

Basi di Dati

UniShare

Davide Cozzi
@dlcgold

Gabriele De Rosa
@derogab

Federica Di Lauro
@f_dila

Indice

1	Introduzione	2
2	Introduzione al Corso	3
3	Progettazione Concettuale:ER	11
4	Modello Relazionale	27
5	SQL	28
5.0.1	Interrogazioni in SQL	39

Capitolo 1

Introduzione

Questi appunti sono presi a ldurante le esercitazioni in laboratorio. Per quanto sia stata fatta una revisione è altamente probabile (praticamente certo) che possano contenere errori, sia di stampa che di vero e proprio contenuto. Per eventuali proposte di correzione effettuare una pull request. Link: <https://github.com/dlcgold/Appunti>.

Grazie mille e buono studio!

Capitolo 2

Introduzione al Corso

Il corso di Basi di Dati affronta gli aspetti e i metodi per lo sviluppo di un database in maniera efficiente, aspetto fondamentale per un informatico e per lo sviluppo ottimale di software.

Il corso si divide in 7 parti:

1. introduzione generale
2. metodologie e modelli per il progetto delle basi di dati
3. progettazione concettuale
4. modello razionale
5. progettazione logica
6. linguaggio SQL
7. algebra relazionale

Le **informazioni** fanno parte delle risorse di un'azienda, soprattutto negli ultimi anni di clima globale, per cui la gestione efficiente ed ottimale dei dati è fondamentale, come ad esempio Facebook ed Amazon fanno ampio uso dei nostri dati, sia a scopi pubblicitari sia a scopi di marketing.

Un sistema informativo è una componente di un'organizzazione che gestisce le informazioni d'interesse, non per forza attraverso un'automatizzazione e/o supporto di un calcolatore, infatti sin dall'antichità le banche tenevano traccia dei depositi tramite un archivio cartaceo.

Una porzione automatizzata del sistema informativo si chiama sistema informatico, che si divide in:

- acquisizione e memorizzazione

- aggiornamento
- interrogazione
- elaborazione

Le informazioni vengono gestite in vari modi, attraverso il linguaggio naturale, graficamente con schemi e/o numeri, con il tempo si è arrivati a codifiche standard per quasi tutte le tipologie di informazioni, e sono rappresentate nei sistemi informatici dai *dati*, la cui differenza è che i dati sono valori senza alcun valore mentre le informazioni stabilisce un'interpretazione attribuendo una semantica ai valori.

I dati sono una risorsa strategica in quanto sono stabili nel tempo, infatti solitamente i dati sono immutati durante una migrazione tra un sistema e un altro, per questo lo sviluppo progettazione di un database rimane stabile in teoria, senza notevoli cambiamenti durante la durata di un sistema informativo.

Un **Data Base** è una collezione di dati usati per rappresentare le informazioni di interesse di un sistema informativo, definite solo una volta a cui un insieme di applicazioni ed utenti può accedere ad essi, mentre un **DBMS** è un software per la gestione di un database.

I dati presenti in un database sono molti, indipendenti dal programma in cui vengono utilizzati e cui si cerca di evitare la ridondanza, per garantire la consistenza delle informazioni.

Per la creazione di un database, al fine di garantire privacy, affidabilità, efficienza ed efficacia, sono presenti le seguenti tre fasi:

1. definizione
2. creazione e popolazione
3. manipolazione

Un'organizzazione è divisa in vari settori e ogni settore ha un suo sottosistema informativo, non necessariamente disgiunto, e i database solitamente sono condivisi, al fine di ridurre la ridondanza delle informazioni, per cui sono presenti meccanismi di autorizzazione e di controllo della concorrenza.

Un database deve essere conservato a lungo termine e si ha una gestione delle **transazioni**: insieme di operazioni da considerare indivisibile, atomico, corretto anche in presenza di concorrenza e con effetti definitivi.

La sequenza di operazioni nel database deve essere eseguita nella sua interezza, per cui l'effetto di transazioni concorrenti deve essere coerente, infatti la conclusione positiva di una transazione corrisponde ad un impegno,

commit, a mantenere traccia del risultato, anche in presenza di guasti e di esecuzioni concorrente.

Come tutti i software, i DBMS devono essere efficienti, utilizzando al meglio memoria e tempo, efficaci e produttivi.

Si hanno delle caratteristiche nell'approccio alla base dati:

- natura autodescrittiva di un sistema di basi di dati: il sistema di basi di dati memorizza i dati con anche una descrizione completa della sua struttura (metadati), per consentire ai DBMS di lavorare con qualsiasi applicazione.
- separazione tra programmi e dati, infatti è possibile cambiare la struttura dati senza cambiare i programmi.
- astrazione dei dati: si usa un modello dati per nascondere dettagli e presentare all'utente una visione concettuale del database.
- supporto di viste multiple dei dati, per cui ogni utente può usare una vista (view) differente del database, contenente solo i dati di interesse per quell'utente.
- condivisione dei dati e gestione delle transazioni con utenti multipli.

I DBMS estendono le funzionalità dei file system, fornendo più servizi ed in maniera integrata.

In ogni base di dati si ha:

- lo **schema**, sostanzialmente invariante nel tempo, che ne descrive la struttura, l'aspetto intensionale.
- l'**istanza**, i valori attuali, che possono cambiare anche molto rapidamente, l'aspetto estensionale "concreto".

Per lo sviluppo delle basi di dati si hanno due tipi di modelli, ambedue importanti:

- **modelli logici**, adottati nei DBMS esistenti per l'organizzazione dei dati, utilizzati dai programmi e sono indipendenti dalle strutture fisiche
- **modelli concettuali** che permettono di rappresentare i dati in modo indipendente da ogni sistema, con il fine di descrivere i concetti del mondo reali; sono usati nelle fasi preliminari di progettazione e il più diffuso è il modello **Entity-Relationship**.

Un database è organizzato solitamente attraverso i tre schemi dell'architettura ANSI/SPARC:

- schema logico: descrizione dell'intera base di dati nel modello logico “principale” del DBMS, ossia si definisce la struttura concettuale del database, senza considerare l'implementazione fisica nel DBMS.
- schema fisico: rappresentazione dello schema logico per mezzo di strutture fisiche di memorizzazione
- schema esterno: descrizione di parte della base di dati in un modello logico (“viste” parziali, derivate, anche in modelli diversi)

L'accesso ai dati avviene solo mediante il livello esterno, il quale a volte coincide con quello logico, e si hanno 2 forme di indipendenza: quella fisica, in cui è possibile interagire con il DBMS senza conoscere la struttura fisica e quella logica, in cui si può accedere al livello esterno senza interagire con lo schema logico.

Per la definizione dei database ci sono quattro tipologie di linguaggi:

- **DLL**(Data Manipulation Languages) linguaggio per definire i dati
- **DML**(Data Manipulation Languages) linguaggio per la manipolazione dei dati
- **DCL**(Data Control Languages) linguaggio per il controllo degli accessi al database
- **DQL**(Data Query Languages) linguaggio per effettuare delle interrogazioni al database

Noi vedremo SQL per definire i database, linguaggio basato sull'algebra relazionale che implementa tutti e 4 le tipologie di linguaggi per la gestione di un database.

Si hanno due tipi di utenti:

1. **utenti finali (terminalisti)**: eseguono applicazioni predefinite (transazioni)
2. **utenti casuali**: eseguono operazioni non previste a priori, usando linguaggi interattivi

inoltre si hanno:

- progettisti e realizzatori di DBMS

- progettisti della base di dati e amministratori della base di dati (DBA), persona o gruppo di persone responsabile del controllo centralizzato del database, in tutti i suoi aspetti.
- progettisti e programmatori di applicazioni



Come visto anche nel corso di analisi e progettazione del software e nella figura X, durante lo sviluppo di un sistema informatico si hanno le seguenti fasi:

- **studio di fattibilità**, definizione di costi, priorità e competenze per un progetto
- **raccolta e analisi dei requisiti**, ovvero lo studio delle proprietà del sistema
- **progettazione** di dati e funzioni
- **implementazione**
- **validazione e collaudo**, che comprendono anche test da parte del cliente
- **funzionamento**, ovvero lo stadio finale dove il sistema diventa effettivamente operativo

In questo corso ci occuperemo soltanto delle fasi di progettazione ed implementazione di base di dati, lasciando l'analisi delle altre fasi al corso ed ai libri di Ingegneria del Software.

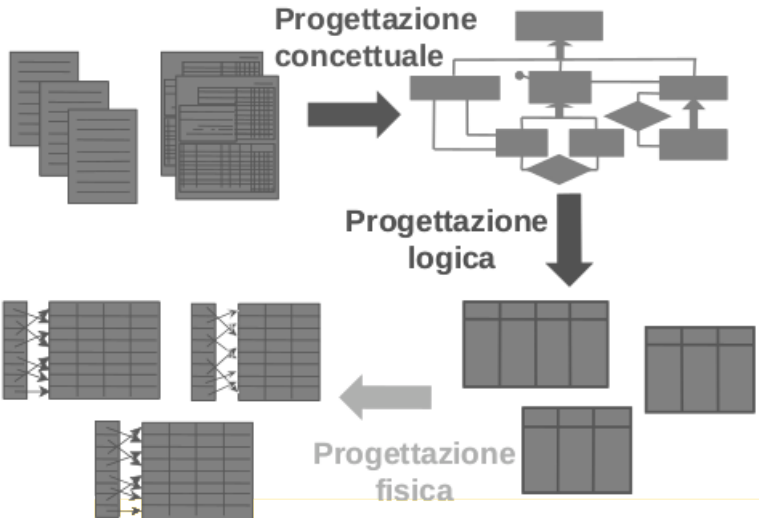
In questo processo a rimanere stabili sono prettamente i **dati** infatti prima si progetta la base dati, con una **metodologia di progetto**, e poi il software.

