Massimo Stefanizzi

Le Basi di Dati

INDICE

I DATA BASE

1. NASCITA DEI DATA BASE

Uno degli scopi dell'informatica è quello di 'conservare' dati. Il concetto di 'conservare' è di disporre in modo ordinato i dati, in modo tale che nel momento in cui si ha bisogno di reperire un determinato dato, si può reperirlo facilmente.

Il termine *archivio* (file) è quello che storicamente ha individuato ed individua una collezione ordinata di documenti.

Nel passato vi erano alcune figure 'mitiche' negli uffici: gli archivisti. Organizzati in una struttura che andava dal responsabile – il capo archivista – agli impiegati archivisti senior e junior, erano la memoria 'storica' dell'ufficio. Ad essi era demandato il compito di conservare, catalogare, ordinare e reperire, migliaia di documenti.

Come la rivoluzione industriale del XIX secolo ha sostituito la bottega con l'industria e l'artigiano con l'operaio, così il computer e l'informatica hanno sostituito alcune figure professionali e macchine dell'industria con altre: specialisti in informatica, computer, robot.

In realtà i computer sono nati principalmente per velocizzare i calcoli. Von Neumann, riuscì a realizzare una macchina in grado non solo di svolgere calcoli velocemente, ma anche di risolvere gli algoritmi, mettendo in pratica l'idea di un grande matematico inglese: Alan Turing.

Von Neumann realizza la struttura del computer come lo conosciamo oggi; in particolare progetta la memoria suddivisa in 'celle': ogni cella è suddivisa in due parti (indirizzo e dati). Il dato viene individuato nella memoria grazie all'indirizzo della cella ove è conservato.

Solo recentemente le dimensioni e costo delle memorie sono scesi vertiginosamente e ciò ha permesso di trasformare il computer da mera macchina veloce di calcolo, in macchina adatta anche a conservare moli enormi di dati.

Ritorniamo ai nostri archivi. Il nostro bravo archivista per non perdersi nella miriade di documenti diversi, li ordina per 'tipo' di documento. Così, ad esempio, i certificati di nascita dei dipendenti dell'azienda in cui lavora, li inserisce in uno o più faldoni (magari ordinati in ordine alfabetico) e li ripone su uno scaffale su cui scrive 'CERTIFICATI DI NASCITA'; analogamente fa con gli 'ordini di servizio' o le 'circolari', in questo caso ordinandoli per data.

Ma cos'hanno in comune i documenti dello stesso tipo? Sostanzialmente due cose: hanno struttura simile e contengono la stessa tipologia di dati. Ad esempio il certificato di nascita ha come dati il cognome, il nome, la data ed il luogo di nascita e il sesso della persona a cui si riferisce.

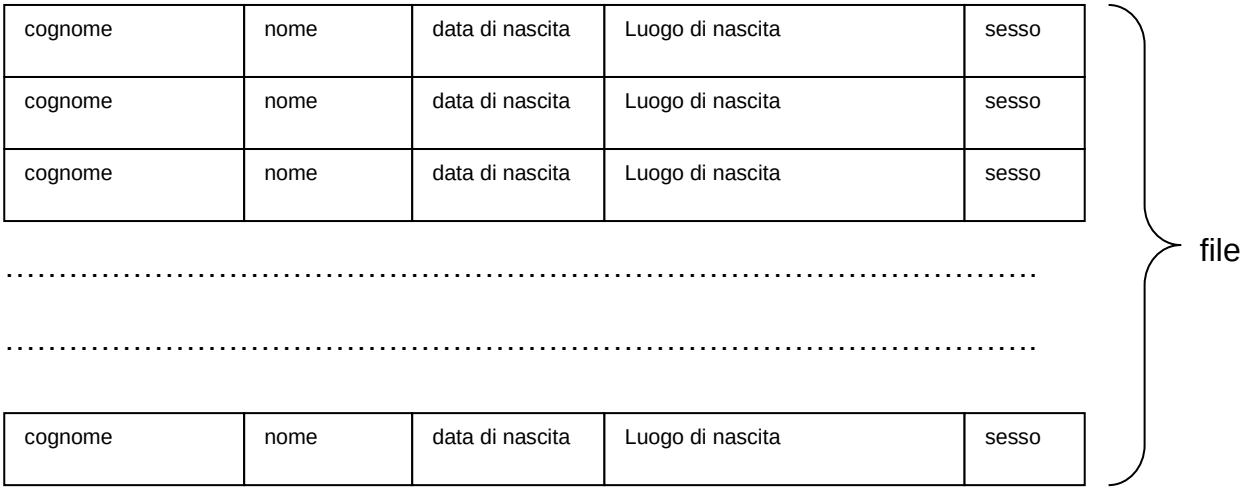
Se devo memorizzare in formato elettronico i dati, cerco di fare lo stesso lavoro del bravo archivista; cambia solo il supporto su cui memorizziamo i dati: dal cartaceo ai bit, dall'archivio cartaceo all'archivio elettronico.

Comincio, quindi, a realizzare la 'struttura' del documento, intendendo per struttura il modo in cui sistemo i dati, ad esempio, del nostro certificato di nascita; avrò bisogno di un certo numero di byte per memorizzare il cognome, altri byte per memorizzare il nome, quindi la data, il luogo ed il sesso.

cognome		data di nascita	Luogo di nascita	sesso

In informatica, come sappiamo, per definire un dato devo essere preciso: devo dire quant'è grande (di quanti byte ho bisogno) e se il dato è alfanumerico, numerico, se è una data, se è un valore booleano… ossia devo definirne il tipo.
La struttura così definita si chiama ***record,*** i dati nella struttura ***campi*** (field) e di ogni campo è stato definito il ***tipo*** (type).

Adesso non devo far altro che inserire nel record i dati del certificato di nascita… ed il gioco è fatto! Ho memorizzato nel computer un certificato di nascita!!! Ma degli altri certificati cosa ne faccio? Ma faccio come faceva l'archivista! Realizzo un contenitore per più record: il ***file.***



A questo contenitore associo un nome (es. certificati) e lo sistemo sullo 'scaffale', ossia in una 'parte' di una memoria di massa.

A questo punto all'archivista arrivano richieste di inserire nuovi certificati, trovare il certificato di un dipendente, modificare qualche dato in un certificato, distruggere qualche certificato non più necessario. Il nostro archivista ha organizzato bene i documenti: si reca allo scaffale su cui ha riposto i faldoni con i certificati di nascita, controlla il cognome della persona sul documento che deve inserire/modificare/cancellare e, siccome i faldoni sono organizzati in ordine alfabetico, estrae quello classificato con la lettera iniziale del cognome uguale a quella del cognome presente sul certificato. Quindi esegue l'operazione richiesta.
Il nostro informatico-archivista, non può, ovviamente, operare manualmente sugli archivi elettronici; realizza quindi un programma che gestirà l'archivio che compierà tutte le azioni che compiva l'archivista di un tempo manualmente.
L'archivista era anche il custode dei segreti. Non permetteva a tutti di accedere ai dati che lui aveva diligentemente conservato ma solo a chi aveva diritto di accedere alle informazioni. Il nostro informatico-archivista deve fare la stessa cosa: non tutti possono accedere o operare sui file che ha archiviato. Deve quindi arricchire il programma con istruzioni per il controllo degli accessi ai file. In poche parole questo programma deve essere in grado di 'fare' tutte le cose che venivano espletate dall'archivista di un tempo: fare copie dei documenti, reperire tutti i documenti inerenti un dipendente, controllare la correttezza dei documenti e, se richiesto, distruggere tutti i documenti relativi ad un dipendente….

Tutte queste cose venivano fatte dai programmi ed i file erano soggetti passivi nel senso che servivano solo per conservare i dati e null'altro.

Lo sviluppo dell'informatica non si esaurisce solo nello sviluppo dell'architettura hardware e nella realizzazione di componenti sempre più piccoli e sempre più potenti, ma anche nello sviluppo del software.
Negli anni '70 E. F. Codd pose le basi per lo sviluppo dei database relazionali (che saranno l'oggetto principale di questo testo), anche se i primi database nacquero intorno gli anni '60; caratteristica dei database è che la gestione dei dati in essi contenuti è effettuata da particolari applicazioni software che si interfacciano con il programmatore/utente: DBMS (Data Base Management System). I DBMS, come

vedremo, è uno "strato" software che assolve a molte funzioni nella gestione e controllo dei dati, funzioni che i programmatori dovevano codificare e scrivere nei programmi aumentando sia il lavoro di stesura che la complessità. Questo è uno dei motivi principali che ha decretato il successo dei data base.

2. PROGETTAZIONE DEI DATA BASE

Supponiamo di voler costruire la nostra casetta ove abitare. Ovviamente la vogliamo comoda, ben esposta, grande quanto ci serve e soprattutto solida.

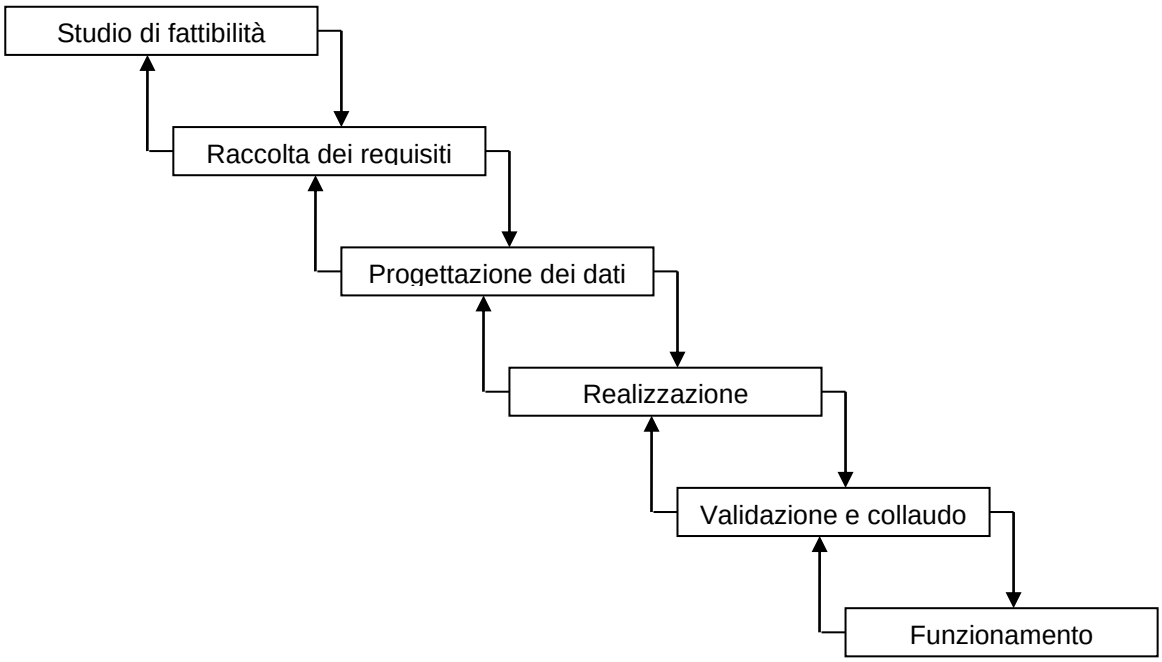
Credo che la prima cosa che faremmo è recarci da un ingegnere civile o da un architetto per commissionargli il progetto. L'ingegnere realizza il progetto che consiste in un insieme di fogli sui quali sono disegnate le strutture della casa, fondazioni, travi, colonne, mura, spessore intonaco ecc. ecc. strutture che devono rispettare i calcoli strutturali affinché la casa stia su bene e non crolli.

Solo dopo che il progetto è completo in ogni sua parte ed averlo visionato ed approvato, diamo incarico ad un'impresa che, con la sua squadra di geometri, muratori, carpentieri, elettricisti, idraulici, realizza la nostra casetta basandosi e rispettando i dettami presenti sul progetto.

Se vogliamo che in nostro Database sia 'solido' e ben costruito, anche noi dobbiamo progettarlo ed alla fine realizzarlo.

2.1 Progettazione dei dati.

In generale, per progettare una base dati si devono eseguire le seguenti fasi: ***raccolta dei requisiti*** (le richieste che vengono espresse dal committente), ***progettazione dei dati*** (dall'esame dei requisiti vengono individuati i dati che devono essere conservati e come conservarli), ***realizzazione*** del data base, fase di ***validazione e collaudo*** e ***funzionamento***. Di seguito diamo lo schema delle fasi:



Lo ***studio di fattibilità*** è la fase di analisi dei costi/benefici nel realizzare il database.

Come possiamo notare dal grafico le diverse fasi sono a 'cascata' nel senso che da una fase posta più in alto, si passa alla successiva in sequenza. Le frecce che vanno dalle fasi più in basso a quelle poste più in alto sono le cosiddette *validazioni intermedie*: se in una fase ci si accorge che è stato commesso un errore in una fase precedente, si interviene nella fase precedente per rimuoverlo e quindi ripassare alla fase successiva. Più un errore si intercetta in fasi lontane dalla fase in cui si è verificato, più complessa risulta la sua rimozione. La fase di correzione viene detto *ciclo di feedback*.
Lo schema delle fasi viene detto anche ***ciclo di vita del sistema***.

