

BASI DI DATI

TEORIA - A.A. 2023/2024

Argomenti:

1. Introduzione alle basi di dati
2. Modello relazionale
3. Progettazione ER (modello concettuale)
4. Progettazione modello logico
5. Ristrutturazione ER
6. Gestione del tempo dello schema ER
7. Normalizzazione
8. Algebra relazionale
9. Fondamenti SQL
10. Query nidificate
11. Operatori insiemistici
12. Query avanzate
13. SQL manipolazione dei dati
14. SQL gestione delle tabelle
15. SQL costrutti avanzati
16. Trigger

1. INTRODUZIONE ALLE BASI DATI

Base di dati: una base di dati è una collezione di dati che viene gestita da un DBMS (DataBase Management System).

Un DBMS è un software in grado di gestire collezioni di dati che sono:

- grandi
- condivise
- persistenti

assicurando affidabilità e sicurezza. Essi hanno dimensioni maggiori rispetto alla memoria centrale disponibile, sfruttando la memoria secondaria, inoltre consentono la condivisione **integrata** tra dati, consentendo una riduzione di ridondanze, inconsistenze dei dati ed offrendo un meccanismo di controllo dell'accesso concorrente.

Principali caratteristiche di un DBMS.

- **Persistenza dei dati:** dati sopravvivono anche dopo esecuzione programma.
- **Affidabilità in caso di guasti hw/sw:** backup e recovery.
- **Privatezza:** autorizzazioni per gli utenti.
- **Efficienza:** operazioni svolte in un tempo accettabile.
- **Efficacia:** produttività degli utenti.

Si può dire che i DBMS estendono le caratteristiche dei file system.

Modello dei dati.

È un insieme di concetti usati per organizzare i dati di interesse e descrivere la struttura in modo comprensibile all'elaboratore.

Il modello relazionale è il più diffuso, i dati sono organizzati in record omogenei (struttura fissa) come le tabelle. Dopo il modello relazionale vi sono state evoluzioni come modello ad oggetti, XML, db NOSQL.

Nel modello relazionale, definiamo:

- **Schema:** descrive la struttura dati (intestazione di ogni tabella, quindi nome tabella + nome colonne);
- **Istanza:** è costituita dal contenuto di ogni tabella, ovvero dai valori assunti dai dati (rappresenta le righe della tabella).

Esempio modello NOSQL

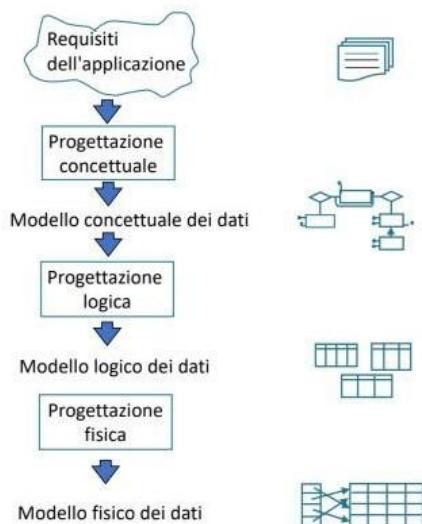
Un database è un insieme di raccolte. Ogni collezione contiene un insieme di documenti, in cui ogni documento è descritto da una lista di campi chiave/valore ed ogni campo può contenere qualsiasi tipo di dato, inoltre i documenti di una stessa collezione possono essere eterogenei.

DB Relazionale	DB NOSQL
Tabella	Collezione
Riga	Documento
Colonna	Campo

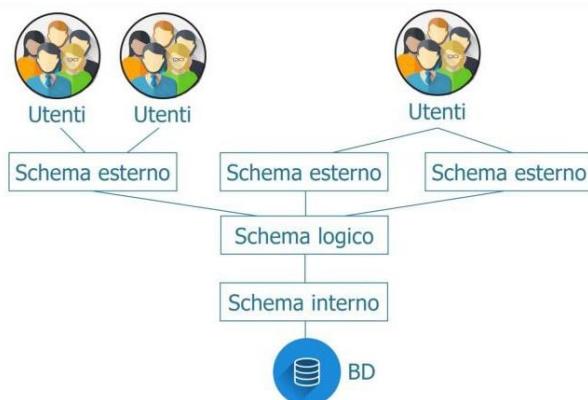
Tipi di modello

Concettuale	Logico
<ul style="list-style-type: none"> Rappresentazione dati indipendente dal modello logico Describe concetti reali Usato durante la progettazione Il più comune è il modello E-R 	<ul style="list-style-type: none"> Describe la struttura dati nel DBMS Usato dai programmi che accedono ai dati Indipendente da strutture fisiche Un esempio è il modello relazionale

Progettazione



Livelli di astrazione



Architettura a 3 livelli

1. Schema logico: descrive dati mediante modello logico del DBMS
2. Schema interno: rappresentazione dello schema logico mediante strutture fisiche di memorizzazione
3. Schema esterno: descrizione di parti della base di dati, le “viste”, definite per particolari utenti, definite a partire dal modello logico.

Indipendenza dei dati.

L'indipendenza dei dati serve a garantire che utenti e programmi che usano la base di dati siano indipendenti dai dettagli usati nella costruzione di quest'ultima. Deriva da una divisione in livelli di astrazione e individuiamo indipendenza logica e fisica.

L'**indipendenza fisica** dei dati consente di interagire con il DBMS sempre nello stesso modo, indipendentemente dalla struttura fisica dei dati ed è possibile modificare la parte fisica senza influire sui programmi che utilizzano i dati.

L'**indipendenza logica** invece consente di interagire con il livello esterno in modo indipendente dal modello logico. È possibile quindi modificare il modello logico senza toccare le strutture esterne ed aggiungere / modificare viste esistenti senza toccare lo schema logico.

Accesso ai dati.

Vi sono diversi linguaggi di accesso ai dati, come interfacce user-friendly, linguaggi testuali interattivi (SQL), linguaggi ORM e linguaggi ospite (introdotti in C, C++, Python, Java).

Sono divisi in due categorie:

- Linguaggi di definizione dei dati **DDL** ([Data Definition Language](#)) utili a definire schemi logici, esterni, fisici ed autorizzazioni
- Linguaggi di manipolazione dei dati **DML** ([Data Manipulation Language](#)) utile ad interrogare, le istanze delle basi di dati.

Utenti.

Riconosciamo il **DBAdmin**, responsabile del controllo e gestione base di dati, il quale garantisce prestazioni sufficienti, affidabilità del sistema ed autorizzazioni di accesso ai dati. Seguono **progettisti e programmatore**, che definiscono e garantiscono la struttura ed i programmi che accedono alla base di dati. Gli **utenti** usano la base per le proprie attività, possono essere **finali** (usano transazioni) o **casuali** (formulano interrogazioni).

Transazioni.

Sono programmi che realizzano attività frequenti e predefinite, come un bonifico bancario, prenotazione di un volo aereo.

Vantaggi DBMS

- Riduzione di ridondanze ed inconsistenze
- Modello dati unificato e preciso
- Controllo centrale dei dati
- Indipendenza dei dati

Svantaggi DBMS

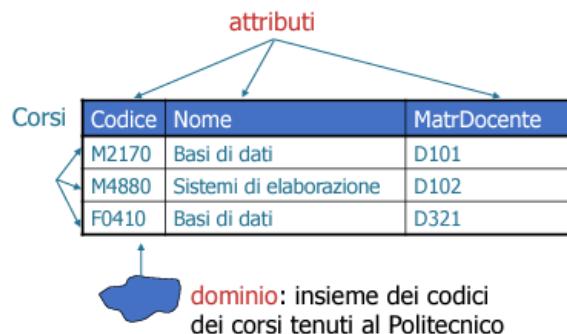
Sono costosi, complessi, richiedono investimenti diretti ed indiretti, come le risorse hw/sw ed il personale, forniscono insieme di servizi integrati che non sono possibili da scorporare per ridurre le prestazioni.

2. MODELLO RELAZIONALE

Fu proposto da E.F. Codd nel 1970 per astrarre ulteriormente i modelli precedenti, è il modello dominante nel mercato dei DBMS ed è basato sul concetto matematico di relazione.

Prime definizioni.

- **Attributo:** nome di una colonna della tabella;
- **Dominio:** insieme di valori assumibili da un attributo;
- **N-Upla:** riga della tabella;
- **Cardinalità:** numero di n-uple della relazione;
- **Grado:** numero di attributi nella relazione.



- **Schema:** descrive la struttura dati e non varia nel tempo;
- **Istanza:** è costituita dal contenuto di ogni tabella, ossia dai valori dei dati e varia nel tempo.

Inoltre le n-uple (righe) **non** sono ordinate, ma sono **distinte** tra di loro (non esistono duplicati) ed anche gli attributi non sono ordinati, non vi si accede per posizione ma per valore.

Il modello relazionale è **basato sui valori**, i riferimenti tra i dati sono rappresentati per mezzo dei valori nei domini. I vantaggi sono:

- Indipendenza dalle strutture fisiche
- Rappresentazione dell'informazione rilevante
- Maggiore portabilità dei dati tra diversi sistemi
- Legame non orientato, a differenza dei puntatori

Esempio:

Corsi	Codice	Nome	MatrDocente
M2170	Fondamenti di informatica	D101	
M4880	Sistemi di elaborazione	D102	
F0410	Basi di dati	D321	

Docenti	Matricola	Nome	Dipartimento	Telefono
D101	Verdi	Informatica	123456	
D102	Blanchi	Elettronica	636363	
D321	Neri	Informatica	414243	

Valore nullo.

Il valore nullo, **NULL** non fa parte di alcun dominio, è un valore ignoto o non definito, può esser assunto da qualsiasi attributo, ma va usato con cautela (no su chiavi primarie).

Chiavi primarie.

Una **chiave** è un insieme di attributi che **identifica in modo univoco** le tuple di una relazione ed è **minimale**. Se viene rispettata solo la proprietà di univocità parliamo di **superchiave**.

Se una chiave può assumere il valore NULL si perde il concetto di univocità, quindi una chiave è **primaria** se consente univocità, è minimale ed è **NON-NULLABILE**.

Vincoli di integrità.

Sono proprietà che devono esser soddisfatte da tutte le istanze corrette della base di dati. A seconda del tipo si suddividono in:

- **Vincolo intra-relazionale**: definiti su attributi di una sola relazione (unicità, di dominio e di tupla);
- **Vincolo inter-relazionale**: definiti su più relazioni in contemporanea (vincoli di integrità referenziale).

Vincoli di integrità referenziale.

Informazioni in relazioni diverse sono correlate attraverso i **valori comuni** di uno o più attributi.

Esempio:

Corsi (relazione referenziante)		Docenti (relazione referenziata)	
Codice	Nome	MatrDocente	Matricola
M2170	Fondamenti di informatica	D101	D101
M4880	Sistemi di elaborazione	D102	D102
F0410	Basi di dati	D101	D321

In questo esempio l'attributo *MatrDocente* nella relazione *Corsi* (**relazione referenziante**) fa riferimento all'attributo *Matricola* nella relazione *Docenti* (**relazione referenziata**). I valori assunti da *MatrDocente* in *Corsi* devono essere **esclusivamente** i valori assunti da *Matricola*, chiave primaria della relazione *Docente*. *MatrDocente* in *Corsi* costituisce la **chiave esterna (foreign key)** di *Corsi*.

Più in generale possiamo dire che il vincolo di integrità referenziale tra due relazioni R ed S tali che:

- R è la referenziata
- S è referenziante, fa riferimento ad R mediante alcuni attributi X

N.B.: l'insieme di attributi X di S costituisce una **chiave esterna** di S.

I vincoli di integrità referenziale sono **fondamentali** per garantire la correttezza dei riferimenti, basato sui valori.

Esempio:

Docenti (relazione referenziata)		Corsi (relazione referenziante)	
Matricola	Nome	Dipartimento	Telefono
D101	Verdi	Informatica	123456
D102	Bianchi	Elettronica	636363
D321	Neri	Informatica	414243

Corsi (relazione referenziante)		
Codice	Nome	MatrDocente
M2170	Fondamenti di informatica	D101
M4880	Sistemi di elaborazione	D102
F0410	Basi di dati	D322

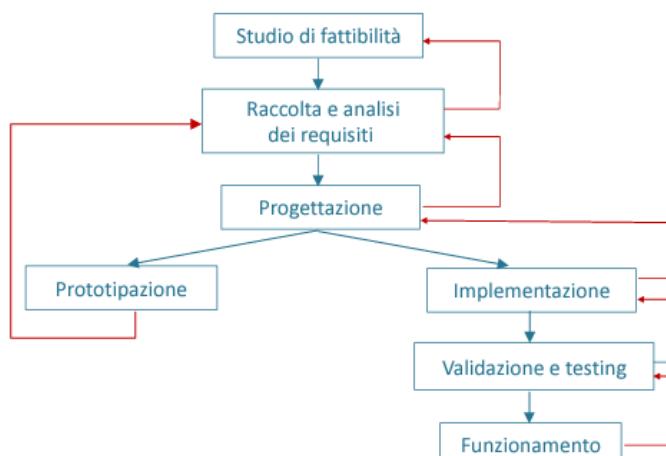
3. PROGETTAZIONE ER

Progettazione basi di dati.

È una delle attività del processo di sviluppo di un sistema informatico.

Vi sono diversi punti:

- Studio di fattibilità: determinazione dei costi delle diverse alternative e delle priorità di realizzazione;
- Raccolta e analisi dei requisiti: definizione delle proprietà e funzionalità del sistema interagendo con utente e descrivendo informalmente il sistema da realizzare;
- Progettazione: suddivisa in progettazione dei dati e delle applicazioni, produce descrizioni formali;
- Implementazione: realizzazione del sistema informativo secondo le caratteristiche definite in fase di progettazione;
- Validazione e testing: verifica del corretto funzionamento e della qualità del sistema. Può portare a modifiche dei requisiti o a revisione progetto;
- Funzionamento: operatività del sistema, gestione e manutenzione;
- Prototipazione: rapida versione semplificata del sistema per valutarne le caratteristiche e può portare a modifiche dei requisiti e/o revisione del progetto.



Metodologia di progettazione.

Una metodologia di progettazione consiste in:

- Decomposizione dell'attività di progetto in fasi successive ed indipendenti;
- Strategie da seguire nelle varie fasi ed i criteri di scelta delle strategie;
- Modelli di riferimento per descrivere i dati di ingresso ed uscita delle varie fasi.

La metodologia ha 3 **proprietà principali**:

1. **Generalità**: possibilità di utilizzo indipendentemente dal problema e dagli strumenti disponibili;
2. **Qualità del risultato**: in termini di correttezza, completezza ed efficienza rispetto alle risorse utilizzate;
3. **Facilità di utilizzo**: sia delle strategie che dei modelli di riferimento.

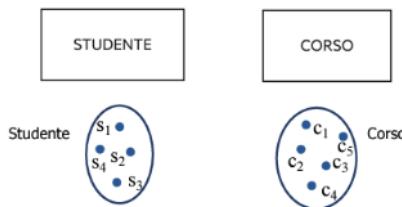
Inoltre occorre distinguere **cosa** rappresentare (modello concettuale) e **come** rappresentarlo (modello logico e fisico).

Si ha quindi un flusso del seguente tipo:

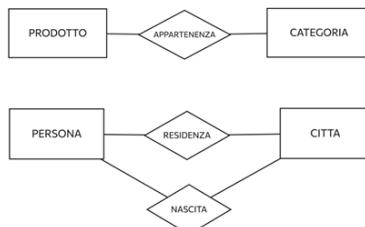
- 1) **Schema concettuale** dove si rappresenta il **contenuto informativo** della base dati, indipendente da aspetti implementativi, con una descrizione formale e completa.
- 2) **Traduzione nello schema logico** facendo riferimento al modello di dati prescelto, controllando la **qualità dello schema** (normalizzazione) ed adottando **criteri di ottimizzazione**.
- 3) Specifica dei **parametri fisici di memorizzazione** dei dati (organizzazione di file ed indici) producendo un **modello fisico** in funzione del DBMS scelto.

Modello Entità-Relazione.

1. **Entità:** rappresenta classi di oggetti del mondo reale che hanno proprietà comuni ed una esistenza autonoma.

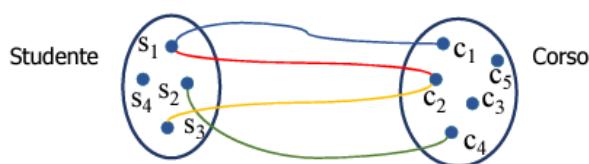


1. **Relazione:** costituisce un legame logico tra due o più entità.



Occorrenze di una relazione.

Un'occorrenza di una relazione è una n-upla (coppia per relazioni binarie) costituita da occorrenze di entità, una per ciascuna delle entità coinvolte.



Cardinalità delle relazioni binarie.

Sono specificate per ogni entità che partecipa alla relazione, descrivono il numero minimo e massimo di occorrenze a cui può partecipare una occorrenza di entità:

minima:

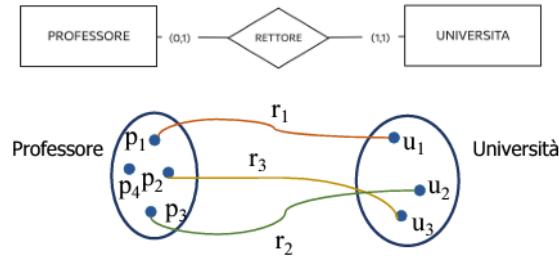
- 0 (**partecipazione opzionale**)
- 1 (**partecipazione obbligatoria**)

massima:

- 1 (**al più un'occorrenza**)
- N (**numero arbitrario**)

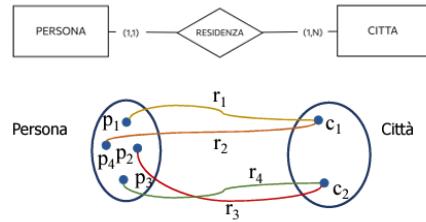
Corrispondenza 1 ad 1.

2 entità che partecipano entrambe con cardinalità massima 1 ad 1. In questo esempio, non tutti i professori che partecipano alla relazione rettore, al più c'è solo un rettore per ogni università, mentre in un'università deve esserci per forza un rettore (card min) e solo 1 (card max).



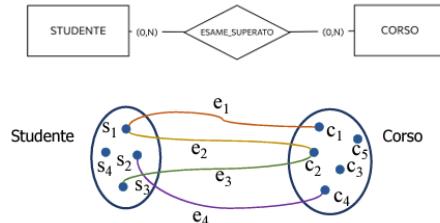
Corrispondenza 1 ad N (uno a molti).

Persona partecipa obbligatoriamente (min=1) e ha (max=1) può risiedere in una città, viceversa la città può avere una o più persone residenti (min=1, max=N).



Corrispondenza N ad N (molti a molti).

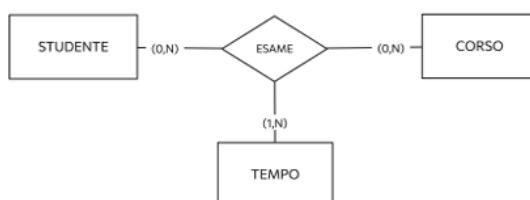
Uno studente può non aver superato un esame (min=0) e può superare N esami (max=N), viceversa un esame può esser nuovo e non esser stato superato (min=0) ed è associato ad N studenti (max=N).



Relazione ternaria.

Uno studente può sostenere lo stesso esame più volte in date diverse.

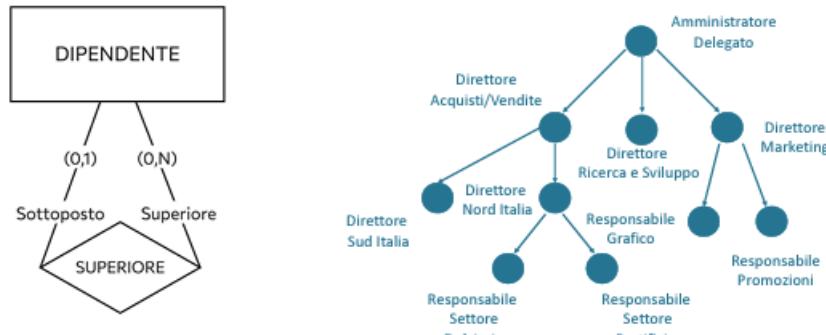
N.B.: la relazione che discrimina le diverse istanza, in questo caso il tempo **deve partecipare con cardinalità (1,N)**.



Raramente le cardinalità minime sono 1 per tutte le entità coinvolte nella relazione.

Relazione ricorsiva.

Relazione di un'entità con se stessa, se non è simmetrica occorre definire i ruoli. In questo esempio l'entità dipendente si associa a sé con le associazioni superiore/sottoposto. Sottoposto è opzionale , un dipendente può non esser sottoposte, al massimo è sottoposto da un dipendente. Un dipendente può non esser un superiore, ma potrebbe al massimo esser superiore di più dipendenti (0,N).



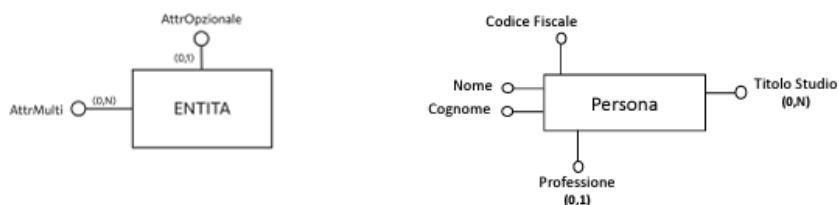
- Attributo: descrive una proprietà elementare di un'entità o di una relazione, ogni attributo ha un dominio di valori ammissibili.



Possono esserci degli attributi composti, i quali aggregano attributi affini per significato o per uso (componenti di un indirizzo / di un'anagrafica).



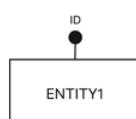
Gli attributi hanno una propria cardinalità, se è omessa corrisponde ad (1,1), mentre se la minima è pari a 0 significa che ho un attributo **opzionale** e quindi che ammette il valore null, se la massima è N allora ho un **attributo multivaleure**.



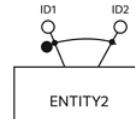
- Identificatore: è specificato per ogni entità ed è composto dal sottoinsieme di attributi che identifica in modo univoco le occorrenze di un'entità. Può essere interno o esterno, semplice o composto.

ID interno.

