMS	79
8.9.3	Tipi di malfunzionamento	79
8.9.4	Protezione dei dati da malfunzionamento	79
8.9.5	Checkpoint	79
8.9.6	Ripresa dai malfunzionamenti (disfare-rifare)	80
8.9.7	Disfare	80
8.9.8	Rifare	80
8.9.9	Ripartenza dopo un fallimento	81
8.9.10	Serialità e serializzabilità	81
8.9.11	Serializzatore 2pl stretto	81

1 Introduzione

Basi di Dati è una tecnologia di base di gestione delle attività quotidiane dell'organizzazione

Titolo	Codice	Syllabus
Basi di Dati	AA024	Progettazione e interrogazione
Reti di Calc.	AA019	Realizzazione e uso di reti

Materia	AA	Semestre	Titolare
AA024	2007	1	Matteo
AA024	2007	1	Fabio
AA019	2007	1	Giorgio

Tabella 1: Esempi di Basi di Dati (prima tabella materie,seconda corsi)

1.1 Sistemi informativi

Un sistema informativo di un'organizzazione è una combinazione di risorse, umane e materiali, e di procedure organizzative per :

- la raccolta
- l'archiviazione
- l'elaborazione e lo scambio

delle informazioni necessarie alle attività :

- operative (informazioni di servizio)
- di programmazione e controllo (informazioni di gestione)
- di pianificazione strategica (informazioni di governo)

1.2 DataBase Management System (DBMS)

Il **DataBase Management System** è sostanzialmente uno strato software che si frappone fra l'utente ed i dati veri e propri. Grazie a questo strato intermedio l'utente e le applicazioni non accedono ai dati così come sono memorizzati effettivamente, cioè alla loro rappresentazione fisica, ma ne vedono solamente una rappresentazione logica. Ciò permette un elevato grado di indipendenza fra le applicazioni e la memorizzazione fisica dei dati. L'amministratore del database, se ne sente la necessità, può decidere di memorizzare i dati in maniera differente o anche di cambiare il DBMS senza che le applicazioni, e quindi gli utenti, ne risentano. La cosa importante è che non venga cambiata la rappresentazione logica di quei dati, che è la sola cosa che i loro utilizzatori conoscono. Questa rappresentazione logica viene chiamata ‘Schema del database’ ed è la forma di rappresentazione dei dati più a basso livello a cui un utente del database può accedere.

1.3 Sistemi informatici

Il **sistema informatico** è l'insieme delle tecnologie informatiche e della comunicazione a supporto delle attività di un'organizzazione. Mentre il **sistema informativo automatizzato** è quella parte del sistema informativo in cui le informazioni sono raccolte, elaborate, archiviate e scambiate usando un sistema informatico.

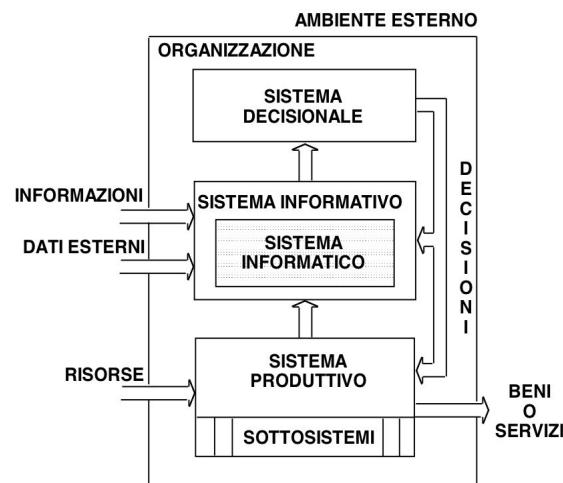


Figura 1: Sistema Informatico Nelle Organizzazioni

1.3.1 Sistemi informatici operativi

I dati sono organizzati in **Basi di Dati** (BD) e le applicazioni si usano per svolgere le classiche attività strutturate e ripetitive dell'azienda nelle aree amministrativa e finanziaria, vendite, produzione, risorse umane ecc.

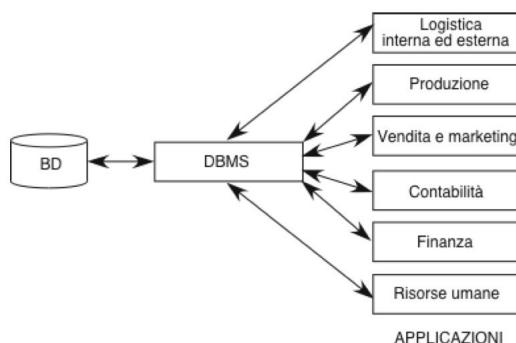


Figura 2: Sistema Informatico Operativo

1.3.2 Elaborazioni su BD : OLTP

Uso principale dei **DBMS**, che comporta una tradizionale elaborazione di transazioni, che realizzano i processi operativi per il funzionamento di organizzazioni e hanno le seguenti caratteristiche:

- Operazioni predefinite e relativamente semplici
- ogni operazione coinvolge "pochi" dati
- Dati al dettaglio, aggiornati ogni volta che ne viene inserito uno nuovo

1.3.3 Sistemi informatici direzionali

I dati sono organizzati in **Data Warehouse** (DW) (*si intende in generale una collezione o aggregazione di dati strutturati, provenienti da fonti interne operazionali (DBMS) ed esterne al sistema informativo*) e gestiti da un opportuno sistema. Le applicazioni, dette di Business intelligence, sono strumenti di supporto ai processi di controllo delle prestazioni aziendali e di decisione manageriale.

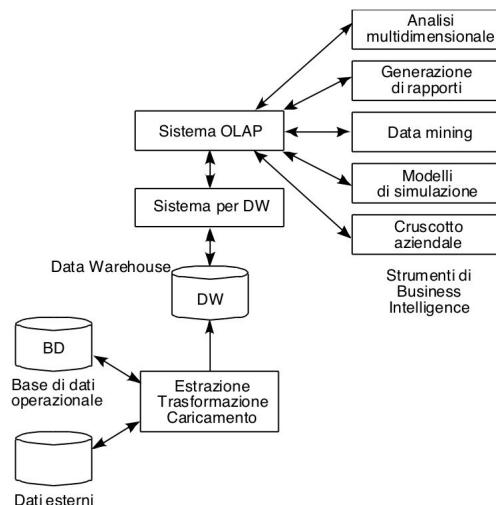


Figura 3: Sistema Informatico Direzionale

1.3.4 Elaborazioni su DW: OLAP

Uso principale dei **Data Warehouse**, che comporta un analisi dei principali dati di supporto alle decisioni con operazioni complesse e casuali che possono coinvolgere molti dati, con aggiornamenti periodici che portano ad avere dati aggregati, storici, anche non attualissimi

1.3.5 Differenze tra OLTP e OLAP

	OLTP	OLAP
Scopi	Supporto operatività	Supporto decisioni
Utenti	Molti, esecutivi	Pochi, dirigenti e analisti
Dati	Analitici, relazionali	Sintetici, multidimensionali
Usi	Noti a priori	Poco prevedibili
Quantità di dati per attività	Bassa (decine)	Alta (Milioni)
Orientamento	Applicazione	Soggetto
Aggiornamenti	Frequenti	Rari
Visione dei dati	Corrente	Storica
Ottimizzati per	Transazioni	Analisi dei dati

Tabella 2: Differenze tra OLTP e OLAP

1.4 Analisi dei dati

Dei dati che ci vengono forniti ci interessano i dati **aggregati**, cioè non il singolo dato, ma la somma, la media, il minimo, il massimo di una misura. Dobbiamo avere anche una presentazione **multidimensionale**, cioè ci interessa incrociare le informazioni, per analizzare da punti di vista diversi e valutare i risultati del business per intervenire sui problemi critici o per cogliere nuove opportunità. Bisogna anche **analizzare a diversi livelli di dettaglio**, cioè passare ad un analisi dettagliata nell'area di interesse. I sistemi informatici operativi non sono adatti per le analisi interattive dei dati.

1.4.1 Big Data

Big Data è un termine ampio, riferito a situazioni in cui l'approccio 'schema-first' tipico di BD o DW risulta troppo restrittivo o troppo lento. Una caratteristica dei Big Data sono le 3 V (**Volume**, **Varietà**, **Velocità**). I Big Data sono spesso associati a sistemi NoSQL, machine learning, approccio Data Lake.

1.4.2 Sistemi per basi di dati

Un **DBMS** è un sistema centralizzato o distribuito che offre opportuni linguaggi per :

- Definire lo **schema** di una basi di dati (lo schema va definito prima di creare i dati).
- scegliere le **strutture dati** per memorizzare dei dati.
- memorizzare i dati rispettando i **vincoli** definiti dallo schema.
- recuperare e modificare i dati interattivamente (linguaggio di interrogazione o **query language**) o da programmi.

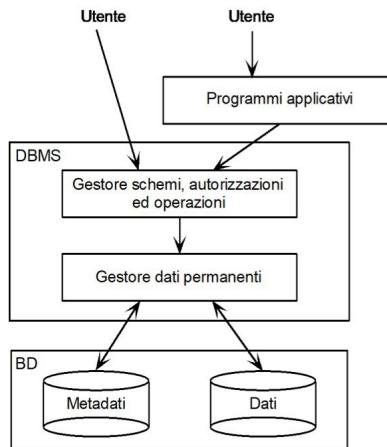


Figura 4: Le Basi Di Dati Gestite Dai DBMS

1.4.3 Le basi di dati gestite dai DBMS

Una base di dati è una raccolta di dati permanenti suddivisi in due categorie :

- i **metadati** : descrivono fatti sullo schema dei dati, utenti autorizzati, applicazioni, parametri quantitativi sui dati, ecc. I metadati sono descritti da uno schema usando il modello dei dati adottato dal DBMS e sono interrogabili con le stesse modalità previste per i dati;
- i **dati** : le rappresentazioni di certi fatti conformi alle definizioni dello schema, con le seguenti caratteristiche.

1.4.4 Caratteristiche dei dati gestiti dai DBMS

Sono organizzati in **insiemi strutturati e omogenei**, fra i quali sono definite delle **relazioni**. La struttura dei dati e le relazioni sono descritte nello schema usando i meccanismi di astrazione del modello dei dati del DBMS. Sono **molti**, in assoluto e rispetto ai metadati, e non possono essere gestiti in memoria temporanea. Sono accessibili mediante **transazioni**, unità di lavoro atomiche che possono avere effetti parziali. Sono **protetti** sia da accesso da parte di utenti non autorizzati, sia da corruzione dovuta a malfunzionamento hardware e software e infine sono **utilizzabili contemporaneamente** da utenti diversi.

1.4.5 Funzionalità dei DBMS

Ha un linguaggio per la definizione della base di dati, dei linguaggi per l'uso dei dati, dei meccanismi per il controllo dei dati con degli strumenti per il responsabile della base di dati e per lo sviluppo delle applicazioni.

1.5 Linguaggio per la definizione della base dei dati (DDL)

È utile distinguere tre diversi livelli di descrizione dei dati (schemi):

- Il livello di vista logico.
- Il livello logico.
- Il livello fisico.

L'approccio con tre livelli di descrizione dei dati è stato proposto come un modo per garantire le proprietà di indipendenza logica e fisica dei dati, che sono fra gli obiettivi più importanti dei DBMS.

Ci sono 2 tipi di indipendenza :

- **Indipendenza fisica** : i programmi applicativi non devono essere modificati in seguito a modifiche dell'organizzazione fisica dei dati.
- **Indipendenza logica** : i programmi applicativi non devono essere modificati in seguito a modifiche dello schema logico.

1.5.1 Livello vista logica

Il livello vista logica: descrive come deve apparire la struttura della base di dati ad una certa applicazione (Schema esterno o vista).

Esempio di schema esterno:

InfCorsi(IdeC char(8), Titolo char(20), NumEsami int)

1.5.2 Livello logico

Il livello logico descrive la struttura degli insiemi di dati e delle relazioni fra loro, secondo un certo modello dei dati, senza nessun riferimento alla loro organizzazione fisica nella memoria permanente (Schema logico).

Esempio di schema logico:

*Studenti(Matricola char(8), Nome char(20),
login char(8), AnnoNascita int, Reddito real)*

Corsi(IdeC char(8), Titolo char(20), Credito int)

Esami(Matricola char(8), IdeC char(8), Voto int)

1.5.3 Livello fisico

Il livello fisico descrive come vanno organizzati fisicamente i dati nelle memorie permanenti e quali strutture dati ausiliarie prevedere per facilitarne l'uso (Schema fisico o interno).

Esempio di schema fisico:

*Relazioni Studenti e Esami organizzate in modo seriale,
Corsi organizzata sequenziale con indice*

Indice su Matricola, (Matricola, IdeC)

1.5.4 Funzionalità dei DBMS : linguaggi per l'uso dei dati

Un DBMS deve prevedere più modalità d'uso per soddisfare le esigenze delle diverse categorie di utenti che possono accedere alla base di dati (dati e catalogo):

- Un'interfaccia grafica per accedere ai dati.
- Un linguaggio di interrogazione per gli utenti non programmati.
- Un linguaggio di programmazione per gli utenti che sviluppano applicazioni :
 - integrazione DDL e DML nel linguaggio ospite: procedure predefinite, estensione del compilatore, precompilazione.
 - comunicazione tra linguaggio e DBMS.
- Un linguaggio per lo sviluppo di interfacce per le applicazioni.

1.5.5 Funzionalità dei DBMS : meccanismi per il controllo dei dati

Una caratteristica molto importante dei DBMS è il tipo di meccanismi offerti per garantire le seguenti proprietà di una base di dati: **Integrità, sicurezza e affidabilità**.

- **Integrità** : mantenimento delle proprietà specificate in modo dichiarativo nello schema (vincoli d'integrità).
- **Sicurezza** : protezione dei dati da usi non autorizzati.
- **Affidabilità** : protezione dei dati da malfunzionamenti hardware o software (fallimenti di transazione, di sistema e disastri) e da interferenze indesiderate dovute all'accesso concorrente ai dati da parte di più utenti.

1.5.6 Controllo dei dati : le transazioni

Una transazione è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea, con le seguenti proprietà:

- **Atomicità** : Le transazioni che terminano prematuramente (aborted transactions) sono trattate dal sistema come se non fossero mai iniziata; pertanto eventuali loro effetti sulla base di dati sono annullati.
- **Serializzabilità** : Nel caso di esecuzioni concorrenti di più transazioni, l'effetto complessivo è quello di una esecuzione seriale.
- **Persistenza** : Le modifiche sulla base di dati di una transazione terminata normalmente sono permanenti, cioè non sono alterabili da eventuali malfunzionamenti.

1.5.7 Riepilogo vantaggi e svantaggi dei DBMS

Vantaggi :

- Indipendenza dei dati
- Recupero efficiente dei dati
- Integrità e sicurezza dei dati
- Accessi interattivi, concorrenti e protetti dai malfunzionamenti
- Amministrazione dei dati
- Riduzione dei tempi di sviluppo delle applicazioni
- La riduzione dei costi della tecnologia e i possibili tipi di DBMS disponibili sul mercato facilitano la loro diffusione.

Svantaggi :

- Prima di caricare i dati è necessario definire uno schema
- Possono essere costosi e complessi da installare e mantenere in esercizi
- Possono gestire solo dati strutturati e omogenei
- Sono ottimizzati per le applicazioni OLTP, non per quelle OLAP

2 La progettazione di basi di dati

Progettare una basi di dati vuol dire progettare la struttura dei dati e le applicazioni. La progettazione di dati è L'attività più importante, infatti per progettarla al meglio è necessario che i dati siano un modello fedele del dominio del discorso.

2.1 Modelli informatici

Un modello astratto è la rappresentazione formale di idee e conoscenze relative a un fenomeno. Gli aspetti di un modello sono :

- il modello è la rappresentazione di certi fatti
- la rappresentazione è data con un linguaggio formale
- il modello è il risultato di un processo di interpretazione, guidato dalle idee e conoscenze possedute dal soggetto che interpreta

La stessa realtà può utilmente essere modellata in modi diversi, e a diversi livelli di astrazione.

2.2 Fasi della progettazione



Ciascuna delle fasi è centrata sulla modellazione

2.3 Aspetti del problema

Quale conoscenza del dominio del discorso si rappresenta? *Aspetto ontologico*

Con quali meccanismi di astrazione si modella? *Aspetto logico*

Con quale linguaggio formale si definisce il modello? *Aspetto linguistico*

Come si procede per costruire un modello? *Aspetto pragmatico*

2.3.1 Aspetto ontologico : cosa si modella

Nell'aspetto ontologico abbiamo 3 tipi di conoscenza : quella **concreta** (i fatti), quella **astratta** (struttura e vincoli sulla conoscenza concreta), e infine quella **procedurale** (le operazioni di base, le operazioni degli utenti e come si comunica con il sistema informatico)

2.3.2 Aspetto logico : il modello dei dati a oggetti

Un **modello dei dati** è un insieme di meccanismi di astrazione per descrivere la struttura della conoscenza concreta. La descrizione della struttura della conoscenza concreta e dei vincoli di integrità è un particolare modello dei dati chiamato **schema**. Useremo come notazione grafica una variante dei cosiddetti diagrammi a oggetti o diagrammi ER (Entità-Relazione).