Lo scopo di questo testo è quello di analizzare e studiare le fasi ***progettazione dei dati*** e ***realizzazione.***

Queste fasi si realizzano con la costruzione dei seguenti modelli:

1. realizzazione del modello concettuale;
2. realizzazione del modello logico;
3. realizzazione del modello fisico.

Il ***modello concettuale*** è la rappresentazione dei dati e dei *'legami'* che legano tra loro i dati, che devono essere conservati/manipolati in un database, rappresentazione più vicina alla *realtà di interesse o dominio applicativo*.

> *Il legame, ad esempio, tra uno studente e un professore, è l'insegnamento. Studente e professore, o meglio i loro dati, sono legati da questa relazione.*
>
> *Per realtà d'interesse o dominio applicativo intendiamo il sottosistema della realtà che deve essere informatizzato; in parole povere realtà d'interesse è, ad esempio, il sistema contabilità, gestione del personale, gestione buste paga, ecc. ecc. ossia quei sistemi che vogliamo informatizzare: essi sono formati da dati (es. fatture, tasse, imposte… per la contabilità, paga base, contingenza, scatti d'anzianità, ritenute, anni di servizio, straordinari… per le buste paga e così via) e funzioni (es. lo stipendio = paga base + contingenza + straordinari – ritenute) ossia trasformazioni sui dati. Le funzioni vengono realizzate con i linguaggi di programmazione che 'ospitano' le istruzioni per l'accesso al data base.*

Il ***modello logico*** è una rappresentazione più vicina alla struttura del data base ed alle sue regole, in questo modello anche i legami, espressi nel modello concettuale si trasformano in dati.

Il ***modello fisico*** è quello in cui si definiscono le caratteristiche utili per l'ottimizzazione della prestazione della memoria o per la gestione del DBMS.

2.2 Il modello concettuale.

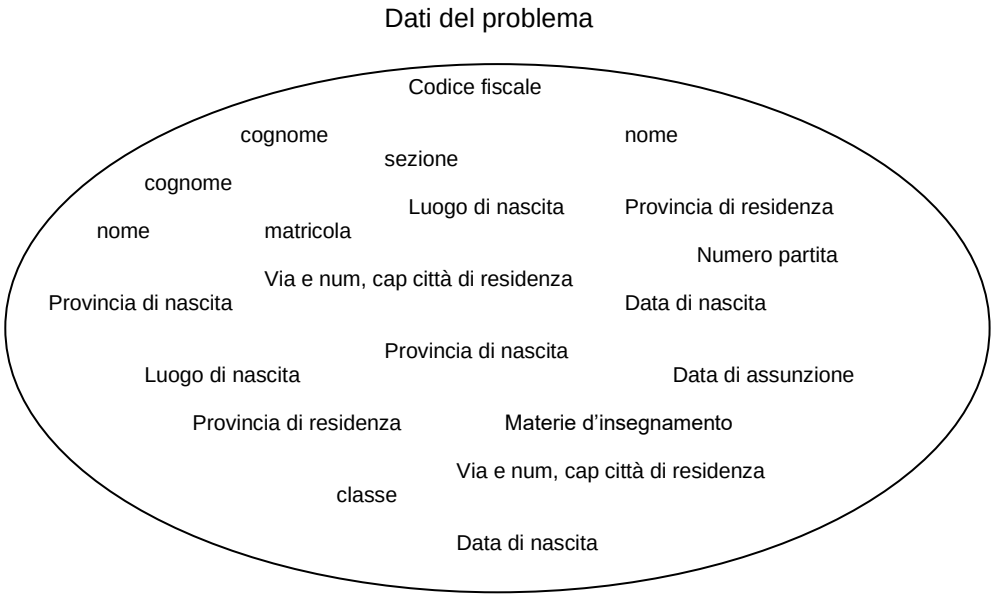
Supponiamo che la fase della raccolta dei requisiti sia stata effettuata. L'analista ci ha descritto la realtà d'interesse su un documento. Per evitare di appesantire la trattazione, supponiamo che tale documento sia in forma discorsiva, non ci siano, cioè, standard di comunicazione tra i progettisti software. Siccome stiamo progettando un data base, ciò che ci interessa è 'estrarre' i dati che dovranno essere conservati dal documento.

Esempio:

> *Una scuola ha la necessità di conservare i dati dei suoi alunni e degli insegnanti in un data base. Agli alunni, quando si iscrivono alla scuola, viene assegnata una matricola. Il segretario, inoltre, riempie una scheda con il cognome, nome, indirizzo di residenza, luogo di nascita, province di residenza e nascita, data di nascita, la classe e la sezione dell'alunno.*
> *Per il personale il segretario amministrativo, ha delle schede sulle quali riporta il codice fiscale, cognome, nome, indirizzo di residenza, luogo di nascita con le rispettive province, data di nascita, il numero di partita, la data di assunzione, la/le materie d'insegnamento degli insegnanti…*

Cerchiamo di individuare, evidenziandoli, i dati che, secondo noi, sono presenti nella porzione di testo

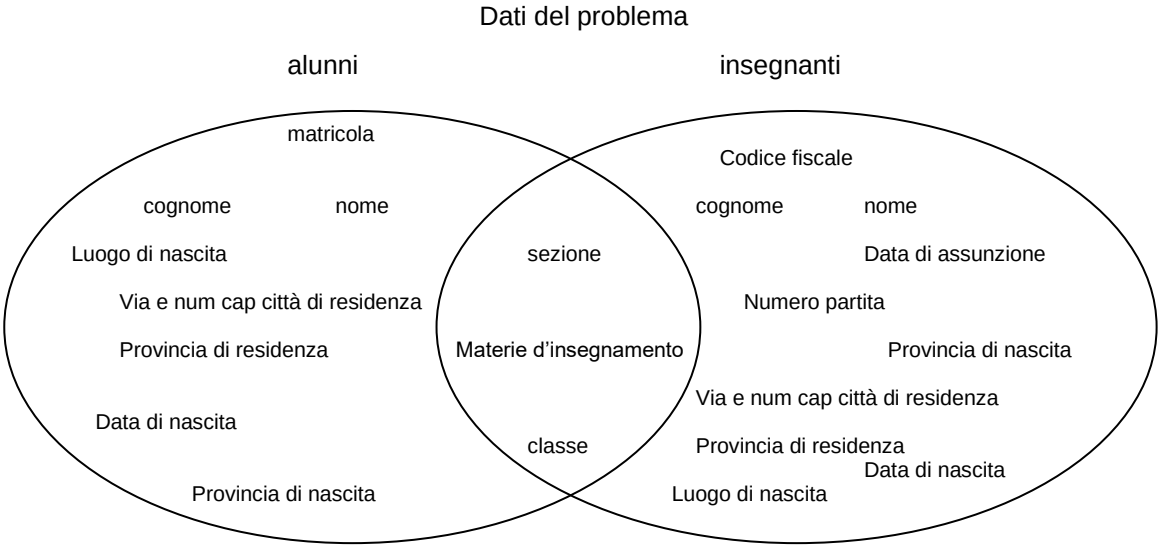
> *Una scuola ha la necessità di conservare i dati dei suoi alunni e degli insegnanti in un data base. Agli alunni, quando si iscrivono alla scuola viene assegnata una matricola. Il segretario, inoltre, riempie una scheda con il cognome, nome, via numero civico, cap e città di residenza, luogo di nascita, province di residenza e nascita, data di nascita, la classe e la sezione dell'alunno.*
> *Per il personale il segretario amministrativo, ha delle schede sulle quali riporta il codice fiscale, cognome, nome, indirizzo di residenza, luogo di nascita con le rispettive province, data di nascita, il numero di partita, la data di assunzione, la/le materie d'insegnamento degli insegnanti…*



Bene, è facile intuire che, raccolti così, non siamo in grado di distinguere i dati a chi appartengono, a meno che non siano specifici degli alunni (matricola) o degli insegnanti (data di assunzione, numero

partita), altri addirittura sono comuni (materie d'insegnamento - dagli alunni vengono studiate, dagli insegnanti, insegnate -, classi e sezioni – dagli alunni frequentate, agli insegnanti assegnate -).

Logico sarebbe, quindi, suddividere i dati secondo l'appartenenza:



Ci accorgiamo, infatti, che i dati appartengono a due insiemi distinti, che chiameremo *alunni* ed *insegnanti*, ed alla loro intersezione.

Cosa succede se nel testo troviamo anche dei legami tra gli insiemi di dati? Nel nostro esempio i requisiti continuano con:

*…inoltre, agli alunni **insegnano** gli insegnanti assegnati alle loro classi….*

probabilmente è necessario che il legame **insegnano** sia in qualche modo rappresentato prima, memorizzato dopo, per determinare, ad esempio, chi sono gli insegnanti di un certo alunno. Questi legami è ciò che in precedenza ho chiamato 'concetti'.

Il modello Entità-Relazioni – E/R (Entity-Relationship).

Per rappresentare il modello concettuale il prof. Peter Chen nel 1976 propose un modello che mette in relazione gli insiemi di dati tra loro.

Le entità e gli attributi.

Definiamo ENTITÀ un sottoinsieme omogeneo di oggetti.

Sottoinsieme omogeneo: significa che, come nel nostro esempio, i dati si riferiscono univocamente al sottoinsieme; cognome, ad esempio, è presente sia nel sottoinsieme alunni che in quello insegnanti, ma, a livello concettuale, non è lo stesso dato: nel primo caso il dato si riferisce ai cognomi degli alunni nel

secondo a quello dei professori, matricola è un dato che si riferisce agli alunni ma non ai professori.

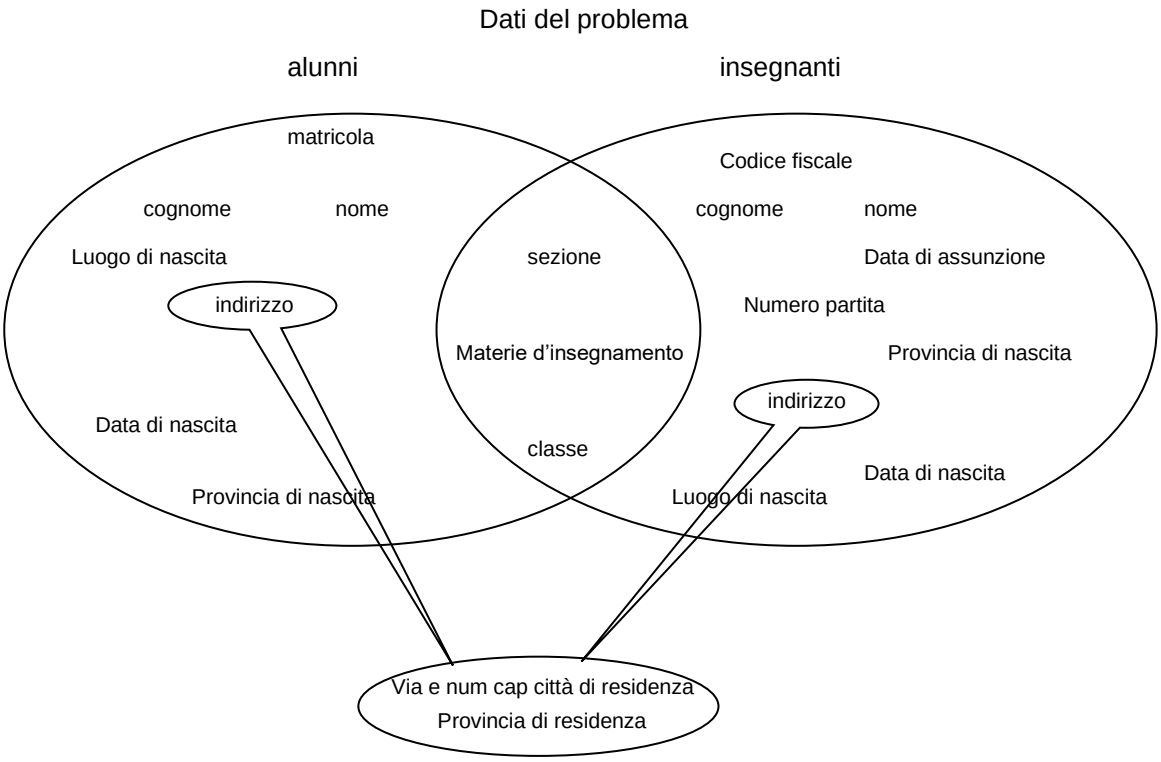
Quando siamo in presenza dei requisiti, dobbiamo quindi individuare questi sottoinsiemi, le **entità**. Nel nostro esempio ***alunni*** ed ***insegnanti*** sono due entità.