Semplice
• costituito da un solo attributo

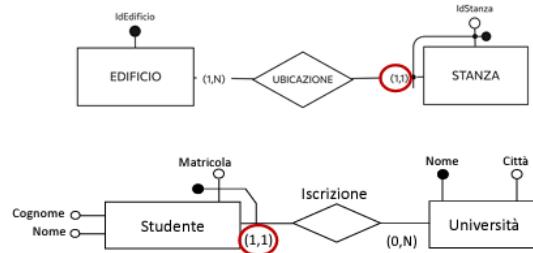


Composto
• costituito da più attributi

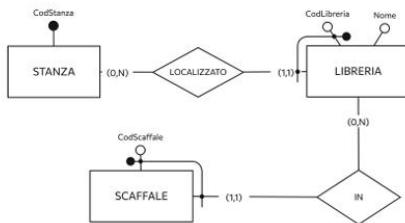


ID esterno.

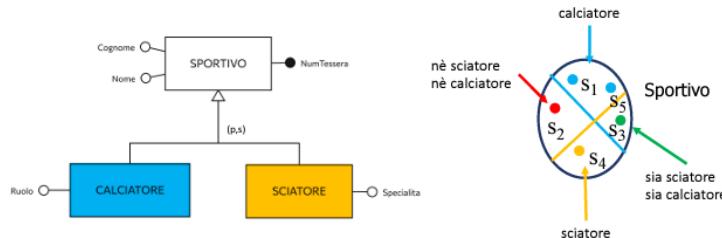
L'entità che non dispone di attributi sufficienti per definire un identificatore viene detta **entità debole** e **deve** partecipare con **cardinalità (1,1)** in ognuna delle relazioni che forniscono parte dell'identificatore.



Osservazione: un identificatore esterno può coinvolgere un'entità che a sua volta identificata esternamente, l'importante è non creare cicli di identificazione!



4. **Generalizzazione:** descrive un collegamento logico tra un'entità E, ed una o più entità E_1, \dots, E_N in cui E comprende come casi particolari di E_1, \dots, E_N . E viene detta **entità padre**, **generalizzazione** di E_1, \dots, E_N le quali sono dette **entità figlie**, ovvero **specializzazioni** di E.



Ogni occorrenza di un'entità figlia è anche un'occocrenza dell'entità padre, si verifica la proprietà di **ereditarietà**, ogni occorrenza dell'entità figlia riceve identificatore, attributi e relazioni dell'entità padre.

Caratteristiche ortogonali.

Una generalizzazione si dice **totale** se ogni occorrenza dell'entità padre è un'occocrenza di almeno una delle figlie, altrimenti si dice **parziale**. Si dice **esclusiva** se ogni occorrenza dell'entità padre è al più un'occocrenza delle entità figlie, **sovrapposta** altrimenti.

Sottoinsieme.

Una generalizzazione con solo un'entità figlia è sempre parziale ed esclusiva.

Documentazione schemi E-R.

Il **dizionario dei dati** permette di arricchire lo schema E-R con descrizioni di linguaggio naturale di entità, relazioni, attributi.

Esempio:

Entità	Descrizione	Attributi	Identificatore
Studente	Studente dell'università	Matricola, Cognome, Nome, Crediti acquisiti, Media voti	Matricola
Docente	Docente dell'università	Codice docente, Dipartimento, Cognome, Nome	Codice docente
Corso	Corsi offerti dall'università	Codice corso, Nome, Crediti	Codice corso
Tempo	Date in cui sono stati sostenuti esami	Data	Data

Relazione	Descrizione	Entità coinvolte	Attributi
Esame	Associa uno studente agli esami che ha sostenuto e memorizza il voto conseguito	Studente (0,N), Corso (0,N), Tempo (1,N)	Voto
Titolare	Associa ogni corso al suo docente titolare	Corso (1,1), Docente (0,N)	

I **vincoli di integrità** non sempre possono esser indicati esplicitamente in uno schema ER, ma possono esser scritti in linguaggio naturale.

Vincoli d'integrità	
RV1	Il voto di un esame può assumere esclusivamente valori compresi tra 0 e 30
RV2	Ogni studente non può superare due volte con esito positivo lo stesso esame
RV3	Uno studente non può sostenere più di tre volte l'esame relativo allo stesso corso nell'arco dello stesso anno accademico

Le regole di derivazione dei dati indicano come ottenere un concetto dello schema, a partire da altri concetti dello schema.

Regole di derivazione	
RD1	Il numero di crediti acquisiti da uno studente si ottiene sommando il numero di crediti dei corsi per cui lo studente ha superato l'esame
RD2	La media voti di uno studente si ottiene calcolando la media dei voti degli esami superati dallo studente

UML VS ER.

- UML (Unifield Modeling Language): modellazione di un'applicazione software, formalismo ricco, aspetti strutturali e comportamentali.
- ER: specifico per modellare basi di dati e funzionale.

Differenze UML da ER.

- Assenza di notazione standard per definire gli identificatori
- Possibilità di commenti per i diagrammi
- Possibilità di indicare verso di navigazione di un'associazione (non rilevante per progettare basi di dati)

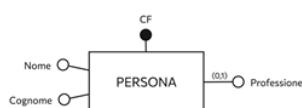
4) PROGETTAZIONE MODELLO LOGICO

Essa si esegue su uno schema ER **ristrutturato**, ovvero senza gerarchie, attributi multivale e composti.

Traduzione di entità.

Ad ogni entità corrisponde una tabella con gli stessi attributi, che costituiscono lo **schema della tabella**. L'identificatore (semplice o composto) dell'entità diventa la **chiave primaria** della tabella. Gli attributi opzionali dell'entità sono attributi che possono assumere il valore NULL, indicati con '*' nello schema della tabella.

Modello concettuale



Modello logico

Persona(CodiceFiscale, Nome, Cognome, Professione*)

- Chiave primaria sottolineata
- Attributi opzionali indicati con asterisco

Traduzione di relazioni.

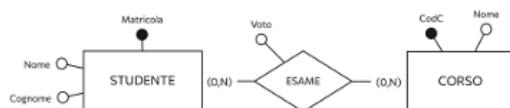
Per la traduzione di una relazione vi sono 3 passi fondamentali:

1. Vengono prima tradotte le entità che partecipano alla relazione;
2. Viene quindi tradotta la relazione (a seconda di binarie o ternarie);
3. Occorre tenere conto di cardinalità massima e minima con cui le entità partecipano alla relazione.

Traduzione di relazione binaria.

Relazione molti a molti.

Modello concettuale



Modello logico

Studente (Matricola, Nome, Cognome)

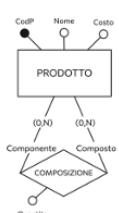
Corso (CodC, Nome)

Esame (Matricola, CodC, Voto)

Ogni relazione molti a molti corrisponde ad una tabella, la cui chiave primaria è la combinazione degli identificatori delle due entità collegate, è possibile ridenominare gli attributi della tabella che corrisponde alla relazione.

Relazione molti a molti ricorsiva.

Modello concettuale



Modello logico

Prodotto (CodP, Nome, Costo)

Composizione (CodComposto, CodComponente, Quantità)

Ogni relazione molti a molti corrisponde ad una tabella, la cui chiave primaria è combinazione degli identificatori delle due entità.

Relazione binaria uno a molti.

Sono possibili due modalità di traduzione:

- Mediante attributi

Modello concettuale **Modello logico**



Person (CE, Nome, Cognome, **NomeComune**, Data)
Comune (**Nome**, Provincia)

Si effettua quando l'entità che ha cardinalità massima ad 1, partecipa alla relazione con cardinalità minima pari ad 1.

- Mediante una nuova tabella

Modello concettuale



Modello logico

Alternativa 1: Traduzione mediante attributi
Studente (Matricola, Nome, Cognome, **NomeFacoltà***, Data*)
Facoltà (**Nome**, Città)

Alternativa 2: Traduzione mediante una nuova tabella
Studente (Matricola, Nome, Cognome)
Facoltà (**Nome**, Città)
Laurea (Matricola, **NomeFacoltà**, Data)

Si effettua quando l'entità che partecipa con cardinalità massima pari ad 1 alla relazione, partecipa optionalmente, con cardinalità minima pari a 0.

Relazione binaria uno ad uno.

Sono possibili più traduzioni che dipendono a seconda del valore della cardinalità minima.

- Esempio 1:

Modello concettuale



Modello logico

Alternativa 1
Rettore (Matricola, Nome, Cognome, **NomeFacoltà**, Data)
Università (**Nome**, Città)

Alternativa 2
Rettore (Matricola, Nome, Cognome)
Università (**Nome**, Città, **Matricola**, Data)

si effettua quando entrambe le entità partecipano con cardinalità massima pari ad 1 alla relazione, ed entrambe le entità partecipano obbligatoriamente alla relazione.

- Esempio 2:

Modello concettuale

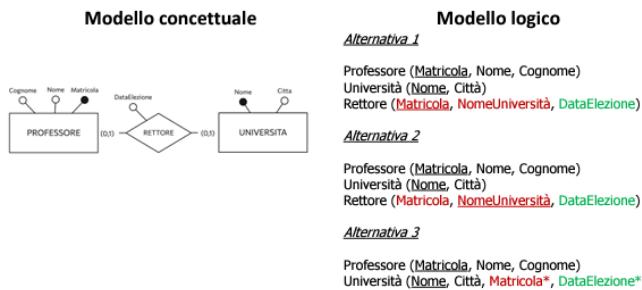


Modello logico

Professore (Matricola, Nome, Cognome)
Università (**Nome**, Città, **Matricola**, DataElezioni)

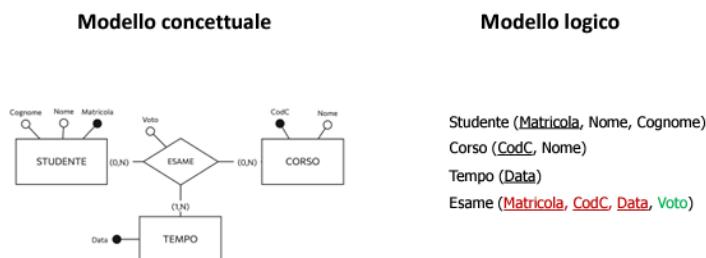
si effettua quando entrambe le entità partecipano con cardinalità massima pari ad 1 alla relazione, ma solo con una delle entità partecipa obbligatoriamente alla relazione.

- Esempio 3:

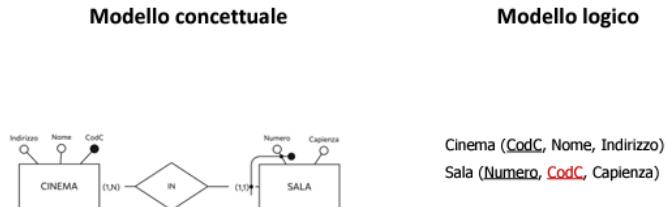


si effettua quando entrambe le entità partecipano con cardinalità massima pari ad 1 alla relazione, ed entrambe partecipano opzionalmente alla relazione (cardinalità minima pari a 0).

Traduzione di relazione ternaria.

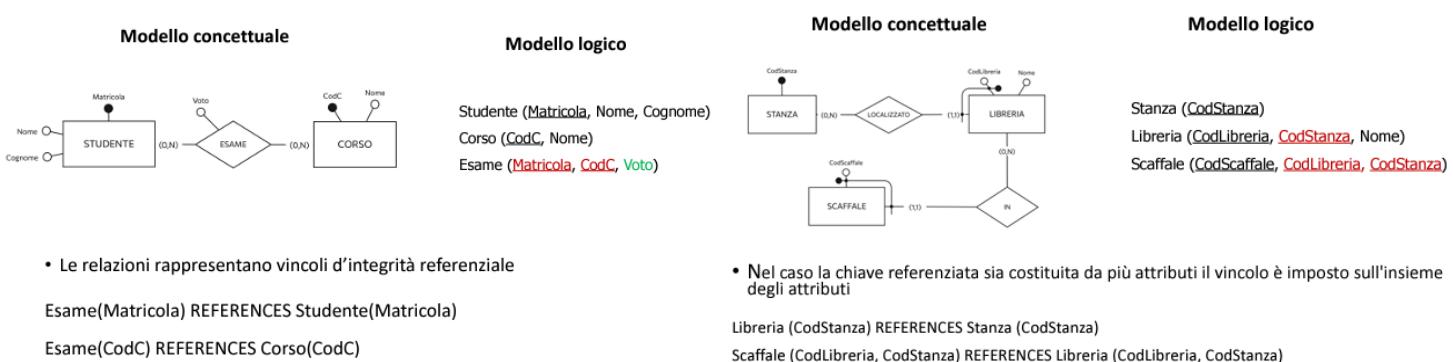


Traduzione di entità con identificatore esterno.



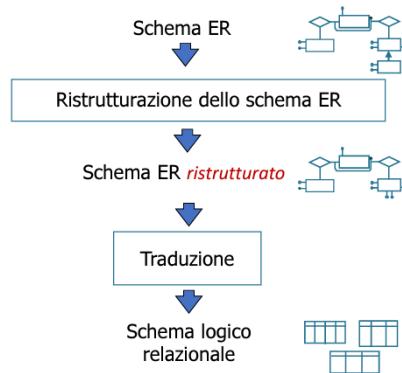
La relazione è rappresentata insieme all'identificatore, la relazione contribuisce alla definizione dell'identificatore dell'entità debole.

Vincoli di integrità referenziale.



5) RISTRUTTURAZIONE ER

Flusso della progettazione logica:



Lo schema ER ristrutturato tiene conto di aspetti realizzativi: non è più uno schema concettuale. Gli obiettivi della ristrutturazione sono i seguenti:

- 1) Eliminazione dei costrutti per cui non esiste una rappresentazione diretta nel modello relazionale;
- 2) Aumentare le operazioni di accesso ai dati.

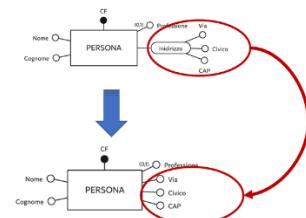
Le **attività di ristrutturazione** sono:

- Eliminazione attributi composti;
- Eliminazione attributi multivaleure;
- Eliminazione delle generalizzazioni;
- Analisi delle ridondanze;
- Partizionamento di entità e relazioni;
- Scelta degli identificatori primari.

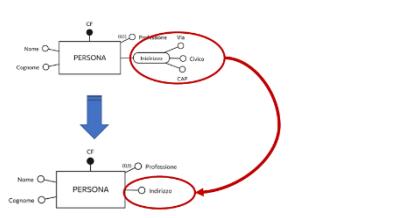
Eliminazione attributi composti.

Gli attributi composti, non essendo rappresentabili all'interno del modello relazionale, subiscono due possibili trasformazioni:

- 1) Si rappresentano in **modo separato** gli attributi componenti (adatta nel caso in cui sia necessario accedere ad ogni attributo separatamente);
- 2) Si rappresentano con un **unico attributo** rappresenta la concatenazione degli attributi componenti (adatta se basta accedere all'info complessiva).



rappresentazione separata



rappresentazione con singolo attributo

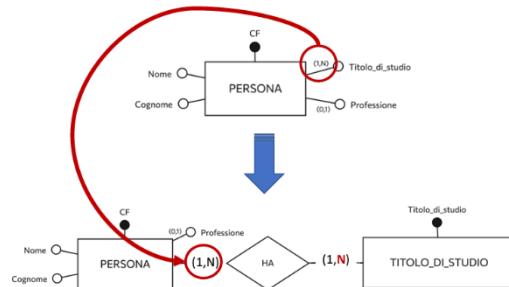
Eliminazione attributi multivalore.

Non possono esser rappresentati nel modello relazionale, vengono rappresentati utilizzando una **relazione** tra entità iniziale ed una nuova entità.

Attenzione alla cardinalità della nuova relazione!

Informazione condivisa.

Si suppone che una persona possa avere più di un titolo di studio, ma che il titolo di studio possa esser in possesso di più persone. Nella traduzione verrà eliminata la tabella **titolo di studio**, in quanto è una proprietà di persona ed esiste solo associato ad essa.

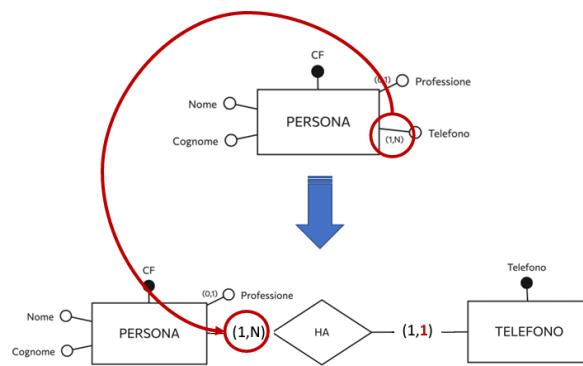


Informazione univoca.

Si suppone che una persona possa avere più di un numero di telefono, ma che **lo stesso numero di telefono sia in possesso di una sola persona**. Il numero di telefono al massimo è associato ad una persona ed ha anche cardinalità minima pari ad 1. Si suppone quindi un'associazione 1 ad N, pilotata da telefono in cui la traduzione sarà data da:

Persona (CF, Nome, Cognome, Professione*)

Telefono (Telefono, CF)



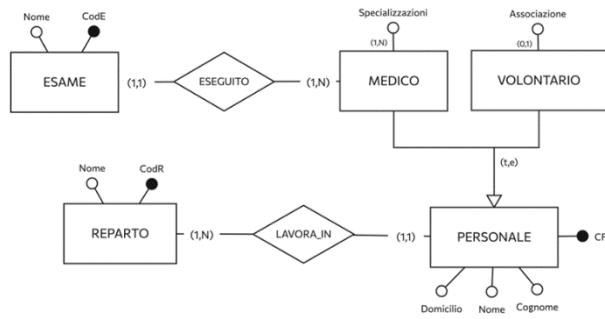
Eliminazione delle generalizzazioni.

Non sono rappresentabili nel modello relazionale, costituite da entità e relazioni. Ci sono dei metodi di ristrutturazione:

- 1) Accorpamento delle entità figlie nell'entità padre;
- 2) Accorpamento dell'entità padre nelle figlie;
- 3) Sostituzione della gerarchia con relazioni.

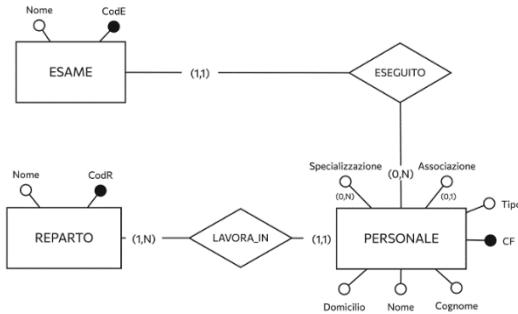
N.B.: la scelta dipende dal carico applicativo interno.

Schema di esempio di riferimento.



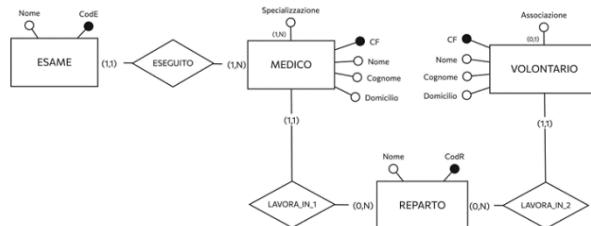
1. Accorpamento nell'entità padre.

Si ereditano le relazioni con delle entità figlie con cardinalità minima pari a 0; si introduce l'attributo **discriminante** Tipo che consente di distinguere se medico o volontario; gli attributi delle entità figlie passano all'entità padre diventando opzionali.



2. Accorpamento nelle entità figlie.

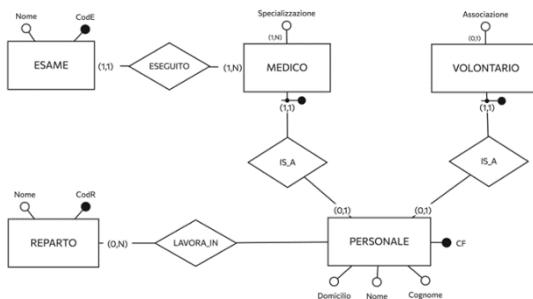
Gli attributi dell'entità padre (compreso identificatore) passano alle entità figlie e si **sdoppiano** le relazioni dell'entità padre, ponendo le cardinalità minime pari a 0.



N.B.: non è adatta per copertura parziale, ci si può ricondurre ad una generalizzazione totale inserendo un'entità figlia 'Altri'. **Non adatta per copertura sovrapposta**, ci sarebbero problemi con identificatori duplicati.

3. Sostituzione con relazione tra entità padre e figlie.

È la soluzione più generale e **sempre applicabile** ma dispendiosa per ricomporre l'informazione di partenza.



Accorpare le entità figlie in quella padre è appropriato quando:

- Le entità figlie hanno pochi valori nulli;
- Le operazioni di accesso non distinguono tra occorrenze dell'entità padre e delle figlie.

Accorpare l'entità padre in quelle figlie è appropriato quando:

- La generalizzazione è totale;
- Le operazioni di accesso distinguono tra occorrenze delle diverse entità figlie.

La soluzione “mista” è appropriata quando:

- Le operazioni di accesso distinguono tra occorrenze di alcune entità figlie.

Si procede nello stesso modo per le generalizzazioni a più livelli, partendo dal livello inferiore.

Analisi delle ridondanze.

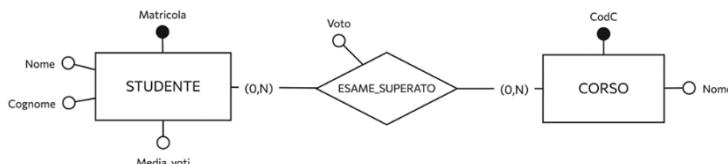
Le ridondanze sono ripetizioni di informazioni derivabili da altri concetti: occorre decidere se conservarle, tenendo conto che esse come effetto sullo schema logico comportano:

- Semplificazione e velocizzazione delle interrogazioni;
- Maggiore complessità e rallentano gli aggiornamenti;
- Maggiore occupazione di spazio.

La dimensione delle tabelle incide significativamente sul costo delle operazioni sui dati.

Esempio attributo ridondante.