Ciclo di vita (modello a spirale)



Una **metodologia** è un'articolazione in fasi di guida ad un'attività di progettazione, in caso di una base dati la metodologia, con il fine di separare il cosa rappresentare dal come, è la seguente:

- suddivida la progettazione in fasi indipendenti
- fornisca strategie e criteri di scelta in caso di alternative
- fornisca modelli di riferimento (i linguaggi)
- garantisca generalità rispetto al problema
- garantisca qualità e facilità d'uso



Come si può notare nella figura X1, la metodologia corrente di modellazione di un database prevede la definizione e l'esecuzione delle seguenti fasi:

- la **progettazione concettuale** consiste nel tradurre i requisiti del sistema informatico in una descrizione formale, integrata e indipendente dalle scelte implementative (DBMS, SW e HW)
- la **progettazione concettuale** consiste nella traduzione dello schema concettuale nel modello dei dati scelto per la modellazione, ottenendo uno schema logico, espresso nel DDL del DBMS.
In questa fase si considerano anche aspetti legati ai vincoli ed all'efficienza. Si hanno due sotto-fasi:

- ristrutturazione dello schema concettuale
- traduzione verso il modello logico
- la **progettazione fisica** completa lo schema logico ottenuto con le specifiche proprie del DBMS scelto. Il risultato è lo schema fisico che descrive le strutture di memorizzazione ed accesso ai dati

Incominciamo nel prossimo capitolo a considerare la progettazione concettuale, per poi analizzare nei successivi capitoli in dettaglio anche la fase di progettazione concettuale e il linguaggio SQL.

Capitolo 3

Progettazione Concettuale:ER

Come abbiamo già introdotto nel precedente capitolo, affrontiamo ora la progettazione concettuale di una base dati attraverso il modello **ER** (Entity Relationships), modello concettuale che fornisce una serie di strutture atte a descrivere in maniera semplice e facile la realtà di interesse da modellare.

Si hanno dei vantaggi con la progettazione concettuale, prevale infatti l'aspetto intensionale indipendente dalla tecnologia ed è una rappresentazione grafica ed è utile per la documentazione, in quanto facilmente comprensibile anche da persone poco avvezze alla tecnologia e ai database.

In uno schema ER si hanno i seguenti costrutti, come si nota nella figura Y, la cui rappresentazione effettiva varia in quanto vi sono più versioni di ER:

- **entità**: classe di oggetti con proprietà comune ed esistenza "autonoma", della quale si vogliono specificare fatti specifici; ogni entità ha un nome univoco, espressivo e al singolare.
- **relazione**: rappresentano legami logici, significativi per la realtà da modellare, tra due o più entità.
Un'occorrenza di relazione è un n-upla costituita da occorrenze di entità, una per ciascuna delle entità coinvolte, e ogni relazione ha un nome univoco, in cui è preferibile assegnare un sostantivo per evitare di stabilire un verso alla relazione.
Essendo una relazione matematica tra le entità coinvolte non è possibile avere delle n-uple identiche, con conseguenze per la realtà da rappresentare, per esempio non è possibile attraverso una relazione il fatto che è possibile ripetere un esame, obbligando a rappresentare l'esame come entità e non più come relazione.

- **attributo semplice:** associa ad ogni istanza di entità o associazione un valore, definito su un dominio di valori, specificato nella documentazione associata, con il fine di descrivere le proprietà elementari di entità e/o relazioni disegnate per rappresentare la realtà d'interesse.
- **attributo composto:** raggruppamento di attributi di una medesima entità/relazione con affinità di significato e/o uso come ad esempio possiamo raggruppare gli attributi Via, Numero Civico e Cap dall'entità persona per formare l'attributo composto Indirizzo.
- **cardinalità delle relazioni:** vengono specificate per ogni relazione e descrivono il numero minimo e massimo di occorrenze di relazione, a cui una occorrenza dell'entità può partecipare alla relazione, ossia quante volte un'occorrenza di un'entità può essere legata ad occorrenze delle altre entità coinvolte.
È possibile assegnare un qualunque intero non negativo, con l'unico vincolo che la cardinalità minima sia minore o uguale alla cardinalità massima e di solito si usano i valori $0, 1eN$, indicanti zero, una o molte occorrenze, senza preoccuparsi in caso di N del numero effettivo di occorrenze.
- **cardinalità di un attributo:** descrivono il numero minimo e massimo di valori dell'attributo associati all'entità e/o relazione, con la cardinalità $(1, 1)$ stabilita come default, che può essere vista come funzione che associa ad ogni occorrenza di entità un solo valore dell'attributo; si hanno le stesse consuetudini delle cardinalità delle relazioni.
- **identificatore interno:** permette di identificare in maniera univoca un'entità ed un identificatore è interno in caso sia uno o più attributi di un'entità, tutti con cardinalità $(1, 1)$.
- **identificatore esterno:** un identificatore è esterno, in caso un'entità E viene identificata da un'attributo di un'entità F , cui esiste una relazione uno a uno tra l'entità E e F .
È possibile, anche se molto raro, avere la definizione dell'identificatore usando entità di entità, ossia l'identificatore dell'entità E viene definito nell'entità G , cui esiste una relazione con l'entità F che a

sua volta ha una relazione con l'entità E , ma si può capire da quanto è contorto il ragionamento qual'è la sua percentuale d'utilizzo nella modellazione dello schema ER.

- **generalizzazione:** rappresentano legami logici tra un'entità E , detta padre, e una serie di entità E_1, E_2, \dots, E_n , dette figlie, di cui l'entità E rappresenta un caso generale della serie di entità figlie.

Tra le entità coinvolte in una generalizzazione valgono le seguenti proprietà:

- ogni occorrenza di un'entità figlia è anche un'occorrenza dell'entità genitore.
- ogni proprietà dell'entità genitore è anche una proprietà delle entità figlie, quindi nello schema ER non devono essere rappresentate, e ciò prende il nome di ereditarietà.




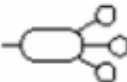




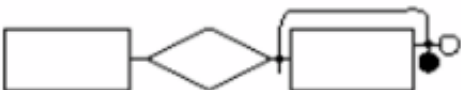


Le generalizzazioni possono essere classificate sulla base di due proprietà ortogonali:

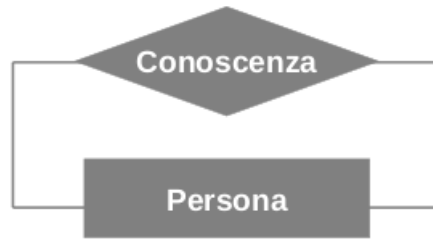
- una generalizzazione è totale se ogni occorrenza del genitore è un'occorrenza di almeno uno dei figli, altrimenti è parziale.
 - una generalizzazione è esclusiva se ogni occorrenza del genitore è al più un'occorrenza di una delle entità figlie, altrimenti è sovrapposta.
- **sottoinsieme:** generalizzazione con soltanto un'entità figlia, di cui solitamente rappresenta una parte dell'entità genitore come ad esempio gli studenti sono un sottoinsieme delle persone.

Un'occorrenza, o istanza, di un'entità, è un oggetto della classe che l'entità rappresenta, ma noi rappresentiamo le entità dello schema concettuale non le singole istanze, in quanto esse sono variabili nel tempo a differenza della struttura, concetto fondamentale per definire un modello funzionale.

Nessuno impone un tipo per un certo attributo, possono essere anche complessi di cui però non ci interessano le informazioni che lo rappresentano, come ad esempio una foto può essere un attributo, ma se ho bisogno dell'autore della foto, non sarà più un attributo ma un'altra entità.

Un'**istanza di associazione** è una combinazione o aggregazione di istanze di entità che prendono parte all'associazione (per esempio "prof. Schettini")

Construct	Graphical representation
Entity	
Relationship	
Simple attribute	
Composite attribute	
Cardinality of a	
Cardinality of an attribute	
Internal identifier	 
External identifier	
Generalization	
Subset	

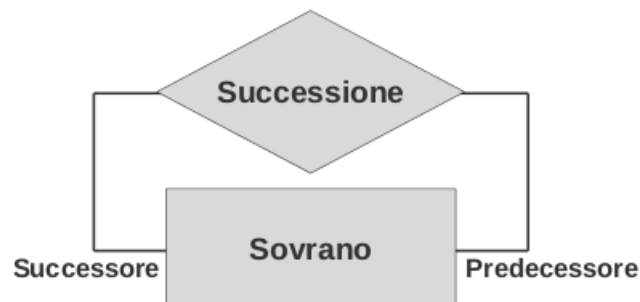


è istanza di associazione per l'entità docente).