Ciò che caratterizza l'entità sono gli **ATTRIBUTI**. L' attributo, in poche parole, è il nome a cui sarà associato il dato: **cognome** è l'attributo, **Rossi** è il dato.

Gli attributi possono essere *semplici*, *composti* e *chiave*.

Un attributo è semplice se non è *decomponibile*. L'attributo composto, in realtà, lo definisce il progettista del data base. Nel nostro esempio, gli attributi semplici *via, numero civico, cap e città e provincia di residenza,* presenti sia nell'entità **alunni** che **insegnanti**, potevano essere 'raggruppati' in un solo attributo chiamato, ed. es., *indirizzo*: ***indirizzo*** così definito è un attributo composto, decomponibile cioè nei suoi attributi semplici.

Gli attributi composti vengono utilizzati principalmente per comodità: quando un progettista trova, in diverse entità, gli stessi attributi semplici raggruppabili in un attributo composto, definisce l'attributo composto e lo userà al posto del gruppo degli attributi semplici:



Ci sono altri attributi semplici raggruppabili in attributi composti?

Prima di passare alla definizione di attributi chiave, definiamo cosa s'intende per ***istanza*** di un'entità.

Disponiamo in forma tabellare i dati associati agli attributi; nella prima riga scriviamo il nome dell'entità, nella seconda il nome degli attributi, nelle successive i dati associati agli attributi:

Alunni			
Matricola	Cognome	Nome	indirizzo
0001254	Rossi	Mario	Via dei Ciclamini 123 00100 Roma
0002145	Bianchi	Marco	Viale del Tramonto 11 87100 Cosenza
0003132	Neri	Giulia	C.so Colombo 281 84121 Salerno

Una riga di dati viene detta ***istanza*** dell'entità.

*Definiamo **chiave** di un'entità, l'attributo o gli attributi che identificano univocamente un'istanza.*

Nel nostro esempio, la matricola è un attributo chiave. Infatti essa è diversa per ogni alunno così che la matricola 0002145 identifica univocamente l'alunno Bianchi Marco. Qualcuno potrebbe osservare che anche il cognome o addirittura il nome o l'indirizzo nel nostro esempio identificano univocamente l'istanza.
Ma cosa succederebbe se si iscrivesse nella nostra scuola un alunno il cui cognome è identico ad uno già esistente? Per non parlare dei nomi! E se due alunni abitano nello stesso palazzo per cui via, numero civico cap e città coincidono? Certo, dalla definizione, abbiamo detto che possiamo usare come chiave più attributi, al limite tutti gli attributi dell'entità e sicuramente questi fornirebbero la chiave, ma, in questo caso, avremmo qualche problema a gestire chiavi così complesse!
Esempi di chiave già 'pronte all'uso' ne esistono già nella vita di tutti i giorni: il codice fiscale, il numero della patente di guida, il numero della carta d'identità, la matricola prima citata… queste chiavi, però, si riferiscono alle persone. Per oggetti e cose di solito si usano dei codici, tra i più diffusi è, ad esempio, il codice a barre che identifica però non il singolo oggetto ma la categoria merceologica cui l'oggetto appartiene.

Mentre leggiamo i requisiti, il problema principale è quindi quello di individuare le entità e gli attributi. Successivamente, per ogni entità cerchiamo di identificare l'attributo o gli attributi chiave. Si possono verificare i due casi:

1. non riusciamo ad individuare alcun attributo chiave
2. ne individuiamo più di uno.

diciamo subito che le istanze di un'entità devono avere necessariamente una chiave per poterle individuare.

Nel caso 1. il progettista ne 'costruisce' una: l'attributo ***chiave artificiale***. Il consiglio è quella di creare una chiave numerica di tipo 'progressivo'.

Esempio:

> Un'azienda ha la necessità di catalogare i suoi macchinari di cui si conoscono il nome delle aziende costruttrici, chi si occupa della manutenzione, la potenza elettrica impegnata, la descrizione.In questo caso l'entità è quella dei macchinari che chiamiamo, con poca fantasia, *macchinari* i suoi attributi: *azienda costruttrice*, *descrizione*, *manutentore*, *potenza elettrica*.

Macchinari			
Azienda costruttrice	Descrizione	Manutentore	Potenza elettrica
Siemens	Simatic s7-1200	Rossi	2000Kw
Bauer	Compressore	Bianchi	1500Kw
Siemens	Simatic pcs7	Rossi	1500Kw
Bauer	Compressore	Bianchi	2000 Kw

È facile rendersi conto che, in questo caso, nessuno degli attributi individuati possono essere definiti chiave. Il progettista deve quindi aggiungere l' attributo chiave (chiave artificiale) *codice macchinario.*

Macchinari				
Codice macchinario	Azienda costruttrice	Descrizione	Manutentore	Potenza elettrica
1	Siemens	Simatic s7-1200	Rossi	2000Kw
2	Bauer	Compressore	Bianchi	1500Kw
3	Siemens	Simatic pcs7	Rossi	1500Kw
4	Bauer	Compressore	Bianchi	2000 Kw

Nel caso 2. le chiavi individuate prendono il nome di ***chiavi candidate;*** tra queste il progettista ne sceglie una, quella che ritiene più significativa relativamente al dominio applicativo d'interesse.

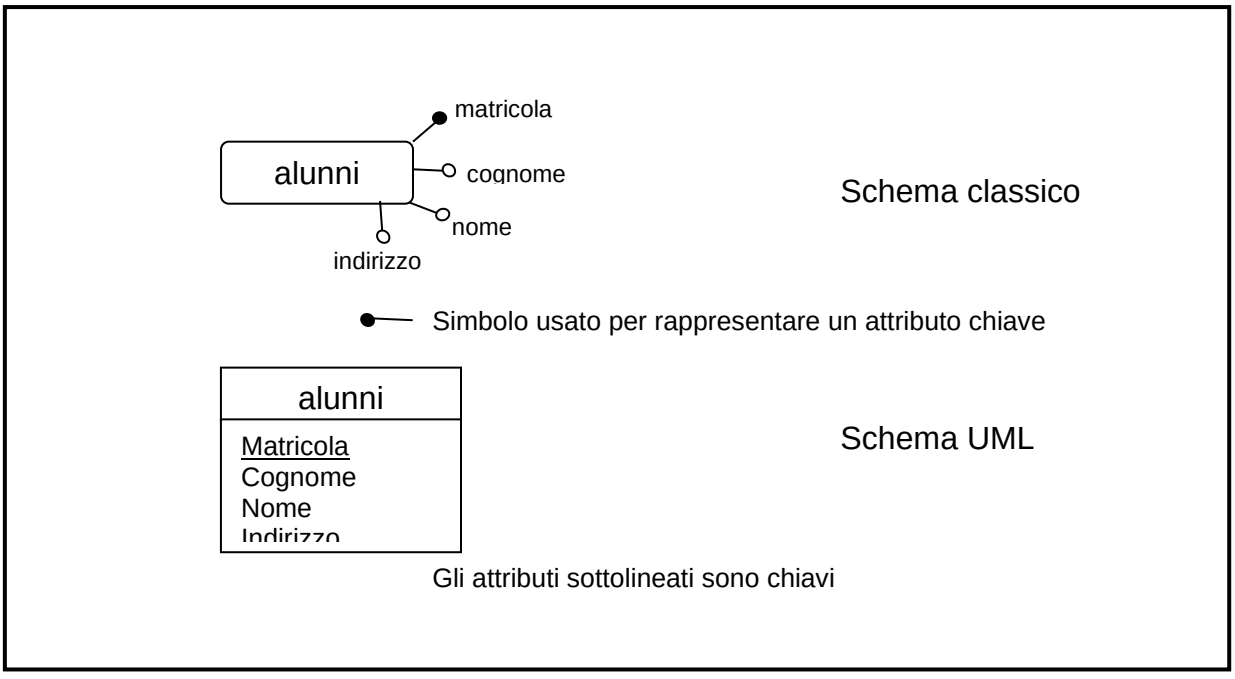
Esempio:

un'azienda deve censire in un database i suoi dipendenti; di questi conosciamo il numero di matricola aziendale, il cognome, nome, via, numero civico, cap e città di residenza, la data di nascita, il codice fiscale, il numero di carta d'identità…

per l'entità **dipendenti** abbiamo, quindi, i seguenti attributi: *matricola, cognome, nome, via, numero civico, cap e città di residenza, la data di nascita, il codice fiscale, il numero di carta d'identità*

le chiavi candidate sono: *matricola, codice fiscale, numero di carta d'identità.* Il progettista in questo caso dovrebbe scegliere come chiave dell'entità dipendenti l'attributo *matricola* in quanto il dominio applicativo si riferisce ad una azienda.

Per rappresentare graficamente le entità e gli attributi indichiamo due schemi tra i più utilizzati:



Dopo aver individuato le entità e, di queste, gli attributi, sarebbe buona norma definirne le caratteristiche di quest'ultimi in una tabella chiamata **tabella dei dati**:

- il **formato:** la tipologia dell'attributo (es. numerico, alfanumerico, booleano..);
- la **dimensione:** quanto è 'grande' (in termini di caratteri o cifre);

e vincoli:

- l'**opzionalità:** se il dato deve essere sempre valorizzato o meno;
- il **range di valori ammissibili:** la valorizzazione del dato appartiene ad un insieme definito (es. livello contrattuale compreso tra 1° livello e 7° livello).

Le relazioni e la cardinalità.

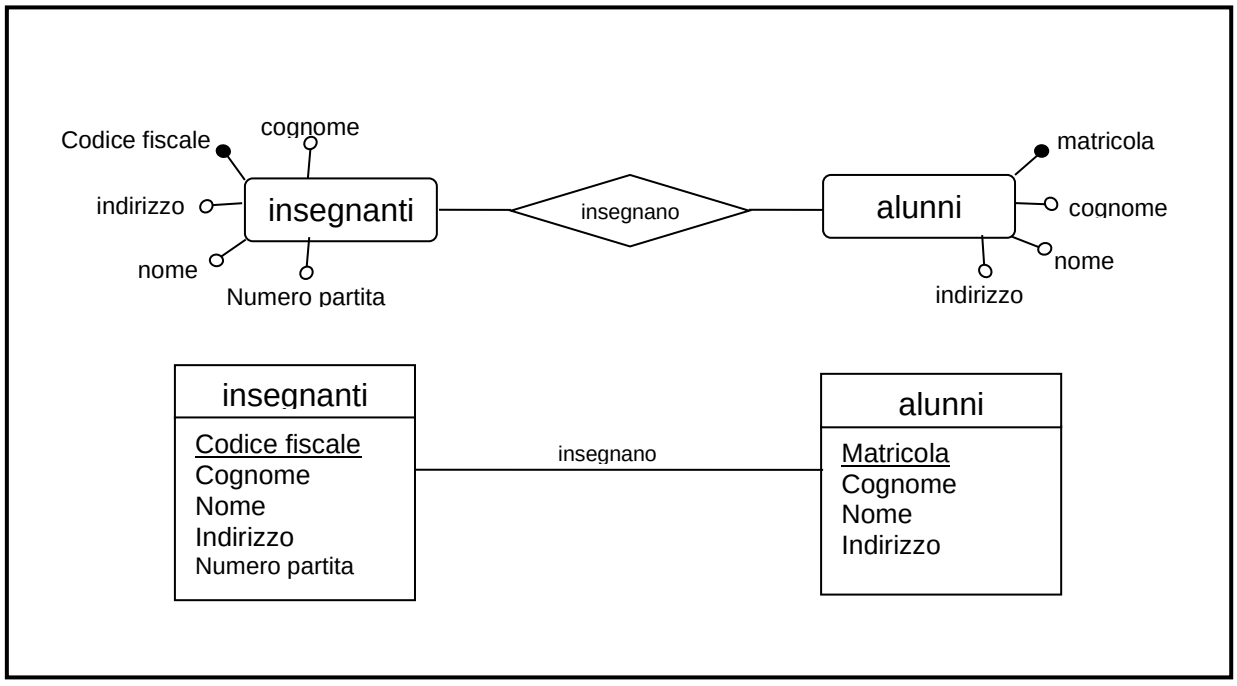
Come detto in precedenza, nel modello concettuale devono essere rappresentati anche i "legami" le entità.

Definiamo **relazione (o associazione)** un *legame logico* tra entità.

Come visto nei requisiti dell'esempio della scuola, abbiamo usato il verbo **insegnano** come un *legame* tra i due insiemi, le due entità, **insegnanti** ed **alunni**. Individuare questi legami è molto importante.