Notazioni fondamentali: *Oggetto, Tipo di oggetti, Classe, Eredità, Gerarchia fra i tipi, Gerarchia fra classi*

2.4 Conoscenza concreta : cosa si modella

Sono fatti specifici che si vogliono rappresentare con :

- le **entità** con le loro proprietà
- le **collezioni** di entità omogenee
- le **associazioni** fra entità

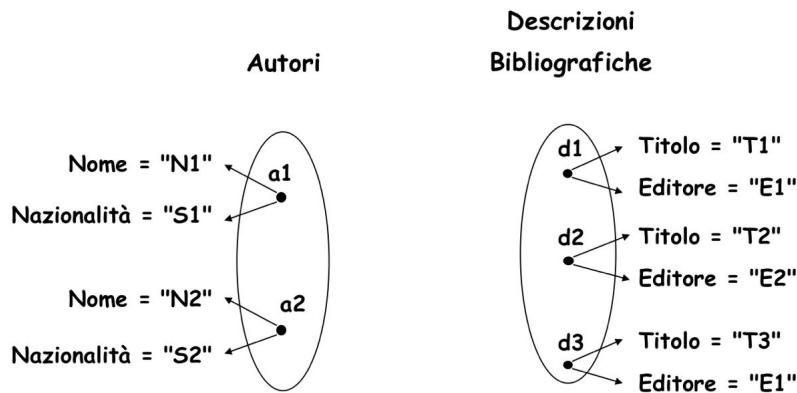


Figura 5: Esempio Di Collezioni

2.4.1 Entità e proprietà

le *entità* sono ciò di cui interessa rappresentare alcuni fatti (o proprietà), invece le *proprietà*, che sono una coppia (Attributo, valore di un certo tipo), sono fatti che interessano solo in quanto descrivono caratteristiche di determinate entità.

2.5 Conoscenza concreta : collezione di entità

Ci sono vari *tipi di entità* dove ogni entità appartiene ad un tipo che ne specifica la natura, invece una *collezione* è un insieme variabile nel tempo di entità omogenee (dello stesso tipo).

Tipo di entità	Attributi
Studente	Nome, AnnoNascita, Matricola, E-mail, ...
Esame	Materia, Candidato, Voto, Lode, Data, ...
Auto	Modello, Colore, Cilindrata, Targa, ...
Descrizione bibliografica	Autori, Titolo, Editore, LuogoEdizione, ...

Tabella 3: Esempi di tipi di entità

Certi fatti possono essere interpretati come proprietà in certi contesti e come entità in altri come ad esempio :

- Descrizione bibliografica con attributi ...
- Autore con attributi Nome, Nazionalità, AnnoNascita ...
- Editore con attributi Nome, Indirizzo, e-mail, ...

2.6 Conoscenza concreta : Le associazioni

Un'istanza di associazione è un fatto che correla due o più entità, stabilendo un legame logico tra di loro. Un'associazione $R(X, Y)$ fra due collezioni di entità X ed Y è un insieme di istanze di associazione tra elementi di X e Y , che varia in generale nel tempo. Il prodotto cartesiano $(X \times Y)$ è detto dominio dell'associazione.

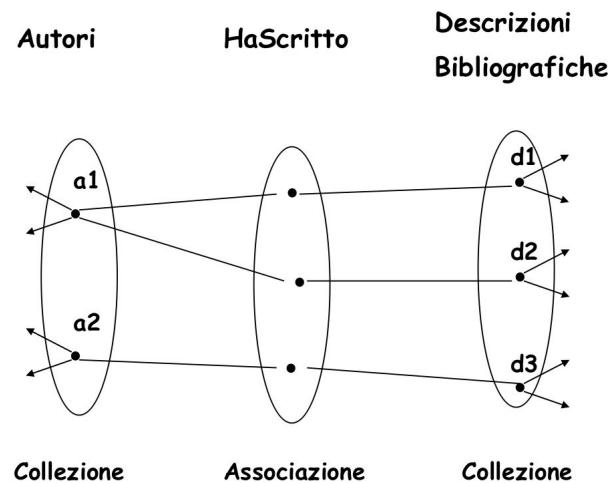


Figura 6: Esempio Associazioni

Esempi :

- *a descrizione bibliografica con titolo “Basi di Dati” riguarda il documento fisico con collocazione “d3-55-2”*
- *l’utente Tizio ha in prestito una copia della Divina Commedia*

2.6.1 Tipi di associazioni

Un'associazione è caratterizzata dalle seguenti proprietà strutturali : **molteplicità e totalità**. **Vincolo di univocità** : Un'associazione $R(X, Y)$ è univoca rispetto ad X se per ogni elemento x di X esiste al più un elemento di Y che è associato ad x ; se non vale questo vincolo, l'associazione è multi valore rispetto ad X.

Esempi :

- $R(X, Y)$ è $(1:N)$ se essa è multi valore su X ed univoca su Y
- $R(X, Y)$ è $(N:1)$ se essa è univoca su X e multi valore su Y
- $R(X, Y)$ è $(N:M)$ se essa è multi valore su X e multi valore su Y
- $R(X, Y)$ è $(1:1)$: se essa è univoca su X e univoca su Y
- Frequentia(Studenti, Corsi) ha cardinalità $(M:N)$
- Insegna(Professori, Corsi) ha cardinalità $(1:N)$
- SuperatoDa(Esami, Studenti) ha cardinalità $(N:1)$
- Dirige(Professori, Dipartimenti) ha cardinalità $(1:1)$

2.6.2 Vincoli

Vincolo di totalità : Un'associazione $R(X, Y)$ è totale (o surgettiva) su X se per ogni elemento x di X esiste almeno un elemento di Y che è associato ad x ; se non vale questo vincolo, l'associazione è parziale rispetto ad X.

Ad esempio, Insegna(Professori, Corsi) è totale su Corsi in quanto non può esistere un corso del piano di studi senza il corrispondente docente che lo tiene

2.6.3 Esempi

Tipi di associazioni fra Persone e Città:

- NataA(Persone, Città) **ha cardinalità (N:1)**, totale su Persone e parziale su Città
- HaVisitato(Persone, Città) **ha cardinalità (N:M)**, parziale su Persone e Città
- E'SindacoDi(Persone, Città) **ha cardinalità (1:1)**, parziale su Persone e Città
- E'VissutaA(Persone, Città) **ha cardinalità (N:M)**, parziale su Persone e Città

2.6.4 Questioni terminologiche

- Dominio del discorso → *Modello Informatico*
- entità → *oggetto (entity instance)*
- tipo entità → *tipo oggetto (entity type)*
- collezione → *classe (entity)*
- associazione → *associazione (relationship)*

2.7 Conoscenza concreta : gerarchie di classi

Spesso le classi di entità sono organizzate in una gerarchia di specializzazione/generalizzazione. Una classe della gerarchia minore di altre viene detta sottoclasse (le altre sono superclassi). Due importanti caratteristiche delle gerarchie sono :

- ereditarietà delle proprietà
- gli elementi di una sottoclasse sono un sottoinsieme degli elementi della superclasse

2.7.1 Scelta delle sottoclassi

La classe degli studenti universitari è una generalizzazione delle classi:

- matricole e dei laureandi.
- studenti in corso e degli studenti fuori corso.
- studenti pisani e degli studenti fuori sede.
- studenti maschi e delle studentesse.

2.8 Modellazione a oggetti : gli oggetti

L'**oggetto** è un entità software con stato, comportamento e identità. Ad ogni entità del dominio corrisponde un oggetto del modello. Lo stato è modellato da un insieme di costanti o variabili con valori di qualsiasi complessità. Un oggetto può rispondere a dei messaggi, restituendo valori memorizzati nello stato o calcolati con una procedura locale.

2.8.1 Tipo oggetto

Il primo passo nella costruzione di un modello consiste nella classificazione delle entità del dominio con la definizione dei tipi degli oggetti che le rappresentano. Un **tipo oggetto** definisce l'insieme dei messaggi (interfaccia) a cui può rispondere un insieme di possibili oggetti. I nomi dei messaggi sono detti anche attributi degli oggetti.

Il tipo oggetto nei diagrammi ER :

- I tipi oggetti non si rappresentano nei diagrammi, dove invece l'attenzione è sulle collezioni e sulle associazioni
- Tuttavia, la rappresentazione grafica di una collezione indica anche gli attributi del tipo oggetto associato

2.8.2 Le classi

Una classe è un insieme di oggetti dello stesso tipo, modificabile con operatori per includere o estrarre elementi dell'insieme.

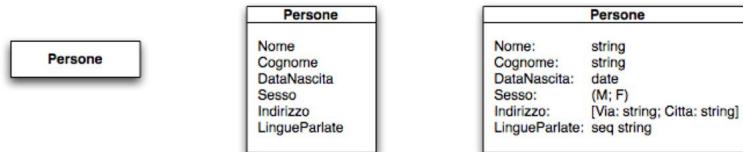


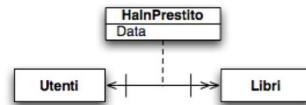
Figura 7: Le Classi

2.9 Modello a oggetti : le associazioni

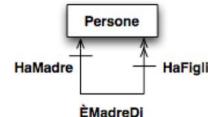
- Le associazioni si modellano con un costrutto apposito



- Le associazioni possono avere delle proprietà



- Le associazioni possono essere ricorsive



2.10 Modello a oggetti : gerarchia tra tipi oggetto

Fra i tipi oggetto è definita una relazione di sottotipo, con le seguenti proprietà:

- È asimmetrica, riflessiva e transitiva (relazione di ordine parziale)
- Se T è sottotipo di T', allora gli elementi di T possono essere usati in ogni contesto in cui possano apparire valori di tipo T' (sostitutività), in particolare :
 - gli elementi di T hanno tutte le proprietà degli elementi di T'
 - per ogni proprietà p in T', il suo tipo in T è un sottotipo del suo tipo in T'

2.11 Ereditarietà

L'ereditarietà (inheritance) permette di definire :

- un tipo oggetto a partire da un altro.
- implementazione di un tipo oggetto a partire da un'altra implementazione.

Normalmente l'eredità tra tipi si usa solo per definire sottotipi, e l'ereditarietà tra implementazioni per definire implementazioni di sottotipi (ereditarietà stretta); in questo caso :

- gli attributi possono essere solo aggiunti
- gli attributi possono essere ridefiniti solo specializzandone il tipo

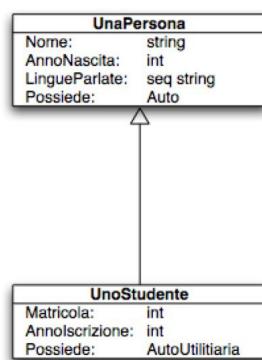


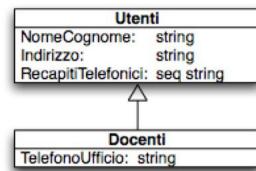
Figura 8: Tipi Definiti Per eredità

2.12 Gerarchie fra le classi

Fra le classi può essere definita una relazione di sottoclasse, detta anche Sottoinsieme, con le seguenti proprietà :

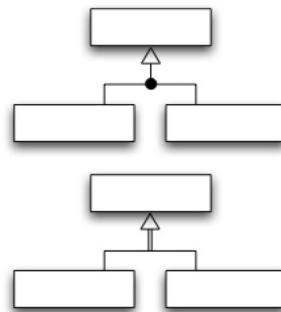
- È asimmetrica, riflessiva e transitiva.
- Se C è sottoclasse di C', allora il tipo degli elementi di C è sottotipo del tipo degli elementi di C' (*vincolo intenzionale*)
- Se C è sottoclasse di C', allora gli elementi di C sono un sottoinsieme degli elementi di C' (*vincolo estensionale*).

- Inclusione



- Vincoli su insiemi di sottoclassi:

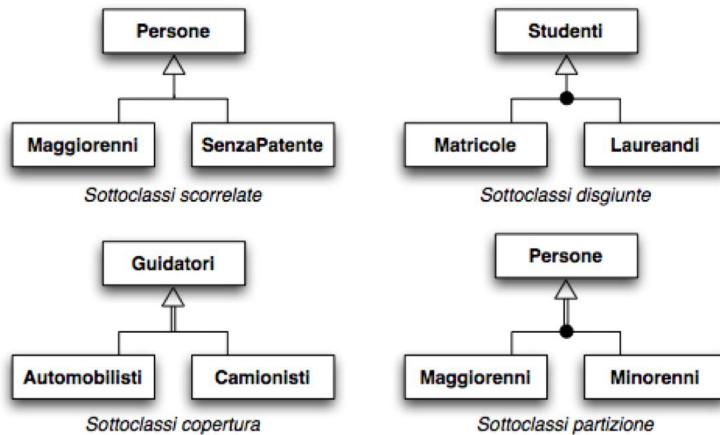
- Disgiunzione



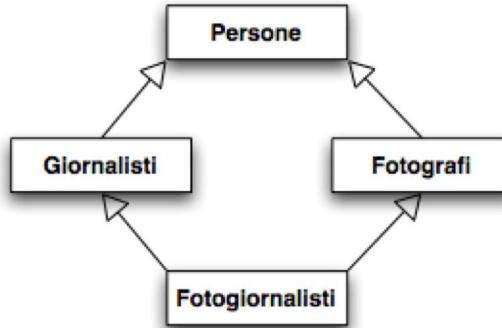
- Copertura

Figura 9: Esempi Gerarchie Fra Le Classi

2.12.1 Relazione tra sottoinsiemi



2.12.2 Gerarchia multipla



2.13 La conoscenza astratta

La conoscenza astratta è formata da fatti generali che descrivono la struttura della conoscenza concreta (collezioni, tipi di entità, associazioni) e restrizioni sui valori possibili della conoscenza concreta sui modi in cui essi possono evolvere nel tempo (vincoli integrità).

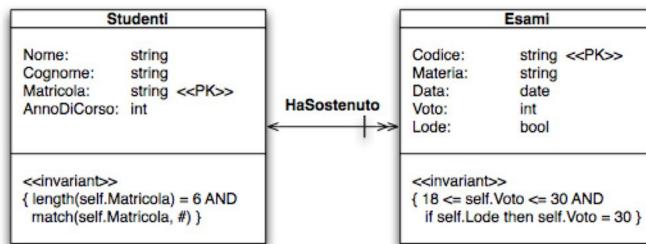


Figura 10: Descrittore Di Classe Con Vincoli

2.14 La costruzione di una base di dati

La costruzione di una base di dati si suddivide in diverse fasi :

- Analisi dei requisiti
- Progettazione :
 - Progettazione concettuale, logica, fisica dei dati
 - Progettazione delle applicazioni
- Realizzazione

Noi spesso considereremo *l'analisi dei requisiti* una parte della progettazione.