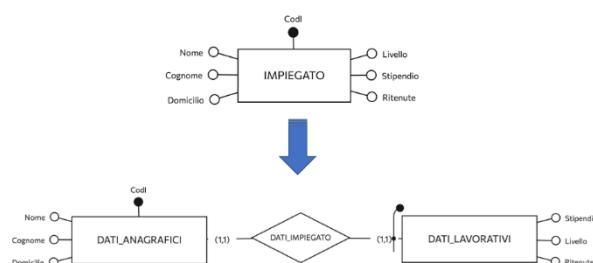
L'attributo Media_Voti è ridondante in quanto è utile per velocizzare le query relative al calcolo della media, ma se conservato occorre integrare lo schema relazionale con l'indicazione di ridondanza.



Partizionamento di concetti.

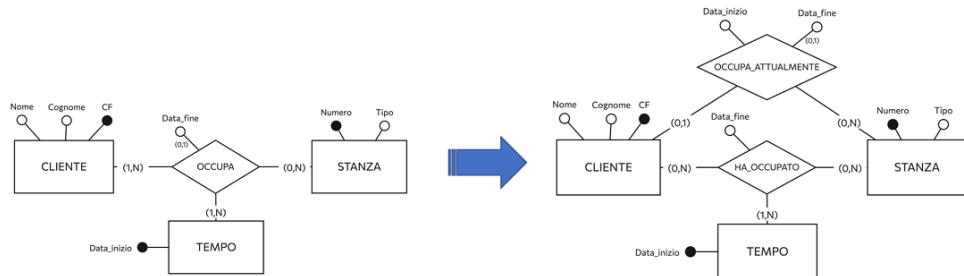
Il partizionamento di entità o relazioni consente nella rappresentazione migliore di concetti separati, separazione di attributi di uno stesso concetto usati da operazioni diverse e maggiore efficienza nelle operazioni.

Esempio partizionamento di entità.



Diversi tipi di dato, alcune app vogliono vedere i dati anagrafici, altre quelli lavorativi.

Esempio partizionamento di relazioni.



In questo caso si vuole tenere conto anche delle occupazioni precedenti.

Scelta degli identificatori primari.

Sono necessari per la chiave primaria delle tabelle.

Un buon identificatore:

- non assume valore nullo
- è costituito da pochi attributi
- è preferibilmente interno
- è usato da molte operazioni di accesso

Talvolta è utile inserire dei codici identificativi.

6) GESTIONE DEL TEMPO NELLO SCHEMA ER

È sicuramente un aspetto complesso nella gestione delle basi di dati. Occorre gestire un evento che si ripete nel tempo, con occorrenze di un'entità oppure di una relazione.

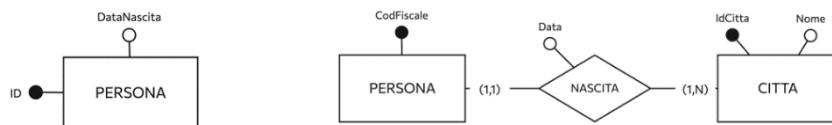
Vi è la necessità di tracciare eventi e l'evoluzione temporale di eventi e/o relazioni.

Le tipologie di modellazione di evoluzione dello storico nel tempo sono date da:

- Attributi temporali
- Relazione binaria
- Relazione ternaria
- Entità storicizzata

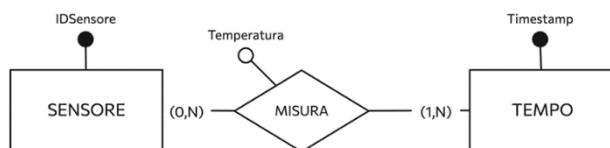
Attributi temporali.

Contengono informazioni su eventi che coinvolgono un'unica entità/relazione, ad esempio la data di nascita, anno produzione di un film.



Relazione binaria.

Esempio 1: sono disponibili un insieme di sensori, ciascuno identificato da un codice univoco e presenti dentro un edificio. Si vuole memorizzare i diversi valori di temperatura rilevati da ciascun sensore in **diversi istanti temporali**.



Il cui modello logico sarà

SENSORE (IDSensore)

TEMPO(Timestamp) → **N.B: è eliminabile!**

MISURA(IDSensore, Timestamp, Temperatura)

Lo stesso evento si ripete nel tempo

S1 ad istante T1 misura temperatura TEMP1

S2 ad istante T1 rileva TEMP2

In generale, si vuole rappresentare una serie temporale di eventi legati ad un'entità E del diagramma ER.

L'informazione è rappresentata mediante l'introduzione di:

- Entità **Tempo** che contiene indicazione temporale su **quando** si verifica/inizia l'evento

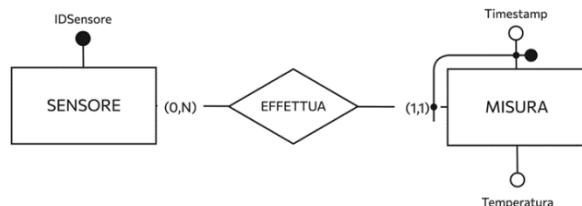
- **Relazione binaria R** che collega il **Tempo** all'entità E. Le eventuali informazioni sulla durata e/o istante temporale di fine dell'evento e/o su altri aspetti che caratterizzano il verificarsi dell'evento nei diversi istanti temporali, sono attributi di R.

Esempio 2(entità debole):

lo schema logico corrispondente è

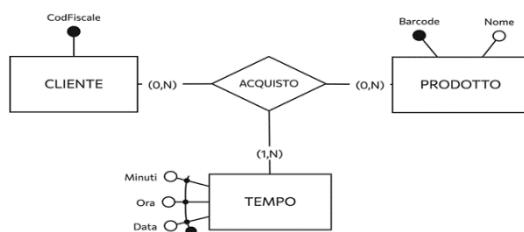
SENSORE(IDSensore)

MISURA(TimeStamp,IDSensore,Temperatura)



Relazione ternaria.

Esempio 1: si vogliono memorizzare gli acquisti di prodotti effettuati da ciascun cliente. Ogni cliente è univocamente identificato dal codice fiscale ed ogni prodotto dal barcode. Si suppone che ogni cliente possa acquistare lo stesso prodotto in istanti diversi della giornata.



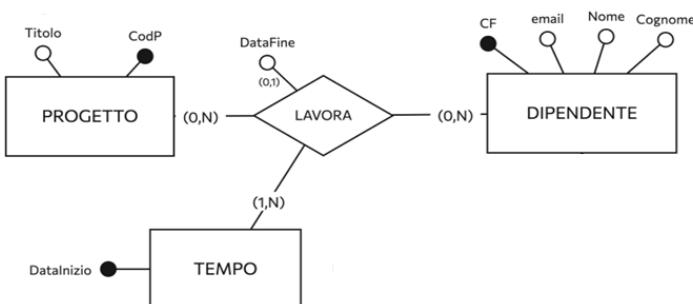
In tale modo l'acquisto è identificato dalla tripletta formata da: data, dal codice fiscale, dal bar code.

Si vuole memorizzare una serie di eventi temporali espressi mediante associazione tra entità E1 ed E2.

L'informazione è rappresentata mediante introduzione di:

- Un'entità tempo (identificata su quando si verifica/inizia l'evento)
- **Relazione ternaria R** che collega tempo, E1 ed E2, le eventuali informazioni sulla durata sono caratteristiche, ovvero attributi di R.

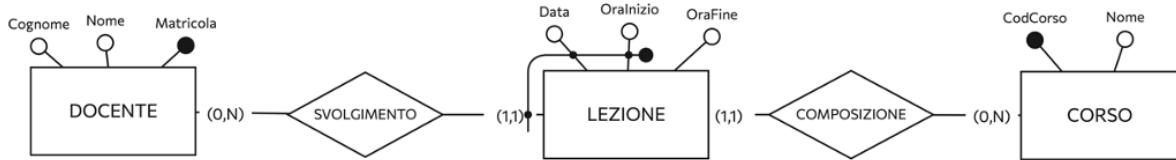
Esempio 2: una ditta fornisce consulenza informatica vuole memorizzare i lavori effettuati dai suoi dipendenti per ogni progetto. Ciascun progetto è identificato da un codice alfanumerico, i dipendenti sono identificati dal codice fiscale. Si vuole tenere traccia dei periodi di tempo nei quali un dipendente lavora su un progetto. Più dipendenti possono lavorare nello stesso periodo sullo stesso progetto.



Il lavoro sul progetto è identificato dalla tripletta codice fiscale, codice progetto, data di inizio ed è caratterizzato dalla data di fine. Le informazioni sulla data di fine sono gestite mediante un trigger.

Entità storicizzata.

Esempio: si vogliono memorizzare le lezioni erogate da ciascun docente per ogni corso. I docenti sono identificati dalla matricola, il corso dal codice, la lezione dalla fascia oraria. Si suppone che ogni docente possa erogare una lezione nella stessa fascia oraria.



Si vuole memorizzare il concetto di lezione erogata.

CORSO	DOCENTE	DATA	ORAINIZIO	ORAFINE
C1	D1	DA1	OI1	OF1
C1	D1	DA1	OI2	OF2
C1	D1	DA2	OI2	OF2 → ERRORE
C2	D1	DA2	OI2	OF2 → ERRORE

Docente è l'entità, il tempo serve a caratterizzare la lezione. La lezione è l'evento che si ripete nel tempo e deve avere come caratteristica il corso. Lezione diventa un'**entità storicizzata**.

Traduzione:

DOCENTE(Matricola,Nome,Cognome)

CORSO(CodCorso,Nome)

LEZIONE(Data,Ora,Matricola,OraFine,CodCorso)

Con l'entità storicizzata si vuole rappresentare un evento che coinvolge due entità, con vincoli sulla partecipazione da parte di un'entità a più eventi.

L'informazione di interesse è rappresentata mediante l'introduzione di:

un'entità debole E identificata dall'informazione temporale su quando si verifica/inizia l'evento ed esternamente mediante la relazione collegata all'entità che **non** può partecipare contemporaneamente a due eventi. Le caratteristiche dell'evento sono attributi dell'entità debole E, la quale partecipa con **cardinalità (1,1)** alle relazioni con le altre entità.

7) NORMALIZZAZIONE

La normalizzazione è un procedimento che, a partire da uno schema relazionale non normalizzato permette di agire su di esso producendone uno normalizzato. La normalizzazione **non è un metodo di progettazione**, ma uno **strumento di verifica**. La progettazione di schemi ER corretti genera schemi relazionali normalizzati, le cui verifiche possono essere applicate ad altri schemi ER.

Esempio:

Esame Superato

MatrStudente	Residenza	CodCorso	NomeCorso	Voto
s94539	Milano	04FLCY	Calcolatori elettronici	30
s94540	Torino	01FLTCY	Basi di dati	26
s94540	Torino	01KPNCY	Reti di calcolatori	28
s94541	Pescara	01KPNCY	Reti di calcolatori	29
s94542	Lecce	04FLCY	Calcolatori elettronici	25

Tale esempio riporta della **ridondanza**, in ogni riga in cui compare uno stesso studente è ripetuta la sua residenza, in ogni riga in cui compare un corso è ripetuto il suo nome, mentre questi attributi sono caratteristiche e funzioni solamente dello studente e del corso.

Tale schema comporta diverse **anomalie**.

- Di **aggiornamento**: se cambia la residenza di uno studente, occorre modificare tutte le righe in cui compare contemporaneamente
- Di **inserimento**: se uno studente si iscrive all'università, non può esser inserito nella base fino a quando non supera un esame
- Di **cancellazione**: se rinuncia agli studi, non si può tenerne traccia.

In generale, la **ridondanza** si manifesta quando un'unica relazione è usata per rappresentare **informazioni eterogenee**.

Le informazioni ridondanti devono esser aggiornate in modo atomico (contemporaneamente), la cancellazione di una tupla comporta la cancellazione di tutti i concetti in essa rappresentati, l'inserimento di un valore è possibile solo se esiste almeno l'informazione completa relative alla chiave primaria.

Forma normale di Boyce-Codd.

La dipendenza funzionale è un particolare vincolo di integrità che descrive legami di tipo funzionale tra attributi di una relazione.

Nell'esempio precedente, la residenza è unica per ogni studente ma ogni volta che compare lo stesso studente il valore è ripetuto. Ciò non va bene poiché è il valore di MatrStudente che **determina** il valore di Residenza.

Definizione: una relazione r soddisfa la dipendenza funzionale $X \rightarrow Y$ se, per ogni coppia t_1, t_2 di tuple di r, aventi uguali valori per tutti gli attributi X, t_1 e t_2 hanno gli stessi valori anche per gli attributi in Y.

X determina Y (in r)

Esempi sono:

$$\begin{aligned} \text{MatrStudente} &\rightarrow \text{Residenza} \\ \text{CodCorso} &\rightarrow \text{NomeCorso} \\ \text{MatrStudente CodCorso} &\rightarrow \text{NomeCorso} \end{aligned}$$

La dipendenza MatrStudente CodCorso → CodCorso è banale poiché CodCorso fa parte di entrambi i lati. Una dipendenza funzionale X → Y è non banale se nessun attributo X compare tra gli attributi in Y.

Data una chiave K di qualsiasi relazione r: K → qualsiasi altro attributo di r

Esempio:

MatrStudente CodCorso → Voto

Le **anomalie** sono causate da proprietà degli attributi coinvolti in dipendenza funzionali.

Esempi:

- MatrStudente → Residenza
- CodCorso → NomeCorso

Le **dipendenze funzionali** dalle chiavi non originano anomalie

Esempi:

- MatrStudente CodCorso → Voto

Formalmente le anomalie sono causate dall'inclusione di concetti indipendenti tra loro nella stessa relazione da dipendenze funzionali X → Y che permettono la presenza di più tuple con lo stesso valore di X, X non contiene la chiave.

BCNF = Boyce Codd Normal Form

Una relazione r è in BCNF se, per ogni dipendenza funzionale (non banale) X → Y definita su di essa, X contiene una chiave di r (X è superchiave di r).

Per la decomposizione in BCNF, si deve scomporre una relazione contenente più concetti indipendenti in relazioni più piccole, una per ogni concetto, per mezzo delle dipendenze funzionali. Le nuove relazioni sono ottenute mediante proiezioni sugli insiemi di attributi corrispondenti alle dipendenze funzionali. Le chiavi delle nuove relazioni sono le parti sinistre delle dipendenze funzionali.

Esempio:

Esame Superato				
MatrStudente	Residenza	CodCorso	NomeCorso	Voto
s94539	Milano	04FLYCY	Calcolatori elettronici	30
s94540	Torino	01FLTCY	Basi di dati	26
s94540	Torino	01KPNKY	Reti di calcolatori	28
s94541	Pescara	01KPNKY	Reti di calcolatori	29
s94542	Lecce	04FLYCY	Calcolatori elettronici	25

Dipendenze funzionali nell'esempio

- MatrStudente → Residenza
- CodCorso → NomeCorso
- MatrStudente CodCorso → Voto

Si modifica:

R(MatrStudente,Residenza,CodCorso,NomeCorso,Voto)

Le relazioni in BCNF sono

R1 (MatrStudente, Residenza) = $\pi_{\text{MatrStudente}, \text{Residenza}} R$

R2 (CodCorso, NomeCorso) = $\pi_{\text{CodCorso}, \text{NomeCorso}} R$

R3 (MatrStudente, CodCorso, Voto) = $\pi_{\text{MatrStudente}, \text{CodCorso}, \text{Voto}} R$

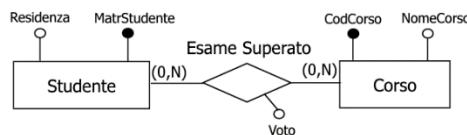
Decomposizione in 3 tabelle:

MatrStudente	Residenza
s94539	Milano
s94540	Torino
s94541	Pescara
s94542	Lecce

CodCorso	NomeCorso
04FLCY	Calcolatori elettronici
01FLTCY	Basi di dati
01KPNYC	Reti di calcolatori

MatrStudente	CodCorso	Voto
s94539	04FLCY	30
s94540	01FLTCY	26
s94540	01KPNYC	28
s94541	01KPNYC	29
s94542	04FLCY	25

Schema ER corrispondente con traduzione:



Studente (MatrStudente, Residenza)
Corso (CodCorso, NomeCorso)
Esame Superato (MatrStudente, CodCorso, Voto)

Proprietà delle composizioni.

Alcuni possibili **problemi** possono derivare da **perdita di informazione** e delle dipendenze.

Esempio 1:

Impiegato	Categoria	Stipendio
Rossi	2	1800
Verdi	3	1800
Bianchi	4	2500
Neri	5	2500
Bruni	6	3500

R (Impiegato, Categoria, Stipendio)

Impiegato \rightarrow Categoria
Impiegato \rightarrow Stipendio
Categoria \rightarrow Stipendio

Decomposizione senza perdita.

Decomposizione N1:

Impiegato \rightarrow Stipendio

Categoria \rightarrow Stipendio

Quindi decomponendo:

$$R_1 (\underline{\text{Impiegato}}, \text{Stipendio}) =$$

$$\pi_{\text{Impiegato}, \text{Stipendio}} R$$

Impiegato	Stipendio
Rossi	1800
Verdi	1800
Bianchi	2500
Neri	2500
Bruni	3500

$$R_2 (\underline{\text{Categoria}}, \text{Stipendio}) =$$

$$\pi_{\text{Categoria}, \text{Stipendio}} R$$

Categoria	Stipendio
2	1800
3	1800
4	2500
5	2500
6	3500

E ricomponendo:

$$R_1 \bowtie R_2$$

Impiegato	Categoria	Stipendio
Rossi	2	1800
Rossi	3	1800
Verdi	2	1800
Verdi	3	1800
Bianchi	4	2500
...

tuple
"spurie"

Si ottiene una **perdita di informazione**, facendo il join su un attributo comune.

La decomposizione di una relazione su due insiemi di attributi X_1 e X_2 si dice senza perdita di informazione se il join delle proiezioni di r su X_1 e X_2 è uguale ad R stessa.

Data la relazione $r(X)$ e gli insiemi di attributi X_1 e X_2 tali che

$$X = X_1 \cup X_2$$

$$X_0 X_1 \cap X_2$$

Se r soddisfa la dipendenza funzionale

$$X_0 \rightarrow X_1 \text{ oppure } X_1 \rightarrow X_2$$

La decomposizione di r su X_1 e X_2 è senza perdita

Gli attributi formano una chiave per almeno una delle relazioni decomposte.

Esempio (perdita di informazione):

$$R_1(\underline{\text{Impiegato}}, \text{Stipendio})$$

$$R_2(\underline{\text{Categoria}}, \text{Stipendio})$$

$$X_1 = \text{Impiegato}, \text{Stipendio}$$

$$X_2 = \text{Categoria}, \text{Stipendio}$$

$$X_0 = \text{Stipendio}$$

L'attributo stipendio non soddisfa la decomposizione senza perdita.

Esempio 2:

$R(\underline{\text{Impiegato}}, \underline{\text{Categoria}}, \text{Stipendio})$

Decomposizione in:

$\text{Impiegato} \rightarrow \text{Categoria}$

$\text{Impiegato} \rightarrow \text{Stipendio}$

$$R_1(\underline{\text{Impiegato}}, \underline{\text{Categoria}}) = \\ \pi_{\text{Impiegato}, \text{Categoria}} R$$

Impiegato	Categoria
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

$$R_2(\underline{\text{Impiegato}}, \text{Stipendio}) = \\ \pi_{\text{Impiegato}, \text{Stipendio}} R$$

Impiegato	Stipendio
Rossi	1800
Verdi	1800
Bianchi	2500
Neri	2500
Bruni	3500

L'attributo **impiegato** soddisfa la condizione per la decomposizione senza perdita.

Se proviamo ad inserire la tupla (Gialli, 3500) in R_2 :

- Nella relazione originaria l'inserimento è vietato poiché causa la violazione della dipendenza $\text{Categoria} \rightarrow \text{Stipendio}$
- Nella decomposizione non è possibile riconoscere alcuna violazione, poiché Categoria e Stipendio sono in relazione separate.

Si perde la dipendenza tra Categoria e Stipendio .

Esempio 3:

$R(\underline{\text{Impiegato}}, \underline{\text{Categoria}}, \text{Stipendio})$

Decomposizione basata sulle dipendenze funzionali

$\text{Impiegato} \rightarrow \text{Categoria}$

$\text{Categoria} \rightarrow \text{Stipendio}$

Decomponendo:

$$R_1(\underline{\text{Impiegato}}, \underline{\text{Categoria}}) = \\ \pi_{\text{Impiegato}, \text{Categoria}} R$$

Impiegato	Categoria
Rossi	2
Verdi	3
Bianchi	4
Neri	4
Bruni	5

$$R_2(\underline{\text{Categoria}}, \text{Stipendio}) = \\ \pi_{\text{Categoria}, \text{Stipendio}} R$$

Categoria	Stipendio
2	1800
3	1800
4	2500
5	2500
6	3500

Ricomponendo:

l'attributo **categoria** soddisfa la condizione per la decomposizione **senza perdita**.

Quindi le dipendenze funzionali sono conservate

$\text{Impiegato} \rightarrow \text{Categoria}$

$\text{Categoria} \rightarrow \text{Stipendio}$

La dipendenza funzionale

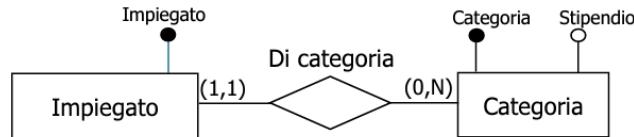
$$\text{Impiegato} \rightarrow \text{Stipendio}$$

Può esser ricostruita mediante:

$$\text{Impiegato} \rightarrow \text{Categoria}$$

$$\text{Categoria} \rightarrow \text{Stipendio}$$

Lo schema logico relazionale è:



Impiegato (Impiegato, Categoria)

Categoria (Categoria, Stipendio)

In generale, le decomposizioni devono sempre soddisfare le proprietà

- Decomposizione senza perdita: garantisce che le info nella relazione originaria siano ricostruibili con precisione a partire da quelle nelle relazioni decomposte;
- Garantisce che le relazioni decomposte abbiano la stessa capacità della relazione originaria di rappresentare i vincoli di integrità.

8) ALGEBRA RELAZIONALE

Estende l'algebra degli insiemi per il modello relazionale, definendo operatori che operano su relazioni e producono **come risultato una relazione**. Gode della proprietà di chiusura: il risultato di qualunque operazione algebrica su relazioni è a sua volta una relazione. È un **linguaggio procedurale** usato dal DBMS per estrarre e combinare le soluzioni.