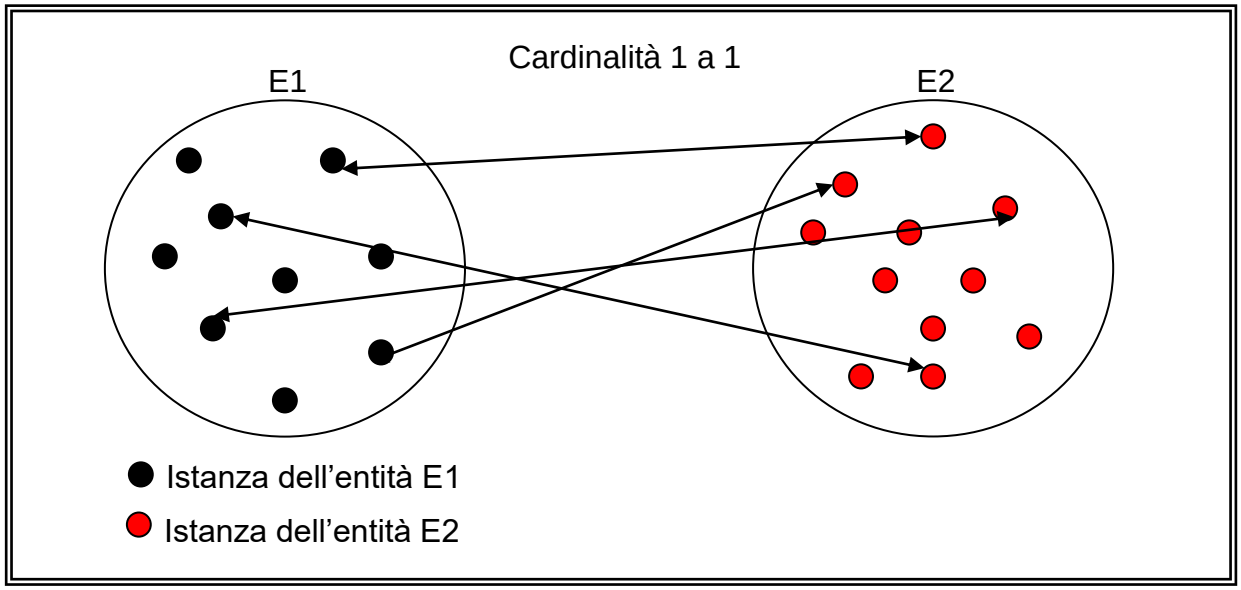
Come risulterà anche dalla vostra esperienza, nel vostro istituto vi sono molti alunni che frequentano classi diverse e molti insegnanti che vi insegnano, ma non tutti gli insegnanti del vostro istituto sono i vostri insegnanti ma solo alcuni.
Bene, come si può intuire, anche la conoscenza di questi legami è importante se, ad esempio volessimo rispondere alla domanda: quali sono gli alunni del prof. Tal dei Tali? O analogamente: chi sono i professori dell'alunno Pinco Pallo?
In poche parole dobbiamo 'trasformare' questi legami in dati.

Per rappresentare graficamente le relazioni tra le entità, le entità relazionale si legano come mostrato in figura:



Resta ora da stabilire come le istanze delle entità si relazionano tra loro, ossia ad una istanza di una entità, quante istanze dell'altra corrispondono e viceversa.
Ci sono tre tipi di associazioni (cardinalità) tra le istanze di due entità:

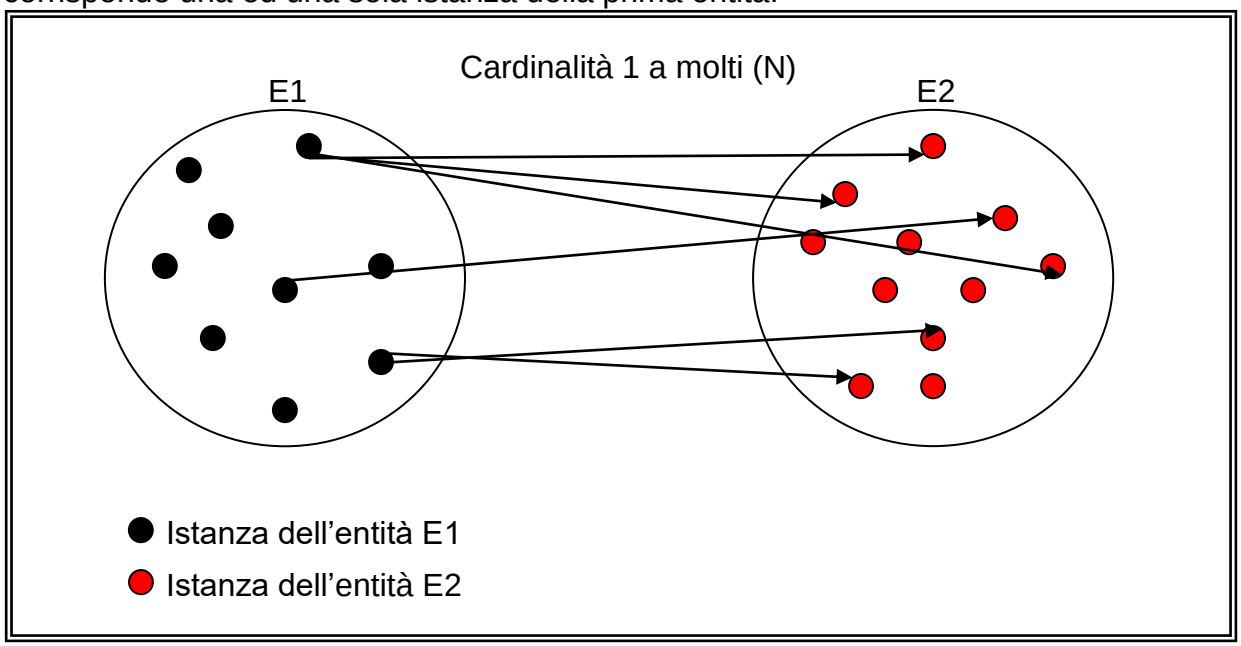
Cardinalità 1 a 1: questa associazione si verifica quando ad un'istanza di un'entità corrisponde una e una sola istanza dell'altra e viceversa.



Ad esempio: sia E1 l'entità ***mariti*** i cui attributi sono i dati anagrafici di persone di sesso maschile sposati ed E2 l'entità ***mogli*** i cui attributi siano i dati anagrafici di persone di sesso femminile e sposate supponiamo che la relazione che lega le due entità sia ***sposato/a***: ad un'istanza di mariti corrisponde un'istanza dell'entità mogli e ad un'istanza dell'entità mariti corrisponde, tramite la relazione, un'istanza dell'entità mariti.

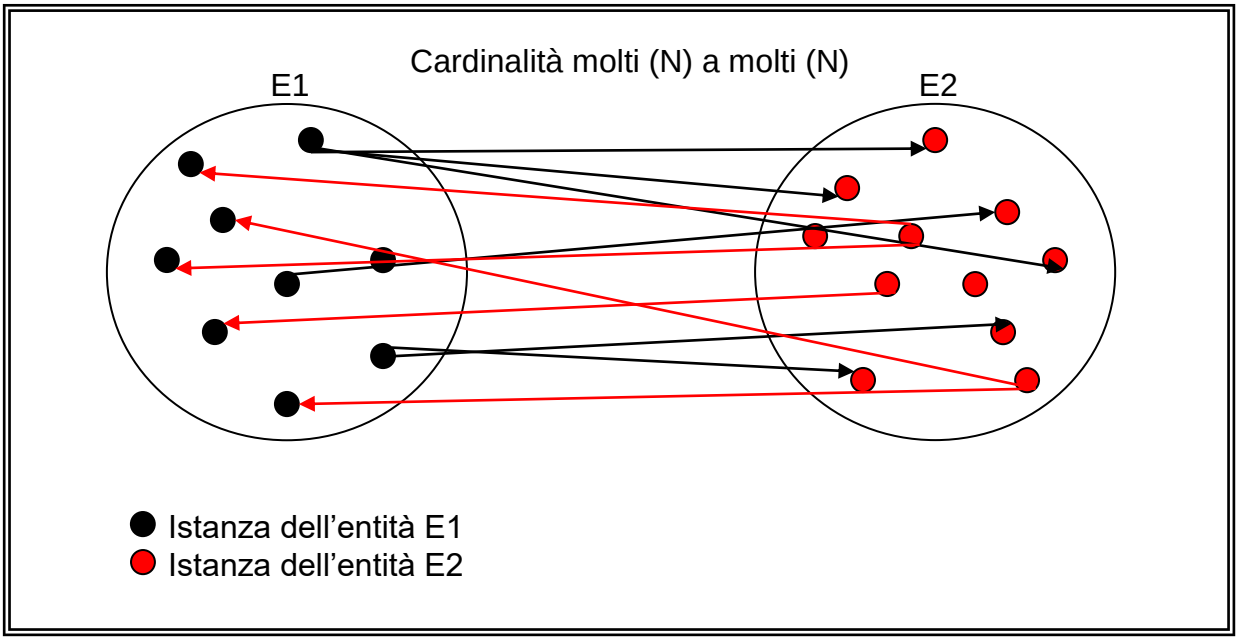
N.B. A prescindere della religione praticata, in Italia è vietata la poligamia! 

Cardinalità 1 a molti (1 a N): questa associazione si verifica quando ad un'istanza di un'entità corrispondono più istanze dell'altra ed a una istanza della seconda entità corrisponde una ed una sola istanza della prima entità.



Ad esempio: sia E1 l'entità ***madri*** ed E2 l'entità ***figli*** i cui attributi sono le anagrafiche (codice fiscale, cognome, nome, data di nascita, indirizzo di residenza ecc. ecc.) per entrambe le entità; le due entità sono legate dalla relazione ***generato***: è ovvio che un'istanza dell'entità madri è legata, tramite la relazione, ad una o più istanze dell'entità figli, mentre un'istanza dell'entità figli è legata ad una ed una sola istanza dell'entità madri.

Cardinalità molti a molti (N a N): questa associazione si verifica quando ad un'istanza di un'entità corrispondono più istanze dell'altra e viceversa.



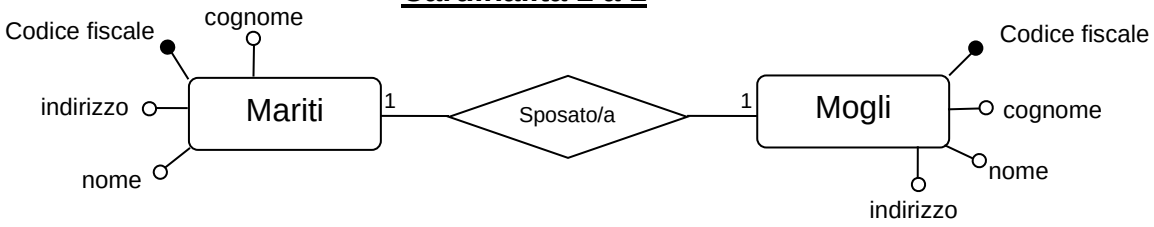
Prendiamo le entità dell'esempio della scuola: ***insegnanti*** ed ***alunni*** legate dalla relazione **insegnano**: un'istanza dell'entità insegnanti è relazionato (insegna) a più istanze dell'entità alunni; un'istanza dell'entità alunni è relazionato (è 'insegnato' - consentitemi la licenza poetica – al posto di 'impara dal') da più istanze dell'entità insegnanti.

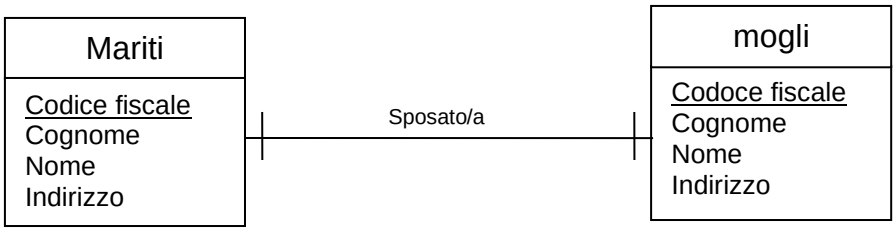
Osservazione 1. La cardinalità tra le entità non dipende dal tipo di entità ma dalla relazione tra esse; ciò significa che se cambiamo la relazione tra due entità cambia la cardinalità: se ad esempio tra le entità mariti e mogli la relazione è amicizia e non sposato/a, ad un'istanza dell'entità mariti può corrispondere più istanze dell'entità mogli e viceversa: la relazione passa da cardinalità 1 a 1 (sposato/a) a N a N (amicizia).

Osservazione 2. Per individuare correttamente le cardinalità di una relazione tra due entità, ricordatevi sempre di verificare come si relaziona una sola istanza della prima entità con le istanze della seconda entità quindi verificare come si relaziona una sola istanza della seconda entità con le istanze della prima.