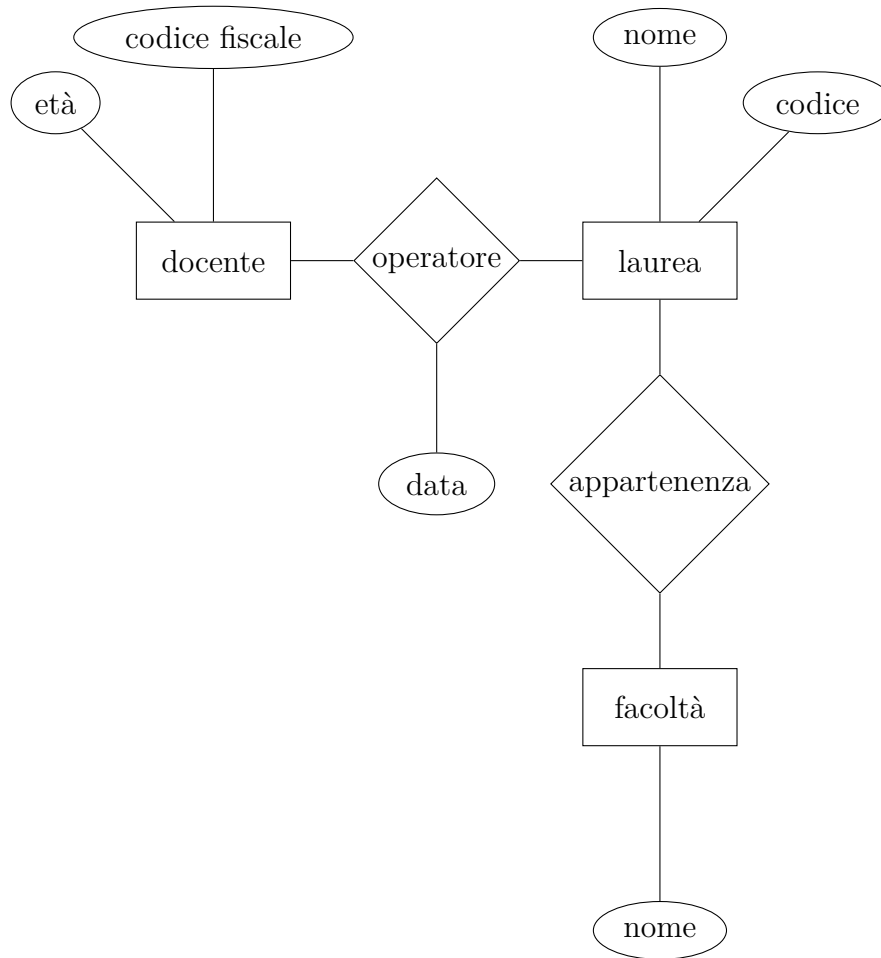
Le relazioni possono avere attributi, con valori specificati in un certo dominio, che modella una proprietà del legame tra tutte le entità rappresentato dalla relazione.

Una associazione può coinvolgere “due o più volte” la stessa entità e ciò si chiama *associazione ricorsiva o ad anello*, come si nota nella figura XXXX, ed è molto utile per definire le relazioni subordinazione e/o comunicazioni tra istanze di una stessa entità, come nel caso della relazione sovrano, indicante la lista dei sovrani con i predecessori e successori; in queste associazioni è necessario aggiungere la specifica dei **ruoli**, come nell'esempio SDERR, per specificare in maniera chiara e non ambigua quale è il verso della relazione.

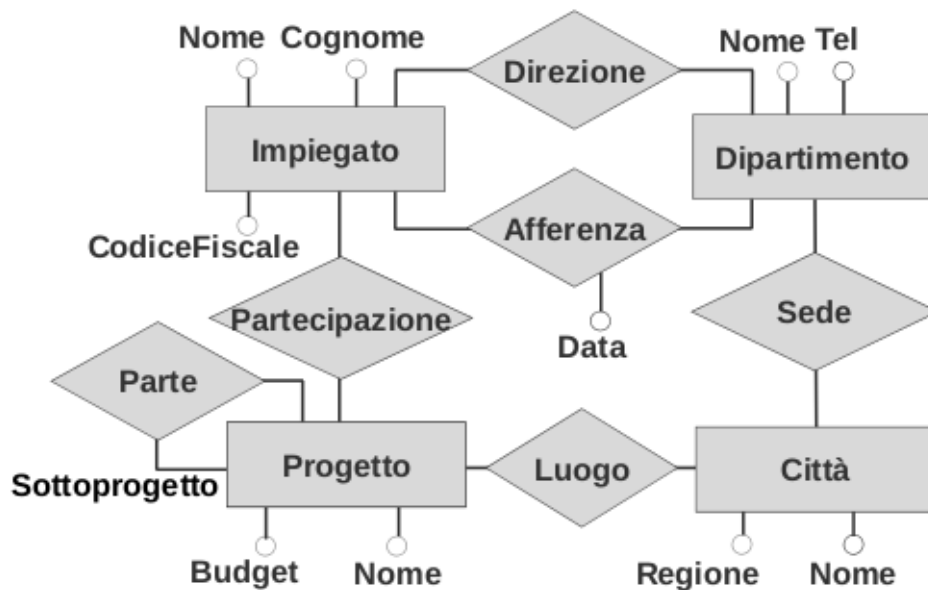
Un'associazione ad anello, ovviamente essendo una relazione, può avere delle proprietà come ad esempio la relazione della figura SJDFHJF è irriflessiva, intransitiva e asimetrica.



Descrivere lo schema concettuale della seguente realtà: I docenti hanno un codice fiscale ed una età. I docenti operano nei corsi di laurea (si dice che afferiscono ai corsi di laurea). Interessa la data di afferenza dei docenti ai corsi di laurea. I corsi di laurea hanno un codice ed un nome, ed appartengono alle facoltà. Ogni facoltà ha un nome



Esempio 1. Descrivere lo schema concettuale della seguente realtà: Degli impiegati interessa il codice fiscale, il nome, il cognome, i dipartimenti ai quali afferiscono (con la data di afferenza), ed i progetti ai quali partecipano. Dei progetti interessa il nome, il budget, e la città in cui hanno luogo le corrispondenti attività. Alcuni progetti sono parti di altri progetti, e sono detti loro sottoprogetti. Dei dipartimenti interessa il nome, il numero di telefono, gli impiegati che li dirigono, e la città dove è localizzata la sede. Delle città interessa il nome e la regione:



Nella scelta del costrutto ideale, per rappresentare in maniera fedele e corretta la realtà, si usano le seguenti "regole":

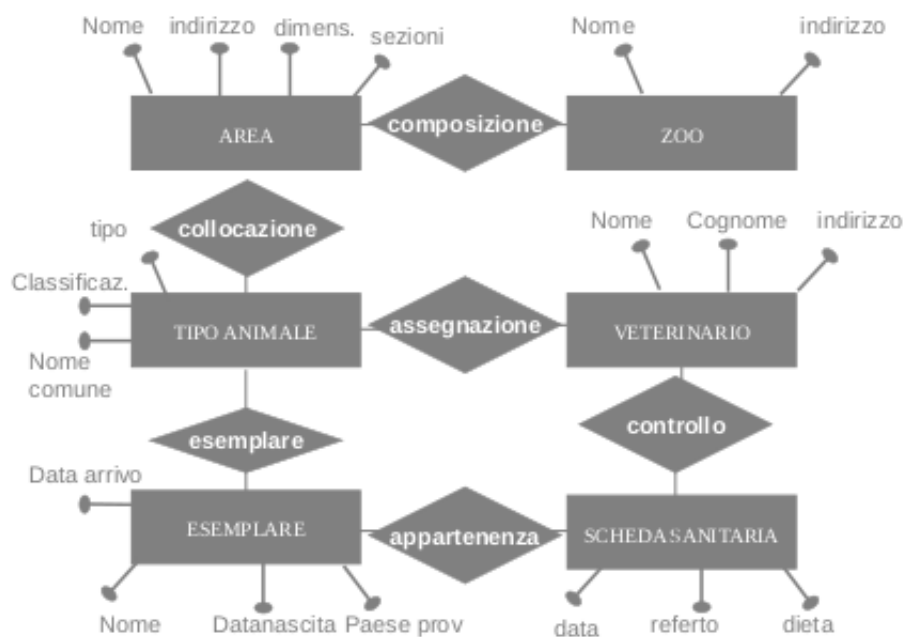
- **entità:** si usa un'entità in caso vengano rispettate le seguenti proprietà: le sue istanze sono significative indipendentemente dalle altre istanze, se si ha o si può avere delle proprietà indipendenti dagli altri concetti oppure se il concetto da rappresentare è importante per la realtà da modellare.
- **attributo:** si usa un'attributo in caso succedono i seguenti avvenimenti: non ha senso considerare una sua istanza a se stante ad altre istanze, le istanze non sono concettualmente significative oppure serve soltanto rappresentare una proprietà locale di un altro concetto.
- **relazione:** si dovrebbe usare una relazione in caso non ha senso pensare alla partecipazione delle sue istanze ad altre relazioni e

le sue istanze non sono significative indipendentemente da altre istanze, ossia una sua istanza ha senso se messa in relazione con altre istanze, provenienti da altri costrutti.