Operatori algebra relazionale:

- Operatori unari
 - selezione (σ)
 - proiezione (π)
- Operatori binari
 - prodotto cartesiano (\times)
 - join (\bowtie)
 - unione (\cup)
 - intersezione (\cap)
 - differenza (-)
 - divisione (/)
- Operatori insiemistici
 - unione (\cup)
 - intersezione (\cap)
 - differenza (-)
 - prodotto cartesiano (\times)
- Operatori relazionali
 - selezione (σ)
 - proiezione (π)
 - join (\bowtie)
 - divisione (/)

Selezione.

Estrae un sottoinsieme “orizzontale” della relazione.

$$R = \sigma_p A$$

Genera una **relazione R** avente:

- Stesso schema di A
- Tuple di A per cui è vero il **predicato p**

N.B. : il predicato p è un'espressione booleana di espressioni di confronto tra attributi o attributi e costanti.

Proiezione.

La proiezione estrae un sottoinsieme “verticale” della relazione.

$$R = \pi_L A$$

L'operatore proiezione genera una **relazione R**:

- Avente come **schema la lista di attributi L** (Sottoinsieme dello schema di A)
- Contenente **tutte le tuple presenti in A**

Sono **eliminati gli eventuali duplicati** dovuti all'esclusione degli attributi non in L (se L include una chiave candidata, non ci sono duplicati).

Prodotto cartesiano.

Il prodotto cartesiano di due relazioni A e B genera tutte le coppie formate da una tupla di A ed una di B.

$$R = A \times B$$

Il prodotto cartesiano di due relazioni A e B genera una relazione R

- Avente come schema l'unione degli schemi di A e B
- Contenente tutte le coppie formate da una tupla di A ed una di B

Il prodotto cartesiano è **commutativo** ed **associativo**.

Join.

Il Join genera tutte le coppie formate da una tupla di A ed una tupla di B semanticamente legati. Nel DBMS ci sono diversi modi per farlo, è in genere un'operazione costosa.

Esempio:

R	Corsi. Codice	Corsi. NomeCorso	Corsi. Semestre	Corsi. MatrDocente	Docenti. MatrDocente	Docenti. NomeDoc	Docenti. Dipartimento
	M2170	Informatica 1	1	D102	D102	Verdi	Informatica
	M2170	Informatica 1	1	D102	D105	Neri	Informatica
	M2170	Informatica 1	1	D102	D104	Bianchi	Elettronica
	M4880	Sistemi digitali	2	D104	D102	Verdi	Informatica
	M4880	Sistemi digitali	2	D104	D105	Neri	Informatica
	M4880	Sistemi digitali	2	D104	D104	Bianchi	Elettronica
	F1401	Elettronica	1	D104	D102	Verdi	Informatica
	F1401	Elettronica	1	D104	D105	Neri	Informatica
	F1401	Elettronica	1	D104	D104	Bianchi	Elettronica
	F0410	Basi di dati	2	D102	D102	Verdi	Informatica
	F0410	Basi di dati	2	D102	D105	Neri	Informatica
	F0410	Basi di dati	2	D102	D104	Bianchi	Elettronica

È un operatore derivato, esprimibile utilizzando gli operatori di selezione, proiezione e prodotto cartesiano. Il join è definito separatamente perché esprime sinteticamente molte operazioni ricorrenti nelle interrogazioni. Esistono diversi tipi di join:

- Natural join
- Theta-join (e l'equi-join)
- Semi-join

Natural-join.

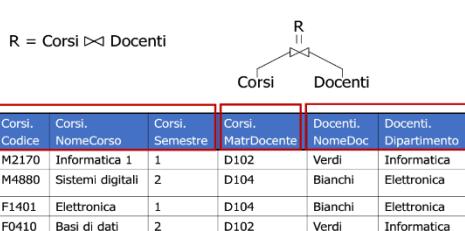
$$R = A \bowtie B$$

Il natural join di due relazioni A e B genera una relazione R avente come schema:

- Gli attributi presenti nello schema di A e non presenti in B
- Gli attributi presenti nello schema di B e non presenti in A
- Una sola copia degli attributi comuni, presenti sia in A ed in B
- Contenente tutte le coppie costituite da una tupla di A ed una di B per cui il valore degli attributi comuni è uguale.

È commutativo ed associativo.

Esempio:



Theta-join.

$$R = A \bowtie_p B$$

Il theta-join di due relazioni A e B genera tutte le coppie formate da una tupla di A ed una di B che soddisfano una generica “condizione di legame” (molto più generico del natural join).

Il theta-join di due relazioni A e B genera una relazione R

- Avente come schema l'**unione degli schemi di A e di B**
- Contenente tutte le coppie costituite da una tupla di A ed una di B per cui è vero il **predicato p**.

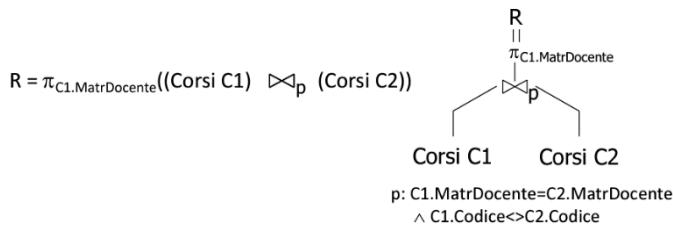
Il predicato p è nella forma $X \Theta Y$

- X attributo di A, Y attributo di B
- Θ è operatore di confronto compatibile con i domini di X ed Y

Il theta-join è **commutativo ed associativo**.

Esempio:

Trovare la matricola dei docenti che sono titolari di almeno due corsi.



Equi-join.

$$R = A \bowtie_p B$$

È un caso particolare del theta-join in cui Θ è l'operatore di uguaglianza (=).

Semi-join.

$$R = A \bowtie_p B$$

Il semi-join di due relazioni A e B seleziona tutte le tuple di A **semanticamente legate** ad almeno una tupla di B (le informazioni non compaiono nel risultato).

Il semi-join di due relazioni A e B genera una relazione R

- Avente lo stesso schema di A
- Contenete tutte le tuple di A per cui è vero il predicato specificato da **p**

Il predicato p è espresso nella stessa forma del theta-join (confronto tra attributi di A e B).

Può esser espresso anche in funzione del theta-join:

$$A \bowtie_p B = \pi_{\text{schema}(A)}(A \bowtie_p B)$$

esso **non gode** della proprietà commutativa!

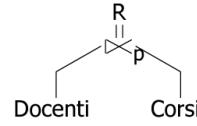
(Il semi-join si comporta come un doppio ciclo for)

Esempio:

trovare le informazioni relative ai docenti titolari di almeno un corso

$R = \text{Docenti} \bowtie_p \text{Corsi}$
 $p: \text{Docenti.MatrDocente} = \text{Corsi.MatrDocente}$

R	Docenti. MatrDocente	Docenti. NomeDoc	Docenti. Dipartimento
D102	Verdi	Informatica	
D104	Bianchi	Elettronica	



Outer join.

È una variante del join che permette di conservare l'informazione relativa alle tuple non semanticamente legate da predicato di join, completate con valori nulli per le tipe prive di controparte.

Ne esistono 3 tipi:

1. **Left**: sono completate le tuple del primo operando
2. **Right**: sono completate le tuple del secondo operando
3. **Full**: sono completate le tuple di entrambi gli operandi

Left outer-join.

Il left outer-join di due relazioni A e B generano le coppie formate da:

- Una tupla di A ed una di B **semanticamente legate**
- Una tupla di **A non semanticamente legata** a tuple di B completata con valori nulli per tutti gli attributi di B

$$R = A \bowtie_p B$$

Il left outer-join di due relazioni A e B genera una relazione R avente come schema:

- Una tupla di A ed una di B per cui è vero **p**
- Una tupla di A che non è correlata mediante il predicato **p** a tuple di B completata con valori nulli per tutti gli attributi di B

Il **left outer-join non è commutativo**.

Esempio:

trovare informazioni sui docenti e sui corsi che tengono

$R = \text{Docenti} \bowtie_p \text{Corsi}$
 $p: \text{Docenti.MatrDocente} = \text{Corsi.MatrDocente}$

R	Docenti. MatrDocente	Docenti. NomeDoc	Docenti. Dipartimento	Corsi. Codice	Corsi. NomeCorso	Corsi. Semestre	Corsi. MatrDocente
D102	Verdi	Informatica	M2170	Informatica 1	1	D102	
D102	Verdi	Informatica	F0410	Basi di dati	2	D102	
D104	Bianchi	Elettronica	M4680	Sistemi digitali	2	D104	
D104	Bianchi	Elettronica	F1401	Elettronica	1	D104	
D105	Neri	Informatica	null	null	null	null	

Right outer-join.

$$R = A \bowtie_p B$$

(speculare al precedente)

Il right outer-join di due relazioni A e B genera una relazione R avente come schema:

- Una tupla di A ed una di B per cui è vero p
- Una tupla di B che non è correlata mediante il predicato p a tuple di A completata con valori nulli per tutti gli attributi di A

Il **right outer-join** non è commutativo.

Full outer-join.

$$R = A \bowtie_p B$$

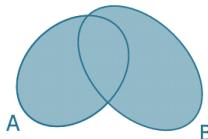
Genera una relazione R avente come schema l'unione degli schemi di A e di B, contenente coppie formate da:

- Una tupla di A ed una di B per cui è vero p
- Una tupla di A che non è correlata mediante il predicato p a tuple di B completata con valori nulli per tutti gli attributi di B
- Una tupla di B che non è correlata mediante il predicato p di tuple di A completata con valori nulli per tutti gli attributi di A

Il full outer-join è commutativo.

Unione.

L'unione di A e B seleziona tutte le tuple presenti in almeno una delle due relazioni.



Esempio:

DocentiLaurea		
MatrDocente	NomeDoc	Dipartimento
D102	Verdi	Informatica
D105	Neri	Informatica
D104	Bianchi	Elettronica

DocentiMaster		
MatrDocente	NomeDoc	Dipartimento
D102	Verdi	Informatica
D101	Rossi	Elettrica

→

R		
MatrDocente	NomeDoc	Dipartimento
D102	Verdi	Informatica
D105	Neri	Informatica
D104	Bianchi	Elettronica
D101	Rossi	Elettrica

N.B. : i duplicati sono eliminati.

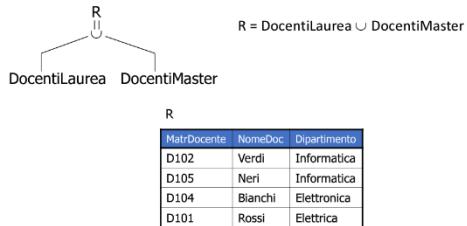
$$R = A \cup B$$

L'unione di due relazioni di A e B genera una relazione R avente:

- Avente stesso schema di A e B
- Contenente tutte le tuple appartenenti ad A e tutte le tuple appartenenti a B

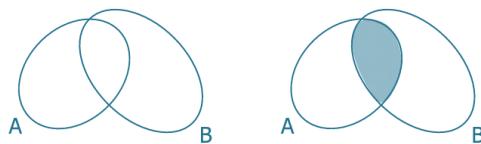
Deve esserci la **compatibilità** (A e B) **devono avere lo stesso schema**.

Le tuple duplicate sono eliminate. L'unione è associativa e commutativa.



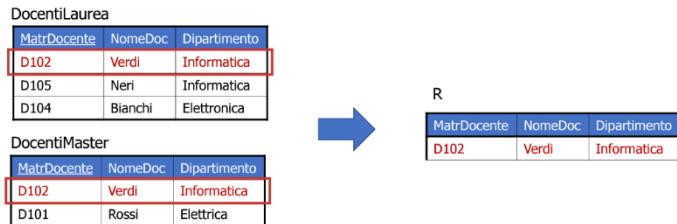
Intersezione.

L'intersezione di due relazioni A e B seleziona tutte le tuple presenti sia in A che in B.



Esempio:

trovare le informazioni relative ai docenti sia di corsi di laurea, sia di master



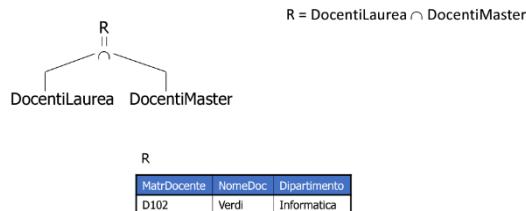
$$R = A \cap B$$

L'intersezione di due relazioni A e B genera una relazione R che ha come schema:

- Lo stesso schema di A e B
- Contenente le tuple sia in A sia in B

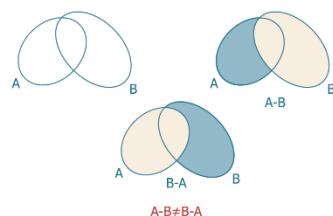
È inoltre richiesta la **compatibilità**, A e B devono avere uguale schema.

È commutativa ed associativa.



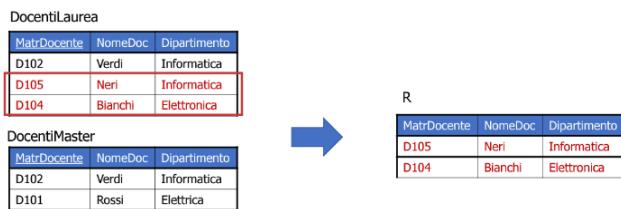
Differenza.

La differenza di due relazioni A e B seleziona tutte le tuple presenti **esclusivamente** in A e non in B. Occorre capire quando eseguire la differenza).



Esempio 1:

trovare i docenti di corsi di laurea ma non di master



dall'insieme voglio escludere i docenti che sono docenti anche dei corsi di master.

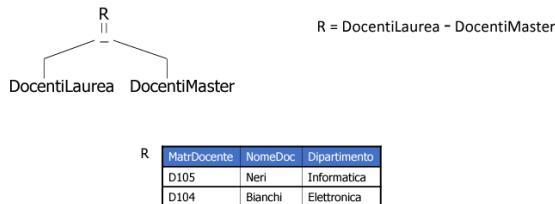
$$R = A - B$$

La differenza di due relazioni A e B genera una relazione R avente come schema:

- Stesso schema di A e di B
- Contenente tutte le tuple appartenenti ad A che non appartengono a B

È richiesta la **compatibilità**, ovvero le relazioni A e B devono avere lo stesso schema.

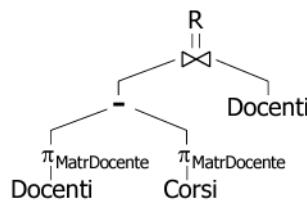
Non gode né della proprietà commutativa, né di quella associativa.



In tale modo si trovano i docenti che insegnano nei corsi di laurea ma non di magistrale.

Esempio 2:

Trovare Matricola, Nome e Dipartimento dei docenti che non tengono corsi.



Anti-join.

$$R = A \bowtie_p B$$

L'anti-join tra due relazioni A e B seleziona tutte le tuple di A **“semanticamente non legate”** a tuple di B.

L'anti-join di due relazioni A e B genera una relazione R avente:

- Avente lo stesso schema di A
- Contenente tutte le tuple di A per cui non esiste nessuna tupla in B per cui è vero il predicato p

Il **predicato p** è espresso nella stessa forma del theta-join e del semi-join ed è la **clausola di esclusione**, non ho bisogno di compatibilità (permette la differenza senza compatibilità).

L'anti-join **non gode** né della proprietà commutativa, né della proprietà associativa.

Esempio:

Trovare Matricola, Nome e Dipartimento dei docenti che non tengono corsi

Docenti

MatrDocente	NomeDoc	Dipartimento
D102	Verdi	Informatica
D105	Neri	Informatica
D104	Bianchi	Elettronica

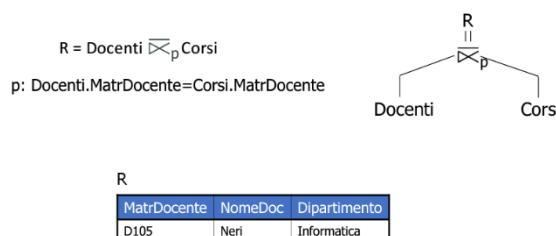
Corsi

Codice	NomeCorso	Semestre	MatrDocente
M2170	Informatica 1	1	D102
M4880	Sistemi digitali	2	D104
F1401	Elettronica	1	D104
F0410	Basi di dati	2	D102

R

MatrDocente	NomeDoc	Dipartimento
D105	Neri	Informatica

Inoltre



Divisione.

Esempio:

trovare gli studenti che hanno superato gli esami di **tutti** i corsi del primo anno.

EsamiSuperati

MatrStudente	CodCorso
S1	C1
S1	C2
S1	C3
S1	C4
S1	C5
S1	C6
S2	C1
S2	C2
S3	C2
S4	C2
S4	C4
S4	C5

CorsiPrimoAnno

CodCorso
C1
C2
C3
C4
C5
C6

R

MatrStudente
S1

$R = \text{EsamiSuperati} / \text{CorsiprimoAnno}$

$$R = A / B$$

La divisione della relazione A per la relazione B genera una relazione R avente come schema:

- **Schema (A) – Schema (B)**
- Contente tutte le tuple di A tali che per ogni tupla $(Y:y)$ presente in B esiste una tupla $(X:x, Y:y)$ in A, ovvero per tutte le tuple in B esiste la tupla in A ed è associata ai valori nell'operando divisore.

La divisione **non gode** né della proprietà commutativa, né di quella associativa.

9) FONDAMENTI SQL

È il linguaggio per gestire le basi di dati relazionali: Structured Query Language

Esso include istruzioni per:

- Definire lo schema di una base dati relazionale
- Leggere e scrivere i dati
- Definire lo schema di tabelle derivate
- Definire i privilegi di accesso degli utenti
- Gestire le transazioni

È utilizzabile in modalità

- **Interattiva**
- **Compilata**: un linguaggio host che contiene le istruzioni SQL, che si differenziano per mezzo di opportuni artifici sintattici

È un linguaggio definito a **livello di set**:

- Operatori operano su relazioni
- Il risultato è sempre una relazione

Il linguaggio SQL è **dichiarativo**:

- Si descrive **cosa fare** e non come fare
- Si pone ad un livello di astrazione superiore rispetto ai linguaggi di programmazione tradizionali

Può essere diviso in:

- **DML** (Data Manipulation Language) manipolazione dei dati
- **DDL** (Data Definition Language) definisce la struttura della base di dati

DML.

Usata per estrarre i dati di interesse attraverso la SELECT

Usata per modificare una base di dati:

INSERT: inserimento di nuove informazioni in una tabella

UPDATE: aggiornamento dati presenti nella base dati

DELETE: cancellazione di dati obsoleti

DDL.

Definizione dello schema di una base di dati:

- Creazione, modifica e cancellazione di tabelle: CREATE, ALTER, DROP TABLE

Definizione di tabelle derivate

- Creazione, modifica e cancellazione di tabelle il cui contenuto è ottenuto da altre tabelle della base di dati: CREATE, ALTER, DROP VIEW

Definizione di strutture dati accessorie per recuperare efficientemente i dati

- Creazione e cancellazione di indici: CREATE, DROP INDEX

Definizione dei privilegi di accesso degli utenti

- Concessione e revoca di privilegi sulle risorse: GRANT, REVOKE

Definizione di transazioni

- Terminazione di una transazione: COMMIT, ROLLBACK

Sintassi.

Grammatica

- Parentesi angolari <> : isolano un termine della sintassi
- Parentesi quadre [] : indicano termine all'interno opzionale
- Parentesi graffe {} : indicano che il termine racchiuso non può comparire o esser ripetuto un numero arbitrario di volte
- Barra verticale | indica che deve esser scelto uno tra i termini separati dalle barre

Select

Schema generale:

```
SELECT [DISTINCT] ElencoAttributiDaVisualizzare
FROM ElencoTabelleDaUtilizzare
[WHERE CondizioniDiTupla ]
[GROUP BY ElencoAttributiDiRaggruppamento ]
[HAVING CondizioniSuAggregati ]
[ORDER BY ElencoAttributiDiOrdinamento ]
```

Clausola WHERE.

Permette di esprimere condizioni di selezione espresse singolarmente ad ogni tupla attraverso espressione booleana di predicati.

Predicati semplici sono:

- Espressioni di confronto tra attributi e costanti
- Ricerca testuale
- Valori NULLI

Ricerca testuale.

Vi è l'operatore **LIKE**

NomeAttributo LIKE StringaDiCaratteri

- Il carattere _ rappresenta un singolo carattere qualsiasi (obbligatoriamente presente)
- Il carattere **%** rappresenta una sequenza qualsiasi di n caratteri (anche vuota)

Esempio 1:

trovare il codice ed il nome dei prodotti il cui nome inizia con la lettera 'C'

```
SELECT CodF
FROM F
WHERE Sede='Milano';
```

F → R

CodF	NomeF	NSoci	Sede
F1	Andrea	2	Torino
F2	Luca	1	Milano
F3	Antonio	3	Milano
F4	Gabriele	2	Torino
F5	Matteo	3	Venezia

CodF
F2
F3

Esempio 2:

- attributo contenente la stringa 'Torino'
Indirizzo LIKE %'Torino%'
- il codice fornitore è pari a 2 ed è preceduto da un carattere ignoto ed è costituito da 2 caratteri
CodF LIKE '_2'

Ricerca di valori NULL.

Operatore speciale IS

NomeAttributo IS [NOT] NULL

In presenza di valori NULL qualsiasi predicato di confronto è falso.

Esempio 1:

trovare il codice e nome dei prodotti per cui la taglia non è indicata

```
SELECT CodP, NomeP
FROM P
WHERE Taglia IS NULL;
```

P → R

CodP	NomeP	Colore	Taglia	Magazzino
P1	Maglia	Rosso	40	Torino
P2	Jeans	Verde	48	Milano
P3	Camicia	Blu	48	Roma
P4	Camicia	Blu	44	Torino
P5	Gonna	Blu	NULL	Milano
P6	Bermuda	Rosso	42	Torino

CodP	NomeP
P5	Gonna

Esempio 2:

trovare codice e nome dei prodotti con taglia maggiore di 44 o che potrebbero avere taglia maggiore di 44.

```
SELECT CodP, NomeP
FROM P
WHERE Taglia>44 OR Taglia IS NULL;
```

P → R

CodP	NomeP	Colore	Taglia	Magazzino
P1	Maglia	Rosso	40	Torino
P2	Jeans	Verde	48	Milano
P3	Camicia	Blu	48	Roma
P4	Camicia	Blu	44	Torino
P5	Gonna	Blu	NULL	Milano
P6	Bermuda	Rosso	42	Torino

CodP	NomeP
P2	Jeans
P3	Camicia
P5	Gonna

Ordinamento risultato.