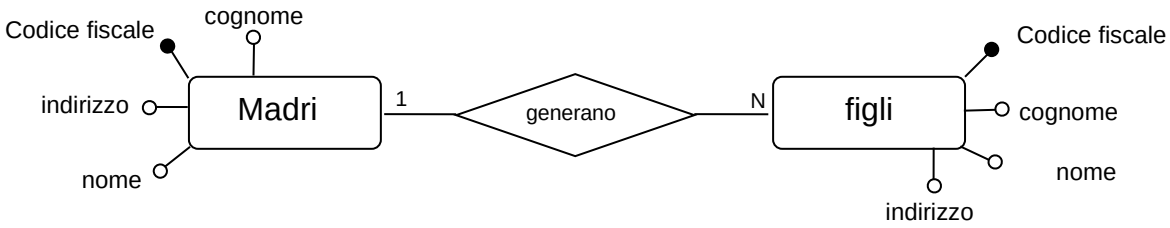
Vediamo come si rappresenta graficamente la cardinalità delle relazioni:

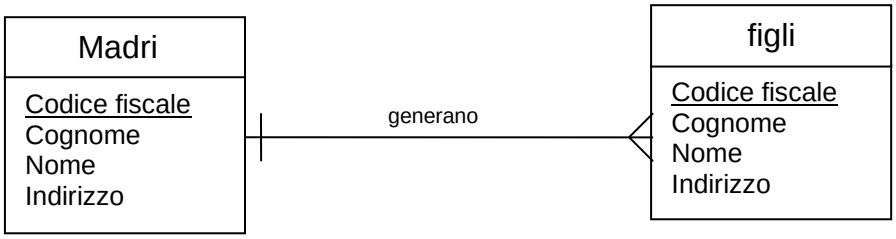
Cardinalità 1 a 1



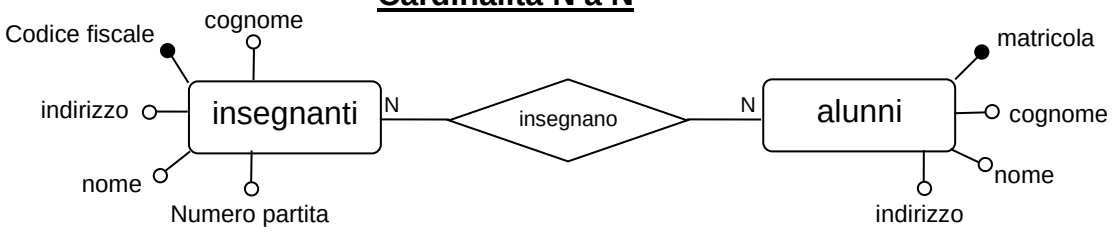


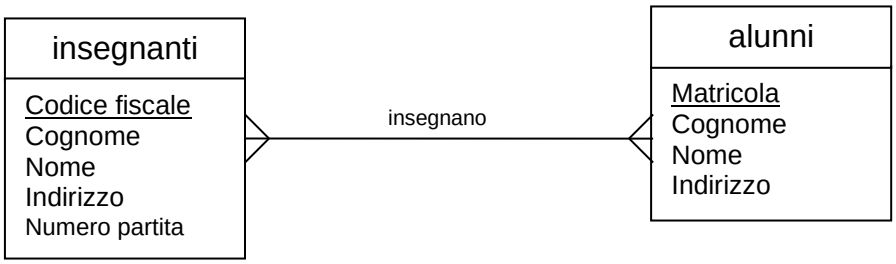
Cardinalità 1 a N





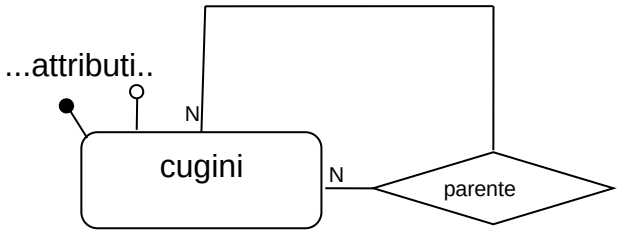
Cardinalità N a N





Le relazioni ricorsive.

Le relazioni ricorsive sono relazioni di un'entità su se stessa. Ad esempio definiamo l'entità ***cugini*** e la relazione ***parente***: la relazione è ricorsiva in quanto un cugino è parente di un altro cugino



Gli attributi delle relazioni.

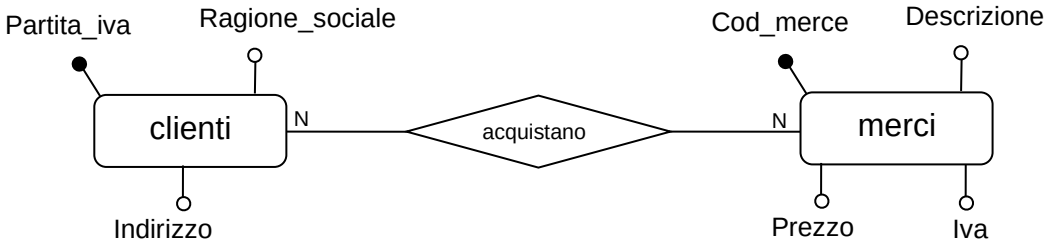
Gli attributi, come detto, caratterizzano ed appartengono strettamente all'entità cui si riferiscono.

Ad esempio, l'attributo cognome dell'entità persona è una caratteristica peculiare dell'entità e si riferisce strettamente ad una precisa istanza dell'entità persona, io ho il mio cognome e non uno a caso!

Supponiamo di avere i seguenti requisiti:

> *Dei clienti acquistano le merci da un grossista; i clienti sono individuati dalla partita iva, dalla ragione sociale, dall'indirizzo, dal telefono. Le merci da un codice, dalla descrizione dal prezzo, dall'iva. I clienti acquistano una certa quantità di merci in date diverse….. le date in cui vengono effettuate gli acquisti dai clienti e la quantità di ogni merce acquistata in quelle date devono essere memorizzate….*

Individuiamo le entità ***merci*** e ***clienti*** legate dalla relazione **acquistano**. La relazione è N : N, infatti un cliente (un'istanza di clienti) può acquistare più merci, una merce (un'istanza di merci) può essere acquistata da più clienti.



dobbiamo, inoltre, memorizzare le date in cui un cliente ha acquistato delle merci e la quantità di merci che in quelle date il cliente ha acquistato.

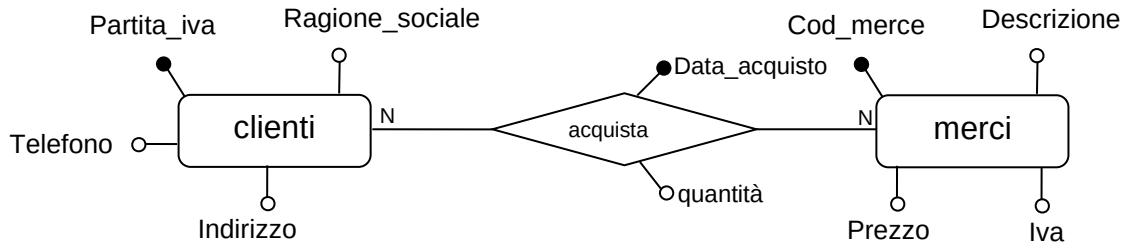
Analizziamo le entità. Entità **clienti:** il cliente è una persona o una società che è caratterizzato dalla partita iva (il codice fiscale delle aziende), la ragione sociale (il nome e cognome di un'azienda), l'indirizzo, il numero di telefono, dati, in poche parole, che si riferiscono all'azienda alla sua anagrafica.

Analogamente l'entità **merci**: la merce è individuata da un codice che la identifica, dalla descrizione , dal prezzo, dal'iva che deve essere applicata quando si fattura la merce acquistata, cioè dati che si riferiscono strettamente alle merci.

La data di acquisto è un dato che possiamo riferire all'anagrafica del cliente? È forse un dato che descrive una merce? Credo che possiamo affermare che la risposta per entrambe le domande sia NO! Stesso ragionamento per la quantità di merce acquistata.

La data in cui un cliente acquista una merce o quanto merce acquista, non sono attributi né dell'entità clienti né merci. Questi attributi, infatti, non appartengono alle entità ma alla relazione ***acquista.***

Tra l'altro posso accorgermi che gli attributi appartengono alla relazione anche dal fatto che spesso nelle specifiche tali attributi sono riferiti direttamente alla relazione con frasi del tipo: "….*le date in cui vengono effettuate gli* ***acquisti*** *dai clienti e la quantità di ogni merce* ***acquistata****…*."



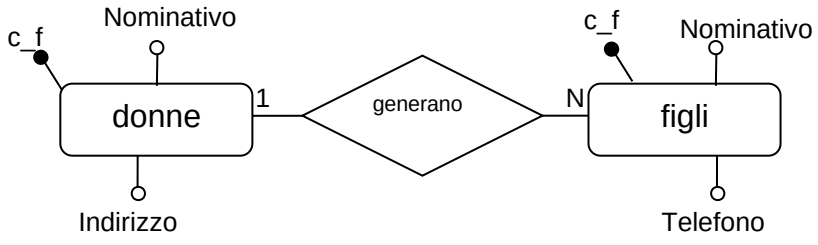
Vediamo un altro esempio:

Gli Alunni di una scuola, individuati da Matricola (univoco), Nome, Cognome, Data di Nascita, Indirizzo di residenza, frequentano durante gli anni scolastici di cui vogliamo tener traccia, le Classi individuati dalla classe, sezione ed indirizzo.

Le relazioni forti e relazioni deboli.

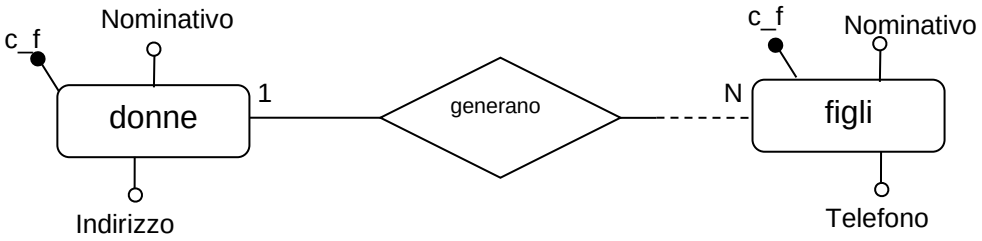
Abbiamo visto che una relazione associa ad un'istanza di un'entità con una o più istanze di un'altra entità e negli esempi che abbiamo finora mostrato ad ogni istanza di un'entità corrisponde *almeno una* istanza dell'altra entità.

Supponiamo di avere:



È facile intuire che non tutte le donne hanno figli mentre i figli sicuramente sono generati da una donna.

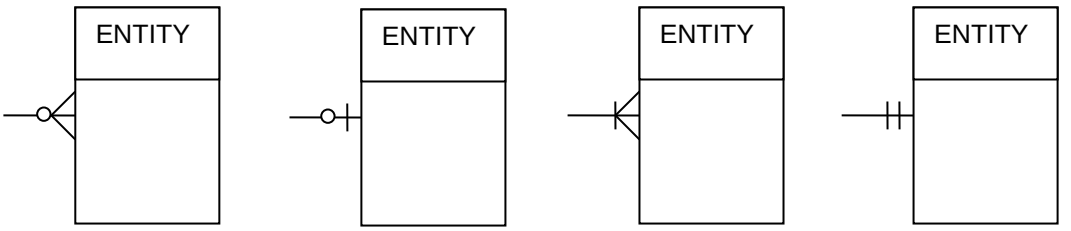
In questo esempio si dice che la relazione *è debole* e si indica con una linea tratteggiata che va verso l'entità in cui non tutte le istanze sono associate:



In questo caso la relazione è detta avere cardinalità (1,1):(0,N) (si legge: 1 a 0,N); analogamente per le altre cardinalità: la coppia (R, C) è un altro modo per indicare la cardinalità della relazione e se la relazione è debole o forte; precisamente R=0 relazione debole, R=1 relazione forte, C = cardinalità 1 o N. Possiamo quindi avere le coppie (0, 1), (1, 1), (0, N), (1, N).

È ovvio, che quando tutte le istanze sono relazionate tra loro, l'associazione si dice *forte*.

N.B. un altro modo per indicare le relazioni deboli e forti è il seguente:



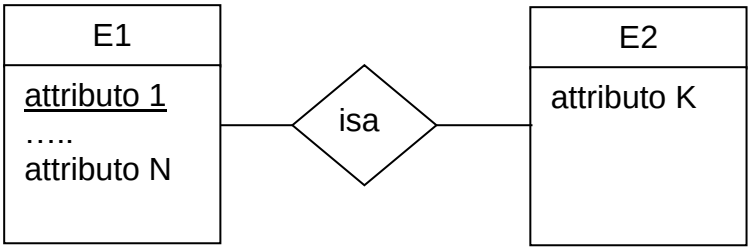
Relazioni 0,N *Relazioni 0,1* *Relazioni 1,N* *Relazioni 1,1*

La cardinalità degli attributi.