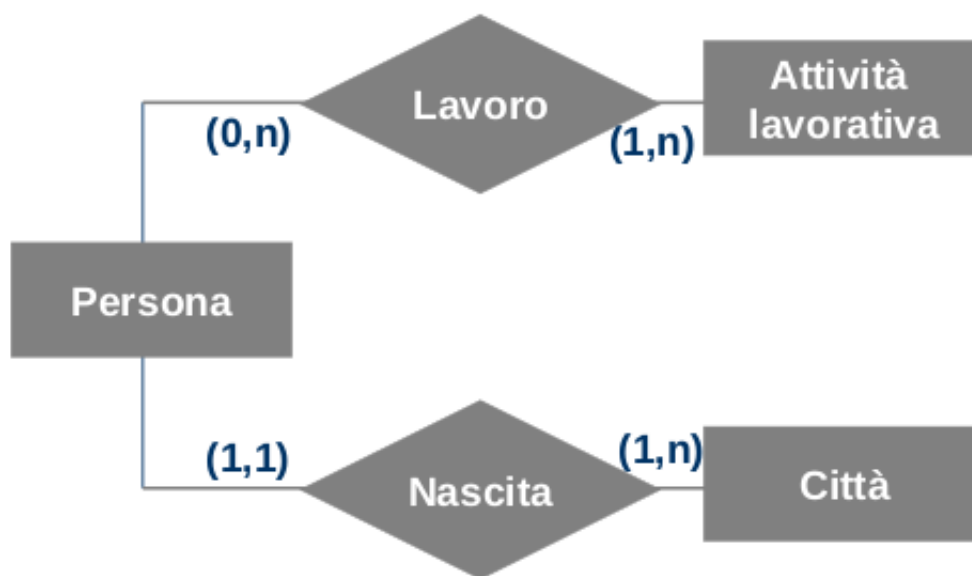
•Calcio: un giocatore gioca in una squadra



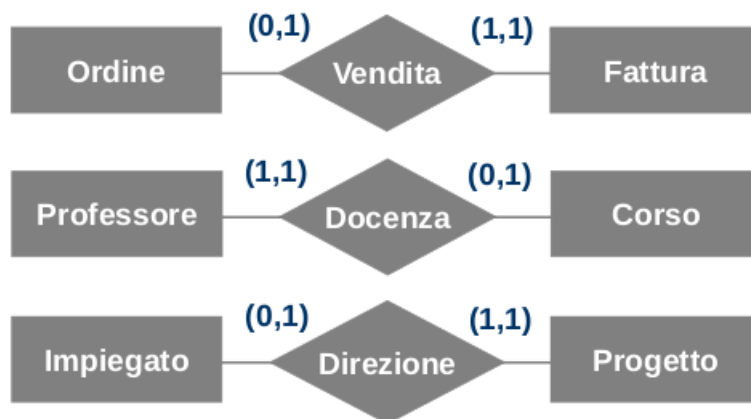
- Esempio 2.**
- ogni zoo è diviso in aree diverse a seconda che si tratti di rettili, pesci, uccelli, scimmie, grandi mammiferi, ... Ogni area è dotata di: nome, indirizzo, dimensione, numero di sezioni.
 - per ogni tipo di animale ci sono informazioni che riguardano: classificazione zoologica, nome comune (giraffa, elefante, serpente, tartaruga, ...), habitat, alimentazione, ... Per ogni tipo di animale c'è un diverso veterinario specialista, dipendente dello zoo.
 - ogni tipo di animale è rappresentato da esemplari e relativi dati anagrafici: nome proprio (giraffa Enrico, giraffa Giulia, ...), data di nascita, Paese di provenienza, data di arrivo allo zoo, ...
 - ogni esemplare è dotato di più schede sanitarie contenenti ognuna: la data della visita, referto, dieta, nome del veterinario, ...



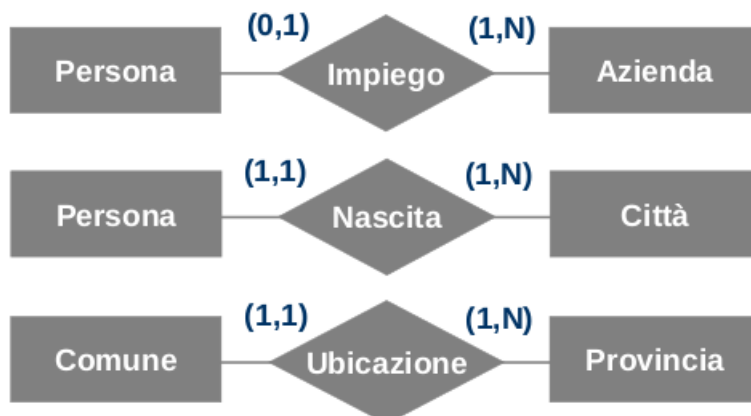
lo zoo ha degli attributi che vanno indicati anche se non richiesti dalla traccia per evitare ambiguità. Dato che ogni scheda è collegata ad un veterinario, di cui ho un'entità la collego a veterinario con una relazione e non ne faccio un attributo



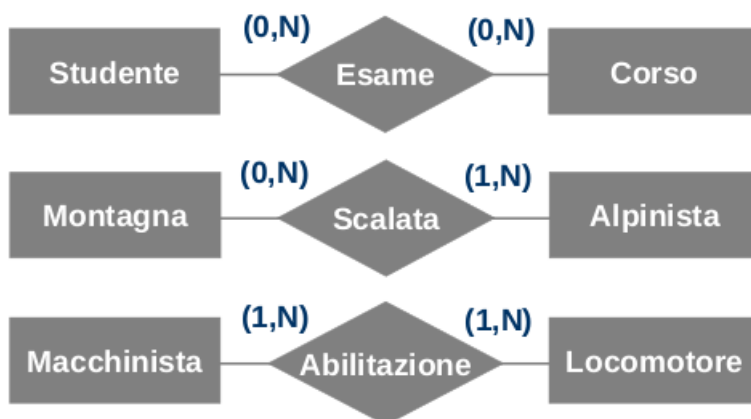
Relazioni “uno a uno”



Relazioni “uno a molti”



Relazioni “molti a molti”



vediamo un esempio:

Esempio 3. *si ha:*



- $\text{min-card}(\text{Automobile}, \text{Proprietario}) = 0$: esistono automobili non possedute da alcuna persona
- $\text{min-card}(\text{Persona}, \text{Proprietario}) = 0$: esistono persone che non posseggono alcuna automobile
- $\text{max-card}(\text{Persona}, \text{Proprietario}) = n$: ogni persona può essere proprietaria di un numero arbitrario di automobili
- $\text{max-card}(\text{Automobile}, \text{Proprietario}) = 1$: ogni automobile può avere al più un proprietario



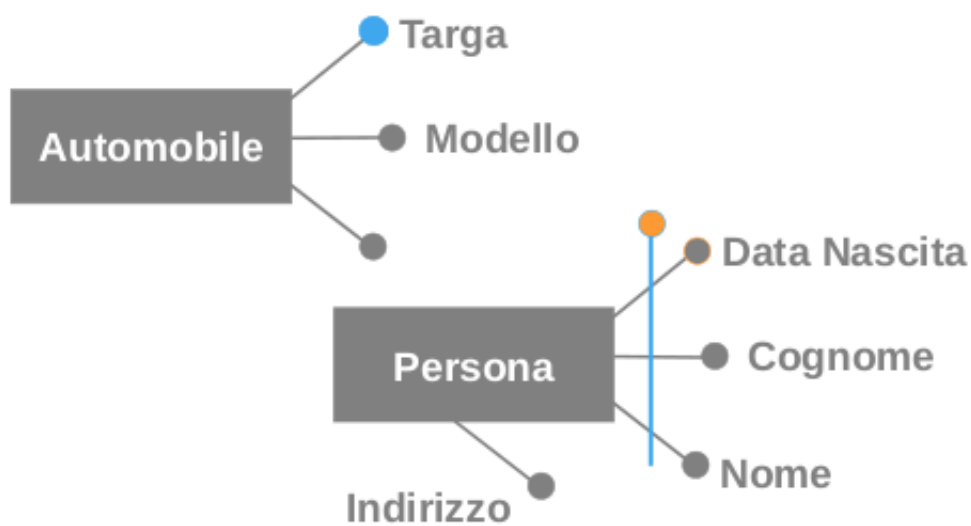
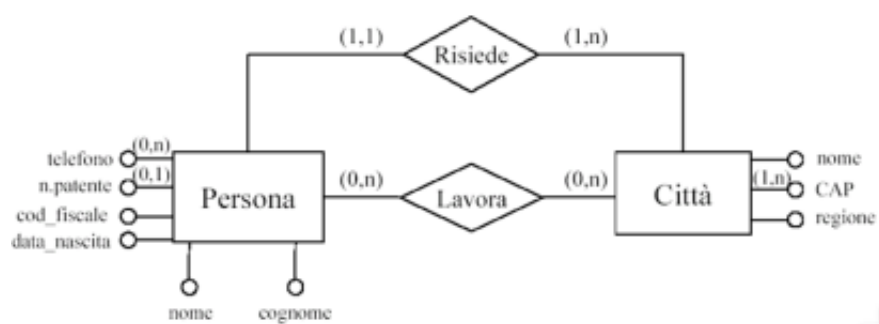
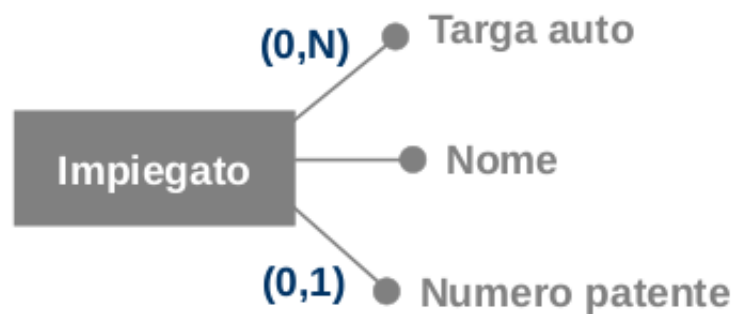
Istanza dello schema:

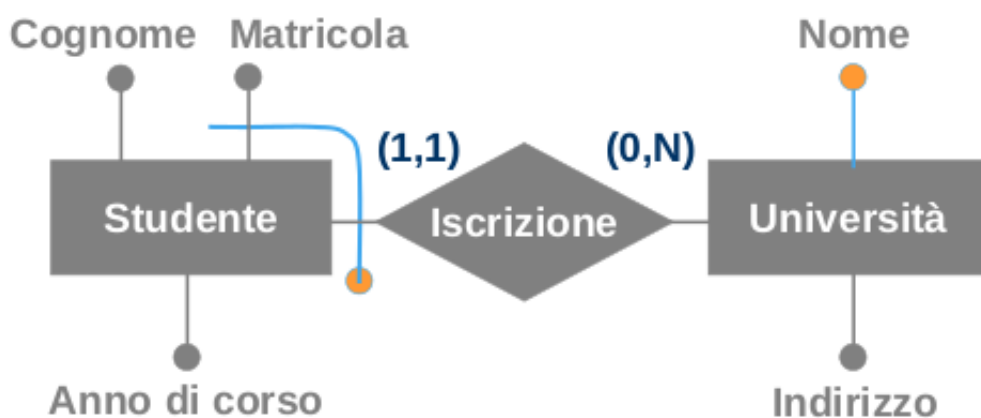
Istanze(Impiegato) = { a,b,c }

istanze(Progetto) = { x,y,v,w,z }

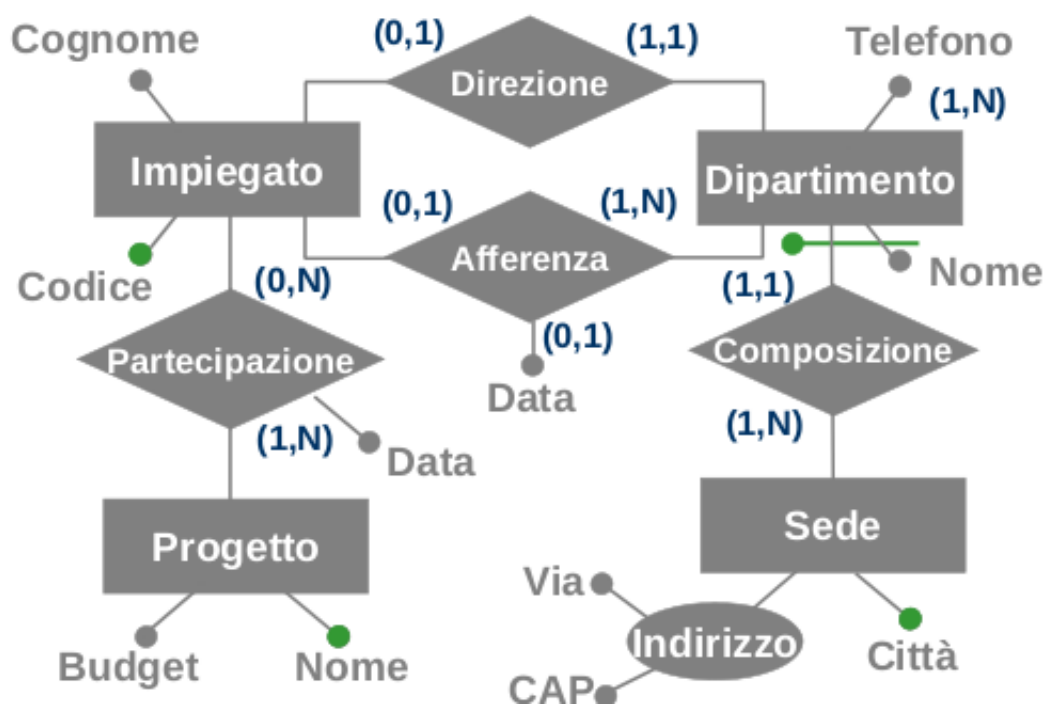
istanze(Assegnazione) = { (a,w), (b,v), (b,w), (c,y), (c,w), (c,z) }

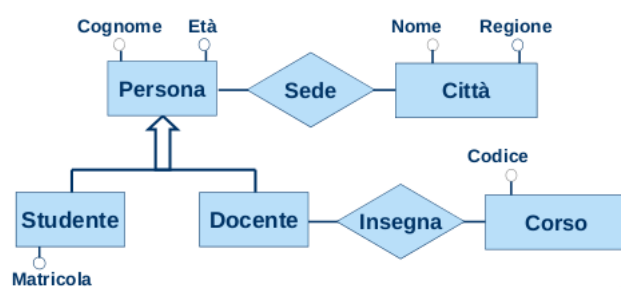
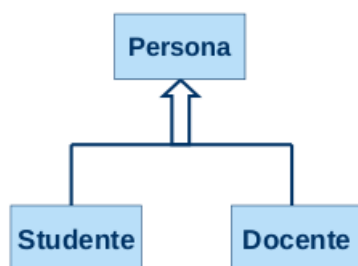
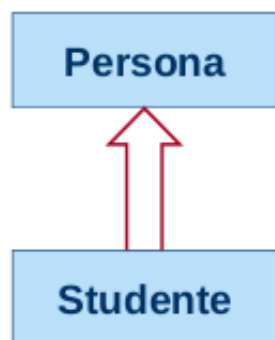
Ad ogni impiegato sono assegnati da 1 a 5 progetti e ogni progetto è assegnato ad al più 50 impiegati. a,b,c compaiono in almeno una istanza di Assegnazione. x non compare nelle istanze di Assegnazione. Infine ci sono progetti (ad esempio lanciati da poco tempo) che possono non essere assegnati a nessun impiegato

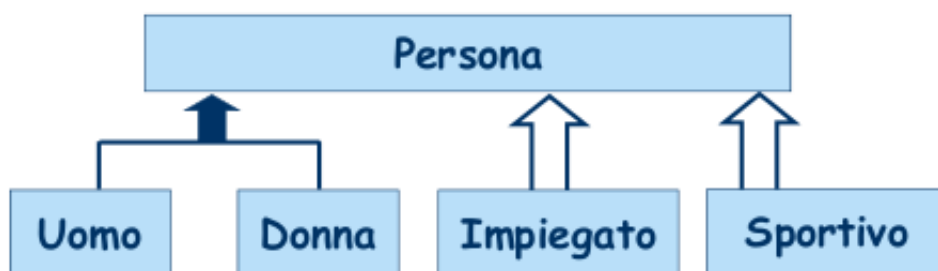
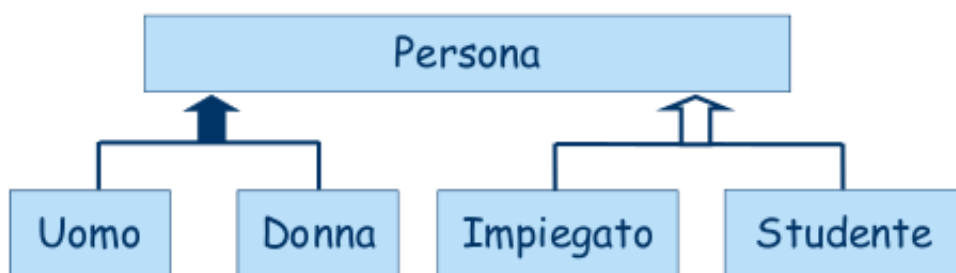
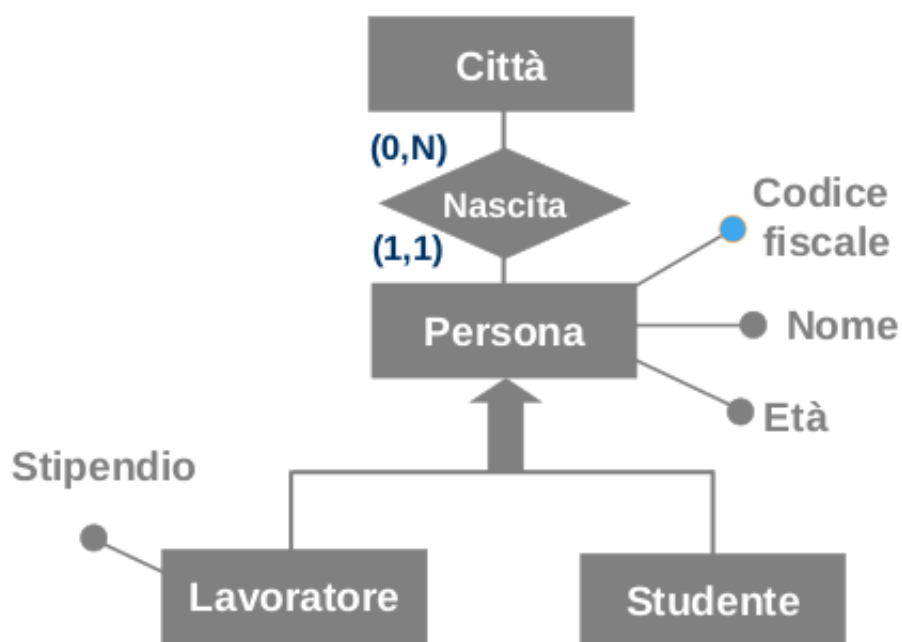