Si usa la clausola ORDER BY

ORDER BY NomeAttributo [ASC | DESC]

{,NomeAttributo [ASC | DESC]}

L'ordinamento può essere crescente (ASC) o decrescente (DESC), quello implicito è crescente (ASC), inoltre gli attributi di ordinamento devono comparire nella clausola SELECT.

Esempio 1:

trovare codice prodotti e la loro taglia ordinando il risultato in ordine di taglia

```
SELECT CodP, Taglia
FROM P
ORDER BY Taglia DESC;
```

P

CodP	NomeP	Colore	Taglia	Magazzino
P1	Maglia	Rosso	40	Torino
P2	Jeans	Verde	48	Milano
P3	Camicia	Blu	48	Roma
P4	Camicia	Blu	44	Torino
P5	Gonna	Blu	40	Milano
P6	Bermuda	Rosso	42	Torino

R

CodP	Taglia
P2	48
P3	48
P4	44
P6	42
P1	40
P5	40

Esempio 2:

trovare le informazioni sui prodotti ordinando il risultato in ordine crescente di nome e decrescente di taglia.

```
SELECT CodP, NomeP, Colore, Taglia, Magazzino
FROM P
ORDER BY NomeP, Taglia DESC;
```

```
SELECT *
FROM P
ORDER BY NomeP, Taglia DESC;
```

R

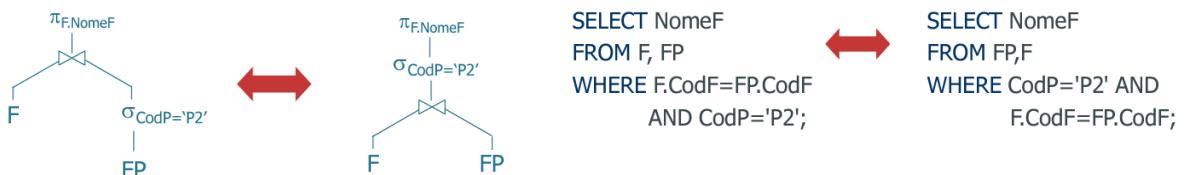
CodP	NomeP	Colore	Taglia	Magazzino
P6	Bermuda	Rosso	42	Torino
P3	Camicia	Blu	48	Roma
P4	Camicia	Blu	44	Torino
P5	Gonna	Blu	40	Milano
P2	Jeans	Verde	48	Milano
P1	Maglia	Rosso	40	Torino

Join.

Definito mediante le clausole **FROM** e **WHERE**. Il risultato e l'efficienza dell'interrogazione, indipendenti dall'ordine delle tabelle nella clausola FORM e nella clausola WHERE. L'ordine di esecuzione ottimale è selezionato dal DBMS, inoltre FROM con N tabelle → N-1 condizioni di join nella clausola WHERE.

Esempio 1:

trovare il nome dei fornitori che forniscono il prodotto P2



Esempio 2:

trovare nome dei fornitori che forniscono almeno un prodotto rosso

```

SELECT NomeF
FROM P INNER JOIN FP ON P.CodP=FP.CodP
    INNER JOIN F ON F.CodF=FP.CodF
WHERE P.Colore='Rosso';

```

Funzione aggregata.

Opera su un insieme di valori producendo un unico valore. È indicata nella clausola SELECT, non si possono indicare anche attributi non aggregati, possono essere richieste più funzioni aggregate contemporaneamente. Esse sono valutate solo dopo l'applicazione di tutti i predicati nella clausola WHERE.

COUNT: conteggio degli elementi in un attributo

SUM: somma dei valori di un attributo

AVG: media dei valori di un attributo

MAX: massimo valore di un attributo

MIN: minimo valore di un attributo

Count.

Conteggia il numero di elementi di un insieme, ovvero le righe di una tabella con i valori eventualmente distinti di uno o più attributi.

COUNT (<*| [DISTINCT | ALL] ListaAttributi >)

Se l'argomento della funzione è preceduto da DISTINCT, conta il numero di valori distinti dell'argomento.

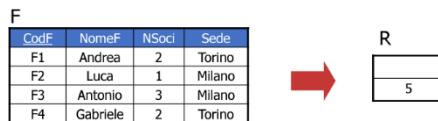
Esempio 1:

trovare numero di fornitori che hanno almeno una fornitura

```

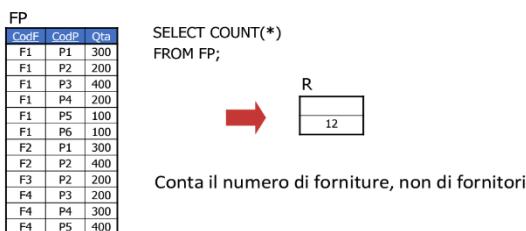
SELECT COUNT(*)
FROM F;

```



Esempio 2:

trovare il numero di fornitori che hanno almeno una fornitura



Sum, Max, Min e AVG.

Ammettono come argomento un attributo o un'espressione

Sum e AVG.

Ammettono come argomento solo attributi di tipo numerico o intervallo di tempo

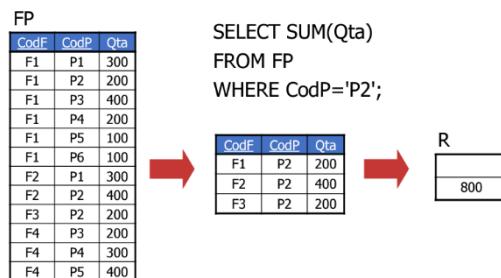
Max e Min.

Richiedono che l'espressione sia ordinabile, possono essere applicate anche su stringhe di caratteri ed istanti di tempo.

Sum.

Esempio :

trovare la quantità totale di pezzi forniti per il prodotto P2



GROUP BY.

È un **partizionamento** di relazione in gruppi di sottoinsiemi disgiunti.

Clausola di raggruppamento

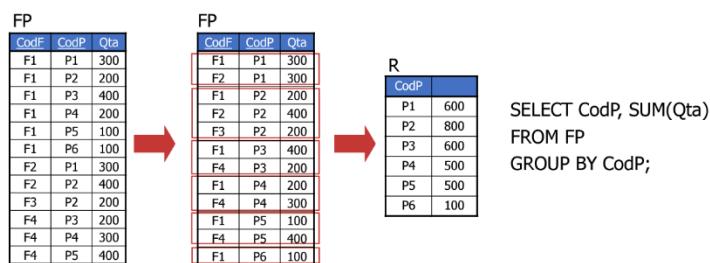
GROUP BY ElencoAttributiDiRaggruppamento

Dove l'ordine degli attributi di raggruppamento è ininfluente.

Nella clausola SELECT possono comparire **solo** attributi presenti nella clausola GROUP BY e funzioni aggregate. Gli attributi univocamente determinati da attributi già presenti nella clausola GROUP BY possono esser aggiunti senza alterare il risultato.

Esempio:

per ogni prodotto, trovare la quantità totale di pezzi forniti



Esempio:

per ogni prodotto, trovare la quantità totale di pezzi forniti da fornitori con sede a Milano.

```
SELECT CodP, SUM(Qta)
FROM FP, F
WHERE FP.CodF=F.CodF AND Sede='Milano'
GROUP BY CodP;
```

Esempio:

per ogni prodotto, trovare il codice, **il nome** e la quantità totale fornita

```
SELECT P.CodP, NomeP, SUM(Qta)
FROM P, FP
WHERE P.CodP=FP.CodP
GROUP BY P.CodP, NomeP
```

Gli attributi univocamente determinati da attributi già presenti nella clausola GROUP BY possono esser aggiunti **senza alterare il risultato**. Ciò che deve esser visualizzato, deve essere nellla GROUP BY e devo controllare la correttezza del gruppo.

Having.

Non è possibile usare la clausola Where per definire condizioni di selezione sui gruppi.

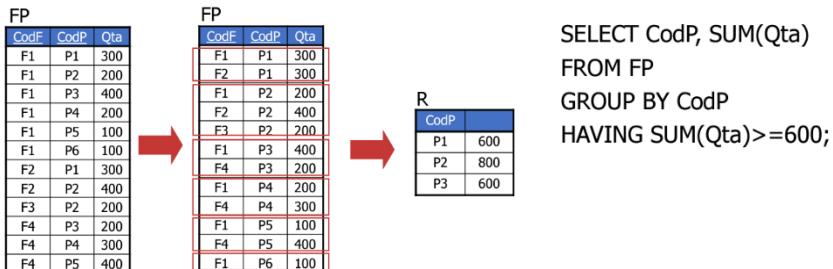
Le condizioni di selezione sui gruppi sono espresse con la clausola HAVING

HAVING Condizioni di gruppo

Permette di specificare condizioni **solo** su funzioni aggregate

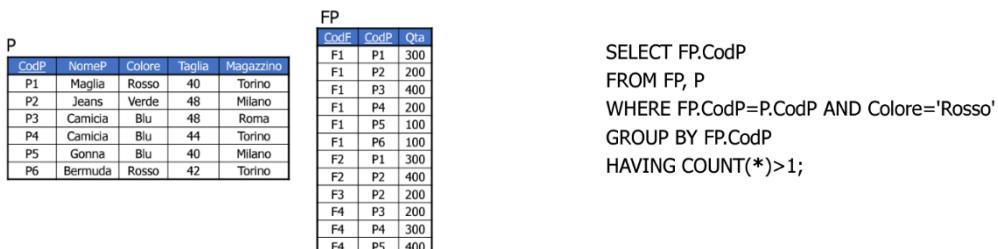
Esempio 1:

trovare la quantità totale di pezzi forniti per i prodotti per cui sono forniti **in totale** almeno 600 pezzi



Esempio 2:

trovare il codice dei prodotti rossi forniti da più di un fornitore



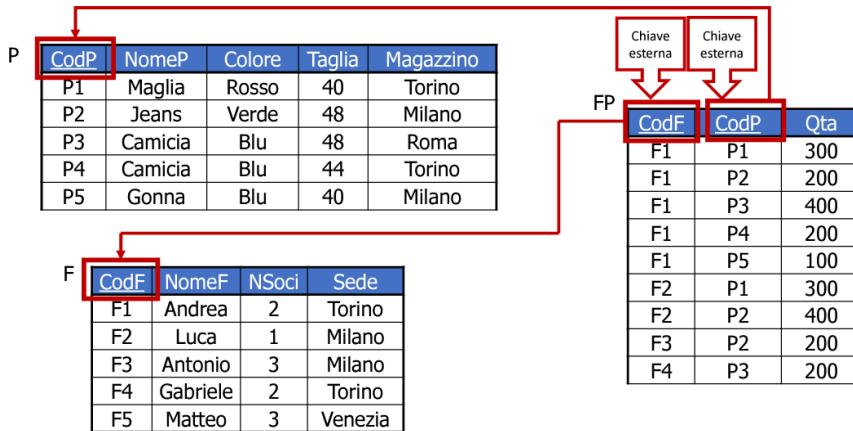
10) INTERROGAZIONI NIDIFICATE

Un'interrogazione nidificata è un'istruzione SELECT contenuta dentro un'altra interrogazione, la nidificazione consente di suddividere un problema complesso in sottoproblemi più semplici.

È possibile introdurre istruzioni **SELECT**

- In un predicato nella clausola **WHERE**
- In un predicato nella clausola **HAVING**
- Nella clausola **FROM**

Schema di riferimento:



Esempio 1:

trovare il codice dei fornitori che hanno sede nella stessa città di F1

Codici dei fornitori con sede nella stessa città di S1

{

 SELECT CodF

 FROM F

 WHERE Sede = (SELECT Sede

 FROM F

 WHERE CodF='F1');

Città del fornitore di S1

N.B.: è possibile utilizzare '=' esclusivamente se è noto a priori che il risultato della SELECT nidificata è sempre **un solo valore**. È possibile una formulazione equivalente con il join.

```

SELECT FY.CodF
FROM F AS FX, F AS FY
WHERE FX.Sede=FY.Sede AND
FX.CodF='F1';
  
```

Esempio 2:

trovare il codice dei fornitori il cui numero di soci è minore del numero massimo dei soci

```

SELECT CodF
FROM F
WHERE NSoci < (SELECT MAX(NSoci)
FROM F);
  
```

non è possibile una formulazione alternativa equivalente con il join

Operatore IN.

Esprime il concetto di appartenenza ad un insieme di valori

NomeAttributo **IN** (InterrogazioneNidificata)

Permette di scrivere l'interrogazione scomponendo il problema in sottoproblemi seguendo un procedimento “bottom-up”. L'interrogazione nidificata può esser sostituita con una lista di valori.

Esempio 1:

trovare il nome dei fornitori che forniscono il prodotto P2

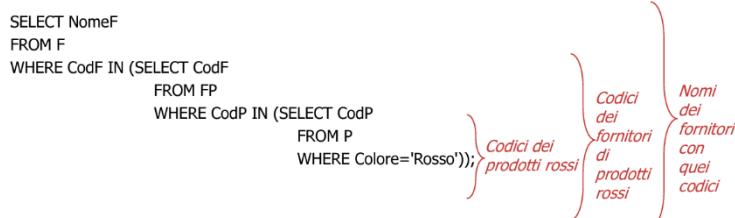
IN	JOIN
<pre>SELECT NomeF FROM F WHERE CodF IN (SELECT CodF FROM FP WHERE CodP='P2');</pre>	<pre>SELECT NomeF FROM F, FP WHERE F.CodF=FP.CodF AND CodP='P2';</pre>

Esempio 2:

trovare il nome dei fornitori che forniscono almeno un prodotto rosso

scomponendo in sottoproblemi:

- 1) Codici dei prodotti rossi
- 2) Codici dei fornitori di quei prodotti
- 3) Nomi dei fornitori aventi quei codici



Invece le formulazioni equivalenti sono:

IN	JOIN
<pre>SELECT NomeF FROM F WHERE CodF IN (SELECT CodF FROM FP WHERE CodP IN (SELECT CodP FROM P WHERE Colore='Rosso'));</pre>	<pre>SELECT NomeF FROM F, FP, P WHERE FP.CodF=F.CodF AND FP.CodP=P.CodP AND Colore='Rosso' ③</pre>

Operatore NOT IN.

Viene usato per la DIFFERENZA, esprime il concetto di esclusione da un insieme di valori

NomeAttributo **NOT IN** (InterrogazioneNidificata)

Richiedere di individuare in modo appropriato **l'insieme da escludere** definito da un'interrogazione nidificata ed una lista di valori.

N.B.: non esiste una formulazione equivalente con il join.

Esempio 1:

trovare il nome dei fornitori che **non** forniscono il prodotto P2

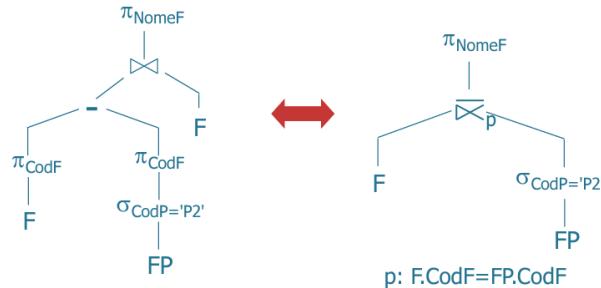
```
SELECT NomeF  
FROM F  
WHERE CodF NOT IN (SELECT CodF  
                     FROM FP  
                     WHERE CodP='P2');
```

Non appartiene

*Codici dei fornitori
che forniscono P2*

Dove il codice dei fornitori che forniscono P2 è l'insieme da escludere.

Similmente in algebra relazionale:



Esempio 2:

trovare il nome dei fornitori che forniscono **solo** il prodotto P2

quindi occorre trovare il nome dei fornitori di P2 che non hanno mai fornito prodotti diversi da P2

l'insieme da escludere → fornitori di prodotti diversi da P2

```
SELECT NomeF  
FROM F, FP  
WHERE F.CodF NOT IN (SELECT F.CodF  
                      FROM FP  
                      WHERE CodP<>'P2')  
      AND F.CodF=FP.CodF;
```

*Codici dei fornitori
che forniscono
almeno un
prodotto diverso
da P2*

Una possibile soluzione alternativa:

```
SELECT NomeF  
FROM F  
WHERE F.CodF NOT IN (SELECT CodF  
                      FROM FP  
                      WHERE CodP<>'P2')  
      AND F.CodF IN (SELECT CodF  
                      FROM FP);
```

*Codici dei fornitori
che forniscono
almeno un
prodotto diverso
da P2*

Esempio 3:

trovare il nome dei fornitori che **non** forniscono prodotti rossi

insieme da escludere → fornitori prodotti rossi (identificati dal loro codice)

```
SELECT NomeF  
FROM F  
WHERE CodF NOT IN (SELECT CodF  
                     FROM FP  
                     WHERE CodP IN (SELECT CodP  
                                    FROM P  
                                    WHERE Colore='Rosso'));
```

*Codici dei fornitori
di prodotti rossi*

Costruttore di tupla.

Permette di definire la struttura temporanea di una tupla: si elencano gli attributi che ne fanno parte tra ()

(NomeAttributo1,NomeAttributo2,...)

Consente di estendere il potere espressivo degli operatori **IN** e **NOT IN**

Esempio 1:

trovare le coppie luogo di partenza e di arrivo per cui nessun viaggio dura più di 2 ore

VIAGGIO (CodV,LuogoPartenza,LuogoArrivo,OraPartenza,OraArrivo)

```

SELECT LuogoPartenza, LuogoArrivo
FROM VIAGGIO
WHERE (LuogoPartenza, LuogoArrivo) NOT IN
      (SELECT LuogoPartenza, LuogoArrivo
       FROM VIAGGIO
       WHERE OraArrivo-OraPartenza>2);
  
```

*Costruttore
di tupla*

Operatore EXISTS.

L'operatore **EXISTS** ammette come **parametro una interrogazione nidificata** e restituisce:

- Il valore **vero** solo se l'interrogazione nidificata fornisce **un insieme non vuoto**
- Il valore **falso** se l'interrogazione interna restituisce **l'insieme vuoto**

Nell'interrogazione interna a **EXISTS**, la clausola **SELECT** è obbligatoria, ma irrilevante in quanto gli attributi non sono visualizzati. La **condizione di correlazione** lega l'esecuzione dell'interrogazione interna al valore di attributi della tupla corrente nell'interrogazione esterna.

Esempio:

trovare il nome dei fornitori del prodotto P2 (trovare il nome dei fornitori **per cui esiste** una fornitura del prodotto P2)

```

SELECT NomeF
FROM F
WHERE EXISTS (SELECT *
              FROM FP
              WHERE CodP='P2'
                AND FP.CodF=F.CodF);
  
```

Condizione di correlazione

Operatore NOT EXISTS.

L'operatore **NOT EXISTS** ammette come **parametro una interrogazione nidificata** e restituisce:

- Il valore **vero** solo se l'interrogazione nidificata fornisce **l'insieme vuoto** (ossia non restituisce nessuna tupla)
- Il valore **falso** se l'interrogazione interna restituisce **l'insieme non vuoto**

Nell'interrogazione interna a **NOT EXISTS**, la clausola **SELECT** è obbligatoria, ma irrilevante in quanto gli attributi non sono visualizzati. La **condizione di correlazione** lega l'esecuzione dell'interrogazione interna al valore di attributi della tupla corrente nell'interrogazione esterna.

Esempio:

trovare il nome dei fornitori che **non** forniscono il prodotto P2

```
SELECT NomeF  
FROM F  
WHERE NOT EXISTS (SELECT *  
                   FROM FP  
                   WHERE CodP='P2'  
                         AND FP.CodF=F.CodF );  
Condizione di correlazione
```

Correlazione tra interrogazioni.

Può esser necessario **legare la computazione di un'interrogazione nidificata** al valore di uno o più attributi in un'interrogazione più esterna: il legame è espresso da una o più condizioni di correlazione.

Una **condizione di correlazione**, indicata nella **clausola WHERE** dell'interrogazione nidificata che la richiede ed è un predicato che lega attributi di tabelle nella **FROM** dell'interrogazione nidificata con attributi di tabelle nella **FROM di interrogazioni più esterne**

Non si possono esprimere condizioni di correlazione in interrogazioni allo stesso livello di nidificazione, contenenti riferimenti ad attributi di una tabella nella **FROM** di un'interrogazione nidificata

Esempio 1:

per ogni prodotto, trovare il codice del fornitore che ne fornisce la quantità massima

```
SELECT CodP, CodF  
FROM FP AS FPX  
WHERE Qta = (SELECT MAX(Qta)  
             FROM FP AS FPY  
             WHERE FPY.CodP=FPX.CodP);  
Condizione di correlazione
```

Quantità massima per il prodotto corrente

Esempio 2:

trovare il codice dei viaggi che hanno una durata inferiore alla durata media dei viaggi sullo stesso percorso (stessa città di arrivo e partenza)

VIAGGIO (CodV, LuogoPartenza, LuogoArrivo,
OraPartenza, OraArrivo)

```
SELECT CodV  
FROM VIAGGIO AS VA  
WHERE OraArrivo-OraPartenza <  
      (SELECT AVG(OraArrivo-OraPartenza)  
       FROM VIAGGIO AS VB  
          Condizioni di correlazione  
       WHERE VB.LuogoPartenza=VA.LuogoPartenza  
             AND VB.LuogoArrivo=VA.LuogoArrivo);
```

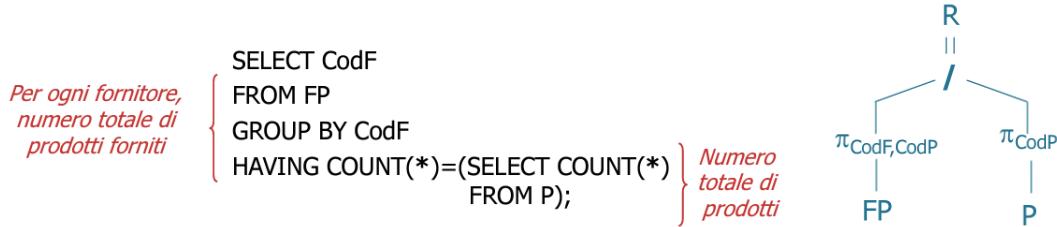
Durata media dei viaggi sul percorso corrente

Operazione di divisione.