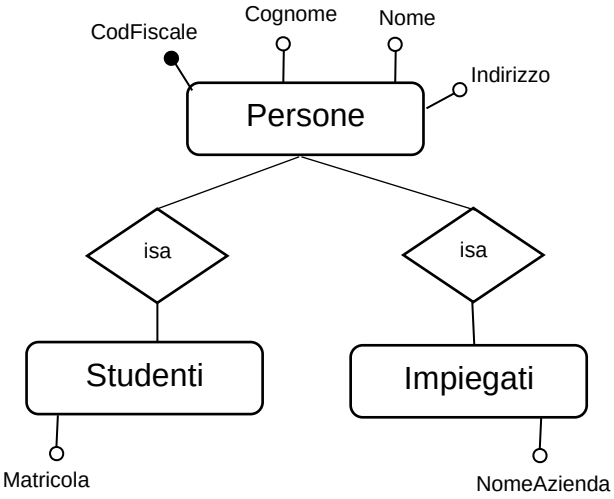
Con cardinalità degli attributi viene indicato il numero minimo e massimo di valori che può assumere quell'attributo all'interno di una entità e si indica con la coppia (min, max) da porre accanto l'attributo: min = 0 significa che l'attributo è opzionale ossia può non assume valori; max assume 1 se l'attributo al massimo assume un valore, N più di uno. Siccome la maggior parte degli attributi assume un solo valore obbligatorio (min = 1, max = 1) viene omesso considerando la coppia (1, 1) come default. Facciamo qualche esempio. Nell'entità *Persona* l'attributo *cognome* può assumere un solo valore ed è ovviamente obbligatorio: quindi se dovessimo indicare la cardinalità di questo attributo dovremmo indicare la coppia (1, 1) (per default non la indichiamo). L'attributo *numTelefono* può essere opzionale perché una persona può avere 0 numeri di telefono (nel caso non abbia telefono) o più di uno: in questo caso la cardinalità viene indicata con (0, N).

Gerarchie ISA.

Se l'entità E2 eredita gli attributi dell'entità E1, ed ha in più rispetto ad essa ulteriori attributi (non chiave) possiamo indicare tale entità con il formalismo:

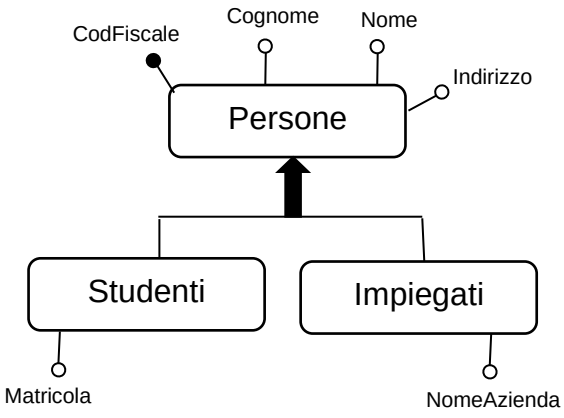


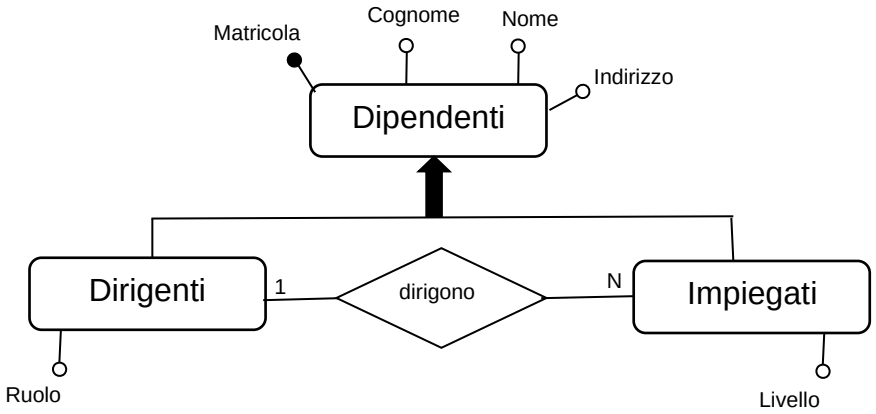
e si dice che **E2 isa E1**. Le gerarchie isa (is a = è un) somigliano molto al concetto di ereditarietà tra classi. E2 viene detta anche entità figlia, E1 entità padre. Facciamo qualche esempio.



L'entità *Studenti* ha tutti gli attributi di *Persone* (compresa la Key ovviamente) ai quali aggiunge l'attributo *Matricola*; così l'entità *Impiegati* ha tutti gli attributi di *Persone* ai quali aggiunge *NomeAzienda.*

Un altro modo per rappresentare una gerarchia isa è:





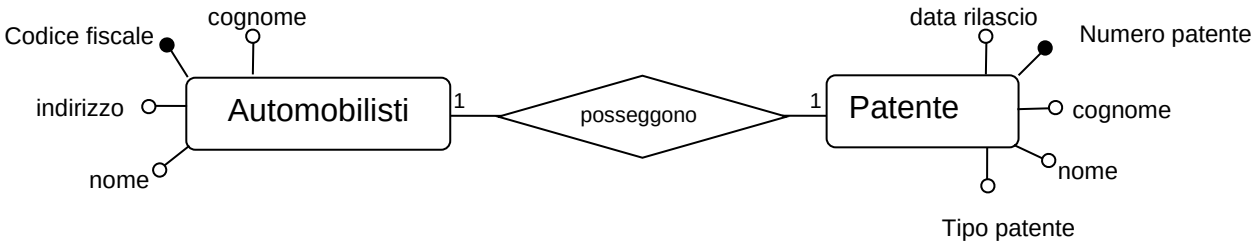
In questo esempio esiste una relazione tra le due entità isa *Dirigenti* e *Impiegati*.

Ottimizzazione del diagramma E-R.

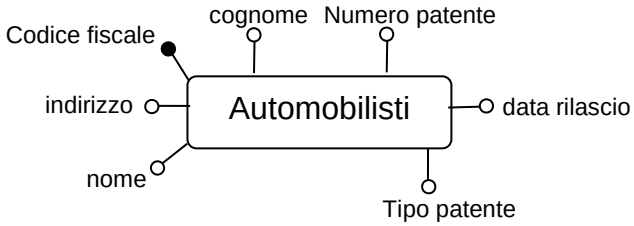
Dopo aver disegnato il diagramma E/R, passiamo ad effettuare la sua ottimizzazione. L'ottimizzazione consiste nella *fusione*, di due entità in una, *scissione* di una entità in più entità, *eliminazione delle relazioni ridondanti*.

Fusione.

Se possibile e se non è chiaramente richiesto dalle specifiche, le entità relazionate con cardinalità 1 a 1 possono essere fuse in un'unica entità. In questo caso gli attributi non comuni di una entità confluiscono nell'altra entità:



L'entità ***Patente*** può confluire nell'entità ***Automobilisti*** che diventa:

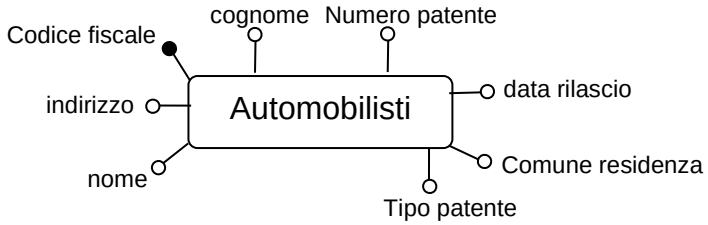


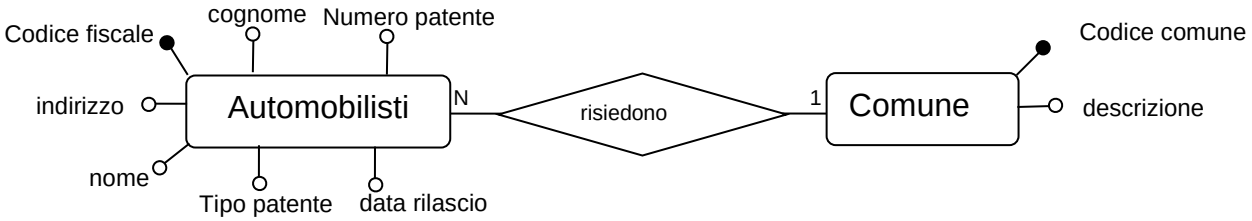
Da notare come il *Numero patente*, chiave nell'entità Patente adesso sia un attributo non chiave nell'entità Automobilista. Il progettista deve infatti scegliere sia l'entità da eliminare che, di conseguenza, la chiave che deve restare.

Scissione.

Alcuni attributi hanno un dominio formato da un range finito di valori. Classico esempio sono, ad esempio, gli attributi di tipo *provincia* (infatti le province sono in Italia un

numero finito), analogamente i *comuni* o tutti quei attributi per i quali nella specifica è enumerato un certo numero di valori che possono assumere come dati (es. Tipo_legno: può assumere i valori pino, quercia, abete, ciliegio…..).
In questo caso dall'attributo si crea un'entità con una chiave (artificiale se non esiste, altrimenti, come nel caso dei comuni italiani e delle province, quelle definita dallo stato italiano) ed una descrizione (che conterrà il nome del comune o il tipo del legno del nostro esempio) e si relazione con l'entità cui apparteneva l'attributo.

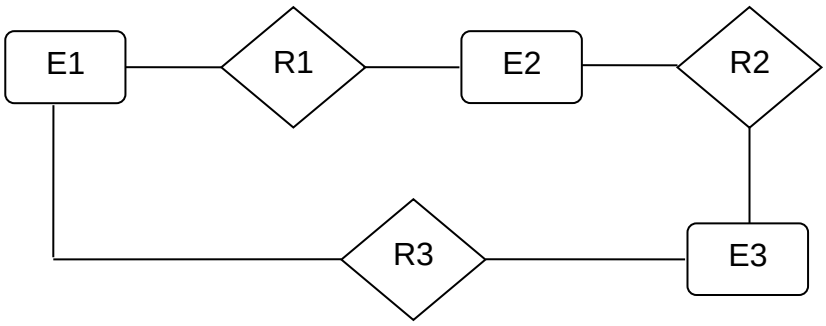




Da notare l'eliminazione dell'attributo *Comune residenza* dall'entità Automobilisti.

Eliminazione delle relazioni ridondanti.

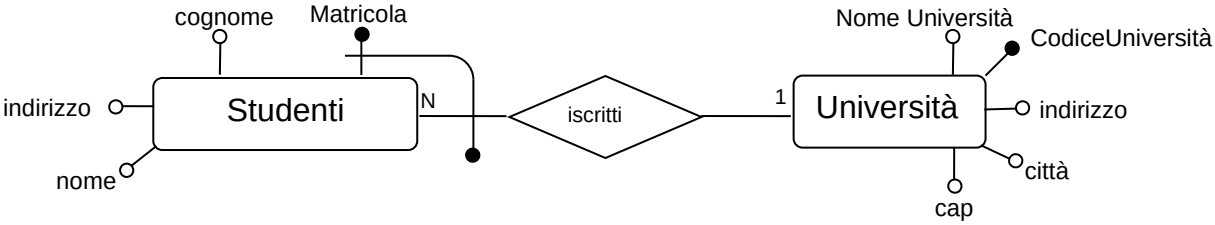
Se ci troviamo davanti uno schema E/R del tipo:



notiamo che le entità E1 ed E3, ad esempio, sono relazionate sia tramite la relazione R3 ma attraverso la 'navigazione' tra le relazioni R1 e R2. In questo caso il progettista può valutare di eliminare la relazione R3.

Identificativo Esterno.

Con la notazione di "***identificativo esterno"*** si indicano i casi in cui la chiave di un'entità può risultare non univoca se, in una relazione con un'altra entità, produce collisioni. Ad esempio:



Lo studente Bianchi iscritto all'Università X con matricola Z (univoca all'interno dell'Università X), può avere la stessa matricola dello studente Rossi iscritto all'Università Y. La matricola Z è univoca all'interno dai due atenei ma risulta non univoca nell'entità Studenti, che raccoglie i dati degli studenti iscritti in diverse Università. Il simbolo utilizzato (il pallino nero legato al segmento ricurvo) indica che la chiave dell'entità Studenti è formata da *Matricola* + *CodiceUniversità*. *CodiceUniversità* viene definito "identificativo esterno".

In realtà Matricola non potrebbe essere definita chiave non possedendo il vincolo dell'univocità. Il progettista dovrebbe, seguendo le indicazioni date in precedenza, creare una chiave artificiale per l'entità **Studenti**, cosa che, lo scrivente, preferisce in quanto più pratica e di migliore gestione.

2.3 Il modello logico.

Il modello concettuale, ci ha permesso di esprimere i dati e le relazioni tra i dati ad un livello alto di astrazione.

Il modello logico è dipendente dal DBMS, ossia da quell'insieme di software che gestiscono le basi di dati.

Nel corso degli anni si sono evoluti principalmente quattro tipi di modelli (quindi quattro tipi di DBMS) di data base:

- modello gerarchico
- modello reticolare
- modello relazionale (RDBMS)
- modello ad oggetti (ODBMS)

quello ad oggi più usato ed oggetto di questo testo, e il modello relazionale.