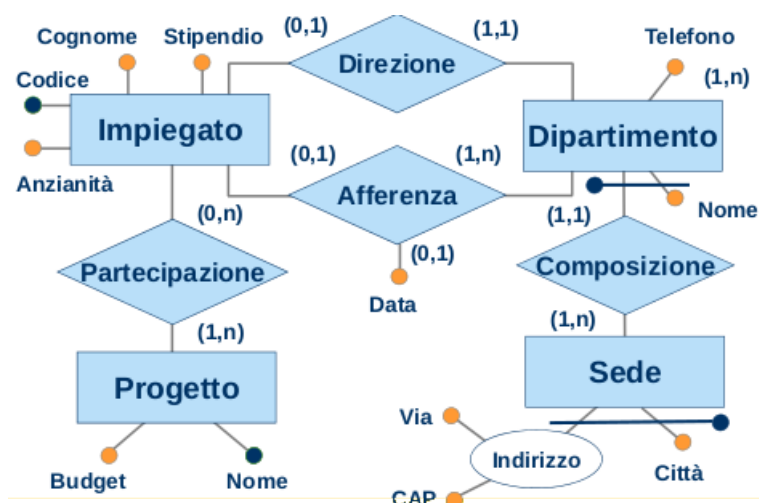


vediamo un esempio complesso, che è preso da un vecchio esempio:

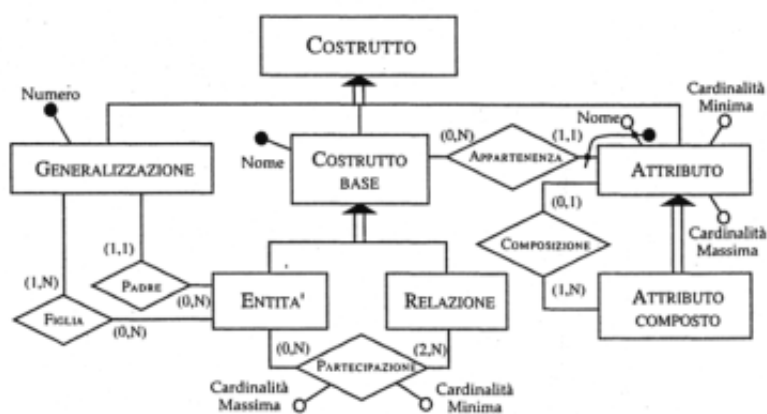








Vincoli di integrità esterni
(1) Il direttore di un dipartimento deve afferire a tale dipartimento da almeno 5 anni
(2) Un impiegato non deve avere uno stipendio maggiore del direttore del dipartimento al quale afferisce
(3) Un dipartimento con sede a Roma deve essere diretto da un impiegato con più di dieci anni di anzianità
(4) Un impiegato non può partecipare ad un numero di progetti maggiore di due volte il numero di dipartimenti ai quali afferisce



Capitolo 4

Modello Relazionale

Dopo aver introdotto il modello concettuale, specifichiamo ed analizziamo ora il modello logico necessario per rappresentare in maniera efficace ed efficiente la base dati modellata.

Analizziamo il modello relazionale, il più diffuso dei modelli dei dati, in cui si utilizzano le tabelle e le relazioni per ottenere la rappresentazione voluta, introdotto negli anni '70 con l'obiettivo di rendere indipendenti i dati, ossia separare la modellazione “concettuale” dall'effettiva implementazione fisica sul DBMS.

Il modello relazionale, presenta una notevole differenza dai modelli precedenti, dato che i riferimenti fra dati in strutture diverse sono rappresentati per mezzo dei valori stessi, garantendo così l'indipendenza dei dati.

Il concetto di relazione, ovviamente proviene dalla teoria degli insiemi, rappresenta un sottoinsieme del prodotto cartesiano ma nel modello relazionale si utilizza una variante di esso, ossia ad ogni dominio, su cui è definito il prodotto cartesiano, viene associato un nome, al fine di superare la struttura posizionale della relazione matematica.

Questo aspetto di utilizzare un'attributo con una notazione non posizionale ci garantisce di poter accedere ai campi di una tupla mediante un nome e soprattutto ci evita di stabilire un ordine tra i diversi campi, ad esempio chi stabilisce che la matricola sia prima del nome e non viceversa.

Capitolo 5

SQL

Il linguaggio SQL è un linguaggio per la definizione e la manipolazione dei dati in database relazionali, sviluppato originariamente presso il laboratorio IBM a San Jose' (California) e adottato nel sistema System R. L'SQL è stato poi adottato da molti altri DBMS e quindi è stato soggetto ad un'intensa attività di standardizzazione. È un linguaggio con varie funzionalità che contiene:

- **DDL:** definizione di domini, tabelle, autorizzazioni, vincoli, procedure, ecc.
- **DML:** linguaggio di query, modifica, ...

Si studierà una versione vecchia di SQL in quanto è la più implementata. Nello specifico studieremo SQL-2. SQL-3 è comunque compatibile con SQL-2 e introduce il concetto di oggetto. Noi useremo SQL per definire, aggiornare e interrogare la base di dati. Sono sempre più frequenti in realtà sistemi dotati di interfacce più facili da usare. Questi programmi generano le istruzioni SQL corrispondenti.

Esistono diverse implementazioni di SQL, per esempio il DBMS di Oracle, Access, Mysql...

SQL è relazionale completo e ogni espressione dell'algebra relazionale può essere tradotta in SQL. Il modello dei dati di SQL è basato su tabelle anziché relazioni e possono essere presenti righe duplicate (le tuple) e adotta la logica a 3 valori dell'algebra relazionale (gestendo il null e aggiungendo il valore di verità U, *unknown*).

Le tabelle di verità diventano quindi:

Tabelle di verità

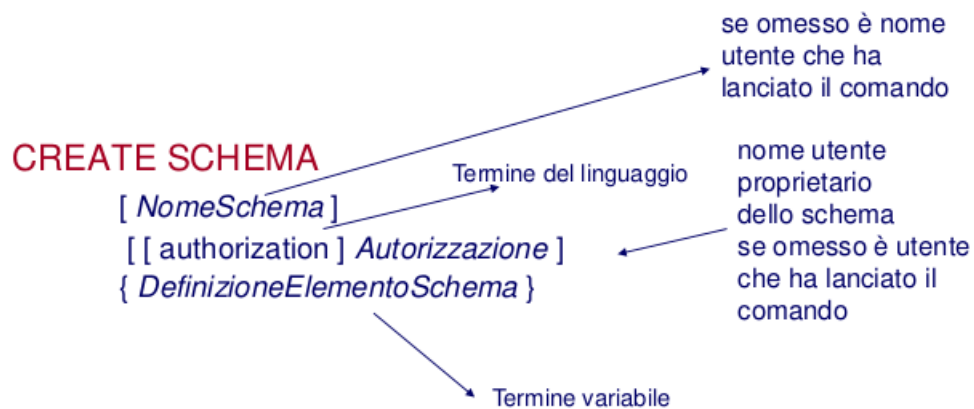
<i>not</i>		<i>and</i>	F T U	<i>or</i>	F T U
T	F	T	F T U	T	T T T
F	T	F	F F F	F	F T U
U	U	U	F U U	U	U T U

Vediamo la notazione per definire la sintassi di SQL:

- $\langle x \rangle$ per isolare un termine x
- $[x]$ indicano che un termine x è opzionale
- $\{x\}$ indicano che un termine x può essere ripetuto 0 volte o un numero arbitrario di volte
- $|$ separa opzioni alternative

Le parentesi tonde () appartengono al linguaggio SQL e non alla notazione sopra descritta.

Uno **schema di base di dati** è una collezione di oggetti e ogni schema ha un nome e un proprietario. SI ha la seguente sintassi:



Dopo il comando CREATE SCHEMA compaiono le definizioni dei vari elementi. Non è necessario che la definizione di tutti gli elementi avvenga contemporaneamente alla creazione dello schema. Può avvenire in più fasi successive.

Una **tabella** è costituita da una collezione ordinata di attributi e da un insieme (eventualmente vuoto) di vincoli:

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

questa istruzione quindi definisce uno schema di relazione e ne crea un'istanza vuota. Per ogni attributo va specificato il dominio, un eventuale valore di default (per questo è comodo il null, che è il valore di default di base) ed eventuali vincoli. Possono essere espressi altri vincoli a livello di tabella (tra più attributi).

I **domini** specificano i valori ammissibili per gli attributi di una relazione. SQL ha 6 domini predefiniti:

1. bit SQL-2 poi eliminato e sostituito parzialmente da BOOLEAN in SQL-3
2. carattere
3. numerico Esatto
4. numerico Approssimato
5. data/Ora
6. intervallo Temporale

più ovviamente i domini definiti dall'utente.