2.15 Fasi della progettazione

- Analisi dei requisiti → *specifiche dei requisiti, schemi di settore*
- Progettazione concettuale → *schema concettuale*
- Progettazione logica → *schema logico*
- Progettazione fisica → *schema fisico*

2.15.1 Analisi dei requisiti

- Analizza il sistema esistente e raccogli requisiti informali
- Elimina ambiguità imprecisioni e disuniformità
- Raggruppa le frasi relative a diverse categorie di dati, vincoli, e operazioni
- Costruisci un glossario
- Disegna lo schema di settore
- Specifica le operazioni
- Verifica la coerenza tra operazioni e dati

2.15.2 Progettazione concettuale di schemi settoriali

- Identificare le classi
- Identificare le associazioni e le loro proprietà strutturali
- Identificare gli attributi delle classi e associazioni e i loro tipi
- Elencare le chiavi
- Individuare le sottoclassi
- Individuare le generalizzazioni

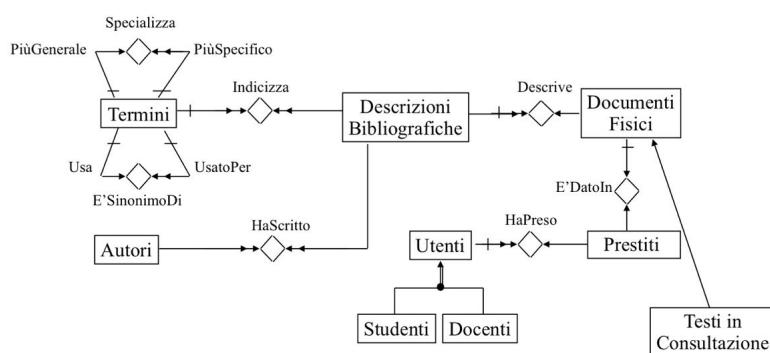


Figura 11: Esempio Grafico

2.15.3 Esempio della biblioteca

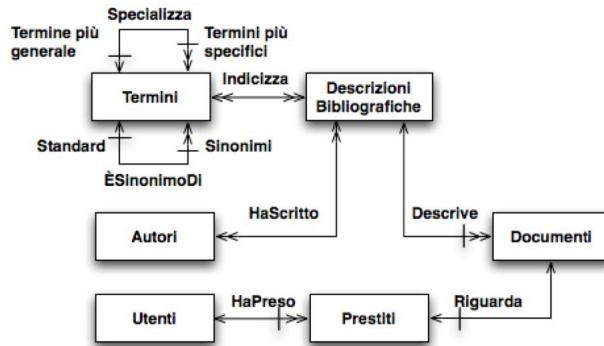


Figura 12: Esempio Biblioteca

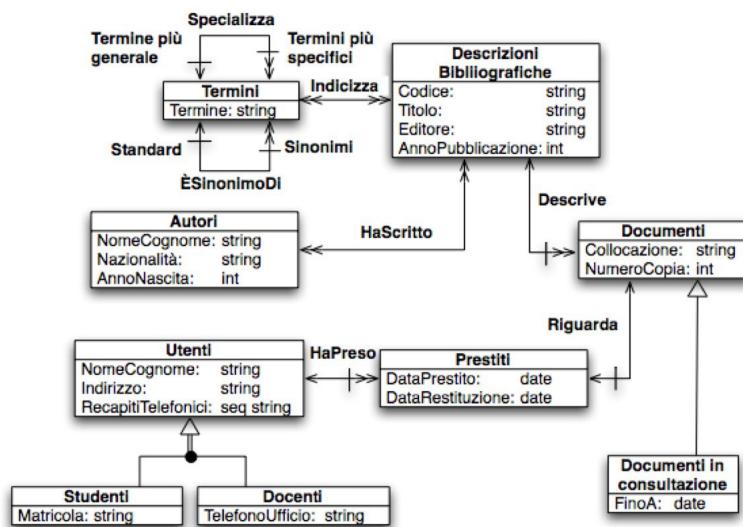


Figura 13: Esempio Biblioteca Con Sottoclassi

3 Il modello relazionale

I meccanismi per definire una base di dati con il modello relazionale sono **l'ennupla** e la **relazione**. Un tipo ennupla T è un insieme finito di coppie (*Attributo, Tipo elementare*) e $R(T)$ è lo schema della relazione R . Lo schema di una base di dati è un insieme di schemi di relazione $R(T_i)$ e un'istanza di uno schema $R(T)$ è un insieme finito di ennupple di tipo T

The diagram shows two relational tables, **Studenti** and **Esami**, with their respective column headers. The **Studenti** table has columns: **Nome**, **Matricola**, **Provincia**, and **AnnoNascita**. The **Esami** table has columns: **Materia**, **Candidato***, **Data**, and **Voto**. Annotations include:
 - A bracket labeled **RELAZIONE** groups both tables.
 - Two curly braces above the tables indicate **SUPERCHIAVE** (superkey) for the first table and **SUPERCHIAVE E CHIAVE** (superkey and key) for the second table.
 - A bracket on the right side of the tables is labeled **SCHEMA DI RELAZIONE** (schema of relation).
 - Another bracket on the far right is labeled **ISTANZA DI RELAZIONE** (instance of relation).

Studenti			
Nome	Matricola	Provincia	AnnoNascita
Isaia	071523	PI	1982
Rossi	067459	LU	1984
Bianchi	079856	LI	1983
Bonini	075649	PI	1984

Esami			
Materia	Candidato*	Data	Voto
BD	071523	12/01/06	28
BD	067459	15/09/06	30
FP	079856	25/10/06	30
BD	075649	27/06/06	25
LMM	071523	10/10/06	18

Figura 14: Esempio Modello Relazionale

3.1 Chiavi e associazioni

Definizioni di vari termini :

- **Superchiave** : Insieme di attributi che la rendono univoca
- **Chiave** : è una superchiave minimale (insieme di un solo attributo)
- **Chiave primaria** : Campo chiave più adatto per identificare (immutabile) (*preferibilmente è un campo non visibile all'utente*)
Esempio: (*Matricola*) e (*Nome,Indirizzo*) sono chiavi in: *Studenti*(*Matricola: Int, Nome: String, Indirizzo: String*)
- **Chiave esterna** : Puntatore di un elemento tra tabelle diverse
Esempio : *Candidato* in *Esami* è chiave esterna verso la tabella esterna solo se appartiene allo stesso numero di matricola
- **Associazioni** : Appartiene al modello a oggetti (schema concettuale)

Studenti(Nome: string, Matricola: string, Provincia: string, AnnoNascita:int)

Esami(Materia: string, Candidato*: string, Data: string, Voto: int)

Relazioni:

```

    graph TD
      Studenti[Studenti] -->|Candidato| Esami[Esami]
  
```

Studenti

Nome	Matricola	Provincia	AnnoNascita
Isaia	071523	PI	1982
Rossi	067459	LU	1984
Bianchi	079856	LI	1983
Bonini	075649	PI	1984

Esami

Materia	Candidato*	Data	Voto
BD	071523	12/01/06	28
BD	067459	15/09/06	30
FP	079856	25/10/06	30
BD	075649	27/06/06	25
LMM	071523	10/10/06	18

Figura 15: Esempio Relazione

3.1.1 Trasformazione di schemi a oggetti in relazionali

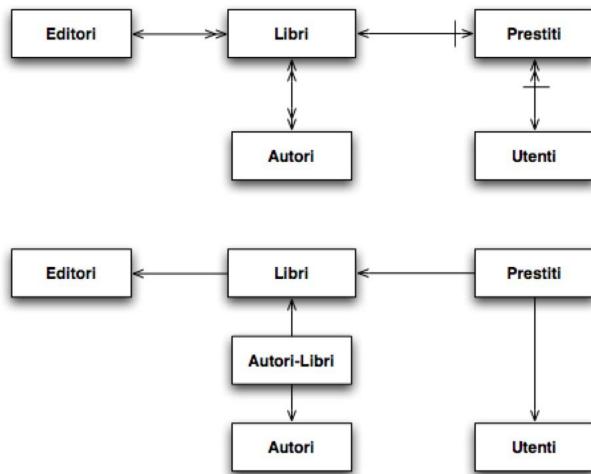


Figura 16: Schemi A Oggetti In Relazionali

3.1.2 Le sottoclassi

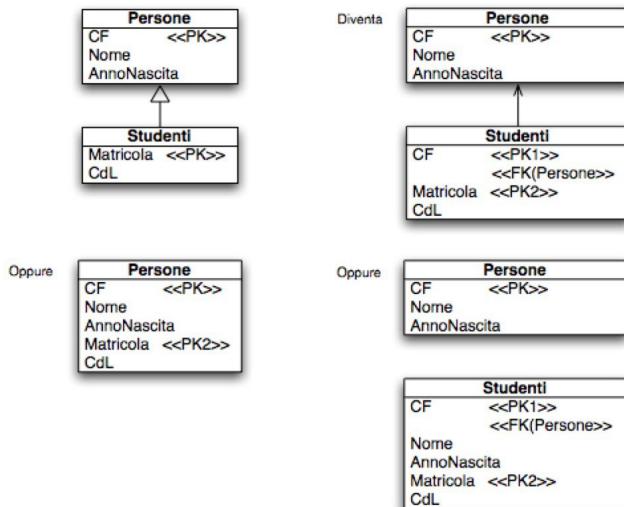


Figura 17: Esempi Sottoclassi

3.1.3 Gli attributi multivalore

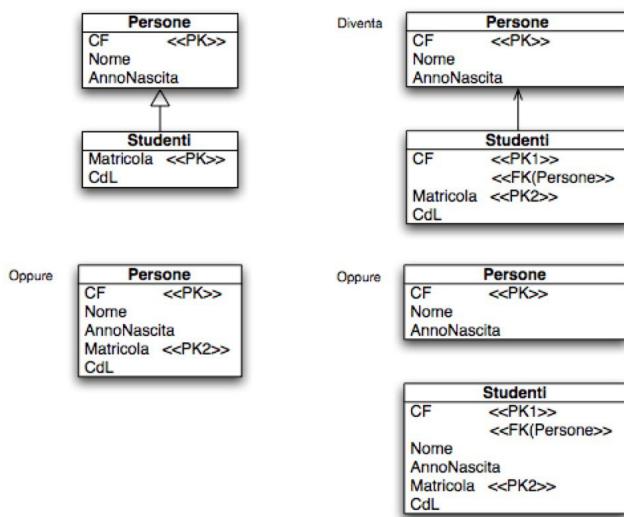


Figura 18: Esempio Attributi Multivalore

3.2 Linguaggi relazionali

Il **calcolo relazionale** è un linguaggio dichiarativo di tipo logico dal quale è stato derivato *SQL*, invece l'**algebra relazionale** è un insieme di operatori su relazioni che danno come risultato relazioni cioè non si usa come linguaggio di interrogazione dei DBMS ma come rappresentazione interna delle interrogazioni.

3.2.1 Proiezione

Scelgo le colonne

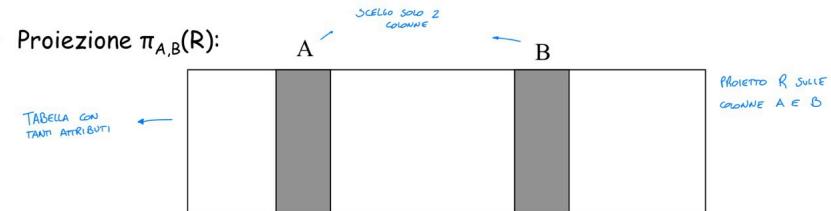


Figura 19: Esempio Proiezione

3.2.2 Restrizione

La condizione può essere anche una espressione

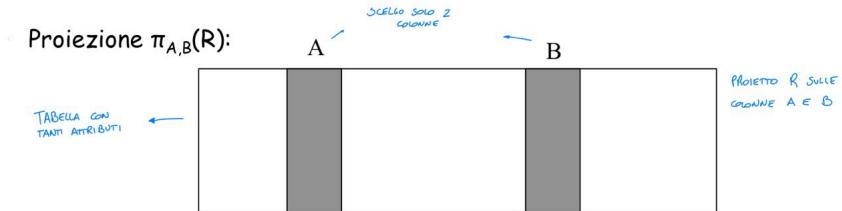


Figura 20: Esempio Restrizione

3.2.3 Unione, differenza e prodotto

- **Unione** : $R \cap S$ (*Ammissibile se e solo se R e S hanno lo stesso tipo*)
- **Differenza** : $R - S$

- **Prodotto**: $R \times S$

a	A	b	B
a1	A1	b1	B1
a1	A1	b2	B2
a1	A1	b3	B3
a2	A2	b1	B1
a2	A2	b2	B2
a2	A2	b3	B3

× =

b	B
b1	B1
b2	B2
b3	B3

Figura 21: Esempio Prodotto

3.2.4 Altri operatori

- Ridenominazione : $\delta_{A \rightarrow B}(R)$
- Intersezione : $R \cap S$
- Giunzione : $R \bowtie_{R.A=S.B} S$
(*Prodotto seguito da restrizione*)
- Giunzione naturale : $R \bowtie S$
Unione di 3 fasi :
 - Ridenominazione degli attributi comuni
 - Giunzione sulla condizione naturale
 - Proiezione sui duplicati

Alla fine per semplicità semplificheremo tutto in una sola fase che è : Fare giunzione sulla condizione naturale (stesso nome stesso valore)

3.2.5 Raggruppamento

Raggruppamento : $(A_i) \gamma_{(F_i)}(R)$

Gli A_i sono attributi di R e le F_i sono espressioni che usano funzioni di aggregazione (*min, max, count, sum, ...*)

Il valore del raggruppamento è una relazione calcolata dal partizionamento delle ennuple di R messe nello stesso gruppo insieme alle ennuple con valori uguali a A_i , poi si calcolano le espressioni F_i per ogni gruppo e infine per ogni gruppo si restituisce una sola ennupla con attributi i valori degli A_i e delle espressioni F_i .

3.2.6 Esecuzione Raggruppamento

- Per ogni candidato: numero degli esami, voto minimo, massimo e medio:

$\{\text{Candidato}\} \gamma_{\{\text{count(*)}, \text{min(Voto)}, \text{max(Voto)}, \text{avg(Voto)}\}} (\text{Esami})$

Materia	Candidato	Voto	Docente		Materia	Candidato	Voto	Docente
DA	1	20	10	→	DA	1	20	10
LFC	2	30	20		MTI	1	24	30
MTI	1	30	30		LFC	2	30	20
LP	2	20	40		LP	2	20	40

Candidato	Count(*)	min(Voto)	max(Voto)	avg(Voto)
1	2	20	24	22
2	2	20	30	25

Figura 22: Esempio Raggruppamento

3.2.7 Trasformazioni algebriche

Le trasformazioni algebriche, basate su regole di equivalenza fra espressioni algebriche, consentono di scegliere diversi ordini di unione e di anticipare proiezioni e restrizioni. Alcuni esempi con la relazione R(A, B, C, D) :

- $\pi_A(\pi_{A,B}(R)) \equiv \pi_A(R)$
- $\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_1 \wedge C_2}$
- $\sigma_{C_1 \wedge C_2}(R \times S) \equiv \sigma_{C_1}(R) \times \sigma_{C_2}(S)$
- $R \times (S \times T) \equiv (R \times S) \times T$
- $(R \times S) \equiv (S \times R)$
- $\sigma_C(x\gamma f(R)) \equiv x\gamma f(\sigma_C(R))$

3.2.8 Operatori algebrici non insiemistici

- $\pi_{A_i}^b(R) : \text{proiezione multinsiemistica (senza eliminazione dei duplicati)}$
- $T_{A_i}(R) : \text{ordinamento}$

3.2.9 Alberi logici e trasformazioni algebriche

- Consideriamo le relazioni $R(A, B, C, D)$ e $S(E, F, G)$ e l'espressione:

$$\pi_{A,F}(\sigma_{A=100 \wedge F>5 \wedge A=E}(R \times S))$$

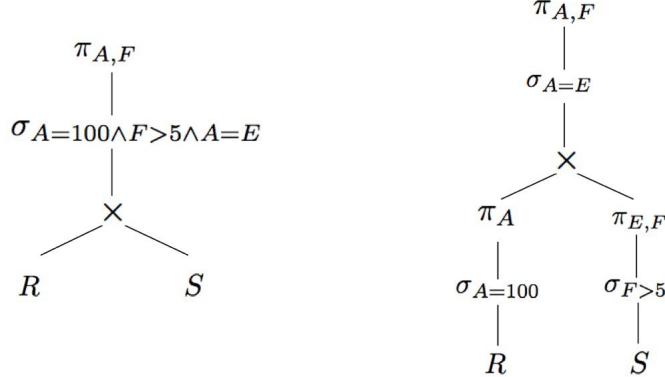


Figura 23: Esempio Alberi Logici

4 SQL : Structured Query Language

SQL è stato definito nel 1973 ed è oggi il linguaggio universale dei sistemi relazionali.

Esempi :

- **SELECT** $s.Nome, e.Data$
FROM Studenti s , Esami e
WHERE $e.Materia = 'BD'$ **AND** $e.Voto = 30$ **AND** $e.Matricola = s.Matricola$
- **SELECT** $s.Nome As Nome, 2018 - s.AnnoNascita As Eta, 0 As NumeroEsami$
From Studenti s
WHERE NOT EXISTS (**SELECT** *
FROM Esami e
WHERE $e.Matricola = s.Matricola$)

4.1 SQL per interrogare : SELECT, FROM, WHERE

SQL è un calcolo su multinsieme e i suoi comandi basi sono :

- **SELECT** [*Distinct*] Attributo {, Attributo}
- **FROM** Tabella [*Ide*] {, Tabella[*Ide*]}
- **[WHERE** *Condizione*]
 - [] → *opzionale*
 - { } → (*opzionale o ripetibile*)

La semantica è formata dal **prodotto + restrizione + proiezione**

4.1.1 Attributi

Un attributo A di una tabella "R x" si denota come : **A oppure R.A oppure x.A**

- **Attributi ::= ***
 - | Expr [[AS] Nuovonome] {, Expr [[AS] Nuovonome]}
- **Expr ::= [Ide.]Attributo | Const**
 - | (Expr) | [-] Expr [Op Expr]
 - | COUNT(*)
 - | AggrFun ([DISTINCT] [Ide.]Attributo)
- **e AS x:** dà un nome alla colonna di **e**
- **AggrFun ::= SUM | COUNT | AVG | MAX | MIN**
- **AggrFun:** o si usano tutte funzioni di aggregazione (e si ottiene un'unica riga) o non se ne usa nessuna.