Il modello logico relazionale

Come abbiamo visto (vedi appunti sui modelli gerarchico e reticolare), le relazioni (associazioni) si riferiscono ai record, 'legano' i record tra loro. L'intuizione di E. F. Codd è stata quella di relazionare non i singoli record, ma **tabelle** (relazioni) di record. Notiamo che il termine relazione assume ora un concetto diverso rispetto a quello visto e definito nel modello E/R. Per evitare confusione utilizziamo il termine

tabella definendola come una disposizione rettangolare di dati suddivisa in righe (tuple) e colonne (campi). Il numero di colonne viene detto ***grado*** della tabella, il numero di righe, ***cardinalità***.

	Campo1	Campo2	Campo3	Campo4	Campo5
tupla 1					
tupla 2					
tupla 3					
tupla 4					

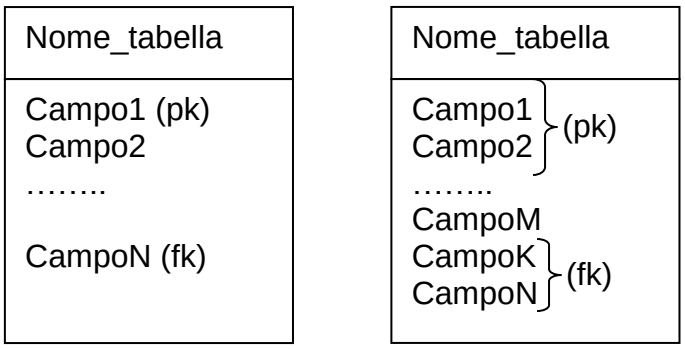
Tabella di grado 5, cardinalità 4

*Definizione 1: definiamo **chiave primaria (primary key – pk)** di una tabella, quel campo o l'insieme dei campi che individua univocamente una tupla della tabella*

*Definizione 2: definiamo **chiave esterna (foreign key – fk)** di una tabella, quel campo o l'insieme dei campi di una tabella che sono chiave primaria in un'altra tabella.*

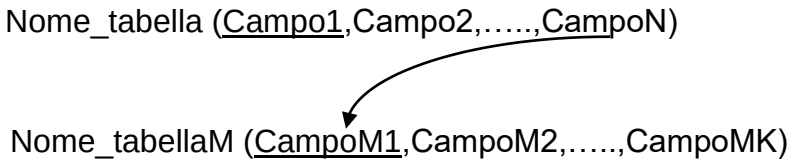
Rappresentazione grafica del modello logico.

Per rappresentare le tabelle del modello logico utilizziamo ancora una volta la simbologia UML:



Le parenesi graffe vengono utilizzate per indicare chiavi formate da più campi.

Alcune volte le tabelle possono essere rappresentate con la seguente notazione:



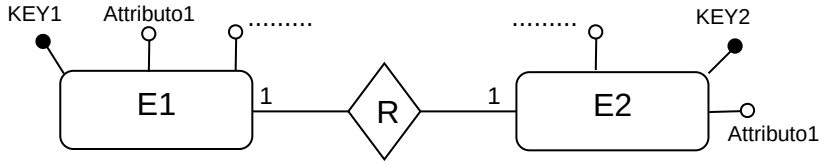
In cui i campi che formano le chiavi primarie sono sottolineate e i campi delle chiave esterne sono associate con le rispettive chiavi primarie tramite una freccia.

Dal modello concettuale al modello logico.

Il passaggio dal modello concettuale al modello logico avviene tramite un procedimento di conversione chiamato **MAPPING** tale procedimento *trasforma* gli attributi delle entità e le relazioni tra esse del modello concettuale in tabelle di dati del modello logico.
Come abbiamo visto nel capitolo precedente, le associazioni tra entità possono essere di tre tipi: associazioni 1 a 1, associazioni 1 a N, associazioni N a N.

Mapping delle associazioni 1 a 1.

Supponiamo di avere due entità E1 ed E2 relazionate da un'associazione R 1:1.



schema concettuale E/R

Per passare al modello logico:

l'entità E1 diventa la tabella T1 in cui:
la l'attributo chiave Key1 diventa il campo primary key della tabella T1
gli attributi Attributo1, Attributo2,…, Attributo N diventano Campo1, Campo2,…, Campo N della tabella T1.

l'entità E2 diventa la tabella T2 in cui:
la l'attributo chiave Key2 diventa il campo primary key della tabella T2
gli attributi Attributo1, Attributo2,…, Attributo M diventano Campo1, Campo2,…, Campo M della tabella T2.

In alternativa, o l'attributo chiave Key1 dell'entità E1 diventa campo foreign key della tabella T2 o l'attributo chiave Key2 dell'entità E2 diventa campo foreign key della tabella T1.

T1
Key1 (pk) Campo 1 Campo 2 …… Campo N

T2
Key2 (pk) Campo 1 Campo 2 …… Campo M Key1 (fk)

oppure

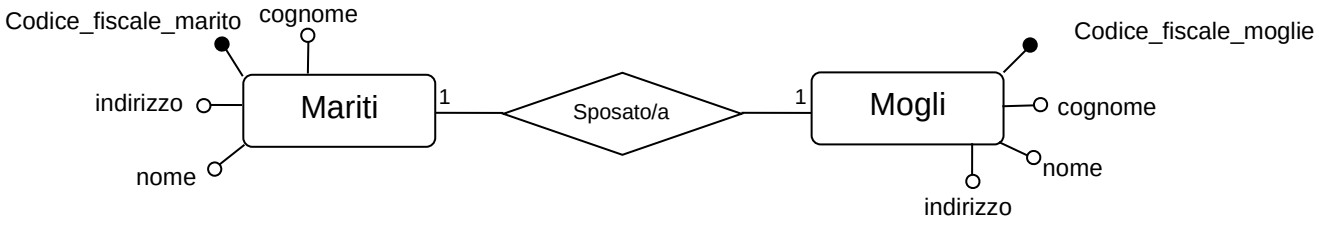
T1
Key1 (pk) Campo 1 Campo 2 …… Campo N Key2 (fk)

T2
Key2 (pk) Campo 1 Campo 2 …… Campo M

schema logico relazionale

Esempio:

Supponiamo di avere il seguente schema E/R:



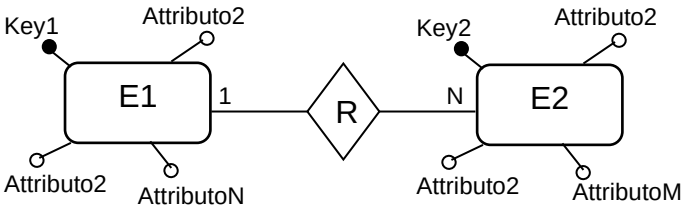
Si trasforma nello schema logico relazionale:

T_mariti
Cod_fiscale_marito (pk) Cognome Nome Indirizzo …… Cod_fiscale_moglie (fk)

T_mogli
Cod_fiscale_moglie (pk) Cognome Nome Indirizzo ……

Mapping delle associazioni 1 a N.

Supponiamo di avere due entità E1 ed E2 relazionate da un'associazione R 1:N.

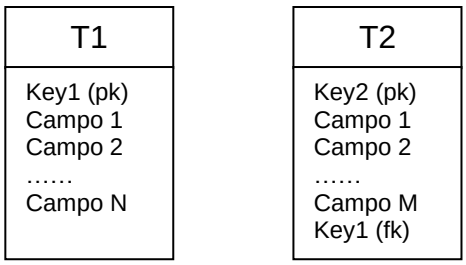


schema concettuale E/R

Per passare al modello logico:

l'entità E1 diventa la tabella T1 in cui:
la l'attributo chiave Key1 diventa il campo primary key della tabella T1
gli attributi Attributo1, Attributo2,…, Attributo N diventano Campo1, Campo2,…, Campo N della tabella T1.
L'entità E2 diventa la tabella T2 in cui:
la l'attributo chiave Key2 diventa il campo primary key della tabella T2
gli attributi Attributo1, Attributo2,…, Attributo M diventano Campo1, Campo2,…, Campo M della tabella T2.

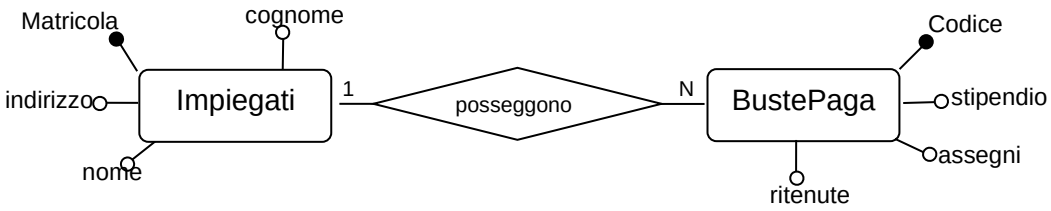
L'attributo chiave Key1 dell'entità E1, dell'entità, per intenderci la cui cardinalità della relazione è 1, diventa campo foreign key della tabella T2.



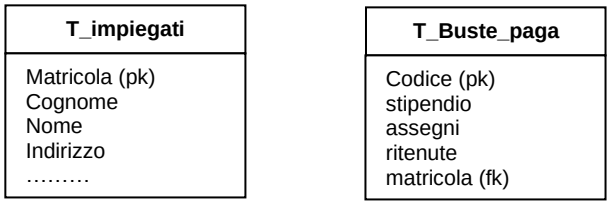
schema logico relazionale

Esempio:

Supponiamo di avere il seguente schema E/R:

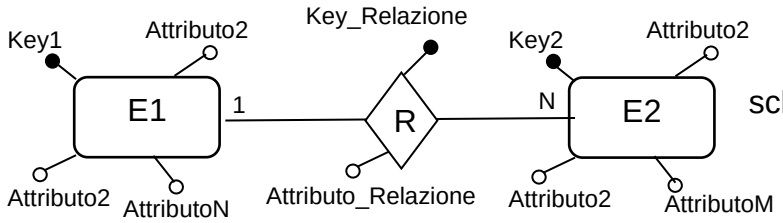


Si trasforma nello schema logico:

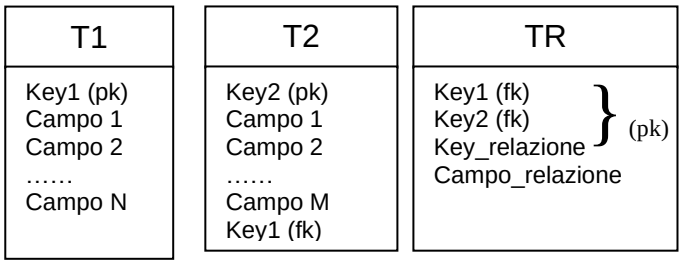


Nel caso in cui siano presenti attributi nella relazione e questi attributi sono tutti non chiave, questi diventano campi nella tabella in cui è stata inserita la foreign key.

Se, invece, abbiamo individuato una chiave tra gli attributi della relazione, allora si crea un'ulteriore tabella (della relazione) la cui primary key è formata dalle due chiavi delle tabelle e dalla chiave della relazione, inoltre, questa nuova tabella, contiene come campi, tutti gli altri eventuali attributi della relazione, in poche parole segue le medesime regole del mapping delle relazioni N a N come vedremo nel paragrafo successivo.

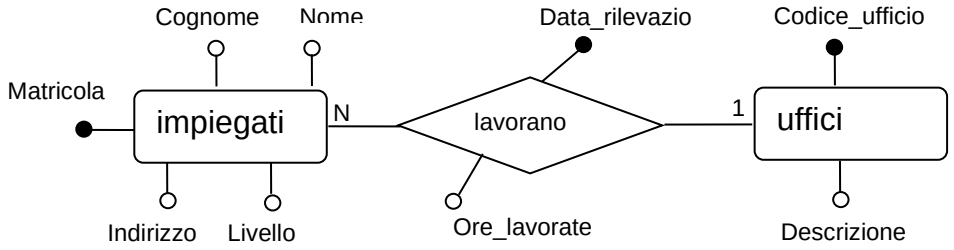


schema concettuale E/R



schema logico relazionale

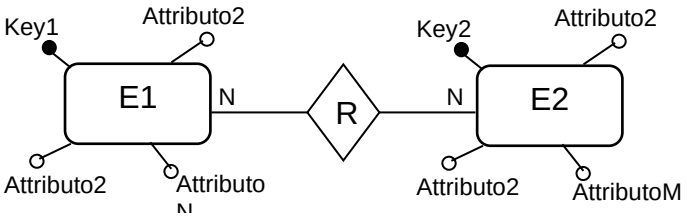
Esempio:



T_impiegati	T_ufficio	T_lavorano
Matricola (pk) Cognome Nome Indirizzo Codice_ufficio(fk)	Codice_ufficio (pk) Descizione	Matricola(fk) Codice_ufficio Data_rilevazione } (pk) Ore_lavorate

Mapping delle associazioni N a N.