Vediamo il **dominio di tipo Bit**. Può essere di lunghezza fissa o variabile. Se la lunghezza non è specificata corrisponde ad 1 singolo valore. Corrisponde ad attributi che possono assumere solo due valori (0,1). Attributi di questo tipo (*flag*) indicano se l'oggetto rappresentato possiede o meno una certa proprietà:

```
bit [varying] [(lunghezza)]  
bit  
bit(lunghezza)  
bit varying (lunghezza)  
varbit (lunghezza)
```

Esempio 4. *Si definisce l'attributo **Lavoratore** nella relazione **STUDENTI** per indicare se lo studente è o meno lavoratore:*

Lavoratore **bit**

Vediamo il **tipo carattere**. Rappresenta singoli caratteri alfanumerici oppure stringhe di lunghezza fissa o variabile:

character [**varying**] [(lunghezza)]
character o **char**
character(lunghezza)
varchar(lunghezza)
character varying (lunghezza)

Esempio 5. *Si definisce l'attributo **Nome** della relazione **IMPIEGATI** come sequenza di caratteri di lunghezza massima 20:*

Nome **char**(20)
Nome **varchar**(20)

con il primo che genera Paolo_Bianchi_ _ _ _ _ e il secondo Paolo_Bianchi

Vediamo i **tipi numerici esatti**. Rappresentano numeri interi o numeri decimali in virgola fissa (con un numero prefissato di decimali, ad esempio i valori monetari). Precision è numero di cifre significative, scala il numero di cifre dopo la virgola:

integer
smallint
numeric **numeric**(precisione) **numeric**(precisione,scala)
decimal **decimal**(precisione) **decimal**(precisione,scala)

Esempio 6. *Si definisce l'attributo **Eta** nella relazione **IMPIEGATI**:*

Eta **decimal**(2) // Rappresenta tutti i numeri fra -99 e +99

*Si definisce l'attributo **Cambio** nella relazione **PAGAMENTO** per il valore del cambio di una certa moneta preciso al centesimo*

Cambio **numeric**(6,2) // Rappresenta tutti i numeri fra -9999,99 e +9999,99

quindi **INTEGER** / **SMALLINT** rappresentano valori interi. La precisione (numero totale di cifre) varia a seconda della specifica implementazione di SQL, non è specificata nello standard. **SMALLINT** richiede minore spazio di

memorizzazione.

NUMERIC / *DECIMAL* rappresentano i valori decimali. La differenza tra *NUMERIC* e *DECIMAL* è che il primo deve essere implementato esattamente con la precisione richiesta, mentre il secondo può avere una precisione maggiore, la precisione segnalata è requisito minimo. Se la precisione non è specificata si usa il valore caratteristico dell'implementazione. Per la scala si usa valore 0. Si hanno ovviamente:

```
float float(precisione)
double precision
real
```

che sono utili per rappresentare valori reali approssimati, ad esempio grandezze fisiche (rappresentazione in virgola mobile, in cui a ciascun numero corrisponde una coppia di valori: mantissa e esponente).

Per la data si ha:

```
date
time time(precisione)
timestamp timestamp(precisione)
```

Ciascuno di questi domini è strutturato e decomponibile in un insieme di campi (anno, mese, giorno, ora, minuti, secondi):

```
DataDiNascita date // 1999-09-18
OraDiConsegna time // 19.24.16
Arrivo timestamp // 2000-09-18 21.15.20
```

SI possono avere intervalli temporali:

```
interval PrimaUnitàDiTempo
interval PrimaUnitàDiTempo to UltimaUnitàDiTempo
```

Infine si hanno i tipi **blob**. Permettono di includere direttamente nel database oggetti molto grandi. *Binary Large Object (BLOB)* e *Character Large Object (CLOB)*:

```
fotografia BLOB(10M)
descrizione CLOB(100k)
```

Definiti solo in SQL-3, ma implementati in diversi DBMS commerciali. Il sistema garantisce solo di memorizzarne il valore. Non possono essere usati come criterio di selezione per le interrogazioni.

Per creare un dominio uso la primitiva *create domain*:

```
CREATE DOMAIN NomeDominio AS DominioElementare
    [ValoreDefault] [Constraints]
```

Un nuovo dominio è caratterizzato dalle seguenti informazioni: nome, dominio elementare, valore di default, insieme di vincoli (constraints). Vediamo un esempio:

```
CREATE DOMAIN Voto AS SMALLINT
    DEFAULT 0
    NOT NULL
```

Il nuovo dominio Voto è definito come uno SMALLINT con valore di default e che non deve essere null.

La definizione di "nuovi domini" è utile perché permette di associare dei vincoli a un nome di dominio: questo è importante quando si deve ripetere la stessa definizione di attributo su diverse tabelle: ad esempio, modifiche alla definizione di Voto si ripercuotono in tutte le occorrenze di questo dominio nello schema del Database.

Esempio 7. *Vediamo un esempio concreto:*

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

```
CREATE TABLE Impiegato (
    Matricola CHAR(6) PRIMARY KEY,
    Nome CHAR(20) NOT NULL,
    Cognome CHAR(20) NOT NULL,
    Dipart CHAR(15)
    Stipendio NUMERIC(9) DEFAULT 0,
    UNIQUE (Cognome, Nome)
)
```

UNIQUE richiede che non ci siano coppie cognome-nome ripetute

Un vincolo (constraint) è una regola che specifica delle condizioni sui valori di un elemento dello schema del database. Un vincolo può essere associato ad una tabella, ad un attributo, ad un dominio. Sono di due tipi:

1. **vincoli intrarelazionali** che si applicano all'interno di una relazione. Possono essere:

```

NOT NULL // (Il valore deve essere non nullo)
UNIQUE // (I valori devono essere non ripetuti)
PRIMARY KEY // (Chiave primaria)
CHECK // (Condizioni complesse)

```

Il vincolo PRIMARY KEY può essere definito una sola volta all'interno della relazione. In alcune implementazioni di SQL potrebbe essere necessario specificare comunque anche il vincolo NOT NULL per tutti gli attributi coinvolti.

2. **vincoli interrelazionali** che si applicano tra relazioni diverse. Possono essere definiti attraverso i costrutti sintattici:

```

REFERENCES // Permettono di definire vincoli di integrità
FOREIGN KEY // referenziale
CHECK // (Vincoli complessi)

```

e si hanno sintassi per singoli o più attributi. È possibile definire politiche di reazione alle violazioni. Si ha l'**integrità referenziale** che esprime un legame gerarchico (padre / figlio) fra tabelle. Alcuni attributi della tabella figlio sono definiti FOREIGN KEY e si devono riferire (REFERENCES) ad alcuni attributi della tabella padre che costituiscono una chiave (devono essere UNIQUE e NOT NULL oppure PRIMARY KEY). I valori contenuti nella FOREIGN KEY devono essere sempre presenti nella tabella padre. Si hanno due sintassi:

- (a) nella parte di definizione degli attributi con il costrutto sintattico REFERENCES:

```
AttrFiglio CHAR(3) REFERENCES TabellaPadre(AttrPadre)
```

- (b) oppure dopo le definizioni degli attributi con i costrutti FOREIGN KEY e REFERENCES:

```

FOREIGN KEY (AttrFiglio)
REFERENCES TabellaPadre(AttrPadre)

```

quindi:

```

CREATE TABLE Impiegato (
  Matricola CHAR(6) PRIMARY KEY,
  Nome CHAR(20) NOT NULL,
  Cognome CHAR(20) NOT NULL,
  Dipart CHAR(15) REFERENCES Dipartimento(NomeDip) ,

```

```

        Stipendio NUMERIC(9) DEFAULT 0,
        UNIQUE (Cognome, Nome)
    )

```

Se si omettono gli attributi destinazione, vengono assunti quelli della chiave primaria.

```

AttrFiglio CHAR(3) REFERENCES TabellaPadre

```

Quando si hanno più attributi da riferire, si utilizza sempre FOREIGN KEY:

```

FOREIGN KEY (AttrFiglio1 { }, AttrFiglio2 { })
REFERENCES TabellaPadre (AttrPadre1 { }, AttrPadre2 { })

```

Esempio 8. Definiamo tre tabelle con le informazioni degli esami sostenuti dagli studenti: *Studente*, *Esame*, *Corso*:

```

CREATE TABLE Studente (
    Matr CHAR(6)
    Nome VARCHAR(30)
    Città VARCHAR(20),
    CDip CHAR(3)
)