Figura 24: Lista Di Attributi

4.1.2 Tabelle

Le tabelle si possono combinare tra loro

- Le tabelle si possono combinare usando:
 - "," (prodotto): FROM T1,T2
 - Giunzioni di vario genere:
 - Studenti s JOIN Esami e ON s.Matricola=e.Matricola
 - Studenti s JOIN Esami e USING Matricola
 - Studenti s NATURAL JOIN Esami e
 - Studenti s LEFT JOIN Esami e ON s.Matricola=e.Matricola
 - LEFT JOIN - USING
 - NATURAL LEFT JOIN
 - RIGHT JOIN
 - FULL JOIN

Figura 25: Lista Delle Tabelle

4.1.3 Condizione

Le condizioni sono combinazioni booleane di predicati

- Combinazione booleana di predicati tra cui:
 - Expr Comp Expr
 - Expr Comp (Sottoselect che torna un valore)
 - [NOT] EXISTS (Sottoselect)
 - Espr Comp (ANY | ALL) (Sottoselect)
 - Expr [NOT] IN (Sottoselect) (oppure IN (v1,...,vn))
 - Comp: <, =, >, <>, <=, >=

Figura 26: Lista Condizioni

4.2 Sintassi della SELECT

Sottoselect :

```
SELECT [Distinct] Attributi  
FROM Tabelle  
[WHERE Condizione]  
[GROUP BY A1,...,An [HAVING Condizione]]
```

Select :

Sottoselect

{ (UNION | INTERSECT | EXCEPT) *Sottoselect* }
[ORDER BY Attributo [DESC] { , Attributo }

Trovare il nome, la matricola e la provincia degli studenti:

```
SELECT Nome, Matricola, Provincia  
FROM Studenti
```

Nome	Matricola	Provincia
Isaia	171523	PI
Rossi	167459	LU
Bianchi	179856	LI
Bonini	175649	PI

Figura 27: Esempio Proiezione

Trovare tutti i dati degli studenti di Pisa:

```
SELECT *  
FROM Studenti  
WHERE Provincia = 'PI'
```

Nome	Matricola	Provincia	AnnoNascita
Isaia	171523	PI	1996
Bonini	175649	PI	1996

Trovare la matricola, l'anno di nascita e il nome degli studenti di Pisa
(Proiezione+Restrizione):

```
SELECT Nome, Matricola, AnnoNascita  
FROM Studenti  
WHERE Provincia = 'PI'
```

Nome	Matricola	AnnoNascita
Isaia	171523	1996
Bonini	175649	1996

Figura 28: Esempio Restrizione

Trovare tutte le possibili coppie Studente-Esame:	<code>SELECT * FROM Studenti, Esami</code>
Trovare tutte le possibili coppie Studente - Esame sostenuto dallo studente:	<code>SELECT * FROM Studenti s, Esami e WHERE s.Matricola = e.Matricola</code>
Trovare il nome e la data degli esami per gli studenti che hanno superato l'esame di BD con 30:	<code>SELECT Nome, Data FROM Studenti s, Esami e WHERE e.Materia = 'BD' AND e.Voto = 30 AND e.Matricola = s.Matricola</code>

Figura 29: Esempio Prodotto E Giunzione

Studenti ordinati per Nome

```
SELECT *
FROM Studenti
ORDER BY Nome;
```

Numero di elementi di Studenti

```
SELECT count(*)
FROM Studenti;
```

Anno di nascita minimo, massimo e medio degli studenti:

```
SELECT min(AnnoNascita), max(AnnoNascita), avg(AnnoNascita)
FROM Studenti;
```

Figura 30: Esempio Ordinamenti E Funzioni Di Aggregazione

4.3 Il valore NULL

Il valore di un campo di un'ennupla può mancare per variate ragioni e proprio per questo SQL fornisce il valore speciale **NULL** per tali situazioni. La presenza di NULL introduce una serie di problemi :

- occorrono dei predicati per controllare se un valore è/non è NULL.
- a condizione "reddito > 8" è vera o falsa quando il reddito è uguale a NULL? Cosa succede degli operatori AND, OR e NOT?

Occorre una logica a 3 valori (vero, falso e unknown) e va definita opportunamente la semantica dei costrutti.

4.4 Il raggruppamento

**SELECT...FROM...WHERE...GROUP BY A₁,..,A_n
[HAVING condizione]**

Semantica :

- Esegue le clausole FROM - WHERE
- Partiziona la tabella risultante rispetto all'uguaglianza su tutti i campi A₁... A_n (solo in questo caso, si assume NULL = NULL)
- Elimina i gruppi che non rispettano la clausola HAVING
- Da ogni gruppo estrae una riga usando la clausola SELECT
 - Per ogni materia, trovare nome della materia e voto medio:
 - Per ogni materia:
 - Un attributo della materia
 - Una funzione aggregata sugli esami della materia
 - Soluzione:

```
SELECT e.Materia, avg(e.Voto)
FROM Esami e
GROUP BY e.Materia
```
 - Per ogni studente, nome e voto medio:

```
SELECT s.Nome, avg(e.Voto)
FROM Studenti s, Esami e
WHERE s.Matricola = e.Matricola
GROUP BY s.Matricola, ...
```
 - È necessario scrivere:
 - GROUP BY s.Matricola, s.Nome
 - Gli attributi espressi non aggregati nella select (s.Nome) devono essere inclusi tra quelli citati nella GROUP BY (s.Matricola, s.Nome)
 - Gli attributi aggregati (avg(e.Voto)) vanno scelti tra quelli non raggruppati

Figura 31: Esempi Raggruppamento

4.4.1 SQL -> algebra

```

SELECT DISTINCT X, F
FROM R1,...,Rn
WHERE C1
GROUP BY Y
HAVING C2
ORDER BY Z

X, Y, Z sono insiemi di attributi
F, G sono insiemi di espressioni
aggregate, tipo count(*) o sum(A)
X,Z ⊆ Y, F ⊆ G, C2 nomina solo attributi
in Y o espressioni in G

```

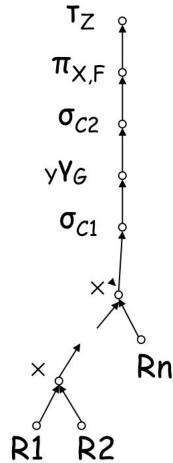


Figura 32: Esempio Da SQL A Algebra

4.4.2 Esecuzione di GROUP BY

```

SELECT Matricola, count(*) AS NEsami, min(Voto), max(Voto), avg(Voto)
FROM Esami
GROUP BY Matricola
HAVING count(*) > 1;

```

Materia	Matricola	Voto	Docente		Materia	Matricola	Voto	Docente
DA	1	20	10	→	DA	1	20	10
LFC	2	30	20		MTI	1	30	30
MTI	1	30	30		LFC	2	30	20
LP	2	20	40		LP	2	20	40

Matricola	NEsami	min(Voto)	max(Voto)	Avg(Voto)
1	2	20	30	25
2	2	20	30	25

Figura 33: Esempio Esecuzione Di Group By

4.4.3 La clausola HAVING

Se la SELECT contiene sia espressioni aggregate (MIN, COUNT...) che attributi non aggregati, allora DEVE essere presente la clausola GROUP BY.

Le clausole HAVING e SELECT citano solo :

- espressioni su attributi di raggruppamento
- funzioni di aggregazione applicate ad attributi non di raggruppamento.

4.5 La quantificazione

Tutte le interrogazioni su di una associazione multivalore vanno quantificate



Non *Gli studenti che hanno preso 30 (Ambiguo!)* Ma :

- Gli studenti che hanno preso sempre (o solo) 30: *universale*
- Gli studenti che hanno preso qualche (almeno un) 30: *esistenziale*
- Gli studenti che non hanno preso qualche 30 (senza nessun 30): *universale*
- Gli studenti che non hanno preso sempre 30: *esistenziale*

Universale negata = esistenziale:

- *Non tutti i voti sono >24 = Almeno un voto >24* (esistenziale)

Esistenziale negata = universale:

- *Non esiste voto diverso da 30 = Tutti i voti sono uguali a 30* (universale)

4.5.1 La quantificazione esistenziale

Gli studenti con almeno un voto sopra 27

Servirebbe un quantificatore $\exists.e \in Esami - Di(s) : e.Voto > 27$ (StileOQL) :

SELECT s.Nome
FROM Studenti s
WHERE EXIST Esami e **WHERE** e.Matricola = s.Matricola : e.Voto>27

Altra query esistenziale: gli studenti in cui non tutti gli esami hanno voto 30, ovvero : gli studenti in cui qualche esame ha voto diverso da 30

SELECT s.Nome
FROM Studenti s
WHERE EXIST Esami e **WHERE** e.Matricola = s.Matricola : e.Voto>27

Le parole in rosso non fanno parte del linguaggio SQL

4.5.2 La quantificazione esistenziale : EXISTS

- Gli studenti con almeno un voto sopra 27 stile OQL:

```
SELECT s.Nome  
FROM Studenti s  
WHERE EXIST Esami e WHERE e.Matricola = s.Matricola : e.Voto >  
27
```

- In SQL diventa:

```
SELECT s.Nome  
FROM Studenti s  
WHERE EXISTS (SELECT *  
    FROM Esami e  
    WHERE e.Matricola = s.Matricola AND e.Voto > 27)
```

Figura 34: Esempio Exists

4.5.3 La quantificazione esistenziale : GIUNZIONE

- Gli studenti con almeno un voto sopra 27, tramite EXISTS:

```
SELECT s.Nome  
FROM Studenti s  
WHERE EXISTS (SELECT *  
    FROM Esami e  
    WHERE e.Matricola = s.Matricola AND e.Voto > 27)
```

- Stessa quantificazione esistenziale, tramite giunzione:

```
SELECT s.Nome  
FROM Studenti s, Esami e  
WHERE e.Matricola = s.Matricola AND e.Voto > 27
```

Figura 35: Esempio Giunzione

4.5.4 La quantificazione esistenziale : ANY

- ANY equivale ad EXISTS

- La solita query:

```
SELECT s.Nome FROM Studenti s  
WHERE EXISTS (SELECT * FROM Esami e  
WHERE e.Matricola = s.Matricola AND e.Voto > 27)
```

- Si può esprimere anche tramite ANY:

```
SELECT s.Nome FROM Studenti s  
WHERE s.Matricola = ANY (SELECT e.Matricola FROM Esami e  
WHERE e.Voto > 27)  
SELECT s.Nome FROM Studenti s  
WHERE 27 < ANY (SELECT e.Voto FROM Esami e  
WHERE e.Matricola = s.Matricola)
```

Figura 36: Esempio Any

4.5.5 La quantificazione esistenziale : IN

- IN è solo un'abbreviazione di =ANY

- La solita query:

```
SELECT s.Nome FROM Studenti s  
WHERE s.Matricola = ANY (SELECT e.Matricola FROM Esami e  
WHERE e.Voto > 27)
```

- Si può esprimere anche tramite IN:

```
SELECT s.Nome FROM Studenti s  
WHERE s.Matricola IN (SELECT e.Matricola FROM Esami e  
WHERE e.Voto > 27)
```

Figura 37: Esempio In

4.5.6 Riassumendo

La quantificazione esistenziale si fa con :

- Exists (*Il più espressivo*)
- Giunzione
- =Any, >Any, <Any (*Non aggiungono potere espressivo*)
- In (*Non aggiungono potere espressivo*)

4.5.7 La quantificazione universale

Esempio : *Gli studenti che hanno preso solo 30*

In SQL diventa :

```
SELECT s.Nome FROM Studenti s
WHERE NOT EXISTS (SELECT *
                   FROM Esami e
                   WHERE e.Matricola = s.Matricola AND e.Voto <> 30)
```

4.5.8 La quantificazione universale con ALL

La query:

```
SELECT s.Nome FROM Studenti s
WHERE FORALL Esami e WHERE e.Matricola = s.Matricola : e.Voto
= 30)
```

Diventa :

```
SELECT s.Nome FROM Studenti s
WHERE 30 = ALL (SELECT e.Voto FROM Esami e
                 WHERE e.Matricola = s.Matricola)
```

Sostituendo EXISTS con =ANY, la solita query (studenti con tutti 30):

```
SELECT s.Nome FROM Studenti s
WHERE NOT EXISTS (SELECT * FROM Esami e
                   WHERE e.Matricola = s.Matricola
                   AND e.Voto <> 30)
```

Diventa:

```
SELECT s.Nome FROM Studenti s
WHERE NOT(s.Matricola=ANY (SELECT e.Matricola
                           FROM Esami e
                           WHERE e.Voto <> 30))
```

Ovvero:

```
SELECT s.Nome FROM Studenti s
WHERE s.Matricola < >ALL(SELECT e.Matricola FROM Esami e
WHERE e.Voto < > 30)
```

Naturalmente, < >**ALL** è lo stesso di **NOT IN**

Esempio:

- Trovare gli studenti che hanno preso solo trenta:

```
SELECT s.Nome
FROM Studenti s
WHERE NOT EXISTS (SELECT *
FROM Esami e
WHERE e.Matricola = s.Matricola AND e.Voto < > 30)
```

- Perché trovo anche Rossi? Cosa cambia se invece di **NOT EXISTS** uso
<>ALL oppure **NOT IN**?

Nome	Matricola	Provincia	AnnoNascita	Mater.	Matricola	Voto
Bianco	1	PI	1996	RC	1	30
Verdi	2	PI	1992	IS	2	30
Rossi	3	PI	1992	RC	2	20

Figura 38: Esempio Quantificazione Universale E Gli Insiemi Vuoti

4.5.9 Gli insiemi vuoti

Se voglio gli studenti che hanno preso solo trenta, e hanno superato qualche esame:

```
SELECT s.Nome
FROM Studenti s
WHERE NOT EXISTS (SELECT *
FROM Esami e
WHERE e. Matricola = s.Matricola AND e.Voto < 27)
AND EXISTS (SELECT *
FROM Esami e
WHERE e. Matricola = s.Matricola)
```

Oppure :

```
SELECT s.Nome
FROM Studenti s, Esami e
WHERE s.Matricola = e.Matricola
GROUP BY s.Matricola, s. Nome
HAVING Min(e.Voto) >= 27
```

4.5.10 Ottimizzare il NOT EXISTS

Loop interno valutato una volta per ogni studente:

```
SELECT s.Nome  
FROM Studenti s  
WHERE NOT EXISTS (SELECT *  
                   FROM Esami e  
                   WHERE e.Matricola = s.Matricola AND e.Voto > 21)
```

Loop interno valutato una sola volta (loop interno decorrelato):

```
SELECT s.Nome  
FROM Studenti s  
WHERE s.Matricola NOT IN (SELECT e.Matricola  
                           FROM Esami e  
                           WHERE e.Voto > 21)
```

4.5.11 NOT IN ED OUTER JOIN

Loop interno valutato una sola volta (loop interno decorrelato):

```
SELECT s.Nome  
FROM Studenti s  
WHERE s.Matricola NOT IN (SELECT e.Matricola  
                           FROM Esami e  
                           WHERE e.Voto > 21)
```

Outer join:

```
SELECT s.Nome  
FROM Studenti s LEFT JOIN (SELECT *  
                            FROM Esami e  
                            WHERE e.Voto > 21) USING Matricola  
WHERE e.Voto IS NULL
```

R	S			
A	B	A	C	
1	a	1	x	
2	b	3	y	
3	c	5	z	

SELECT *
FROM R
NATURAL JOIN
S;

A	B	C
1	a	x
3	c	y

Chiamato anche: natural **inner** join

R	S			
A	B	A	C	
1	a	1	x	
2	b	3	y	
3	c	5	z	

SELECT *
FROM R
NATURAL LEFT JOIN
S;

A	B	C
1	a	x
2	b	
3	c	y

Chiamato anche : natural left **outer** join

Figura 39: Esempio LEFT OUTER JOIN

R	S			
A	B	A	C	
1	a	1	x	
2	b	3	y	
3	c	5	z	

SELECT *
FROM R
NATURAL RIGHT JOIN
S;

A	B	C
1	a	x
3	c	y
5		z

Chiamato anche : natural right **outer** join

R	S			
A	B	A	C	
1	a	1	x	
2	b	3	y	
3	c	5	z	

SELECT *
FROM R
NATURAL FULL JOIN
S;

A	B	C
1	a	x
2	b	
3	c	y
5		z

Chiamato anche : natural full **outer** join

Figura 40: Esempio OUTER JOIN: RIGHT, FULL

4.5.12 SQL per modificare i dati

- INSERT INTO Tabella [(A1,..,An)]
(VALUES (V1,..,Vn) | AS Select)
- UPDATE Tabella
SET Attributo = Expr, ..., Attributo = Expr
WHERE Condizione
- DELETE FROM Tabella
WHERE Condizione

5 SQL per la definizione di basi di dati

SQL non è solo un linguaggio di interrogazione (*Query Language*) ma anche un linguaggio per la definizione di basi di dati (Data-definition language (**DDL**))), per stabilire controlli sull'uso dei dati (GRANT) e per modificare i dati.