In SQL l'operazione di divisione può esser realizzata attraverso l'operatore **COUNT**, per verificare che gli elementi di interesse appartengano **tutti** all'insieme di riferimento

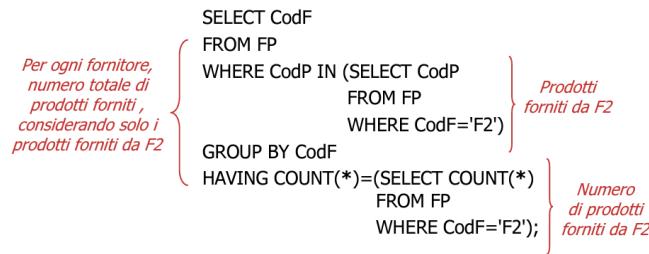
Esempio 1:

trovare il codice dei fornitori che forniscono **tutti** i prodotti, in algebra si utilizza l'operatore di divisione.



Esempio 2:

trovare il codice dei fornitori che forniscono almeno tutti i prodotti forniti dal fornitore F2, si esegue il conteggio del numero di prodotti forniti da F2, il conteggio del numero di prodotti forniti da un fornitore arbitrario ed anche da F2. I due conteggi devono esser uguali.



11. OPERATORI INSIEMISTICI

UNION.

È un operatore insiemistico di unione

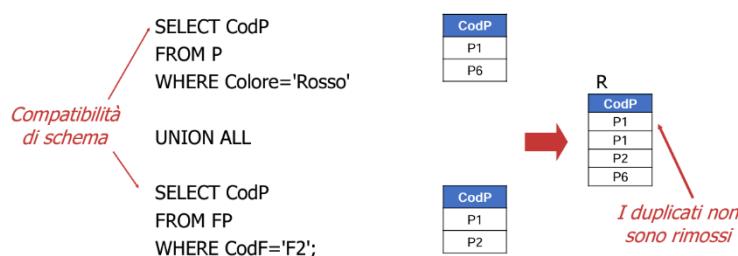
A UNION B

Esegue l'unione delle due espressioni relazionali A e B

- Le espressioni relazionali A e B possono esser generate da istruzioni SELECT
- Richiede la **compatibilità** di schema tra A e B
- Rimozione dei duplicati, UNION rimuove i duplicati, UNION ALL non rimuove i duplicati

Esempio:

trovare codice prodotti di colore rosso forniti dal fornitore F2



Intersect.

Operatore insiemistico di intersezione

A INTERSECT B

Esegue l'intersezione delle due espressioni relazionali A e B

- Le espressioni relazionali A e B possono esser generate da istruzioni SELECT
- Richiede la **compatibilità** di schema tra A e B

Esempio:

trovare le città che sono sia sede di fornitori, sia di magazzino di prodotti



Equivalenza con altri operatori.

L'operazione di intersezione può esser eseguita mediante l'operatore **JOIN** o **IN**.

JOIN	IN
<ul style="list-style-type: none">• La clausola FROM contiene le relazioni interessate dall'intersezione• La clausola WHERE contiene condizioni di join tra gli attributi presenti nella clausola SELECT delle espressioni relazionali A e B	<ul style="list-style-type: none">• Una delle due espressioni relazionali diviene un'interrogazione nidificata mediante l'operatore IN• Gli attributi nella clausola SELECT esterna, uniti da un costruttore di tupla, costituiscono la parte sinistra dell'operatore IN

Esempio 1 equivalenza con il JOIN:

trovare le città sia sedi di fornitori, sia magazzino di prodotti

```
SELECT Sede  
FROM F, P  
WHERE F.Sede=P.Magazzino;
```

Esempio 2 equivalenza con In:

trovare le città che sono sia sede di fornitori, sia magazzino di prodotti

```
SELECT Magazzino  
FROM P  
WHERE Magazzino IN (SELECT Sede  
FROM F);
```

Except.

Operatore insiemistico di differenza

A **EXCEPT** B

Sottrae l'espressione relazionale B all'espressione relazionale A, richiede la **compatibilità** di schema tra A e B.

Esempio:

trovare le città che sono sede di fornitori, ma non magazzino di prodotti

Equivalenza con l'operatore

```
SELECT Sede  
FROM F  
  
EXCEPT  
  
SELECT Magazzino  
FROM P;
```

Sede
Torino
Milano
Milano
Torino
Venezia

Magazzino
Torino
Milano
Roma
Torino
Milano
Torino

R
Venezia

NOT IN.

L'operatore di differenza può esser eseguito anche mediante l'operatore NOT IN

- Espressione relazionale B è nidificata all'interno dell'operatore NOT IN
- Gli attributi nella clausola SELECT dell'espressione relazionale A, uniti da un costruttore di tupla, costituiscono la parte sinistra dell'operatore NOT IN

Esempio:

trovare le città che sono sede di fornitori, ma non magazzino di prodotti

```
SELECT Sede  
FROM F  
WHERE Sede NOT IN (SELECT Magazzino  
FROM P);
```

12. QUERY AVANZATE

Tabelle derivate.

Definisce una tabella temporanea che può esser utilizzata per ulteriori operazioni di calcolo.

La tabella derivata ha

- Struttura di una **SELECT**
- È definita all'interno di una clausola **FROM**
- Può essere referenziata come una normale tabella

La tabella derivata permette di:

- Calcolare più livelli di aggregazione
- Formulare in modo equivalente le interrogazioni che richiedono la correlazione

Esempio 1:

trovare la media massima (conseguita da uno studente)

- 1) Trovare la media per ogni studente
- 2) Trovare il valore massimo della media

```
SELECT MAX(MediaStudente)
  FROM (SELECT Matricola, AVG(Voto) AS MediaStudente
          FROM ESAME-SUPERATO
         GROUP BY Matricola) AS MEDIE;
```

Tabella derivata

Esempio 2:

per ogni anno di iscrizione, trovare la media massima (conseguita da uno studente)

- 1) Trovare la media per ogni studente
- 2) Raggruppare gli studenti per anno di iscrizione e calcolarne la media massima

```
SELECT AnnoIscrizione, MAX(MediaStudente)
  FROM STUDENTE,
       (SELECT Matricola, AVG(Voto) AS MediaStudente
          FROM ESAME-SUPERATO
         GROUP BY Matricola) AS MEDIE
 WHERE STUDENTE.Matricola=MEDIE.Matricola
   GROUP BY AnnoIscrizione
```

*Condizione
di join*

Correlazione con tabella derivata.

Esempio:

per ogni prodotto, trovare il codice del fornitore che ne fornisce la quantità massima

F (CodF, NomeF, NSoci, Sede)
P (CodP, NomeP, Colore, Taglia, Magazzino)
FP (CodP, CodF, Qta)

- 1) Calcolare la qta max per ogni prodotto
- 2) Selezione su fornitori che forniscono la qta massima, prodotto per prodotto

```

SELECT CodP, CodF
FROM FP,
     (SELECT CodP, MAX(Qta) AS MQta
      FROM FP
      GROUP BY CodP) AS TMax
WHERE FP.CodP = TMax.CodP
AND FP.Qta = TMax.MQta;

```

Tabella derivata
Condizione di join
Correlazione

Common Table Expression CTE.

Definisce una tabella temporanea che può essere utilizzata per ulteriori operazioni di calcolo.

La CTE

- Ha la struttura di una **SELECT**
- È definita mediante la clausola **WITH**
- Può esser referenziata come una normale tabella

La CTE inoltre è usata per calcolare più livelli di aggregazione e formulare in modo equivalente le interrogazioni che richiedono la correlazione.

Vi sono riferimenti a CTE **precedentemente** definite nella stessa clausola WITH ed usabile ricorsivamente.

La CTE è preferibile quando:

- È necessario fare riferimento ad una tabella derivata più volte in una singola query
- È necessario eseguire lo stesso calcolo più volte in pi parti della stessa query
- Si vuole aumentare la leggibilità

Esempio 1:

trovare la media massima conseguita da uno studente

- 1) Trovare la media per ogni studente
- 2) Trovare il valore massimo della media (raggruppato per ogni studente)

```

WITH MEDIE AS
    (SELECT Matricola, AVG(Voto) AS MediaStudenti
     FROM ESAME-SUPERATO
     GROUP BY Matricola)
SELECT MAX(MediaStudenti)
     FROM MEDIE;

```

Esempio 2:

trovare tutte le compagnie aeree in cui il salario medio di tutti i piloti di quella compagnia è superiore alla media dei salari totali di tutti i piloti del database

- 1) Trovare il salario medio per ogni compagnia
- 2) Trovare il salario medio considerando tutti i piloti
- 3) Trovare le compagnie con salario medio maggiore del salario medio globale

PILOTI (CodP, Nome, Cognome, Compagnia, Salario)

```

WITH salarioMedioCompagnia AS
    (SELECT Compagnia, AVG(Salario) AS AvgSalComp
     FROM PILOTI
     GROUP BY Compagnia),
mediaSalario AS
    (SELECT AVG(Salario) AS MediaSal
     FROM PILOTI )
SELECT Compagnia
     FROM salarioMedioCompagnia, mediaSalario
     WHERE salarioMedioCompagnia.AvgSalComp >
           mediaSalario.MediaSal;

```

Esempio CTE REFERENZIATE:

considerando le distanze medie percorse per ciascuna città, calcolare la distanza massima percorsa per ciascuna provincia

CITTA (CodC, NomeC, Provincia)
AUTISTA (CodA, NomeA, Cognome, CodC)
CORSA_GIORNALIERA (Data, CodA, Importo, Distanza)

- 1) Calcolare la distanza percorsa per ogni città da ogni autista
- 2) Calcolare la distanza media per ogni città
- 3) Calcolare la distanza massima per provincia

```
WITH totDistanzaAutista AS
    (SELECT SUM(Distanza) AS distanzaTot, CG.CodA, CG.CodC, NomeC, Provincia
     FROM CORSA_GIORNALIERA CG, CITTA C, AUTISTA A
      WHERE CG.CodA=A.CodA AND A.CodC=C.CodC
      GROUP BY CG.CodA, CG.CodC, NomeC, Provincia),
distanzaMedia AS
    ( SELECT AVG(distanzaTot) AS avgDist, CodC, NomeC, Provincia
      FROM totDistanzaAutista
      GROUP BY CodC, NomeC, Provincia )
SELECT MAX(avgDist), Provincia
FROM distanzaMedia
GROUP BY Provincia
```

CTE ricorsive.

Per ciascun impiegato, trovare il boss e il livello nella gerarchia

IMPIEGATI (CodI, Nome, Cognome, BossID*)

<u>CodI</u>	Nome	Cognome	BossId*
1	Domenic	Leaver	5
2	Cleveland	Hewins	1
3	Kakalina	Atherton	7
4	Roxanna	Fairlie	NULL
5	Hermie	Comsty	4
6	Pooh	Goss	8
7	Faulkner	Challiss	5

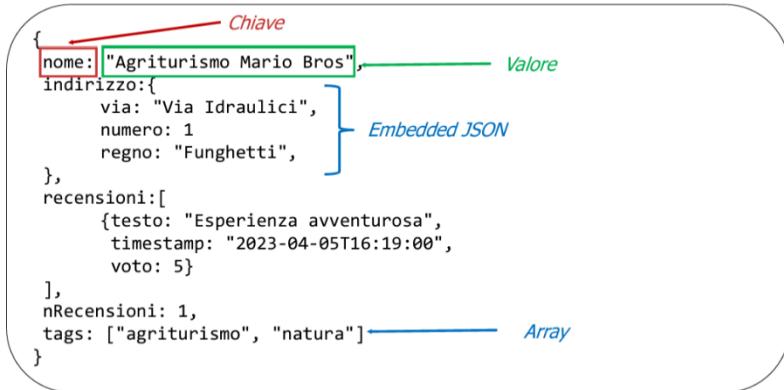
```
WITH RECURSIVE gerarchia AS (
  SELECT CodI, Nome, Cognome, BossID, 0 AS livello
  FROM IMPIEGATI
  WHERE BossID IS NULL
  UNION ALL
  SELECT I.CodI, I.Nome, I.Cognome, I.BossID, livello +1
  FROM IMPIEGATI I, gerarchia G
  WHERE I.BossID = G.CodI
)
SELECT G.Nome, G.Cognome, I.Nome AS NomeBoss, I.Cognome AS CognomeBoss, livello
FROM gerarchia G LEFT JOIN IMPIEGATI I ON G.BossID= I.CodI
ORDER BY livello;
```

Query JSON:

acronimo per JavaScript Object Notation, è un formato per lo scambio dei dati in applicazioni client-server. Funzioni dati JSON dipendono dal DBMS utilizzato, le funzioni dati JSON usate per:

- Creati dati in formato JSON
- Cercare all'interno del JSON in base al path fornito
- Modificare campi del JSON

Esempio:



- **JSON_ARRAY(target, candidate[, path])**
 - valuta un elenco di valori (eventualmente vuoto) e restituisce un array JSON contenente tali valori

```
SELECT JSON_ARRAY(1, "abc", NULL, TRUE, CURTIME()) AS RESULT;
```

RESULT
[1, "abc", null, true, "11:30:24.000000"]

- **JSON_OBJECT([key, val[, key, val] ...])**
 - valuta un elenco (eventualmente vuoto) di coppie chiave-valore e restituisce un oggetto JSON contenente tali coppie

```
SELECT JSON_OBJECT('id', 87, 'name', 'carrot') AS RESULT;
```

RESULT
{"id": 87, "name": "carrot"}

13. MANIPOLAZIONE DEI DATI

Le operazioni di aggiornamento modificano lo stato della base di dati, è necessario verificare che siano rispettati i vincoli di integrità.

Ogni istruzione può aggiornare il contenuto di una sola tabella, sono DML:

- **INSERT**: inserimento di nuove tuple in tabella
- **DELETE**: cancellazione di tuple da una tabella
- **UPDATE**: modifica del contenuto di tuple in una tabella

Insert.

Inserimento di una sola tupla, assegnando un valore costante ad ogni attributo

```
INSERT INTO NomeTabella  
[(ElencoColonne)]  
VALUES (ElencoCostanti);
```

Si possono inserire più tuple lette da altre tabelle mediante un'istruzione **SELECT**, senza contenere la clausola **ORDER BY**

```
INSERT INTO NomeTabella  
[(ElencoColonne)]  
Interrogazione;
```

Delete.

Cancella uno o più record della tabella ma non elimina una tabella.

```
DELETE FROM NomeTabella  
[WHERE predicato];
```

Effettua la cancellazione dalla tabella NomeTabella di tutte le tuple che soddisfano il predicato, ed occorre verificare che la cancellazione non causi la violazione di vincoli di integrità referenziale!

Update.

```
UPDATE NomeTabella  
SET colonna = espressione  
{,colonna = espressione}  
[WHERE predicato];
```

Tutti i record della tabella NomeTabella che soddisfano il predicato nella clausola WHERE sono modificati in base alle assegnazioni colonna=espressione nella clausola SET.

14. GESTIONE TABELLE

Creazione tabella.

```
CREATE TABLE NomeTabella  
  (NomeAttributo Dominio [ValoreDiDefault] [Vincoli]  
   {,NomeAttributo Dominio [ValoreDiDefault] [Vincoli]}  
   AltriVincoli  
 );
```

Permette di definire gli attributi (le colonne) della tabella e definire i vincoli di integrità sui dati della tabella. Il dominio definisce il tipo di dato dell'attributo, domini predefiniti del linguaggio SQL, Vincoli di integrità e specificare vincoli di tipo generale sulla tabella.

Domini elementari.

Tipologia di dato	SQL
Testo	CHARACTER [VARYING] [(Lunghezza)] [CHARACTER SET NomeFamigliaCaratteri] VARCHAR (Lunghezza) TEXT
Binario	BIT [VARYING] [(Lunghezza)] BLOB BINARY
Booleano	BOOLEAN
Numeri interi	INTEGER SMALLINT BIGINT
Numeri reali	NUMERIC [(Precisione, Scala)] DECIMAL [(Precisione, Scala)] FLOAT [(n)] REAL DOUBLE PRECISION
Tipologia di dato	SQL
Tempo	TIMESTAMP [(Precisione)] [WITH TIME ZONE] DATE DATETIME
JSON	JSON
Spaziali	SDO_GEOGRAPHY GEOMETRY POINT LINESTRING POLYGON

Esempio:

```
CREATE TABLE F (  
  CodF      CHAR(5),  
  NomeF     CHAR(20),  
  NSoci      SMALLINT,  
  Sede       CHAR(15));  
• Creazione della tabella forniture  
CREATE TABLE FP (  
  CodF      CHAR(5),  
  CodP      CHAR(6),  
  Qta       INTEGER);  
CREATE TABLE P (  
  CodP      CHAR(6),  
  NomeP     CHAR(20),  
  Colore    CHAR(6),  
  Taglia    SMALLINT,  
  Magazzino CHAR(15));
```



Mancata definizione dei vincoli di integrità

Modifica della tabella.

Sono possibili le seguenti “alterazioni”:

- Aggiunta di una nuova colonna
- Definizione di un nuovo valore di default per una colonna esistente
- Eliminazione di una colonna esistente

- Definizione di un nuovo vincolo di integrità
- Eliminazione di un vincolo di integrità esistente

ALTER TABLE NomeTabella

< ADD COLUMN <Definizione-Attributo> |

ALTER COLUMN NomeAttributo

< SET <Definizione-Valore-Default> | **DROP DEFAULT**>|

DROP COLUMN NomeAttributo

< CASCADE | RESTRICT > |

ADD CONSTRAINT [NomeVincolo]

< definizione-vincolo-unique > |

< definizione-vincolo-integrità-referenziale > |

< definizione-vincolo-check > |

DROP CONSTRAINT[NomeVincolo]

< CASCADE | RESTRICT >

- RESTRICT: l'elemento non è rimosso se è presente in qualche definizione di un altro elemento
- CASCADE: tutti gli elementi che dipendono da un elemento rimosso vengono rimossi, fino a quando non esistono più dipendenze non risolte

Cancellazione di una tabella.

DROP TABLE NomeTabella

[RESTRICT | CASCADE];

Tutte le righe della tabella sono eliminate insieme alla tabella

- RESTRICT: la tabella non è rimossa se è presente in qualche definizione di tabella, vincolo o vista
- CASCADE: se la tabella compare in qualche definizione di vista anche questa è rimossa

Dizionario dei dati.

I metadati sono informazioni (dati) sui dati, ovvero l'immagine del DBMS sulla base dati.

Possono essere memorizzati in tabelle della base di dati. Il dizionario dei dati contiene i metadati di una base dati relazionale:

- Contiene informazioni sugli oggetti della base di dati
- È gestito direttamente dal DBMS relazionale
- Può essere interrogato con istruzioni SQL

Contiene diverse informazioni

- Descrizione di tutte le strutture (tabelle, indici, viste) della base di dati
- Stored procedure SQL
- Privilegi degli utenti

- Statistiche su tabelle basi dati, indici della base dati, viste base dati e sulla crescita della base di dati.

Informazioni sulle tabelle.

Il dizionario dei dati contiene per ogni tabella della base di dati:

- Nome tabella e struttura fisica del file dove è memorizzata
- Nome e tipo dato per ogni attributo
- Vincoli di integrità
- Nome di tutti gli indici creati sulla tabella

Esempio 1:

visualizzare il nome delle tabelle definite dall'utente ed il numero di tuple memorizzate in ciascuna di esse

```
SELECT Table_Name, Num_Rows
FROM USER_TABLES;
```

R	
Table_Name	Num_Rows
F	5
P	6
FP	12

Esempio 2:

per ogni attributo della tabella forniture, visualizzare il nome dell'attributo, il numero di valori diversi e il numero di tuple che assumono valore NULL

```
SELECT Column_Name, Num_Distinct, Num_Nulls
FROM USER_TAB_COL_STATISTICS
WHERE Table_Name = 'FP'
ORDER BY Column_Name;
```

R		
Column_Name	Num_Distinct	Num_Nulls
CodF	4	0
CodP	6	0
Qta	4	0

Integrità dei dati.

I dati all'interno di una base di dati sono corretti se soddisfano un insieme di **regole di correttezza**

- Le regole sono dette **vincoli di integrità**
- Esempio: Qta \geq 0

Le operazioni di modifica dei dati definiscono un nuovo stato della base dati, che non è necessariamente corretto.

La verifica della correttezza dello stato di una base di dati può esser effettuata

- Dalle **procedure applicative**, che effettuano tutte le verifiche necessarie
- Mediante la definizione di **vincoli di integrità** sulle tabelle
- Mediante la definizione dei **trigger**

Procedure applicative.

All'interno di ogni applicazione sono previste le verifiche di correttezza necessarie.

Vantaggi	Svantaggi
<ul style="list-style-type: none">• approccio "flessibile"	<ul style="list-style-type: none">• è possibile "aggirare" le verifiche interagendo direttamente con il DBMS• un errore di codifica può avere un effetto significativo sulla base di dati• la conoscenza delle regole di correttezza è tipicamente "nascosta" nelle applicazioni

Vincoli di integrità sulle tabelle.

I **vincoli di integrità** sono:

- Definiti nelle istruzioni **CREATE** o **ALTER TABLE**
- Memorizzati nel dizionario dati del sistema

Durante l'esecuzione di qualunque operazione di modifica dei dati il DBMS verifica automaticamente che i vincoli siano osservati.

Durante l'esecuzione di qualunque operazione di modifica dei dati il DBMS verifica automaticamente che i vincoli siano osservati.

Vantaggi	Svantaggi
<ul style="list-style-type: none">• definizione dichiarativa dei vincoli, la cui verifica è affidata al sistema<ul style="list-style-type: none">• il dizionario dei dati descrive tutti i vincoli presenti nel sistema• unico punto centralizzato di verifica<ul style="list-style-type: none">• impossibilità di aggirare la verifica dei vincoli	<ul style="list-style-type: none">• possono rallentare l'esecuzione delle applicazioni• non è possibile definire tipologie arbitrarie di vincoli<ul style="list-style-type: none">• esempio: vincoli su dati aggregati

Trigger.