```

Matr	Nome	Città	CDip
34321	Luca	Mi	Inf
53524	Giovanni	To	Mat
64521	Emilio	Ge	Ing
73321	Francesca	Vr	Mat

```

CREATE TABLE Esame (
    Matr CHAR(6),
    CodCorso CHAR(6),
    Data DATE NOT NULL,
    Voto Voto,
    PRIMARY KEY (Matr, CodCorso)
)

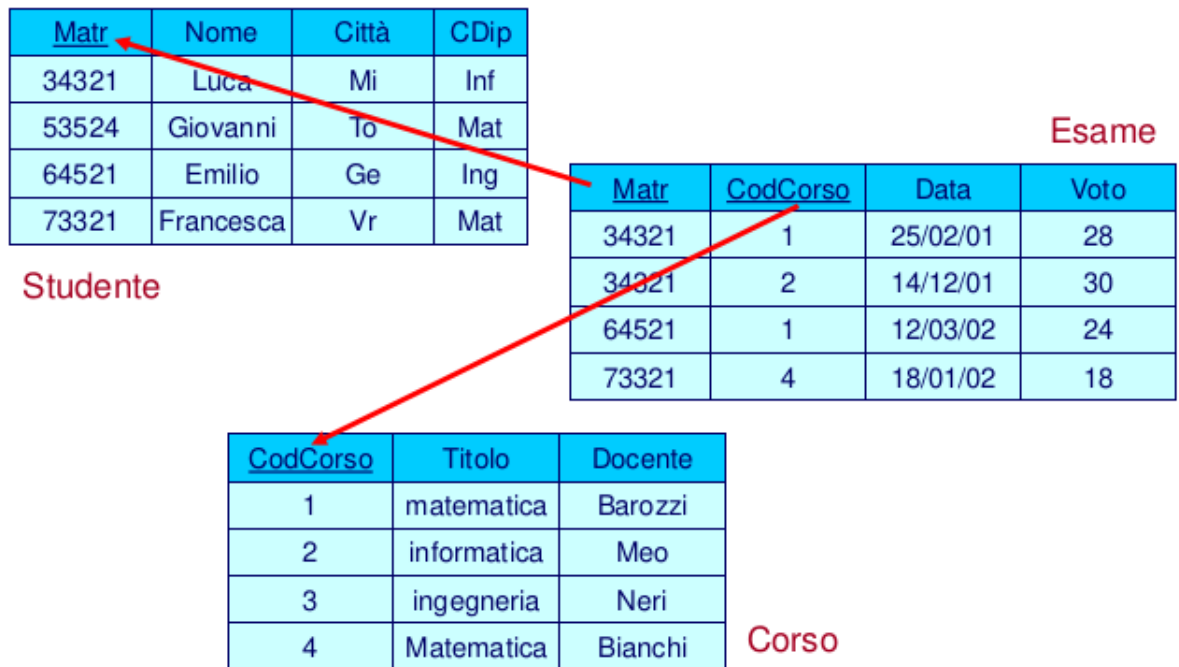
```

<u>Matr</u>	<u>CodCorso</u>	Data	Voto
34321	1	25/02/01	28
34321	2	14/12/01	30
64521	1	12/03/02	24
73321	4	18/01/02	18

```
CREATE TABLE Corso (
  CodCorso CHAR(6) PRIMARY KEY,
  Titolo VARCHAR(30) NOT NULL,
  Docente VARCHAR(20)
)
```

<u>CodCorso</u>	Titolo	Docente
1	matematica	Barozzi
2	informatica	Meo
3	ingegneria	Neri
4	Matematica	Bianchi

con le relazioni:



aggiungiamo un vincolo a esame:

```
CREATE TABLE Esame (
  Matr CHAR(6),
  CodCorso CHAR(6),
  Data DATE NOT NULL,
  Voto Voto,
  PRIMARY KEY (Matr, CodCorso)
  FOREIGN KEY (Matr) REFERENCES Studente
  FOREIGN KEY (CodCorso) REFERENCES Corso
```

oppure:

```
CREATE TABLE Esame (
  Matr CHAR(6), REFERENCES Studente,
  CodCorso CHAR(6), REFERENCES Corso,
  Data DATE NOT NULL,
  Voto Voto,
  PRIMARY KEY (Matr, CodCorso)
```

Si introduce il **problema delle violazioni**.

Per tutti gli altri vincoli visti fino ad ora, a seguito di una violazione, il comando di aggiornamento viene rifiutato segnalando l'errore all'utente.

Per i vincoli di integrità referenziale invece SQL permette di scegliere delle reazioni da adottare in caso di violazioni.

Si possono introdurre violazioni operando sulle righe della tabella padre (tabella esterna) o sulle righe della tabella figlio (tabella interna). Si hanno le seguenti modifiche sulla tabella interna (figlio):

- inserimento di una nuova riga
- modifiche della *foreign key*

Inoltre non vengono proposte reazioni, solo il rifiuto dell'operazione. Per la tabella esterna (padre) si hanno le seguenti modifiche:

- cancellazione di una riga
- modifica dell'attributo riferito

inoltre vengono proposte diverse reazioni.

Si ha anche il problema dell'aggiornamento. Se rimuovi una matricola alla tabella Esame si aggiunge un problema, il **problema degli orfani**.

Si hanno le reazioni alle violazioni, che si introducono con una modifica (update) dell'attributo cui si fa riferimento o con la cancellazione di tuple. Le reazioni operano sulla tabella figlio (es Esami), in seguito a modifiche alla tabella padre (es Studente):

CASCADE: propaga la modifica

SET NULL: annulla l'attributo che fa riferimento

SET DEFAULT: assegna il valore di **default all** attributo

NO ACTION: impedisce che la modifica possa avvenire

A fini diagnostici (e di documentazione) è spesso utile sapere quale vincolo è stato violato a seguito di un'azione sul DB. A tale scopo è possibile associare dei nomi ai vincoli, ad esempio:

```
tipendio INTEGER CONSTRAINT StipendioPositivo  
CHECK (Stipendio > 0),
```

```
CONSTRAINT ForeignKeySedi  
FOREIGN KEY (Sede) REFERENCES Sedi
```

Per le modifiche degli schemi si hanno:

- **ALTER** per la modifica
- **DROP** che cancella oggetti dallo schema

```
ALTER DOMAIN NomeDominio <
  SET DEFAULT ValoreDefault |
  DROP DEFAULT |
  ADD CONSTRAINT DefVincolo |
  DROP CONSTRAINT NomeVincolo >
```

```
ALTER TABLE NomeTabella <
  ALTER COLUMN NomeAttributo <
    SET DEFAULT NuovoDefault |
    DROP DEFAULT > |
  DROP COLUMN NomeAttributo |
  ADD COLUMN DefAttributo |
  DROP CONSTRAINT NomeVincolo
  ADD CONSTRAINT DefVincolo >
```

DROP cancella oggetti DDL, si applica su domini, tabelle, indici, view, asserzioni, procedure,...

```
DROP < schema, domain, table, view, ... > NomeElemento
[ RESTRICT | CASCADE ]
```

- RESTRICT: impedisce drop se gli oggetti comprendono istanze non vuote
- CASCADE applica drop agli oggetti collegati. Potenziale pericolosa reazione a catena

5.0.1 Interrogazioni in SQL

Le interrogazioni si fanno con la SELECT:

```
SELECT ListaAttributi
FROM ListaTabelle
[WHERE Vondizioni]
```

```
SELECT AttrEspr [ [ AS ] Alias ] {, AttrEspr [ [ AS ] Alias ] }
FROM Tabella [ [ AS ] Alias ] {, Tabella [ [ AS ] Alias ] }
[ WHERE Condizione ]
```

con gli AS setto gli alias. Vediamo qualche esempio:

```
-- Individuare lo stipendio degli impiegati di cognome
"Rossi":
```



```
SELECT Stipendio
FROM Impiegato
WHERE Cognome="Rossi"
```

```
-- Selezionare nome, cognome e mansione degli
impiegati dell'ufficio 10
SELECT Nome, Cognome, Mansione
FROM Impiegato
WHERE ufficio='10'
```

```
-- Individuare le informazioni degli impiegati con
stipendio > 40
Individuare le informazioni degli impiegati con
stipendio > 40
```

```
-- Selezionare tutte le informazioni sui dipendenti:
SELECT id_impiegato, Nome, Cognome, Dipart, Ufficio,
stipendio, premioprod, Mansione, Città, idcapo
FROM Impiegato
```

```
SELECT *
FROM Impiegato
```

```
-- Selezionare cognome, stipendio e premio di
produzione per gli impiegati che hanno uno
stipendio compreso tra 38 e 60:
SELECT cognome, stipendio, premioprod AS premio_di_produzione
FROM Impiegato
WHERE ( stipendio>=38 AND stipendio <=60 )
```

```
-- Selezionare i nomi, i cognomi e le città di provenienza
degli impiegati
SELECT Nome, Cognome, città AS città_provenienza
FROM Impiegato
```

```
-- Selezionare i nomi e i cognomi degli impiegati, le città
di provenienza e le città in cui lavorano:
SELECT Nome, Cognome, CittàDip AS città_lavoro,
Città AS città_provenienza
FROM Impiegato, Dip
```

```
WHERE Dipart = NomeDip
```

```
/* Specificare più tabelle su cui operare per determinare il
risultato significa: eseguire il prodotto cartesiano delle tabelle
e poi selezionare le righe che soddisfano clausola WHERE.
Join di algebra relazionale*/
```

```
-- Selezionare i nomi e i cognomi degli impiegati, le città
di provenienza e le città in cui lavorano:
```

```
SELECT Nome, Cognome, CittàDip AS città_lavoro,
Città AS città_provenienza
FROM Impiegato, Dip
WHERE Dipart = NomeDip
```

```
-- Selezionare i nomi e i cognomi degli impiegati, le città
di provenienza e le città in cui lavorano:
```

```
SELECT Nome, Cognome, Città AS città_lavoro,
Città AS città_provenienza
FROM Impiegato, Dip
WHERE Dipart = Nome
```

```
/* Notazione punto: si utilizza il nome della tabella cui
l'attributo fa riferimento */
```

```
SELECT Impiegato.Nome, Cognome, Dip.Città AS città_lavoro,
Impiegato.Città AS città_provenienza
FROM Impiegato, Dip
WHERE Dipart = Dip.Nome
```

```
/* Si usano Alias per ridenominare le tabelle:
```

```
1) Abbreviano riferimento a tabelle
```

```
2) Risolvono ambiguità di riferimento*/
```

```
SELECT I.Nome, Cognome, D.Città AS città_lavoro,
I.Città AS città_provenienza
FROM Impiegato I, Dip D
WHERE Dipart = D.Nome
```

pagina 18