- CREATE SCHEMA Nome AUTHORIZATION Utente
- CREATE TABLE o VIEW, con vincoli
- CREATE INDEX
- CREATE PROCEDURE
- CREATE TRIGGER

5.1 Definizione di tabelle

Le tabelle si creano con il comando **CREATE** e si può eliminare con il comando **DROP** o cambiare con il comando **ALTER**

```
CREATE TABLE Nome  
  (Attributo Tipo [ValoreDefault] [VincoloAttributo])  
  { Attributo Tipo [Default] [VincoloAttributo] }  
  { VincoloTabella }
```

Default := DEFAULT { valore | NULL | username }

Nuovi attributi si possono aggiungere con :

```
ALTER TABLE Nome ADD COLUMN NuovoAttributo Tipo
```

```
CREATE TABLE Impiegati  
  ( Codice CHAR(8) NOT NULL,  
    Nome CHAR(20),  
    AnnoNascita INTEGER CHECK (AnnoNascita < 2000),  
    Qualifica CHAR(20) DEFAULT 'Impiegato',  
    Supervisore CHAR(8),  
    PRIMARY KEY pk_impiegato (Codice),  
    FOREIGN KEY fk_Impiegati (Supervisore)  
    REFERENCES Impiegati )
```

```
CREATE TABLE FamiliariACarico  
  ( Nome CHAR(20),  
    AnnoNascita INTEGER,  
    GradoParentela CHAR(10),  
    CapoFamiglia CHAR(8)  
    FOREIGN KEY fk_FamiliariACarico (CapoFamiglia)  
    REFERENCES Impiegati )
```

Figura 41: Esempio Definizione Tabella

5.2 Tabelle inizializzate e tabelle calcolate

Tabelle inizializzate (*Statica*):

```
CREATE TABLE Nome EspressioneSELECT  
CREATE TABLE Supervisori  
    SELECT Codice, Nome, Qualifica, Stipendio  
    FROM Impiegati  
    WHERE Supervisore IS NULL
```

Tabelle calcolate (viste) (*Dinamica*):

```
CREATE VIEW Nome [(Attributo {, Attributo})]  
    AS EspressioneSELECT [WITH CHECK OPTION];  
CREATE VIEW Supervisori  
AS SELECT Codice, Nome, Qual., Stip.  
    FROM Impiegati  
    WHERE Supervisore IS NULL
```

5.3 Viste modificabili

Le tabelle delle viste si interrogano come le altre, ma in generale non si possono modificare. Deve esistere una corrispondenza biunivoca fra le righe della vista e le righe di una tabella di base, ovvero:

- 1 SELECT senza DISTINCT e solo di attributi
- 2 FROM una sola tabella modificabile
- 3 WHERE senza SottoSelect
- 4 GROUP BY e HAVING non sono presenti nella definizione.

Possono esistere anche delle restrizioni su SELECT su viste definite usando GROUP BY.

5.3.1 Utilità delle viste

Le viste servono a nascondere certe modifiche all'organizzazione logica dei dati (indipendenza logica), offrire visioni diverse degli stessi dati senza ricorrere aduplicazioni e a rendere più semplici, o possibili, alcune interrogazioni.

5.3.2 Esempi di viste e interrogazioni impossibili

- 'Il dipartimento che spende il massimo per gli stipendi':

```
CREATE VIEW SpeseStipendi (Dipartimento, Spesa)
AS SELECT Dipartimento, sum(Stipendio)
FROM Impiegati GROUP BY Dipartimento
SELECT Dipartimento, Spesa
FROM SpeseStipendi
WHERE Spesa = ( SELECT max(Spesa) FROM SpeseStipendi)
```

- equivalente a

```
SELECT Dipartimento, sum(Stipendio)
FROM Impiegati
GROUP BY Dipartimento
HAVING sum(Stipendio) >= all (SELECT sum(Stipendio)
FROM Impiegati
GROUP BY Dipartimento )
```

spesso non ammessa perché HAVING coinvolge una sottoselect.

Figura 42: Esempio Viste E Interrogazioni Impossibili

- "Trovare il numero medio di impiegati dei dipartimenti"

```
CREATE VIEW NumImpegatiDip(Dipart., NumImp)
AS      SELECT          Dipartimento, count(*)
        FROM            Impiegati
        GROUP BY        Dipartimenti
SELECT  avg(NumImp)
FROM    NumImpegatiDip
```

- equivale a

```
SELECT  avg(count(*))
FROM    Impiegati
GROUP BY Dipartimento
```

che non si può scrivere senza view

Figura 43: Esempio Viste E Interrogazioni Impossibili

5.4 Vincoli d'integrità : chiavi e generali

Vincoli su attributi:

- VincoloAttributo :=
[NOT NULL [UNIQUE]] | [CHECK (Condizione)]
[REFERENCES Tabella [(Attributo {, Attributo})]]

Vincoli su tabella:

- VincoloTabella := UNIQUE (Attributo , Attributo)
| CHECK (Condizione) |
| PRIMARY KEY [Nome] (Attributo {, Attributo})
| FOREIGN KEY [Nome] (Attributo {, Attributo})
REFERENCES Tabella [(Attributo {, Attributo})]
[ON DELETE {NO ACTION| CASCADE | SET NULL}]

- **CREATE TABLE Impiegati**
(Codice CHAR(8) NOT NULL,
 Nome CHAR(20) NOT NULL,
 AnnoNascita INTEGER NOT NULL,
 Dipartimento CHAR(20),
 Stipendio FLOAT NOT NULL,
 Supervisore CHAR(8),
 PRIMARY KEY pk_impiagato (Codice),
 FOREIGN KEY fk_ Impiegati (Supervisore)
 REFERENCES Impiegati
 ON DELETE SET NULL
)

Figura 44: Esempio Vincoli Integrità

```
CREATE TABLE FamiliariACarico
( Nome CHAR(20) NOT NULL,
  AnnoNascita INTEGER NOT NULL,
  GradoParentela CHAR(10) NOT NULL,
  CapoFamiglia CHAR(8) NOT NULL,
  PRIMARY KEY pk_FamiliariACarico (CapoFamiglia, Nome)
  FOREIGN KEY fk_FamiliariACarico (CapoFamiglia)
  REFERENCES Impiegati
  ON DELETE CASCADE )
```

Figura 45: Esempio Vincoli Integrità

5.5 Create procedure/function

```
CREATE FUNCTION contaStudenti IS
    DECLARE
        tot INTEGER;
    BEGIN
        SELECT COUNT(*) INTO tot FROM STUDENTI;
        RETURN (tot);
    END
```

5.6 I trigger

I trigger si basano sul paradigma evento-condizione-azione (ECA):

```
CREATE TRIGGER Nome
    PrimaODopoDi Evento {, Evento}
    ON Tabella [WHEN Condizione]
    [Granularità]
    Azione

PrimaODopoDi := BEFORE | AFTER
Evento := INSERT | DELETE | UPDATE OF Attributi
Granularità := FOR EACH ROW | FOR EACH STATEMENT
```

Una proprietà essenziale dei trigger è la *terminazione* e il saper trattare vincoli non esprimibili nello schema, attivando automaticamente azioni sulla base di dati quando si verificano certe condizioni.

```
CREATE TRIGGER ControlloStipendio
    BEFORE INSERT ON Impiegati
    DECLARE
        StipendioMedio FLOAT
    BEGIN
        SELECT avg(Stipendio) INTO StipendioMedio
        FROM Impiegati
        WHERE Dipartimento = :new.Dipartimento;
        IF :new.Stipendio > 2 * StipendioMedio
        THEN RAISE_APPL_ERR(-2061, 'Stipendio alto')
        END IF;
    END;
```

Figura 46: Esempio Di Trigger

5.7 Controllo degli accessi

Chi crea lo schema della BD è l'unico che può fare **CREATE**, **ALTER** e **DROP** e che può stabilire i modi in cui altri possono farne uso:

GRANT Privilegi ON Oggetto TO Utenti [WITH GRANT OPTION]

Tipi di privilegi:

- **SELECT**: lettura di dati
- **INSERT** [(Attributi)]: inserire record (con valori non nulli per gli attributi)
- **DELETE**: cancellazione di record
- **UPDATE** [(Attributi)]: modificare record (o solo gli attributi)
- **REFERENCES** [(Attributi)]: definire chiavi esterne in altre tabelle che riferiscono gli attributi.
- **WITH GRANT OPTION**: si possono trasferire i privilegi ad altri utenti.

Chi definisce una tabella o una VIEW ottiene automaticamente tutti i privilegi su di esse, ed è l'unico che può fare un DROP e può autorizzare altri ad usarla con GRANT. Nel caso di viste, il "creatore" ha i privilegi che ha sulle tabelle usate nella definizione. Quando si toglie un privilegio a U, lo si toglie anche a tutti coloro che lo hanno avuto solo da U.

Le autorizzazioni si annullano con il comando:

• REVOKE [GRANT OPTION FOR] Privilegi ON Oggetto FROM Utenti [CASCADE]

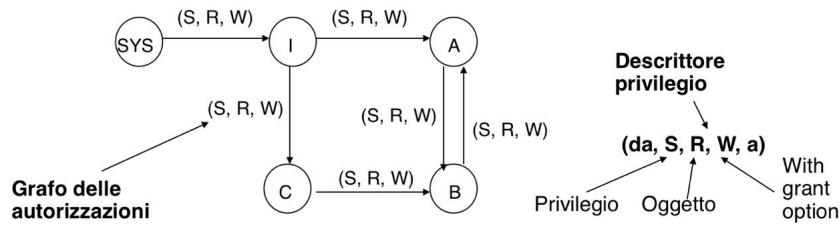
5.7.1 Esempi di GRANT

- **GRANT INSERT, SELECT ON Esami TO Tizio .**
- **GRANT DELETE ON On Esami TO Capo WITH GRANT OPTION**
Capo può cancellare record e autorizzare altri a farlo.
- **GRANT UPDATE (voto) ON Esami TO Sicuro**
Sicuro può modificare solo il voto degli esami.
- **GRANT SELECT, INSERT ON VistaEsamiBD1 TO Albano**
Albano può interrogare e modificare solo i suoi esami.

5.8 Grafo delle autorizzazioni

L'utente I ha creato la tabella R e innesca la seguente successione di eventi:

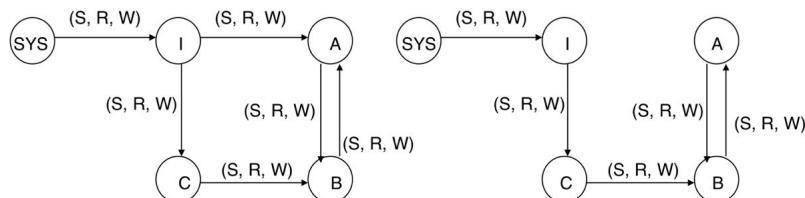
- I: GRANT SELECT ON R TO A WITH GRANT OPTION
- A: GRANT SELECT ON R TO B WITH GRANT OPTION
- B: GRANT SELECT ON R TO A WITH GRANT OPTION
- I: GRANT SELECT ON R TO C WITH GRANT OPTION
- C: GRANT SELECT ON R TO B WITH GRANT OPTION



Se un nodo N ha un arco uscente con un privilegio, allora esiste un cammino da SYSTEM a N con ogni arco etichettato dallo stesso privilegio + WGO.

Effetto del **REVOKE**, ad es:

- I: **REVOKE** SELECT ON R FROM A CASCADE
e poi I: **REVOKE** SELECT ON R FROM C CASCADE



5.9 Creazione di indici

Non è un comando standard dell'SQL e quindi ci sono differenze nei vari sistemi

- CREATE INDEX NomeIdx ON Tabella(Attributi)
- CREATE INDEX NomeIdx ON Tabella
WITH STRUCTURE = BTREE, KEY = (Attributi)
- DROP INDEX NomeIdx

5.10 Catalogo (dei metadati)

Alcuni esempi di tabelle, delle quali si mostrano solo alcuni attributi, sono:

- Tabella delle password:
PASSWORD(username, password)
- Tabella delle basi di dati:
SYSDB(dbname, creator, dbpath, remarks)
- Tabella delle tabelle (type = view or table):
SYSTABLES(name, creator, type, colcount, filename, remarks)

Alcuni esempi di tabelle, delle quali si mostrano solo alcuni attributi, sono:

- Tabella degli attributi:
SYSCOLUMNS(name, tbname, tbcreator, colno, coltype, lenght, default, remarks)
- Tabella degli indici:
SYSINDEXES(name, tbname, creator, uniquerule, colcount)
e altre ancora sulle viste, vincoli, autorizzazioni, etc. (una decina).

5.11 Riepilogo

- DDL consente la definizione di tabelle, viste e indici. Le tabelle si possono modificare aggiungendo o togliendo attributi e vincoli.
- Le viste si possono interrogare come ogni altra tabella, ma in generale non consentono modifiche dei dati.
- I comandi GRANT / REVOKE + viste offrono ampie possibilità di controllo degli usi dei dati.
- SQL consente di dichiarare molti tipi di vincoli, oltre a quelli fondamentali di chiave e referenziale.
- Oltre alle tabelle fanno parte dello schema le procedure e i trigger.
- La padronanza di tutti questi meccanismi — e di altri che riguardano aspetti fisici, affidabilità, sicurezza — richiede una professionalità specifica (DBA)

6 Uso di SQL da programmi

Ci sono 3 tipi di approcci :

- **Linguaggio integrato (dati e DML)** : Linguaggio disegnato ad-hoc per usare SQL. I comandi SQL sono controllati staticamente dal traduttore ed eseguiti dal DBMS.
- **Linguaggio convenzionale + API** : Linguaggio convenzionale che usa delle funzioni di una libreria predefinita per usare SQL. I comandi SQL sono stringhe passate come parametri alle funzioni che poi vengono controllate dinamicamente dal DBMS prima di eseguirle.
- **Linguaggio che ospita l'SQL** : Linguaggio convenzionale esteso con un nuovo costrutto per marcare i comandi SQL. Occorre un pre-compilatore che controlla i comandi SQL, li sostituisce con chiamate a funzioni predefinite e genera un programma nel linguaggio convenzionale + API.

6.1 Linguaggio integrato : PL/SQL di Oracle

Un linguaggio a blocchi con una struttura del controllo completa che contiene l'SQL come sottolinguaggio che serve per manipolare basi di dati che integra DML (SQL) con il linguaggio ospite.

Permette :

- Di definire variabili di tipo scalare, record (annidato), insieme di scalari, insiemi di record piatti, cursore.
- Di definire i tipi delle variabili a partire da quelli della base di dati.
- Di eseguire interrogazioni SQL ed esplorarne il risultato.
- Di modificare la base di dati.
- Di definire procedure e moduli.
- Di gestire il flusso del controllo, le transazioni, le eccezioni.

```
CREATE
PROCEDURE Esempio1(
    p_Mat IN Studenti.Matricola%TYPE) IS
DECLARE
    -- Identificatori per lo scambio dati
    XNome CHAR;
    XProvincia Studenti.Provincia%TYPE;
    XAttributi Studenti%ROWTYPE;
    prv_manca EXCEPTION;
BEGIN
    -- ricerca di ennupla : stampa Nome e Provincia
    SELECT Nome, Provincia INTO XNome, XProvincia
    FROM Studenti WHERE Matricola = p_Mat;
    IF XProvincia IS NULL THEN
        RAISE prv_manca
    ELSE PRINT ....
    END IF
EXCEPTION
    WHEN prv_manca THEN <gestione eccezione>
END;
```

Figura 47: Procedura in PLSQL

6.1.1 Cursore

E' il meccanismo per ottenere uno alla volta gli elementi di una relazione e viene definito con un'espressione SQL, poi si apre per far calcolare al DBMS il risultato e con un opportuno comando si trasferiscono i campi delle ennuple in opportune variabili del programma.

```
PROCEDURE Esempio2 (Prov IN Studenti.Provincia%TYPE) IS
DECLARE
    CURSOR c IS
        SELECT Nome, AnnoNascita
        FROM Studenti WHERE Provincia = Prov;
    Stud_Rec c%ROWTYPE;
BEGIN
    -- ricerca di insieme di ennuple : stampa Nome e
    -- AnnoNascita degli studenti di Pisa
    OPEN c
    LOOP
        FETCH c INTO Stud_Rec;
        EXIT WHEN c%NOTFOUND;
        PRINT ... Stud_Rec.Nome ... Stud_Rec.Provincia
    END LOOP;
    CLOSE c -- rilascio del cursore
END
```

Figura 48: Esempio Uso Cursore + FETCH

```
PROCEDURE Esempio3 (Prov IN Studenti.Provincia%TYPE) IS
BEGIN
    -- ricerca di insieme di ennuple: stampa Nome e
    -- AnnoNascita degli studenti di Pisa  */

    FOR Stud_Rec IN
        (SELECT Nome, AnnoNascita
        FROM Studenti WHERE Provincia = Prov)
    LOOP
        PRINT ... Stud_Rec.Nome ... Stud_Rec.Provincia
    END LOOP; -- rilascio del cursore

END
```

Figura 49: Esempio Uso Cursore Implicito + FOR

6.2 Linguaggio con interfaccia API

Invece di modificare il compilatore di un linguaggio, si usa una libreria di funzioni/oggetti che operano su basi di dati (API) alle quali si passa come parametro stringhe SQL e ritornano il risultato sul quale si opera con una logica ad iteratori.