I trigger sono procedure **eseguite in modo automatico** quando si verificano opportune modifiche dei dati:

- Definiti nell'istruzione **CREATE TRIGGER**
- Memorizzati nel dizionario dati del sistema

Quando si verifica un evento di modifica dei dati sotto il controllo del trigger, la procedura viene eseguita automaticamente.

Riparazione delle violazioni.

Se un'applicazione tenta di eseguire un'operazione che violerebbe un vincolo, il sistema può:

- Impedire l'operazione, causando un errore di esecuzione dell'applicazione
- Eseguire un'**azione compensativa** tale da raggiungere un nuovo stato corretto

Vincoli di integrità SQL.

Possibilità di specificare i vincoli di integrità in modo dichiarativo, si affida al sistema la verifica della loro consistenza.

Tipologie di vincoli:

- Vincoli di tabella (restrizione sui dati)
- Vincoli di integrità referenziale (gestione riferimenti tra tabelle diverse)

Vincoli di tabella.

Sono definiti su una o più colonne di una tabella, sono definiti nelle istruzioni di creazione di tabelle e domini. Le tipologie possono essere:

- Chiave primaria
- Ammissibilità del valore nullo
- Unicità
- Vincoli generali di tupla

Sono verificati dopo ogni istruzione SQL che operano sulla tabella soggetta al vincolo: inserimento nuovi dati o modifica del valore di colonne soggette al vincolo.

Se il vincolo è violato occorre generare un **errore di esecuzione** (ABORT).

Chiave primaria.

La chiave primaria è un insieme di attributi che identifica in modo univoco le righe di una tabella. Può essere specificata una sola chiave primaria per una tabella.

La definizione della chiave primaria viene definita:

NomeAttributo Dominio **PRIMARY KEY**

Composta da uno o più attributi

PRIMARY KEY (ElencoAttributi)

Esempio:

```
un solo attributo
```

```
CREATE TABLE F (CodF CHAR(5) PRIMARY KEY,  
    NomeFCHAR(20),  
    NSoci SMALLINT,  
    Sede CHAR(15));
```



```
uno o più attributi
```

```
CREATE TABLE FP (CodF CHAR(5),  
    CodP CHAR(6),  
    Qta INTEGER  
    PRIMARY KEY (CodF, CodP));
```

Ammissibilità del valore nullo.

Il valore **NULL** indica l'assenza di informazioni, quando è obbligatorio specificare sempre un valore per l'attributo

NomeAttributo Dominio **NOT NULL**

Il valore nullo non è ammesso

Esempio:

```
CREATE TABLE F (CodF CHAR(5),
                NomeFCHAR(20) NOT NULL,
                NSoci SMALLINT,
                Sede CHAR(15));
```

Unique.

Un attributo o un insieme di attributi non può assumere lo stesso valore in righe diverse della tabella, per un solo attributo o per più attributo.

È ammessa la ripetizione del valore **NULL**, considerato sempre diverso

NomeAttributo Dominio **UNIQUE**

UNIQUE (ElencoAttributi)

La chiave candidata è un insieme di attributi che potrebbe assumere il ruolo di chiave primaria, ovvero è:

- Univoca
- Può ammettere il valore nullo

La combinazione **UNIQUE NOT NULL** permette di definire una chiave candidata che non ammette valori nulli.

Esempio:

```
CREATE TABLE P (CodP      CHAR(6),
                NomeP      CHAR(20) NOT NULL UNIQUE,
                Colore     CHAR(6),
                Taglia     SMALLINT,
                Magazzino  CHAR(15));
```

Vincoli generali di tupla.

Permettono di esprimere condizioni di tipo generale su ogni tupla

NomeAttributo Dominio **CHECK** (Condizione)

Possono essere indicati come condizione i predicati specificabili nella clausola WHERE e la base di dati è corretta se la condizione è vera

Esempio:

```
CREATE TABLE F (CodF  CHAR(5) PRIMARY KEY,
                NomeF CHAR(20) NOT NULL,
                NSoci  SMALLINT CHECK (NSoci>0),
                Sede   CHAR(15));
```

Vincoli d'integrità referenziale.

Permettono di gestire il legame tra tabelle mediante il valore di attributi. La chiave esterna è definita nell'istruzione **CREATE TABLE** della tabella referenziante

FOREIGN KEY (ElencoAttributiReferenzianti)

REFERENCES NomeTabella [(ElencoAttributiReferenziati)]

Esempio:

```
CREATE TABLE FP (CodF    CHAR(5),
                 CodP    CHAR(6),
                 Qta     INTEGER,
                 PRIMARY KEY (CodF, CodP),
                 FOREIGN KEY (CodF)
                           REFERENCES F(CodF),
                 FOREIGN KEY (CodP)
                           REFERENCES P(CodP));
```

Politiche di gestione dei vincoli.

I vincoli di integrità sono verificati dopo ogni istruzione SQL che potrebbe causarne la violazione. Non sono ammesse operazioni di inserimento e modifica della tabella referenziante che violino il vincolo. Nell'istruzione CREATE TABLE della tabella referenziante

FOREIGN KEY (ElencoAttributiReferenzianti)

REFERENCES

NomeTabella [(ElencoAttributiReferenziati)]

[ON UPDATE

<CASCADE | SET DEFAULT | SET NULL | NO ACTION>]

[ON DELETE

<CASCADE | SET DEFAULT | SET NULL | NO ACTION>]

Operazioni di modifica o cancellazione della tabella referenziata causano sulla tabella referenziante:

- **CASCADE**: propagazione dell'operazione di aggiornamento o cancellazione
- **SET NULL/DEFAULT**: null o valore di default in tutte le colonne delle tuple che hanno valori non più presenti nella tabella referenziata
- **NO ACTION**: non si esegue azione invalidante

Esempio 1:

- tabella **P**: descrive i prodotti disponibili
 - chiave primaria: CodP
 - nome prodotto non può assumere valori nulli o duplicati
 - la taglia è sempre maggiore di zero
- tabella **F**: descrive i fornitori
 - chiave primaria: CodF
 - nome fornitore non può assumere valori nulli o duplicati
 - numero dei soci è sempre maggiore di zero
- Tabella **FP** (referenziante)
 - insert (nuova tupla) -> No
 - update (CodF) -> No
 - delete (tupla) -> Ok
- Tabella **F** (referenziata)
 - insert (nuova tupla) -> Ok
 - update (CodF) -> aggiornare in cascata (cascade)
 - delete (tupla) -> aggiornare in cascata (cascade)
impedire l'azione (no action)

```

CREATE TABLE P (CodP CHAR(6) PRIMARY KEY,
    NomeP CHAR(20) NOT NULL UNIQUE,
    Colore CHAR(6),
    Taglia SMALLINT CHECK (Taglia > 0),
    Magazzino CHAR(15));

CREATE TABLE F (CodF CHAR(5) PRIMARY KEY,
    NomeF CHAR(20) NOT NULL,
    NSoci SMALLINT CHECK (NSoci>0),
    Sede CHAR(15));

CREATE TABLE FP (CodF CHAR(5),
    CodP CHAR(6),
    Qta INTEGER,
    CHECK (Qta IS NOT NULL and Qta>0),
    PRIMARY KEY (CodF, CodP),
    FOREIGN KEY (CodF)
        REFERENCES F(CodF)
        ON DELETE NO ACTION
        ON UPDATE CASCADE,
    FOREIGN KEY (CodP)
        REFERENCES P(CodP)
        ON DELETE NO ACTION
        ON UPDATE CASCADE);

```

Esempio 2:

- Impiegati (Matr, Nomel, Residenza, DNum)
- Dipartimenti (DNum, DNome, Sede)

- Impiegati (referenziante)
 - insert (nuova tupla) -> No
 - update (DNum) -> No
 - delete (tupla) -> Ok
- Dipartimenti (referenziata)
 - insert (nuova tupla) -> Ok
 - update (DNum) -> aggiornare in cascata (cascade)
 - delete (tupla)
 - > aggiornare in cascata (cascade)
 - impedire l'azione (no action)
 - impostare a valore ignoto (set null)
 - impostare a valore di default (set default)

15. COSTRUTTI AVANZATI

Viste.

La vista è una **tabella virtuale**, uno strumento che consente di accedere ad una porzione della base dati. Il contenuto (tuple) è definito mediante un'interrogazione SQL sulla base dati, ma non è memorizzato fisicamente, è ricalcolato ogni volta che si usa la vista eseguendo l'esecuzione che la definisce.

La vista è un oggetto della base dati, creata con una DDL, salvata nel dizionario dei dati ed è utilizzabile come se fosse una tabella. Se l'interrogazione fa riferimento ad una vista, deve esser riscritta dal DBMS prima dell'esecuzione.

OSS: la vista è un oggetto della base dati, fa parte del dizionario e posso usarla più volte, le CTE vivono localmente nelle query in cui sono chiamate.

Esempio 1:

definizione della vista **piccoli fornitori**, ovvero i fornitori che hanno meno di 3 soci

```
CREATE VIEW PICCOLI_FORNITORI AS  
  SELECT CodF, NomeF, NSoci, Sede  
  FROM   F  
  WHERE  NSoci<3;
```

Nome della vista

Interrogazione associata alla vista

per visualizzare il codice, nome, sede e numero di soci per piccoli fornitori di Torino:

```
SELECT *  
FROM  PICCOLI_FORNITORI  
WHERE Sede='Torino';
```

Esempio 2:

definizione della vista numero di fornitori per prodotto

```
CREATE VIEW NUMFORNITORI_PER_PRODOTTO  
(CodP, NumFornitori) AS  
  SELECT CodP, COUNT(*)  
  FROM   FP  
  GROUP BY CodP;
```

Attributi della vista

Nome della vista

Interrogazione associata alla vista

Vantaggi offerti dalle viste.

- Semplificazione delle interrogazioni: scomposizione di una query complessa in più parti
- Gestione sicurezza: meccanismi di protezione della privacy diversi per ogni utente o gruppo, la vista diviene l'elemento a cui sono associate le autorizzazioni di accesso e quindi ogni utente o gruppo accede alla base di dati solo mediante le viste appropriate per le operazioni che è abilitato a svolgere
- Evoluzione della base di dati: in caso di ristrutturazione di una base dati, è sufficiente ridefinire le viste.

Creazione / cancellazione viste.

```
CREATE VIEW NomeVista [(ElencoAttributi)]  
AS InterrogazioneSQL;
```

I nomi degli attributi devono essere specificati se rappresentano il risultato di una funzione interna, rappresentano il risultato di un'espressione, sono costanti o se due colonne hanno lo stesso nome.

```
DROP VIEW NomeVista;
```

Può avere diversi effetti, in base al DBMS.

Aggiornabilità delle viste.

È possibile eseguire operazioni di aggiornamento dei dati presenti in una vista solo per alcune viste, sono aggiornabili le viste in cui una sola riga di ciascuna tabella di base corrisponde a una sola riga della vista.

Non è aggiornabile una vista che nel blocco più esterno dell'interrogazione che la definisce

- Non contiene la chiave primaria della tabella su cui è definita
- Contiene DISTINCT
- Contiene funzioni aggregate
- Contiene join che rappresentano corrispondenze uno a molti o molti a molti

Esempio 1:

è aggiornabile, non contiene distinct, funzioni aggregate, join e contiene chiave primaria

```
CREATE VIEW FORNITORE_SEDE AS  
SELECT CodF, Sede  
FROM F;
```

inserimento

('F10', 'Roma')

➔ corrisponde all'inserimento in F di

('F10', NULL, NULL, 'Roma')

- gli attributi NomeF, NSoci devono ammettere il valore NULL

cancellazione

('F1', 'Torino')

➔ cancellazione da F di

('F1', 'Andrea', 2, 'Torino')

- l'identificazione della tupla da cancellare è permessa dalla chiave primaria

modificabilità

('F1', 'Torino') in ('F1', 'Milano')

➔ modifica in F di

('F1', 'Andrea', 2, 'Torino') in ('F1', 'Andrea', 2, 'Milano')

- l'identificazione della tupla da modificare è permessa dalla chiave primaria

Esempio 2:

```
CREATE VIEW NUMSOCI_SEDE AS  
SELECT DISTINCT NSoci, Sede  
FROM F;
```

Inserimento

(40, 'Napoli')

➔ impossibile inserire in F

(NULL, NULL, 40, 'Napoli')

- manca il valore della chiave primaria

Cancellazione

(2, 'Torino')

➔ più tuple sono associate alla coppia (2, 'Torino')

- quale tupla deve essere cancellata da F?

Modifica

(2, 'Torino') in (3, 'Milano')

➔ più tuple sono associate alla coppia (2, 'Torino')

- quale tupla deve essere modificata in F?

La vista NUMSOCI_SEDE **non è aggiornabile**, non è presente la chiave primaria della tabella F nella vista, alcune tuple della vista corrispondono a più tuple della tabella F.

Esempio 3:

vista non è aggiornabile, non seleziona in modo esplicito la chiave primaria della tabella F, è sufficiente sostituire al simbolo "*" il nome degli attributi, indicando la chiave primaria.

```
CREATE VIEW FORNITORI_TORINO AS  
SELECT *  
FROM F  
WHERE Sede='Torino';
```

Esempio 4:

la vista non è aggiornabile, è presente un join ed è presente la parola chiave DISTINCT

```
CREATE VIEW FORNITORI_IMPORTANTI (CodF,  
NomeF) AS  
SELECT DISTINCT CodF, NomeF  
FROM F, FP  
WHERE F.CodF=FP.CodF AND  
Qta>100;
```

ma può esser **resa aggiornabile** sostituendo join → IN e il distinct non è più necessario!

```
CREATE VIEW FORNITORI_IMPORTANTI (CodF, NomeF) AS  
SELECT CodF, NomeF  
FROM F  
WHERE CodF IN (SELECT CodF  
FROM FP  
WHERE Qta>100);
```

Transazioni.

Necessaria quando più utenti possono accedere contemporaneamente ai dati, offre meccanismi efficienti per gestire l'accesso concorrente ai dati ed eseguire un recovery a seguito di un malfunzionamento.

Si tratta di **un'unità logica di lavoro non scomponibile**, contenente una sequenza di operazioni di modifica dei dati, portando la base da uno stato consistente ad un altro consistente.

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

Per definire l'inizio di una transazione, il linguaggio SQL prevede l'istruzione

START TRANSACTION

Di solito è implicita

La fine di una transazione può

- Terminare con successo → tutte le transazioni terminano con successo: **commit**
COMMIT [WORK]
- Terminare con insuccesso → si genera un abort
ROLLBACK [WORK]

Commit.

È un'azione eseguita quando una transazione termina con successo e la base di dati è in un nuovo stato (finale) corretto, le modifiche dei dati della transazione divengono permanenti e visibili.

Rollback.

Azione eseguita quando una transazione termina a causa di un errore, tutte le operazioni di modifica vengono annullate e la base torna allo stato precedente l'inizio della transazione.

Esempio:

```
• Trasferire la somma 100
  • dal conto corrente bancario IT92X0108201004300000322229
  • al conto corrente bancario IT32L0201601002410000278976

  START TRANSACTION;

  UPDATE Conto-Corrente
    SET Saldo = Saldo - 100
    WHERE IBAN='IT92X0108201004300000322229';

  UPDATE Conto-Corrente
    SET Saldo = Saldo + 100
    WHERE IBAN= 'IT32L0201601002410000278976';

  COMMIT;
```

Le proprietà principali delle transazioni.

Le proprietà delle transazioni sono riassunte dall'acronimo **ACID**.

1. **Atomicità**: una transazione è un'unità indivisibile (atomica) di lavoro. Devono essere eseguite tutte le operazioni contenute nella transazione oppure nessuna deve esser eseguita. La base non può rimanere in uno stato intermedio durante l'esecuzione di una transazione.
2. **Consistenza**: l'esecuzione di una transazione deve portare la base da uno stato iniziale consistente (corretto) ad uno stato finale consistente. La correttezza è verificata dai vincoli di

integrità definiti sulla base di dati e quando si verifica la violazione di un vincolo di integrità il sistema interviene per annullare la transazione, oppure per modificare lo stato della base di dati eliminando il vincolo.

3. **Isolamento**: l'esecuzione di una transazione è indipendente dalla contemporanea esecuzione di altre transazioni. Gli effetti non sono visibili dalle altre transazioni fino a quando la transazione non è terminata; uno stato intermedio può esser annullato da un rollback successivo.
4. **Persistenza**: l'effetto di una transazione che ha effettuato il commit è salvato in modo permanente in memoria principale e secondaria. Garantisce l'affidabilità delle operazioni di modifica dei dati. I DBMS forniscono meccanismi di ripristino dello stato corretto della base di dati dopo che si è verificato un guasto (file di log).

Sicurezza dei dati.

Protezione dei dati da

- Letture non autorizzate
- Alterazione o distruzione

Il DBMS offre strumenti per realizzare le protezioni, definite dall'**admin della base di dati (DBA)**. Il controllo verifica che gli utenti siano autorizzati ad eseguire le operazioni che richiedono di eseguire, mediante un **insieme di vincoli** specificati dal DBA e memorizzati nel dizionario dei dati.

Risorse.

Qualsiasi componente dello schema di una base dati è una risorsa, come tabelle, viste, attributo, dominio, procedura, .. ecc

Le risorse sono protette mediante la definizione di **privilegi di accesso**.

Privilegi di accesso.

Descrivono i **diritti di accesso alle risorse del sistema**. La SQL offre i meccanismi di controllo dell'accesso molto flessibili mediante i quali è possibile specificare

- Le risorse a cui possono accedere agli utenti
- Le risorse che devono essere mantenute private

Ogni privilegio è caratterizzato dalle seguenti informazioni:

- La risorsa a cui si riferisce
- Il tipo di privilegio (azione permessa sulla risorsa)
- L'utente che concede il privilegio
- L'utente che riceve il privilegio
- La facoltà di trasmettere il privilegio ad altri utenti

Tipi di privilegi:

- | | |
|---|--|
| <ul style="list-style-type: none">• INSERT<ul style="list-style-type: none">• permette di inserire un nuovo oggetto nella risorsa• vale per le tabelle e le viste• UPDATE<ul style="list-style-type: none">• permette di aggiornare il valore di un oggetto• vale per le tabelle, le viste e gli attributi• DELETE<ul style="list-style-type: none">• permette di rimuovere oggetti dalla risorsa• vale per le tabelle e le viste | <ul style="list-style-type: none">• SELECT<ul style="list-style-type: none">• permette di utilizzare la risorsa all'interno di un'interrogazione• vale per le tabelle e le viste• REFERENCES<ul style="list-style-type: none">• permette di far riferimento a una risorsa nella definizione dello schema di una tabella• può essere associato solo a tabelle e attributi• USAGE<ul style="list-style-type: none">• permette di utilizzare la risorsa (per esempio, un nuovo tipo di dato) nella definizione di nuovi schemi |
|---|--|

Creatore della risorsa	Amministratore del sistema
<ul style="list-style-type: none"> Alla creazione di una risorsa, il sistema concede tutti i privilegi su tale risorsa all'utente che ha creato la risorsa Solo il creatore della risorsa ha il privilegio di eliminare una risorsa (DROP) e modificarne lo schema (ALTER) <ul style="list-style-type: none"> Il privilegio di eliminare e modificare una risorsa non può essere concesso a nessun altro utente 	<ul style="list-style-type: none"> L'amministratore del sistema (utente system) possiede tutti i privilegi su tutte le risorse

Gestione dei privilegi in SQL.

I privilegi sono concessi o revocati mediante le istruzioni SQL

- GRANT: concede privilegi su una risorsa ad uno o più utenti
- REVOKE: toglie ad uno o più utenti i privilegi che erano stati loro concessi

Grant.

```
GRANT ElencoPrivilegi ON NomeRisorsa
TO ElencoUtenti
[WITH GRANT OPTION]
```

ElencoPrivilegi specifica l'elenco dei privilegi **ALL PRIVILEGES**

NomeRisorsa: specifica la risorsa sulla quale si vuole concedere il privilegio

ElencoUtenti: specifica gli utenti a cui viene concesso il privilegio

WITH GRANT OPTION: facoltà di trasferire il privilegio ad altri utenti

Esempio:

<pre>GRANT ALL PRIVILEGES ON PRODOTTI TO Neri, Bianchi</pre> <ul style="list-style-type: none"> Agli utenti Neri e Bianchi sono concessi tutti i privilegi sulla tabella PRODOTTI 	<pre>GRANT SELECT ON FORNITORI TO Rossi WITH GRANT OPTION</pre> <ul style="list-style-type: none"> All'utente Rossi è concesso il privilegio di SELECT sulla tabella FORNITORI L'utente Rossi ha facoltà di trasferire il privilegio ad altri utenti
--	---

Revoke.

```
REVOKE ElencoPrivilegi ON NomeRisorsa FROM
ElencoUtenti
[RESTRICT | CASCADE]
```

Può togliere tutti i privilegi che erano stati concessi

RESTRICT: il comando non deve essere eseguito qualora la revoca dei privilegi all'utente comporti qualche altra revoca di privilegi

CASCADE. Revoca anche tutti i privilegi che erano stati propagati agendo in cascata e rimuovendo tutti gli elementi della base di dati creati sfruttando questi privilegi.

Esempio:

REVOKE UPDATE ON PRODOTTI FROM Bianchi	REVOKE SELECT ON FORNITORI FROM Rossi CASCADE
<ul style="list-style-type: none">• All'utente Bianchi è revocato il privilegio di UPDATE sulla tabella PRODOTTI<ul style="list-style-type: none">• il comando non è eseguito se comporta la revoca del privilegio ad altri utenti	<ul style="list-style-type: none">• All'utente Rossi è revocato il privilegio di SELECT sulla tabella FORNITORI• L'utente Rossi aveva ricevuto il privilegio con GRANT OPTION<ul style="list-style-type: none">• se Rossi ha propagato il privilegio ad altri utenti, il privilegio è revocato in cascata• se Rossi ha creato una vista utilizzando il privilegio di SELECT, la vista è rimossa

Concetto di ruolo.

Il ruolo è un profilo di accesso, definito dall'insieme di privilegi che lo caratterizzano.

Ogni utente ricopre un ruolo predefinito e gode dei privilegi associati al ruolo

Vantaggi:

- Controllo dell'accesso più flessibile (utente può ricoprire ruoli diversi in momenti diversi)
- Semplificazione dell'attività di amministrazione: facilità di definire profilo di nuovi utenti e definire un profilo di accesso in un momento diverso dalla sua attivazione.

Create ROLE.