Dovrebbero essere indipendenti dal DBMS : un “driver” gestisce le richieste e le traduce in un codice specifico di un DBMS

```
class StampaNomiStudenti{
    public static void main(String argv[]){
        Class.forName("driver per DBMS");
        Connection con = // connect
            DriverManager.getConnection("url", "login", "pass");
        Statement stmt = con.createStatement(); // crea un oggetto per comando SQL
        String query = "SELECT Nome
                        FROM Studenti WHERE Provincia ='" + argv[0] + "' ";
        ResultSet iter = stmt.executeQuery(query);
        System.out.println("Nomi trovati:");
        try { // gestore eccezioni
            // ciclo sul risultato
            while (iter.next()) {
                String nome = iter.getString("Nome");
                int anno = iter.getInt("AnnoNascita");
                System.out.println(" Nome: " + nome + "; AnnoNascita: " + anno);
            }
        } catch(SQLException ex) {
            System.out.println(ex.getMessage()+ex.getSQLState()+ex.getErrorCode());
        }
        stmt.close(); con.close();
    }
}
```

Figura 50: Esempio SQL API In Java (JDBC)

```
char SQLSTATE[6];
EXEC SQL BEGIN DECLARE SECTION
char c_sname[20]; short c_annoNascita;
EXEC SQL END DECLARE SECTION
short c_Provincia = "Pisa";
EXEC SQL DECLARE sinfo CURSOR FOR
    SELECT S.nome, S.annoNascita
    FROM Studenti S
    WHERE S.Provincia = :c_Provincia
    ORDER BY S.nome;
do {
    EXEC SQL FETCH sinfo INTO :c_sname, :c_annoNascita;
    printf("Nome:%s; AnnoNascita: %s \n ", c_sname,
    c_annoNascita);
} while (SQLSTATE != '02000');
EXEC SQL CLOSE sinfo;
```

Figura 51: Esempio Linguaggio Che Ospita SQL

6.3 SQLJ : Java che ospita SQL

Dialecto di SQL che può essere immerso in programmi Java, che poi vengono tradotti da un precompilatore in programmi Java standard sostituendo i comandi SQL in chiamate di una libreria che usano JDBC.

```
public static void main(String argv[]){
    Oracle.connect("jdbc:oracle:oci8:@", "scott", "tiger");
    #SQL iterator GetNomeIter(String Nome, int AnnoNascita);
    GetNomeIter iter;
    #SQL iter = {SELECT Nome, AnnoNascita AS Anno
                 from Studenti where Provincia = :(argv[0])};
    System.out.println("Nomi trovati:");
    while (iter.next()) {
        String nome = iter.Nome();
        int anno = iter.Anno();
        System.out.println(" Nome = " + nome + \n);
    }
}
iter.close();
Oracle.close(); }
```

Figura 52: Esempio SQLJ

7 Teoria relazionale

La teoria della progettazione relazionale studia cosa sono le “anomalie” e come eliminarle. Ci sono 2 metodi per produrre uno schema relazionale :

- (A). Partire da un buon schema a oggetti e tradurlo.
- (B). Partire da uno schema relazionale fatto da altri e modificarlo o completarlo.

È particolarmente utile se si usa il metodo (B) ed è moderatamente utile quando si usa il metodo (A).

- **Esempio:**
 - *StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)*
- **Anomalie:**
 - Ridondanze
 - Potenziali inconsistenze
 - Anomalie nelle inserzioni
 - Anomalie nelle eliminazioni
- **Schema senza anomalie**
 - *Studenti (Matricola, Nome, Provincia, AnnoNascita)*
 - *Esami (Materia, Matricola, Voto)*

Figura 53: Esempio Schemi Con Anomalia

Obiettivi della teoria :

- Equivalenza di schemi
- Qualità degli schemi (forme normali)
- Trasformazione degli schemi (normalizzazione di schemi)

Tutti i fatti sono descritti da attributi di un'unica relazione (relazione universale), cioè gli attributi hanno un significato globale.

7.1 Dipendenze funzionali

Per formalizzare la nozione di schema senza anomalie, occorre una descrizione formale della semantica dei fatti rappresentati in uno schema relazionale.

Istanza valida di R : è una nozione semantica, che dipende da ciò che sappiamo del dominio del discorso.

Dato uno schema $R(T)$ e $X, Y \subseteq T$, una dipendenza funzionale (DF) è un vincolo su R del tipo $X \rightarrow Y$, i.e. X determina funzionalmente Y o Y è determinato da X , se per ogni istanza valida di R un valore di X determina in modo univoco un valore di Y :

$\forall r$ istanza valida di R .

$\forall t1, t2 \in r. \text{ se } t1[X] = t2[X] \text{ allora } t1[Y] = t2[Y]$

Si dice che un'istanza $r0$ di R soddisfa le DF $X \rightarrow Y$ ($r0 \models X \rightarrow Y$) se la proprietà vale per $r0$, e che un'istanza $r0$ di R soddisfa un insieme F di DF se, per ogni $X \rightarrow Y \in F$, vale $r0 \models X \rightarrow Y$:

$r0 \models X \rightarrow Y$ sse $\forall t1, t2 \in r0. \text{ se } t1[X] = t2[X] \text{ allora } t1[Y] = t2[Y]$

- DotazioniLibri(*CodiceLibro*, *NomeNegozio*, *IndNegozio*, *Titolo*, *Quantità*)
- DF:
 - { *CodiceLibro* → *Titolo* }
 - NomeNegozio* → *IndNegozio*
 - CodiceLibro*, *NomeNegozio* → *IndNegozio*, *Titolo*, *Quantità* }

Figura 54: Esempio Dipendenze Funzionali

7.1.1 Esprimere le dipendenze funzionali

Consideriamo : $\text{NomeNegozio} \rightarrow \text{IndNegozio}$

- Espressione diretta :

– Se in due righe il *NomeNegozio* è uguale, anche l' *IndNegozio* è uguale:
 $\text{NomeNegozio}_\equiv \rightarrow \text{IndNegozio}_\equiv$

- Per contrapposizione :

– Se l' *IndNegozio* è diverso allora il *NomeNegozio* è diverso:
 $\text{IndNegozio}_\neq \rightarrow \text{NomeNegozio}_\neq$

- Per assurdo :

– Non possono esserci due dotazioni con *NomeNegozio* uguale e *IndNegozio* diverso:
 $\text{Not}(\text{NomeNegozio}_\equiv \wedge \text{IndNegozio}_\neq)$
 $\text{NomeNegozio}_\equiv \wedge \text{IndNegozio}_\neq \rightarrow \text{False}$

7.1.2 Manipolazione di clausole

Sono equivalenti :

- $\text{NomeNegozi}_\equiv \rightarrow \text{IndNegozi}_\equiv$
- $\text{IndNegozi}_\neq \rightarrow \text{NomeNegozi}_\neq$
- $\text{NomeNegozi}_\equiv \wedge \text{IndNegozi}_\neq \rightarrow \text{False}$

In generale :

- $A \rightarrow B \iff A \wedge \neg B \rightarrow \text{False} \iff \neg B \rightarrow \neg A$

Più in generale, in ogni clausola $A \wedge B \rightarrow E \vee F$ posso spostare le sottoformule da un lato all'altro, negandole.

Quindi sono equivalenti:

- $\text{NomeNegozi}_\equiv \wedge \text{CodiceLibro}_\equiv \rightarrow \text{Quantita}_\equiv$
- $\text{NomeNegozi}_\equiv \wedge \text{CodiceLibro}_\equiv \wedge \text{Quantita}_\neq \rightarrow \text{False}$
- $\text{CodiceLibro}_\equiv \wedge \text{Quantita}_\neq \rightarrow \text{NomeNegozi}_\neq$
- $\text{NomeNegozi}_\equiv \wedge \text{Quantita}_\neq \rightarrow \text{CodiceLibro}_\neq$

Esempio:

$\text{Orari}(\text{CodAula}, \text{NomeAula}, \text{Piano}, \text{Posti}, \text{Materia}, \text{CDL}, \text{Docente}, \text{Giorno}, \text{OraInizio}, \text{OraFine})$

- In un dato momento, un docente si trova al più in un'aula :

$\text{Docente}, \text{Giorno}, \text{OraInizio} \rightarrow \text{CodAula}$
 $\text{Docente}, \text{Giorno}, \text{OraFine} \rightarrow \text{CodAula}$

- Non è possibile che due docenti diversi siano nella stessa aula contemporaneamente :

$\text{Giorno}, \text{OraFine}, \text{Aula} \rightarrow \text{Docente}$

- Se due lezioni si svolgono su due piani diversi appartengono a due corsi di laurea diversi :

$\text{CodiceAula} \rightarrow \text{Piano}$

7.1.3 Proprietà delle DF

$R <T, F>$ denota uno schema con attributi T e dipendenze funzionali F .

Le DF sono una proprietà semantica, cioè dipendono dai fatti rappresentati e non da come gli attributi sono combinati in schemi di relazione.

Si parla di DF complete quando $X \rightarrow Y$ e per ogni $W \subset X$, non vale $W \rightarrow Y$.

Se X è una superchiave, allora X determina ogni altro attributo della relazione: $X \rightarrow T$

Se X è una chiave, allora $X \rightarrow T$ è una DF completa

Da un insieme F di DF, in generale altre DF sono ‘implicate’ da F .

Definizione : *Sia F un insieme di DF sullo schema R , diremo che F implica logicamente $X \rightarrow Y$ ($F \models X \rightarrow Y$), se ogni istanza r di R che soddisfa F soddisfa anche $X \rightarrow Y$.*

7.2 Regole di inferenza

Come derivare DF implicate logicamente da F, usando un insieme di regole di inferenza.

"Assiomi" (sono in realtà regole di inferenza) di Armstrong :

- Se $y \subseteq x$, allora $x \rightarrow y$ (*Riflessività R*)
- Se $x \rightarrow y, z \subseteq t$, allora $xz \rightarrow yz$ (*Arricchimento A*)
- $x \rightarrow y, y \rightarrow z$, allora $x \rightarrow z$ (*Transitività T*)

7.2.1 Correttezza e completezza degli assiomi di Armstrong

Teorema : Gli assiomi di Armstrong sono corretti e completi.

Correttezza degli assiomi : $\forall f, F \vdash f \rightarrow F \models f$

Completezza degli assiomi : $\forall f, F \models f \rightarrow F \vdash f$

7.3 Derivazione

Sia F un insieme di DF, diremo che $X \rightarrow Y$ si derivabile da F ($F \vdash X \rightarrow Y$), sse $X \rightarrow Y$ può essere inferito da F usando gli assiomi di Armstrong.

Si dimostra che valgono anche le regole :

- $\{X \rightarrow Y, X \rightarrow Z\} \vdash X \rightarrow YZ$ (*Unione U*)
- $Z \subseteq Y\{X \rightarrow Y\} \vdash X \rightarrow Z$ (*Decomposizione D*)
- $R(A B C D)$
- $F = \{A \rightarrow B, BC \rightarrow D\}$
- AC è una superchiave? Ovvero $AC \rightarrow ABCD$?
 1. $A \rightarrow B$ ipotesi 1
 2. $AC \rightarrow BC$ da 1 per **Arr (C)**
 3. $BC \rightarrow D$ ipotesi 2
 4. $BC \rightarrow BCD$ da 3 per **Arr (BC)**
 5. $AC \rightarrow BCD$ da 2+4 per **Trans**
 6. $AC \rightarrow ABCD$ da 5 per **Arr (A)**

Figura 55: Esempio Derivazione

7.4 Chiusura di un insieme F

Definizione : Dato un insieme F di DF, la chiusura di F, denotata come F^+ , è:
 $F^+ = \{ X \rightarrow Y \mid F \vdash X \rightarrow Y \}$

Definizione : Dato R <T, F> e $X \subseteq T$, la *chiusura* di X rispetto ad F, denotata con X_F^+ , (o X^+ , se F è chiaro dal contesto) è : $X_F^+ = \{ A_i \in T \mid F \vdash X \rightarrow A_i \}$

Problema dell'implicazione : controllare se una DF $V \rightarrow W \in F^+$

Un algoritmo efficiente per risolvere il problema dell'implicazione senza calcolare la chiusura di F scaturisce dal seguente teorema.

Teorema : $F \vdash X \rightarrow Y \iff Y \subseteq X_F^+$

7.4.1 Chiusura lenta

Un semplice algoritmo per calcolare X^+ (ne esiste uno migliore di complessità di tempo O(ap)) è :

```

• Algoritmo CHIUSURA LENTA
  input      R<T, F>  $X \subseteq T$ 
  output      $X^+$ 
  begin
     $X^+ = X$ 
    while ( $X^+$  cambia) do
      for  $W \rightarrow V$  in F with  $W \subseteq X^+$  and  $V \not\subseteq X^+$ 
        do  $X^+ = X^+ \cup V$ 
    end
  
```

Figura 56: Algoritmo Chiusura

7.5 Chiavi e attributi primi

Definizione : Dato lo schema R<T, F>, diremo che $W \subseteq T$ è una chiave candidata di R se :

$$\begin{aligned} W \rightarrow T &\in F^+ && (W \text{ superchiave}) \\ \forall V \subset W, V \rightarrow T &\notin F^+ && (\text{se } V \subset W, V \text{ non superchiave}) \end{aligned}$$

Attributo primo : attributo che appartiene ad almeno una chiave.

Il problema di trovare tutte le chiavi di una relazione richiede un algoritmo di complessità esponenziale nel caso peggiore e controllare se un attributo è primo è NPcompleto

7.6 Copertura di insiemi di DF

Definizione : Due insiemi di DF, F e G, sullo schema R sono equivalenti, $F \equiv G$, sse $F^+ = G^+$. Se $F \equiv G$, allora F è una copertura di G (e G una copertura di F).

Definizione : Sia F un insieme di DF :

1 Data una $X \rightarrow Y \in F$, si dice che X contiene un attributo *estraneo* A_i sse $(X - \{A_i\}) \rightarrow Y \in F^+$, cioè $F \vdash (X - A_i) \rightarrow Y$

2 $X \rightarrow Y$ è una dipendenza *ridondante* sse

$$(F - \{X \rightarrow Y\})^+ = F^+, \text{ cioè } F - \{X \rightarrow Y\} \vdash X \rightarrow Y$$

F è detta una copertura *canonica* sse :

- a parte destra di ogni DF in F è un attributo;
- non esistono attributi estranei;
- nessuna dipendenza in F è ridondante.

7.6.1 Esistenza della copertura canonica

Teorema : Per ogni insieme di dipendenze F esiste una copertura canonica.

Algoritmo per calcolare una copertura canonica:

- Trasformare le dipendenze nella forma $X \rightarrow A$
- Eliminare gli attributi estranei.
- Eliminare le dipendenze ridondanti.

7.7 Decomposizione di schemi

In generale, per eliminare anomalie da uno schema occorre decomporlo in schemi più piccoli "equivalenti"

Definizione : Dato uno schema $R(T)$, $\rho = \{ R_1(T_1), \dots, R_k(T_k) \}$ è una decomposizione di R sse $\cup T_i = T$:

- $\{\text{Studenti}(\text{Matr}, \text{Nome}), \text{Esami}(\text{Matr}, \text{Materia})\}$
decomp. di $\text{Esami}(\text{Matr}, \text{Nome}, \text{Materia})$
- $\{\text{Studenti}(\text{Matr}, \text{Nome}), \text{Esami}(\text{Materia})\}$
- $\{\text{Studenti}(\text{Matr}, \text{Nome}), \text{Esami}(\text{Nome}, \text{Materia})\}$

La decomposizione ha anche due proprietà : **conservazione dei dati** (nozione semantica) e la **conservazione delle dipendenze**

Decomposizione che preservano i dati :

Definizione : $\rho = \{ R_1(T_1), \dots, R_k(T_k) \}$ è una decomposizione di $R(T)$ che preserva i dati sse per ogni *istanza valida* r di R :

$$r = (\pi_{T1}r) \bowtie (\pi_{T2}r) \bowtie \dots \bowtie (\pi_{Tk}r)$$

Dalla definizione di giunzione naturale scaturisce il seguente risultato :

Teorema : se $\rho = \{ R_1(T_1), \dots, R_k(T_k) \}$ è una decomposizione di $R(T)$, allora per ogni istanza r di R :

$$r \subseteq (\pi_{T1}r) \bowtie (\pi_{T2}r) \bowtie \dots \bowtie (\pi_{Tk}r)$$

7.7.1 Decomposizioni binarie

Teorema : Sia $R < T, F >$ uno schema di relazione, la decomposizione $\rho = \{R_1(T_1), R_2(T_2)\}$ preserva i dati sse $T_1 \cap T_2 \rightarrow T_1 \in F^+$
oppure $T_1 \cap T_2 \rightarrow T_2 \in F^+$

7.8 Proiezione delle dipendenze

Definizione : Dato lo schema $R < T, F >$, e $T_1 \subseteq T$, la proiezione di F su T_1 è $\pi_{T_1}(F) = \{X \rightarrow Y \in F^+ | XY \subseteq T_1\}$

Esempio :

Sia $R(A, B, C)$ e $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

$\pi_{AB}(F) \equiv \{A \rightarrow B, B \rightarrow A\}$

$\pi_{AC}(F) \equiv \{A \rightarrow C, C \rightarrow A\}$

Algoritmo banale per il calcolo di $\pi_{T_1}(F)$:

for each $Y \subseteq T_1$ **do** ($Z := Y^+; output Y \rightarrow Z \cap T_1$)

7.9 Preservazione delle dipendenze

Definizione : Dato lo schema $R < T, F >$, la decomposizione $\rho = \{R_1, \dots, R_n\}$ preserva le dipendenze sse l'unione delle dipendenze in $\pi_{T_i}(F)$ è una copertura di F .

Proposizione : Dato lo schema $R < T, F >$, il problema di stabilire se la decomposizione $\rho = \{R_1, \dots, R_n\}$ preserva le dipendenze ha complessità di tempo polinomiale.

Teorema : Sia $\rho = \{R_1 < T_i, F_i >\}$ una decomposizione di $R < T, F >$ che preservi le dipendenze e tale che un T_j sia una superchiave per R . Allora ρ preserva i dati.