CREATE ROLE NomeRuolo

Definizione dei privilegi di un ruolo e del ruolo di un utente. Un utente in momenti diversi può ricoprire ruoli diversi.

SET ROLE NomeRuolo

Gestione degli indici.

Organizzazione fisica dei dati.

All'interno di un DBMS relazionale, i dati sono rappresentati come collezioni di record memorizzati in uno o più file. Non esiste un'organizzazione fisica dei dati che sia efficiente per qualunque tipo di lettura e scrittura dei dati.

La progettazione fisica definisce il layout fisico delle tabelle che incide significativamente sul costo di esecuzione per l'accesso ai dati.

Indici.

Gli **indici** sono le strutture fisiche accessorie offerte dai DBMS relazionali per migliorare l'efficienza delle operazioni di accesso ai dati.

Sono realizzate mediante strutture fisiche di tipo diverso:

- Alberi
- Hash table

Le istruzioni per la gestione degli indici non fanno parte dello standard SQL.

Sono dei puntatori fisici.

Il linguaggio SQL offre le seguenti istruzioni per la definizione degli indici

- Creazione di un indice: **CREATE INDEX**
- Cancellazione di un indice: **DROP INDEX**

Il DBMS sceglie se eseguire istruzione caricando la tabella o leggendo l'indice, conoscendo dimensioni della tabella e calcolando il costo di esecuzione dei dati.

Create INDEX.

```
CREATE INDEX NomeIndice  
ON NomeTabella (ElencoAttributi)
```

L'ordine in cui compaiono gli attributi in ElencoAttributi è **importante**.

Esempio:

- Creazione di un indice sulla combinazione di attributi Cognome e Nome della tabella DIPENDENTE

```
CREATE INDEX IndiceCognomeNome  
ON DIPENDENTE(Cognome, Nome)
```

- L'indice è definito congiuntamente sui due attributi
- Le chiavi dell'indice sono ordinate
 - prima in base al valore dell'attributo Cognome
 - a pari valore dell'attributo Cognome, sul valore dell'attributo Nome

Drop INDEX.

```
DROP INDEX NomeIndice
```

Elimina l'indice NomeIndice

Progettazione fisica.

I dati di ingresso sono

- Schema logico della base dati
- Caratteristiche del DBMS (strutture fisiche, indici)
- Volume dei dati (cardinalità delle tabelle)
- Stima del carico applicativo con interrogazioni più importanti e la loro frequenza, operazioni di aggiornamento più importanti e la loro frequenza, requisiti sul tempo di risposta per interrogazioni/aggiornamenti importanti.

Il risultato

- Schema fisico della base di dati con organizzazione fisica delle tabelle e degli indici
- Parametri di memorizzazione e funzionamento

Procedimento: la progettazione fisica è svolta in modo empirico, con approccio per tentativi e non esistono metodologie di riferimento.

Caratterizzazione del carico applicativo: per ogni interrogazione è importante rilevante è necessario definire

- Relazioni a cui accede
- Attributi da visualizzare
- Attributi coinvolti in selezioni/join

- Grado di selettività delle condizioni di selezione

Tuning.

Se il risultato non è soddisfacente, il **Tuning** si svolge aggiungendo e togliendo indici

È un procedimento guidato dalla disponibilità di strumenti che permettano di verificare il piano di esecuzione adottato dal DBMS prescelto

- Il piano di esecuzione definisce la sequenza di attività svolte dal DBMS per eseguire un'interrogazione
- Metodi di accesso ai dati
- Metodi di join

Si riferisce all'insieme di tecniche e strategie utilizzate per ottimizzare le prestazioni di un database. L'obiettivo principale del tuning è migliorare l'efficienza delle operazioni di lettura e scrittura, ridurre i tempi di risposta delle query e garantire che il sistema sia in grado di gestire un carico di lavoro elevato in modo efficace.

Ecco alcune delle principali aree di focus del tuning di un DBMS:

1. ****Ottimizzazione delle query**:**
2. ****Ottimizzazione degli indici**:**
3. ****Gestione della memoria**:**
4. ****Ottimizzazione della concorrenza**:**
5. ****Gestione dell'archiviazione**:**
6. ****Monitoraggio e manutenzione continua**:**

L'ottimizzazione di un DBMS richiede una comprensione approfondita del funzionamento interno del sistema, delle caratteristiche del carico di lavoro e delle esigenze specifiche dell'applicazione. Spesso, il tuning è un processo iterativo che richiede prove ed errori per trovare le configurazioni e le strategie più efficaci.

16. TRIGGER

Il funzionamento tradizionale del DBMS è **passivo** cioè, esegue la richiesta

- Le query e gli aggiornamenti sono richiesti esplicitamente dagli utenti
- La conoscenza dei processi che operano sui dati è tipicamente incorporata nelle applicazioni

Le basi di dati **attive** invece, **reagiscono** alle operazioni INSERT/UPDATE/DELETE eseguendo anche altre operazioni in risposta.

La reattività delle basi di dati attive è assicurata dall'esecuzione automatica di **regole attive o ECA**, poiché espresso nella forma

- Evento: operazione di modifica sui dati
- Condizione: filtrare il caso in cui l'evento deve attivare un'azione
- Azione: sequenza istruzioni SQL o procedura applicativa da eseguire alla richiesta dell'evento

Rule engine.

È un componente del DBMS incaricato di:

- Tracciare gli eventi
- Eseguire le regole quando è appropriato, in base alla strategia di esecuzione del DBMS

L'esecuzione delle regole è alternata all'esecuzione delle transazioni tradizionali.

Esempio:

la regola attiva gestisce il riordino delle scorte di un magazzino:

quando la quantità disponibile di un prodotto scende al di sotto di una soglia, deve essere emesso un nuovo ordine per il prodotto

- Evento: aggiornamento della quantità disponibile per il prodotto x e inserimento di un nuovo prodotto x
- Condizione: la quantità disponibile del prodotto x è inferiore a una determinata soglia e non ci sono ordini in sospeso per il prodotto x
- Azione: emettere un ordine con una determinata quantità di riordino per il prodotto x

Applicazione delle regole attive.

Le applicazioni interne servono al mantenimento di vincoli di integrità complessi (non specificabili nelle CREATE TABLE), gestione delle repliche e manutenzione delle viste materializzate.

Le **business rules** sono sequenze di operazioni ed incorporano nel DBMS la conoscenza delle applicazioni.

Alert utilizzati per le notifiche.

Queste regole sono parte integrante del dato e non possono esser violate. Si riduce trasmissione di dati tra database ed applicazione.

Trigger.

Sono oggetti della base dati, nei DBMS le regole attive sono implementate mediante **i trigger**.

Il linguaggio SQL fornisce istruzioni per la definizione dei trigger, definiti con l'istruzione DDL **CREATE TRIGGER**. La sintassi e la semantica sono definite nello standard SQL3.

Struttura del trigger.

- Evento: inserimento/cancellazione/aggiornamento di una tabella. Inoltre ogni trigger può monitorare una singola tabella
- Condizione: predicato SQL
- Azione: sequenza di istruzioni SQL, codice Java e blocchi di linguaggi di programmazione

Processo di esecuzione.

Quando gli eventi si verificano [attivazione]

Se la condizione è vera [valutazione]

Allora l'azione viene eseguita [esecuzione]

Sembra molto semplice, ma occorre valutare la modalità di esecuzione e della granularità di esecuzione.

Modo di esecuzione.

La modalità può essere

- Immediata: trigger eseguito **immediatamente prima o dopo** l'istruzione di attivazione
- Differita: trigger eseguito **immediatamente prima** del **commit**

Nei sistemi commerciali è disponibile solo l'opzione immediata.

Granularità dell'esecuzione.

Quante volte viene eseguito un trigger?

Può essere a livello di:

- Tupla (o livello di riga): **esecuzione separata** del trigger **per ogni tupla** interessata dall'istruzione di attivazione
- Istruzione: una **singola esecuzione** per **tutte le tuple** interessate dall'istruzione di attivazione

Esempio di granularità.

- Tabella T

A	B
1	5
2	9
8	20

- Evento sulla tabella T

```
UPDATE T  
SET A=A+1  
WHERE B<10;
```

- Esecuzione del trigger definito sulla tabella T

- Un trigger a livello di tupla viene eseguito due volte
- Un trigger a livello di istruzione viene eseguito una sola volta

Trigger in Oracle.

Sintassi:

CREATE TRIGGER TriggerName **OR REPLACE**

Mode Event {**OR** Event}

ON TargetTable

[**[REFERENCING** ReferenceName]**]**

FOR EACH ROW

[**WHEN** Predicate]]

PL/SQL Block

Il modo è **BEFORE** o **AFTER**, anche **INSTEAD IF** ma è da evitare.

L'evento **ON** tabella target può essere **INSERT**, **UPDATE**, **DELETE** [OF Column Name]

FOR EACH ROW specifica la semantica di esecuzione a livello di tupla, se omessa la semantica è a livello di istruzione.

Per rinominare le variabili di stato si usa **REFERENCING OLD AS** OldVariableName (non usata).

La clausola **WHEN** specifica la condizione, è opzionale e vale solo per i trigger a livello di tupla, è possibile specificare facoltativamente una condizione ed è possibile accedere alle variabili di stato vecchie e nuove.

Nel blocco PL/SQL nessuna istruzione è transazionale e DDL, posso usare solo istruzioni DML.

Algoritmo di esecuzione.

1. Vengono eseguiti i trigger di tipo before a livello di istruzione
2. Per ogni tupla nella tabella target (TargetTable) interessata dall'istruzione di trigger
 - a) Vengono eseguiti i trigger di tipo before a livello di tupla
 - b) Viene eseguita l'istruzione di attivazione + vincoli di integrità definiti a livello di tupla
 - c) Vengono eseguiti i trigger di tipo after a livello di tupla
3. Vengono controllati i vincoli di integrità sulle tabelle
4. Vengono eseguiti i trigger di tipo after a livello di istruzione

Semantica.

L'ordine di esecuzione di trigger con lo stesso evento, la stessa modalità e la stessa granularità non viene specificato: è fonte di non determinismo.

Quando si verifica un errore:

- Rollback di tutte le operazioni eseguite dai trigger
- Rollback dell'istruzione di attivazione nella transazione di attivazione

Non terminazione.

L'esecuzione di un trigger può attivare altri trigger e l'attivazione di essi in cascata può portare alla non terminazione dell'esecuzione del trigger.

È possibile impostare la lunghezza massima per l'esecuzione di trigger in cascata e, se il limite massimo viene superato si genera un errore di esecuzione.

Tabella mutante.

Una **tabella mutante** è la tabella target modificata dall'istruzione che attiva il trigger. Essa:

- **Non** è accessibile nei trigger a livello di riga
- Può essere accessibile **solo** nei trigger di istuzione

In Oracle è sempre permesso l'accesso alle tabelle mutanti.

Esempio (riordino di prodotti):

Trigger per gestire il riordino di un inventario di magazzino. Quando la quantità di un prodotto in magazzino scende al di sotto di una determinata soglia deve essere emesso un nuovo ordine per il prodotto.

Inventory (Part# , QtyOnHand, ThresholdQty, ReorderQty)

PendingOrders(Part# , OrderDate, OrderedQty)

Evento:

- Aggiornamento della quantità disponibile per il prodotto x
- Inserimento di un uovo prodotto x

Semantica di esecuzione:

- Dopo evento di modifica
- Esecuzione separata per ogni riga della tabella Inventory

Condizione:

- La quantità disponibile è inferiore ad una determinata soglia e non ci sono ordini in sospeso per il prodotto x

Azione:

- Emette un ordine con una determinata quantità di riordino per prodotto x

Corpo del trigger

```
CREATE TRIGGER Reorder
AFTER UPDATE OF QtyOnHand OR INSERT ON Inventory
FOR EACH ROW
WHEN (NEW.QtyOnHand < NEW.ThresholdQty)
DECLARE
N number;
BEGIN
select count(*) into N
from PendingOrders
where Part# = :NEW.Part#;
If (N==0) then
insert into PendingOrders(Part#,OrderedQty,OrderDate)
values (:NEW.Part#, :NEW.ReorderQty, SYSDATE);
end if;
END;
```

Linee guida per scrivere un trigger in Oracle.

La modalità INSTEAD OF è consentita in Oracle, ma dovrebbe esser evitata.

L'uso dei **trigger before** in Oracle per essere conformi allo standard:

- Le modifiche della variabile NEW nelle tuple interessate dall'istruzione di attivazione sono consentite nei trigger di tipo before
- Altre modifiche della base di dati, oltre a quelle riportate al punto precedente, non sono consentite nei **trigger before non può modificare altre tabelle**
- I trigger di tipo before non possono innescare altri trigger

Progettazione dei trigger.

La comprensione delle interazioni reciproche tra i trigger è più complessa, l'azione di un trigger può essere l'evento di un trigger differente e come detto prima, è possibile un'esecuzione infinita.

Proprietà di esecuzione del trigger.

- Terminazione: per uno stato arbitrario del database e una transazione utente, l'esecuzione del trigger termina in uno stato finale, anche dopo un abort
- Confluenza: per uno stato arbitrario del database e una transazione utente, l'esecuzione di un trigger **termina in un unico stato finale**, indipendentemente dall'ordine di esecuzione dei trigger.

La terminazione è la proprietà più importante e la confluenza è assicurata da un'esecuzione **deterministica** del trigger.

Garantire la terminazione.

La terminazione è garantita in fase di esecuzione, interrompendo l'esecuzione del trigger dopo un determinato numero di attivazione di trigger in cascata. La terminazione può essere verificata in fase di progettazione mediante il **grafo di attivazione dei trigger** con un nodo per ogni trigger, un arco diretto $T_i \rightarrow T_j$, se il trigger T_i sta eseguendo un'azione che innasca il trigger T_j .

Un ciclo nel grafo mostra esecuzioni che potrebbero causare una condizione di non terminazione.

Esempio (gestione dei salari):

Trigger per la gestione degli importi salariali: quando viene superato un determinato valore medio di stipendio, viene applicata automaticamente una riduzione dello stipendio

- Viene fornita la seguente tabella

Employee (Emp# , Ename, ..., Salary)
- Semantica di esecuzione:
 - dopo gli eventi di modifica
 - esecuzione separata per ogni istruzione di aggiornamento
- Nessuna condizione per l'esecuzione

```

CREATE TRIGGER SalaryMonitor
AFTER UPDATE OF Salary ON Employee
FOR EACH STATEMENT
BEGIN
    update Employee
    set Salary = Salary * K
    where 2500 < (select AVG (Salary) from Employee);
END;

```

Il valore di K può essere
K = 0.9 l'esecuzione termina
K = 1.1 esecuzione infinita



Trigger per la gestione dei vincoli.

I trigger vengono sfruttati per imporre vincoli di integrità complessi

Procedura di progettazione

1. Scrivere il vincolo come predicato SQL
2. Identificare eventi violanti
3. Tecnica di gestione dei vincoli nell'azione

Esempio 1:

sono riportate le seguenti tabelle

Supplier S (S# , SName, ...)

Part P (P# , PName, ...)

Supply SP (S# , P# , Qty)

Vincolo da rispettare: un pezzo può essere fornito al massimo da 10 fornitori diversi

- Predicato di vincolo
 - select P#
from SP
group by P#
having count(*) > 10
 - insieme di parti che violano il vincolo
- Eventi
 - insert on SP
 - update of P# on SP
- Azione
 - rifiutare la transazione in violazione
- Semantica di esecuzione
 - *dopo* la modifica
 - *livello di istruzione*
 - per catturare l'effetto dell'intera modifica
 - (Oracle) per consentire l'accesso alla tabella mutante
- (Oracle) Nessuna condizione
 - la condizione non può essere specificata nella clausola WHEN
 - viene verificata nel corpo del trigger
- Progettazione della semantica del trigger Oracle

Codice:

```

CREATE TRIGGER TooManySuppliers
AFTER UPDATE OF P# OR INSERT ON SP
DECLARE
N number;
BEGIN
select count(*) into N
from SP
where P# IN (select P# from SP
              group by P#
              having count(*) > 10);
if (N > 0) then
    raise_application_error (xxx, 'constraint violated');
end if;
END;

```

Esempio 2:

sono riportate le seguenti tabelle

Supplier S (S# , SName, ...)

Part P (P# , PName, ...)

Supply SP (S# , P# , Qty)

Vincoli da rispettare: la quantità di un prodotto fornito non può essere superiore a 1000. Se è maggiore, ridurla a 1000.

I vincoli di controllo non consentono azioni di compensazione, occorre implementare un trigger.

- Predicato di vincolo
 - Qty > 1000
 - è anche la condizione di attivazione
 - Eventi
 - insert on SP
 - update of Qty on SP
 - Azione
 - Qty = 1000
- Semantica dell'esecuzione
 - *prima* che avvenga la modifica
 - il suo effetto può essere modificato prima che il vincolo venga controllato
 - *livello di tupla*
 - ogni tupla viene modificata separatamente

Codice:

```
CREATE TRIGGER ExcessiveQty
BEFORE UPDATE OF Qty OR INSERT ON SP
FOR EACH ROW
WHEN (NEW.Qty > 1000)
BEGIN
:NEW.Qty := 1000;
END;
```

Mantenimento delle viste materializzate:

Le viste materializzati sono query memorizzate in modo persistente nella base dati, forniscono maggiori prestazioni, contengono informazioni ridondanti. I trigger possono essere utilizzati per aggiornare le viste materializzati.

Le tabelle sono:

Studente S (SId , SName, DCId)

Corso di laurea DC (DCId , DCName) •

La vista materializzata è:

Studenti iscritti ES (DCId , TotalStudents)

per ogni corso di laurea, TotalStudents conta il numero totale degli studenti iscritti, definito dalla query

```
SELECT DCId, COUNT(*)
FROM S
GROUP BY DCId;
```

Quindi:

- un nuovo corso di laurea viene inserito nella vista materializzata ES quando il primo studente vi si iscrive
- un corso di laurea viene cancellato dalla vista materializzata ES quando l'ultimo studente lo abbandona

Propagare le modifiche sulla tabella S alla vista materializzata (tabella) ES:

- inserimento di nuove tuple in S (incremento contatore / inserimento nuovo corso laurea)
- cancellazione di tuple da S (decremento / elimino corso laurea)
- aggiornamento dell'attributo DCId in una o più tuple di S (es studente cambia costo di laurea)
- Progettare tre trigger per gestire separatamente ogni modifica dei dati
 - trigger di inserimento, trigger di cancellazione, trigger di aggiornamento
 - tutti i trigger condividono la stessa semantica di esecuzione
- Semantica di esecuzione
 - *dopo* che la modifica ha avuto luogo
 - la tabella ES viene aggiornata dopo che la tabella S è stata modificata
 - *livello di tupla*
 - esecuzione separata per ogni tupla della tabella S
 - significativamente più semplice da implementare
- Evento
 - insert on S
- Nessuna condizione
 - è sempre eseguito
- Azione
 - se la tabella ES contiene il DCId a cui è iscritto lo studente
 - incrementare TotaleStudents
 - altrimenti
 - aggiungere una nuova tupla nella tabella ES per il corso di laurea, con TotalStudents impostato a 1

Trigger di Inserimento.

```
CREATE TRIGGER InsertNewStudent
AFTER INSERT ON S
FOR EACH ROW
DECLARE
N number;
BEGIN
--- verificare se la tabella ES contiene la tupla per il corso di laurea
--- corso NEW.DCId a cui lo studente si iscrive
select count(*) into N
from ES
where DCId = :NEW.DCId;
if (N <> 0) then
    --- la tupla per il corso di laurea NEW.DCId è disponibile in ES
    update ES
    set TotalStudents = TotalStudents +1
    where DCId = :NEW.DCId;
else
    --- nessuna tupla per il corso di laurea NEW.DCId è disponibile in ES
    insert into ES (DCId, TotalStudents)
    values (:NEW.DCId, 1);
end if;
END;
```

Trigger di cancellazione.

- Evento
 - delete from S
- Nessuna condizione
 - è sempre eseguito
- Azione
 - se lo studente era l'unico iscritto al corso di laurea
 - cancellare la tupla corrispondente da ES
 - altrimenti
 - decrementare TotaleStudents

```

CREATE TRIGGER DeleteStudent
AFTER DELETE ON S
FOR EACH ROW
DECLARE
N number;
BEGIN
--- leggere il numero di studenti iscritti al corso di laurea OLD.DCId
select TotalStudents into N
from ES
where DCId = :OLD.DCId;

if (N > 1) then
    --- ci sono molti studenti iscritti
    update ES
    set TotalStudents = TotalStudents - 1
    where DCId = :OLD.DCId;
else
    --- c'è un solo studente iscritto
    delete from ES
    where DCId = :OLD.DCId;
end if;
END;

```

Trigger di aggiornamento.

- Evento
 - Update of DCId on S
- Nessuna condizione
 - è sempre eseguito
- Azione
 - aggiornare la tabella ES per il corso di laurea a cui lo studente *era* iscritto
 - decrementare TotalStudents o cancellare la tupla se l'ultimo studente è stato immatricolato
 - aggiornare la tabella ES per il corso di laurea a cui lo studente *è attualmente* iscritto
 - incrementare TotalStudents, o inserire una nuova tupla se si tratta del primo studente

```

CREATE TRIGGER UpdateDegreeCourse
AFTER UPDATE OF DCId ON S
FOR EACH ROW
DECLARE
N number;
BEGIN
--- leggere il numero di studenti iscritti al corso di laurea OLD.DCId
select TotalStudents into N
from ES
where DCId = :OLD.DCId;
if (N > 1) then
    --- ci sono molti studenti iscritti
    update ES
    set TotalStudents = TotalStudents - 1
    where DCId = :OLD.DCId;
else
    --- c'è un solo studente iscritto
    delete from ES
    where DCId = :OLD.DCId;
end if;
--- controllare se la tabella ES contiene la tupla per il corso di laurea
--- NEW.DCId a cui lo studente è iscritto
select count(*) into N
from ES
where DCId = :NEW.DCId;
if (N <> 0) then
    --- la tupla per il corso di laurea NEW.DCId è disponibile in ES
    update ES
    set TotalStudents = TotalStudents + 1
    where DCId = :NEW.DCId;
else
    --- non ci sono tuple per il corso di laurea NEW.DCId disponibili in ES
    insert into ES (DCId, TotalStudents)
    values (:NEW.DCId, 1);
end if;
END;

```