- **Telefoni(Prefisso, Numero, Località, Abbonato, Via)** $\{P \rightarrow N \rightarrow L \rightarrow A \rightarrow V, L \rightarrow P\}$
- Si consideri la decomposizione:
 $\rho = \{\text{Tel}\langle\{N, L, A, V\}, F1\rangle, \text{Pref}\langle\{L, P\}, F2\rangle\}$ con
 - $F1 = \{LN \rightarrow A \rightarrow V\}$
 - $F2 = \{L \rightarrow P\}$
- Preserva dati ma non le dipendenze: $P \rightarrow N \rightarrow L$ non è deducibile da $F1$ e $F2$.

Figura 57: Esempio Preservazione Dipendenze

7.10 Forme normali

1FN : Impone una restrizione sul tipo di una relazione: ogni attributo ha un tipo elementare.

2FN, 3FN, FNBC : Impongono restrizioni sulle dipendenze. FNBC è la più naturale e la più restrittiva.

FNBC : Intuizione: se esiste in R una dipendenza $X \rightarrow A$ non banale ed X non è chiave, allora X modella l'identità di un'entità diversa da quelle modellate dall'intera R

Ad esempio, in StudentiEdEsami, il Nome dipende dalla Matricola che non è chiave.

7.10.1 FNBC

Definizione : $R <T, F>$ è in BCNF \iff per ogni $X \rightarrow A \in F^+, \text{ con } A \notin X$, X è una superchiave.

Teorema : $R <T, F>$ è in BCNF \iff per ogni $X \rightarrow A \in F$, X è una superchiave.

Esempio:

- Docenti(CodiceFiscale, Nome, Dipartimento, Indirizzo)
 - Impiegati(Codice, Qualifica, NomeFiglio)
 - Librerie(CodiceLibro, NomeNegozio, IndNegozio, Titolo, Quantità)
 - Telefoni(Prefisso, Numero, Località, Abbonato, Via)
- $$F = \{P \ N \rightarrow L \ A \ V, L \rightarrow P\}$$

7.11 L'algoritmo di analisi

$R <T, F>$ è decomposta in: $R_1 (X, Y)$ e $R_2 (X, Z)$ e su di esse si ripete il procedimento esponeziale.

```

 $\rho = \{R <T, F>\}$ 
while esiste in  $\rho$  una  $R_i <T_i, F_i>$  non in BCNF per la DF  $X \rightarrow A$ 
  do
     $T_a = X^+$ 
     $F_a = \pi_{T_a}(F_i)$ 
     $T_b = T_i - X^+ + \textcolor{red}{X} \quad \leftarrow \quad \text{attenzione: errore nel vecchio libro}$ 
     $F_b = \pi_{T_b}(F_i)$ 
     $\rho = \rho - R_i + \{R_a <T_a, F_a>, R_b <T_b, F_b>\}$ 
    ( $R_a$  ed  $R_b$  sono nomi nuovi)
  end
```

Figura 58: Algoritmo Di Analisi

Preserva i dati, ma non necessariamente le dipendenze.

7.12 Terza forma normale

Definizione : $R <T, F>$ è in 3FN se per ogni $X \rightarrow A \in F^+$, con $A \notin X$, X è una superchiave o A è primo.

La 3FN ammette una dipendenza non banale e non-da-chiave se gli attributi a destra sono primi; la BCNF non ammette mai nessuna dipendenza non banale e non-da-chiave.

Definizione : $R <T, F>$ è in 3FN se per ogni $X \rightarrow A \in F$ non banale, allora X è una superchiave oppure A è primo.

- Non sono in 3FN (e quindi, neppure in BCNF)
 - Docenti(CodiceFiscale, Nome, Dipartimento, Indirizzo)
 - Impiegati(Codice, Qualifica, NomeFiglio)
- Sono in 3FN, ma non in BCNF:
 - Telefoni(Prefisso, Numero, Località, Abbonato, Via)
 - $F = \{P \ N \rightarrow L \ A \ V, L \rightarrow P\}$
 - $K = \{PN, LN\}$
 - Esami(Matricola, Telefono, Materia, Voto)
 - Matricola Materia \rightarrow Voto
 - Matricola \rightarrow Telefono
 - Telefono \rightarrow Matricola
 - Chiavi: Matricola Materia, Telefono Materia

Figura 59: Esempio Terza Forma Normale

7.12.1 L'algoritmo di sintesi : versione base

Sia $R <T, F>$, con F **copertura canonica** e tutti gli attributi interessati da qualche DF.

- 1 Si partiziona F in gruppi tali che ogni gruppo ha lo stesso determinante.
- 2 Si definisce uno schema di relazione per ogni gruppo, con attributi gli attributi che appaiono nelle DF del gruppo, e chiavi i determinanti.
- 3 Si eliminano schemi contenuti in altri.
- 4 Se la decomposizione non contiene uno schema i cui attributi sono una superchiave di R , si aggiunge lo schema con attributi W , con W una chiave di R .

8 Architettura Dei DBMS

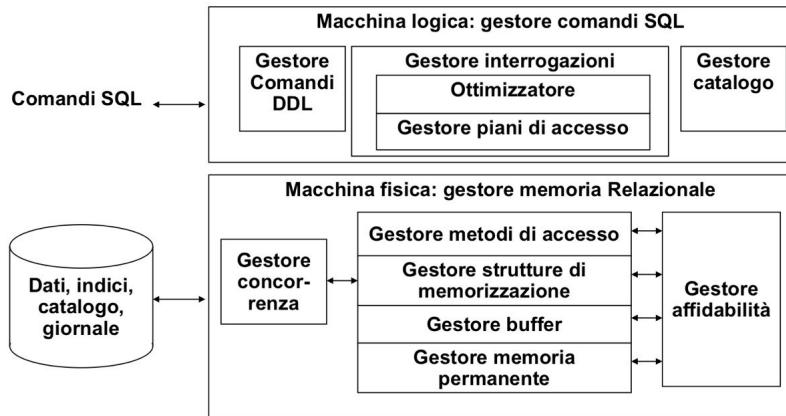


Figura 60: Esempio Struttura DBMS

8.1 Memorie a disco

Un'unità a dischi contiene una pila di dischi metallici che ruota a velocità costante ed alcune testine di lettura che si muovono radialmente al disco.

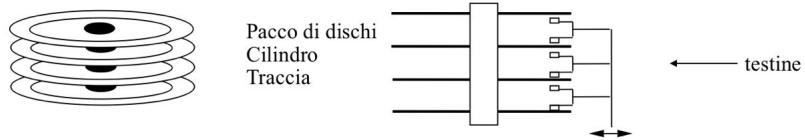


Figura 61: Interno Memoria

Una traccia è organizzata in settori di dimensione fissa; i settori sono raggruppati logicamente in blocchi, che sono l'unità di trasferimento. Trasferire un blocco richiede un tempo di posizionamento delle testine, un tempo di latenza rotazionale e tempo per il trasferimento (trascurabile).

8.2 Gestore memoria permanente e gestore del buffer

Gestore memoria permanente : Fornisce un'astrazione della memoria permanente in termini di insiemi di file logici di pagine fisiche di registrazioni (blocchi), nascondendo le caratteristiche dei dischi e del sistema operativo.

Gestore del buffer : Si preoccupa del trasferimento delle pagine tra la memoria temporanea e la memoria permanente, offrendo agli altri livelli una visione della memoria permanente come un insieme di pagine utilizzabili in memoria temporanea, astraendo da quando esse vengano trasferite dalla memoria permanente al buffer e viceversa

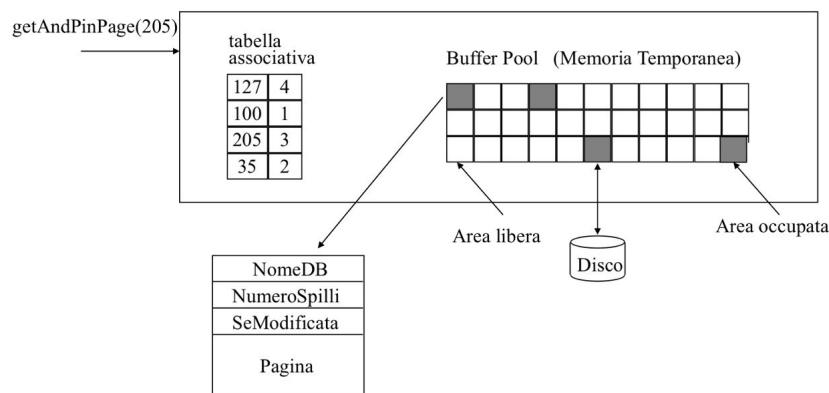


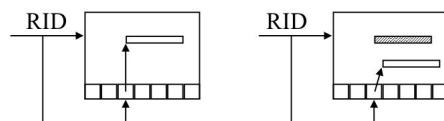
Figura 62: Esempio Area Delle Pagine

8.3 Struttura di una pagina

Struttura fisica : un insieme, di dimensione fissa, di caratteri.

Struttura logica : informazioni di servizio, un'area che contiene le stringhe che rappresentano i record.

Il problema dei riferimenti ai record: coppia (PID della pagina, posizione nella pagina) (RID).



8.4 Gestore strutture di memorizzazione

Tipi di organizzazioni :

- Seriali o Sequenziali
- Per chiave
- Per attributi non chiave

Parametri che caratterizzano un'organizzazione :

- Occupazione di memoria
- Costo delle operazioni di:
 - Ricerca per valore o intervallo
 - Modifica
 - Inserzione
 - Cancellazione

8.4.1 Organizzazioni seriale e sequenziale

Organizzazione seriale (heap file): i dati sono memorizzati in modo disordinato uno dopo l'altro:

- Semplice e a basso costo di memoria
- Poco efficiente
- Va bene per pochi dati
- E' l'organizzazione standard di ogni DBMS

Organizzazione sequenziale : i dati sono ordinati sul valore di uno o più attributi:

- Ricerche più veloci
- Nuove inserzioni fanno perdere l'ordinamento

8.4.2 Organizzazione per chiave

Ci sono 2 tipi di metodi : **procedurale o tabellare**

Metodo procedurale statico :

Parametri di progetto:

- La funzione per la trasformazione della chiave
- Il fattore di caricamento $d=N/(M^*c)$
- La capacità c delle pagine
- Il metodo per la gestione dei trabocchi

Il metodo procedurale è utile per ricerche per chiave ma non per intervallo.

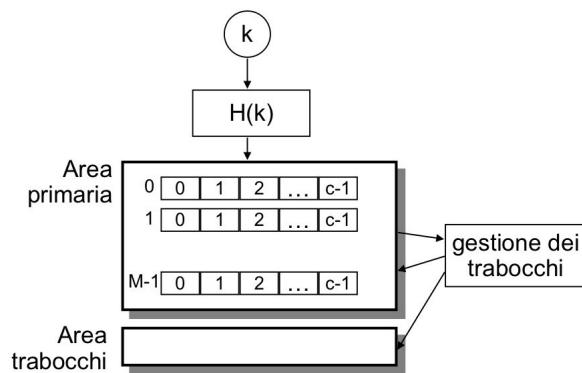


Figura 63: Esempio Tabella Hash

Metodo tabellare :

Utile per ricerche per chiave e per intervallo infatti si usa un indice, ovvero un insieme ordinato di coppie $(k, r(k))$, dove k è un valore della chiave ed $r(k)$ è un riferimento al record con chiave k .

L'indice è gestito di solito con un'opportuna struttura albero detta B^+ -albero, la struttura più usata e ottimizzata dai DBMS.

Gli indici possono essere multi-attributi.

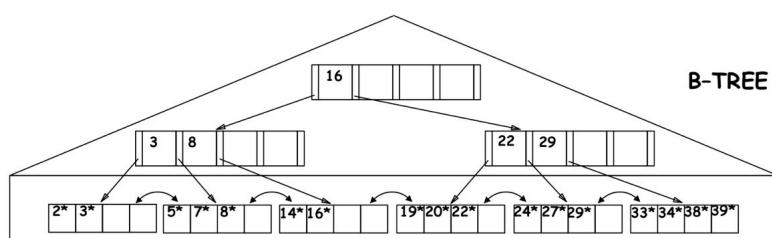


Figura 64: Esempio Albero

- Tabella:

RID	Matr	Prov	An
1	106	MI	1972
2	102	PI	1970
3	107	PI	1971
4	104	FI	1968
5	100	MI	1970
6	103	PI	1972

- Indici

Matr	RID
100	5
102	2
103	6
104	4
106	1
107	3

Indice su Matr

An	RID
1968	4
1970	2
1970	5
1971	3
1972	1
1972	6

Indice su An

Figura 65: Esempio Di Indici Per Attributo Chiave O Non Chiave

8.4.3 Ordinamento di archivi

Si possono ottenere risultati di interrogazioni ordinate (order by), e servono per eseguire alcune operazioni relazionali (join, select distinct, group by).

Algoritmo sort-merge: costo $N^*Log(N)$

8.5 Realizzazione degli operatori relazionali

Si considerano i seguenti operatori :

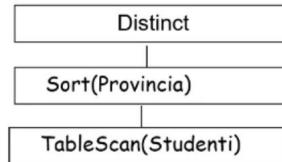
- Proiezione
- Selezione
- Raggruppamento
- Join

8.5.1 Proiezione

Approccio basato sull'ordinamento :

- Si legge R e si scrive T che contiene solo gli attributi della SELECT
- Si ordina T su tutti gli attributi
- Si eliminano i duplicati

```
SELECT DISTINCT Provincia FROM Studenti R
```



8.5.2 Restrizione con condizione semplice

```
SELECT * FROM Studenti R WHERE R.Provincia = 'PI'
```



8.5.3 Operazioni di aggregazione

Senza GROUP BY : Si visitano i dati e si calcolano le funzioni di aggregazione.

Con GROUP BY : Approccio basato sull'ordinamento, cioè Si ordinano i dati sugli attributi del GROUP BY, poi si visitano i dati e si calcolano le funzioni di aggregazione per ogni gruppo.

8.5.4 Giunzione

```
SELECT * FROM Studenti S, Esami E WHERE S.Matricola=E.Matricola
```

Ci sono diversi metodi :

Nested loops : Per ogni record della relazione esterna R, si visita tutta la relazione interna S.

Altri metodi :

- Nested loop con indice:
 - Si usa quando esiste l'indice IS_j sull'attributo di giunzione j della relazione interna S
- Sort-merge:
 - Si usa quando R e S sono ordinate sull'attributo di giunzione: si visitano in R ed S in parallelo

```

IndexNestedLoop
foreach r in R do
  foreach s in get-through-
    index(ISj,r.i)
      aggiungi <r,s> al risultato

SortMerge
r = first(R); s = first(S);
while r in R and s in S do
  if r.i = s.j
    avanza r ed s fino a che r.i ed s.j non
    cambiano entrambe, aggiungendo
    ciascun <r,s> al risultato
  else if r.i < s.j avanza r dentro R
  else if r.i > s.j avanza s dentro S

```

8.6 Operatori logici e fisici

	Operatore logico	Operatore fisico
R		TableScan (R) per la scansione di R;
		IndexScan (R, Idx) per la scansione di R con l'indice Idx;
		SortScan (R, {A_i}) per la scansione di R ordinata sugli {A _i };
$\pi_{\{A_i\}}^b$		Project (O, {A_i}) per la proiezione dei record di O senza l'eliminazione dei duplicati;
	$\pi_{\{A_i\}}$	Distinct (O) per eliminare i duplicati dei record ordinati di O;

Operatore logico	Operatore fisico
σ_{ψ}	<p style="text-align: center;">Filter (O, ψ) per la restrizione senza indici dei record di O;</p>
$\tau_{\{A_i\}}$	<p style="text-align: center;">IndexFilter (R, Idx, ψ) per la restrizione con indice dei record di R;</p>
$\tau_{\{A_i\}}$	<p style="text-align: center;">Sort ($O, \{A_i\}$) per ordinare i record di O sugli $\{A_i\}$, per valori crescenti;</p>

Operatore logico	Operatore fisico
$\{A_i\} \gamma \{f_i\}$	<p style="text-align: center;">GroupBy ($O, \{A_i\}, \{f_i\}$) per raggruppare i record di O sugli $\{A_i\}$ usando le funzioni di aggregazione in $\{f_i\}$.</p> <ul style="list-style-type: none"> • Nell'insieme $\{f_i\}$ vi sono le funzioni di aggregazione presenti nella SELECT e nella HAVING. • L'operatore ritorna record con attributi gli $\{A_i\}$ e le funzioni in $\{f_i\}$. • I record di O sono ordinati sugli $\{A_i\}$;

Operatore logico	Operatore fisico
\bowtie_{ψ_j}	<p style="text-align: center;">NestedLoop (O_E, O_I, ψ_J) per la giunzione con il nested loop e ψ_J la condizione di giunzione;</p>
	<p style="text-align: center;">PageNestedLoop (O_E, O_I, ψ_J) per la giunzione con il page nested loop;</p>
	<p style="text-align: center;">IndexNestedLoop (O_E, O_I, ψ_J) per la giunzione con il index nested loop. L'operando interno O_I è un IndexFilter(R, Idx, ψ_J) oppure Filter (O, ψ'): con O un IndexFilter(R, Idx, ψ_J); per ogni record r di O_E, la condizione ψ_J dell'IndexFilter è quella di giunzione con gli attributi di O_E sostituiti dai valori in r.</p>
	<p style="text-align: center;">SortMerge (O_E, O_I, ψ_J) per la giunzione con il sort-merge, con i record di O_E e O_I ordinati sugli attributi di giunzione.</p>

8.7 Piani di accesso

Un piano di accesso è un algoritmo per eseguire un'interrogazione usando gli operatori fisici disponibili.

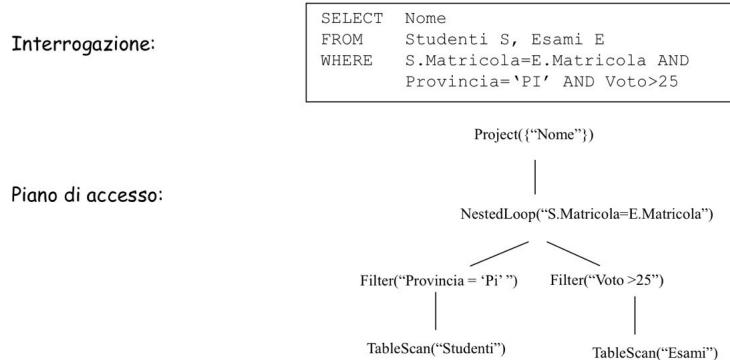


Figura 66: Esempio Piani Di Accessi

8.8 Ottimizzatore delle interrogazioni

L'ottimizzazione delle interrogazione è fondamentale nei DBMS, infatti è necessario conoscere il funzionamento dell'ottimizzatore per una buona progettazione fisica.

Obiettivo dell'ottimizzatore : Scegliere il piano con costo minimo, fra possibili piani alternativi, usando le statistiche presenti nel catalogo.

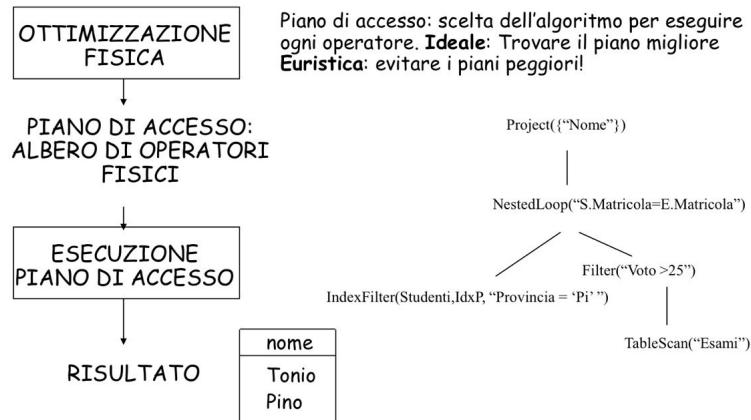


Figura 67: Fasi Del Processo

8.8.1 Esempi di piani di accesso

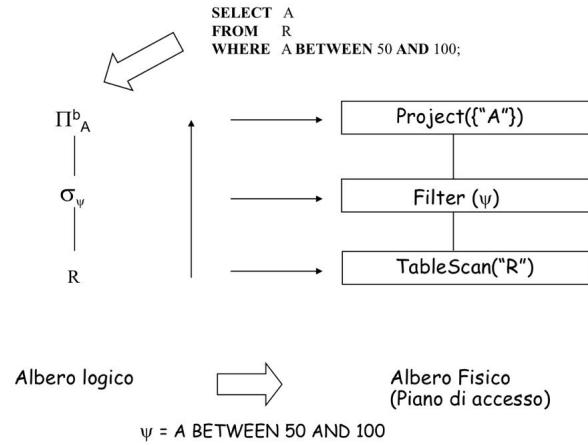


Figura 68: Esempio Di Piano Di Accesso: SFW

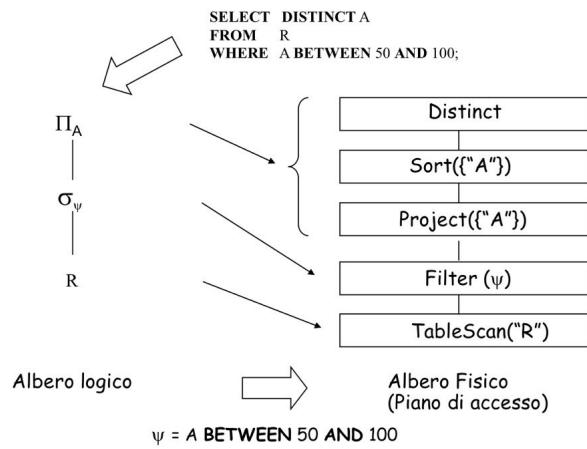


Figura 69: Esempio Di Piano Di Accesso: DISTINCT

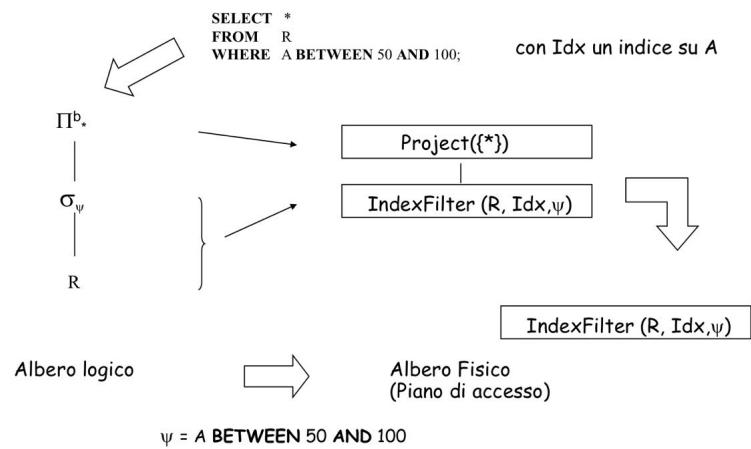


Figura 70: Esempio Di Piano Di Accesso Con Indice 1

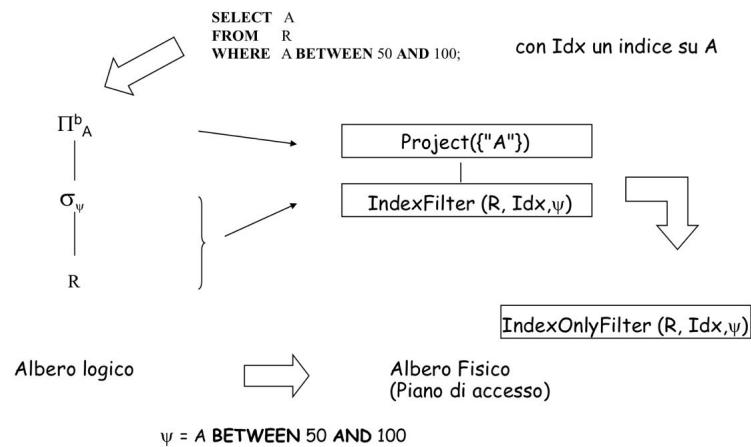


Figura 71: Esempio Di Piano Di Accesso Con Indice 2

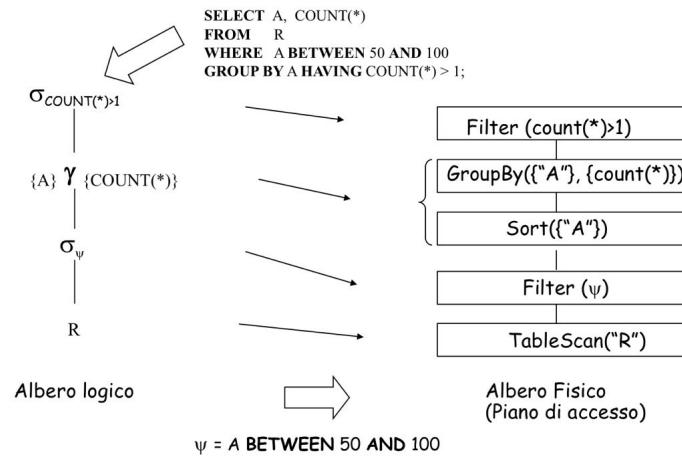


Figura 72: Esempio Di Piano Di Accesso: GROUP BY

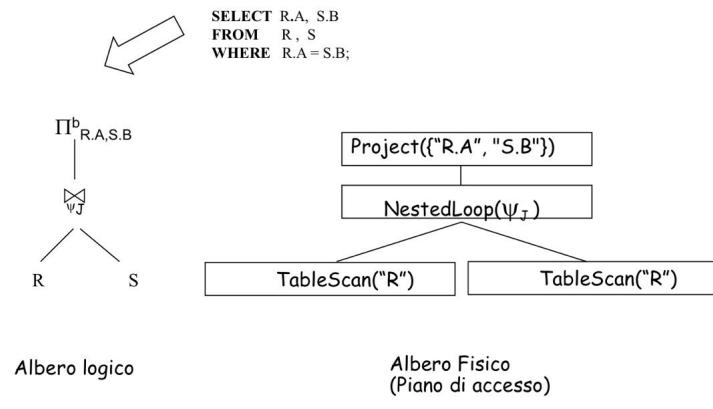


Figura 73: Esempio Di Piano Di Accesso: Giunzione

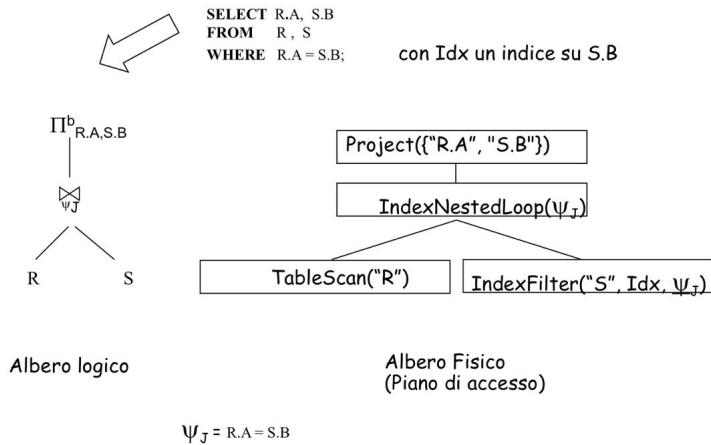


Figura 74: Esempio Di Piano Di Accesso: Giunzione Con Indice

8.9 Gestione delle transizioni

Una funzionalità essenziale di un DBMS è la protezione dei dati da malfunzionamenti e da interferenze dovute all'accesso contemporaneo ai dati da parte di più utenti. Una transazione è un programma sequenziale costituito da operazioni che il sistema deve eseguire garantendo:

- Atomicità
- Serializzabilità
- Persistenza

(Atomicity, Consistency, Isolation, Durability - ACID)

8.9.1 Le transazioni

Una transazione è una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea, con le seguenti proprietà:

- **Atomicità** : Le transazioni che terminano prematuramente (aborted transactions) sono trattate dal sistema come se non fossero mai iniziata; pertanto eventuali loro effetti sulla base di dati sono annullati.
- **Serializzabilità** : Nel caso di esecuzioni concorrenti di più transazioni, l'effetto complessivo è quello di una esecuzione seriale.
- **Persistenza** : Le modifiche sulla base di dati di una transazione terminata normalmente sono permanenti, cioè non sono alterabili da eventuali malfunzionamenti.

8.9.2 La transizione per il DBMS

Una transazione può eseguire molte operazioni sui dati recuperati da una base di dati, ma al DBMS interessano solo quelle di lettura o scrittura della base di dati, indicate con $r_i[x]$ e $w_i[x]$.

- Un'operazione $r_i[x]$ comporta la lettura di una pagina nel buffer, se non già presente.
- Un'operazione $w_i[x]$ comporta l'eventuale lettura nel buffer di una pagina e la sua modifica nel buffer, ma non necessariamente la sua scrittura in memoria permanente. Per questa ragione, in caso di malfunzionamento, si potrebbe perdere l'effetto dell'operazione.

8.9.3 Tipi di malfunzionamento

- **Fallimenti di transizioni** : non comportano la perdita di dati in memoria temporanea né persistente (es.: violazione di vincoli, violazione di protezione, stallo)
- **Fallimento di sistema** : comportano la perdita di dati in memoria temporanea ma non di dati in memoria persistente (es.: comportamento anomalo del sistema, caduta di corrente)
- **Disastri** : comportano la perdita di dati in memoria persistente (es.: danneggiamento di periferica)

8.9.4 Protezione dei dati da malfunzionamento

durante l'uso della BD, il sistema registra nel giornale la storia delle azioni effettuate sulla BD dal momento in cui ne è stata fatta l'ultima copia. Per ogni contenuto di modifica viene registrato :

- a transazione responsabile;
- il tipo di ogni operazione eseguita;
- a nuova e vecchia versione del dato modificato: (T,write, address, oldV, newV);

8.9.5 Checkpoint

Periodicamente si fa un Checkpoint (CKP): si scrive la marca CKP sul giornale per indicare che tutte le operazioni che la precedono sono state effettivamente effettuate sulla BD.

Si scrive sul giornale una marca di inizio checkpoint che riporta l'elenco delle transazioni attive (BeginCkp, T1, ..., Tn)

- In parallelo alle normali operazioni delle transazioni, il gestore del buffer riporta sul disco tutte le pagine modificate
 - Si scrive sul giornale una marca di EndCkp
 - La marca di EndCkp certifica che tutte le scritture avvenute prima del BeginCkp ora sono sul disco. Le scritture avvenute tra BeginCkp e EndCkp forse sono sul disco e forse no.

8.9.6 Ripresa dai malfunzionamenti (disfare-rifare)

Quando si verificano dei fallimenti di transazioni si scrive nel giornale (T, abort) e si applica la procedura disfare.

Mentre se si verifica un fallimento del sistema : La BD viene ripristinata con il comando Restart (ripartenza di emergenza), a partire dallo stato al punto di allineamento, procedendo come segue:

- Le T non terminate vanno disfatte
- Le T terminate devono essere rifatte.

Con i disastri si riporta in linea la copia più recente della BD e la si aggiorna rifacendo le modifiche delle T terminate normalmente (ripartenza a freddo).

8.9.7 Disfare

Quando si portano le modifiche nella BD ?

- Politica della modifica libera : le modifiche possono essere portate nella BD stabile prima che la T termini (disfare o steal).

Regola per poter disfare: prescrittura nel giornale (“Log Ahead Rule” o “Write Ahead Log”):

- Se la nuova versione di una pagina rimpiazza la vecchia sulla BD stabile prima che la T abbia raggiunto il punto di Commit, allora la vecchia versione della pagina deve essere portata prima sul giornale in modo permanente.

8.9.8 Rifare

Come si gestisce la terminazione ?

- Commit libero : una T può essere considerata terminata normalmente prima che tutte le modifiche vengano riportate nella BD stabile (occorre rifare).

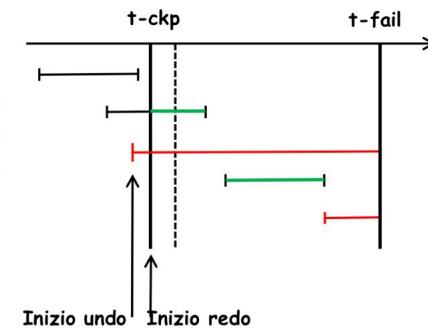
Regola per poter rifare una T: (“Commit Rule”)

- Le modifiche (nuove versioni delle pagine) di una T devono essere portate stabilmente nel giornale prima che la T raggiunga il Commit (condizione per rifare).

8.9.9 Ripartenza dopo un fallimento

L'esecuzione concorrente di transazioni è essenziale per un buon funzionamento del DBMS.

Il DBMS deve però garantire che l'esecuzione concorrente di transazioni avvenga senza interferenze in caso di accessi agli stessi dati.



- T_1 va ignorata
- T_2 e T_4 vanno rifatte
- T_3 e T_5 vanno disfatte

8.9.10 Serialità e serializzabilità

Serialità : Un'esecuzione di un insieme di transazioni T_1, \dots, T_n si dice seriale se, per ogni coppia di transazioni T_i e T_j , tutte le operazioni di T_i vengono eseguite prima di qualsiasi operazione T_j o viceversa.

Serializzabilità : Un'esecuzione di un insieme di transazioni si dice serializzabile se produce lo stesso effetto sulla base di dati di quello ottenibile eseguendo serialmente, in un qualche ordine, le sole transazioni terminate normalmente.

8.9.11 Serializzatore 2pl stretto

Il gestore della concorrenza (serializzatore) dei DBMS ha il compito di stabilire l'ordine secondo il quale vanno eseguite le singole operazioni per rendere serializzabile l'esecuzione di un insieme di transazioni.

Il protocollo del blocco a due fasi stretto (Strict Two Phase Locking) è definito dalle seguenti regole:

- 1 Transazioni diverse non ottengono blocchi in conflitto.
- 2 Ogni transazione, prima di effettuare un'operazione acquisisce il blocco corrispondente.
- 3 I blocchi si rilasciano alla terminazione della transazione.