Supponiamo di avere due entità E1 ed E2 relazionate da un'associazione R N:N.



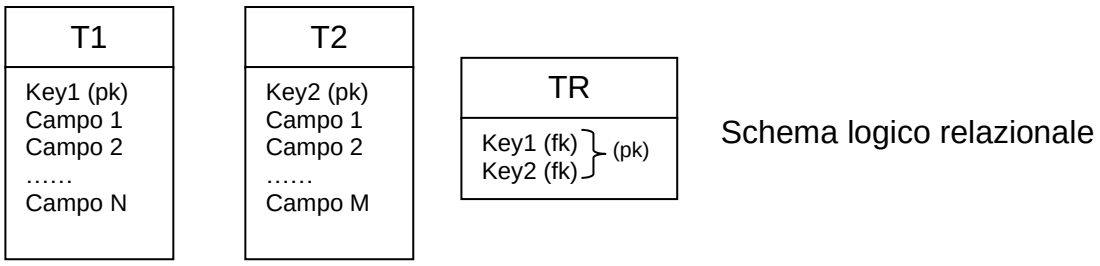
schema concettuale E/R

Per passare al modello logico:

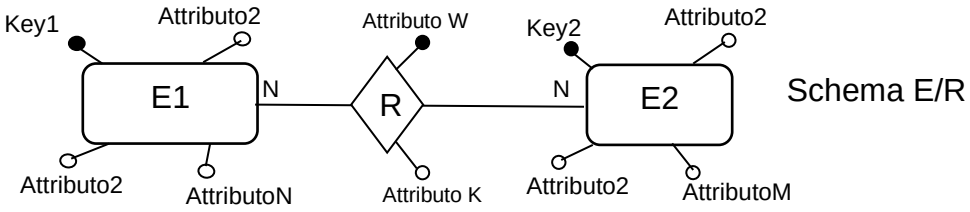
l'entità E1 diventa la tabella T1 in cui:
la l'attributo chiave Key1 diventa il campo primary key della tabella T1
gli attributi Attributo1, Attributo2,…, Attributo N diventano Campo1, Campo2,…, Campo N della tabella T1.

L'entità E2 diventa la tabella T2 in cui:
la l'attributo chiave Key2 diventa il campo primary key della tabella T2
gli attributi Attributo1, Attributo2,…, Attributo M diventano Campo1, Campo2,…, Campo M della tabella T2.

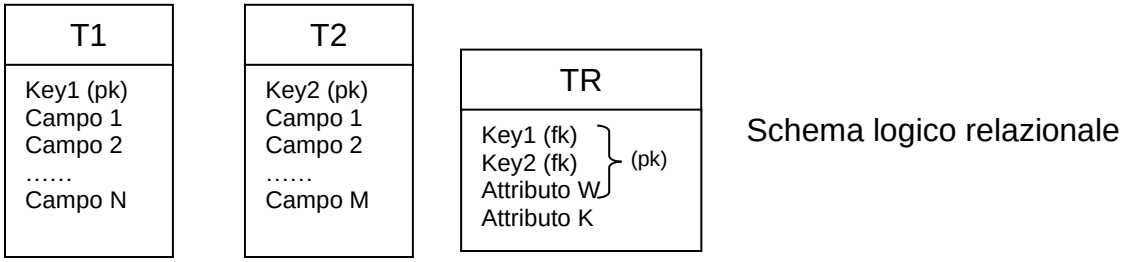
Si crea una tabella, TR, relativa alla relazione in cui la primary key è formata da entrambe le primary key delle tabelle T1 e T2 che, a loro volta, sono foreign key delle tabelle di provenienza; gli eventuali attributi della relazione diventano campi della tabella.



Vediamo il caso in cui sono presenti attributi della Relazione:



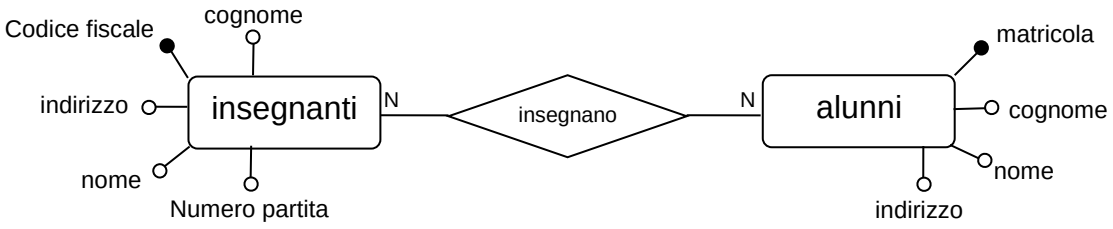
Passando allo schema logico:



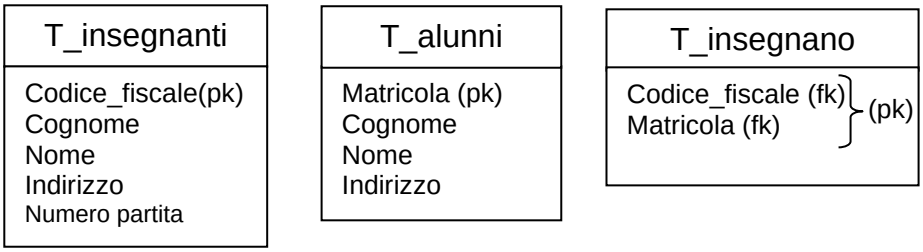
In questo caso, come si vede, gli attributi della relazione diventano campi della tabella TR e gli attributi chiave della relazione entrano nella primary key di TR insieme con le chiavi delle entità E1 ed E2.

Esempio.

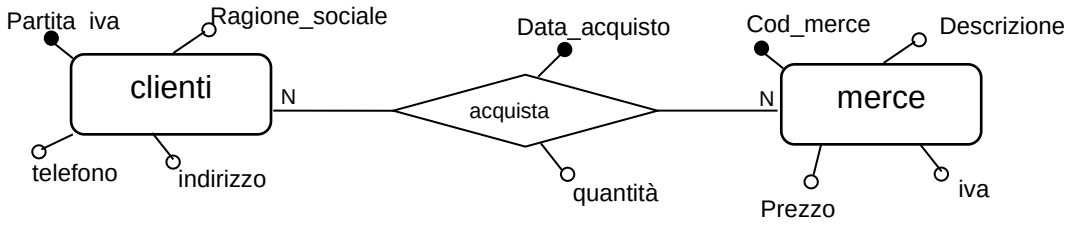
Supponiamo di avere in seguente schema E/R



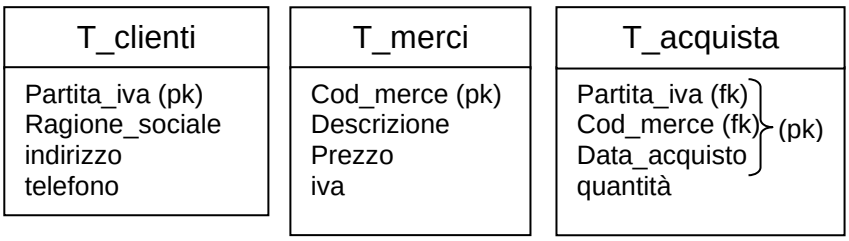
Passando allo schema logico abbiamo:



Esaminiamo il caso di una relazione con attributi:



Passando allo schema logico:



Esaminiamo perché *data_acquisto* fa parte della primary key della tabella *T_acquista*.

Supponiamo che il cliente X la cui partita iva è xxxxxxxxxxx, acquisti in date diverse la stesse merce di codice yyy: se *data_acquisto* non fa parte della primary key della tabella, avremmo due tuple con la stessa chiave formata da xxxxxxxxxxx e yyy partita iva e codice merce rispettivamente delle tabelle T_clienti e T_merci confluite nella tabella T_acquista come primary key, in corrispondenza dei due acquisti effettuati nelle due date diverse.

T_acquista

Partita_iva	***Cod_merce***	***Data_acquisto***	***Quantità***
xxxxxxxxxxx	yyy	10/12/2004	348
xxxxxxxxxxx	yyy	15/12/2004	279
……	…	……	……

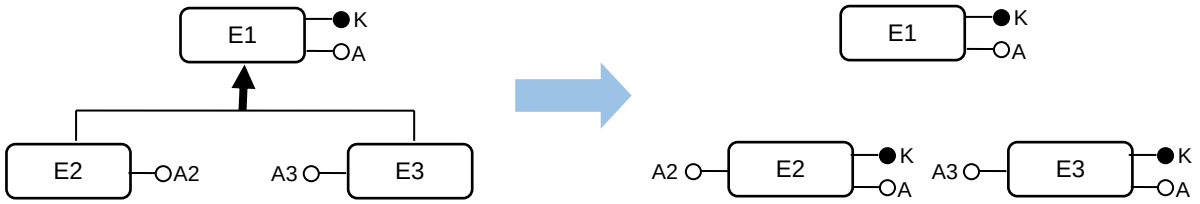
Chiave primaria: errore chiave duplicata!!!
Non individua univocamente la tupla.

Mapping delle gerarchie isa.

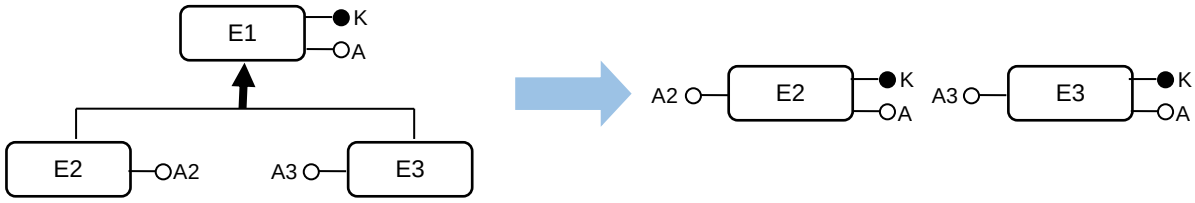
Come abbiamo visto una gerarchia **isa** è una gerarchia tra entità ***padre*** ed entità ***figlie*** che ereditano gli attributi dei padri ai quali aggiungono degli attributi propri.
L'operazione che prelude il mapping di entità legate da gerarchie isa è quella di "eliminare" le gerarchie dal modello E/R. Ci sono tre modi per eliminare le gerarchie:

1. ***Mantenimento delle entità associate gerarchicamente*;**
2. **Collasso verso il basso;**
3. **Collasso verso l'alto.**

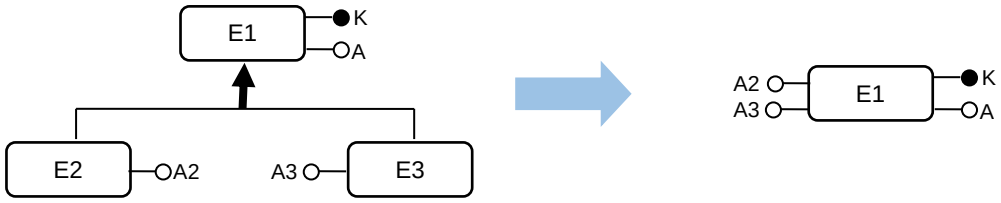
Nel *mantenimento delle entità associate gerarchicamente*, tutte le entità della gerarchia rimangono come entità: le entità figlie si "sganciano" dall'entità padre ereditando tutti gli attributi del padre:



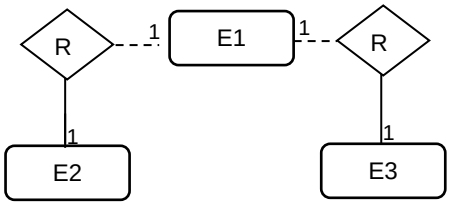
Nel *collasso verso il basso* si elimina l'entità padre e le entità figlie ereditano gli attributi dell'entità padre:



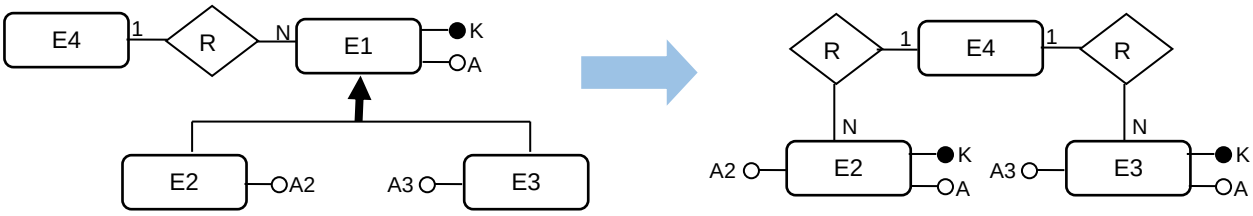
Nel collasso verso l'alto, scompaiono le entità figlie e l'entità padre acquisisce gli attributi delle entità figlie:



Una volta eliminate le gerarchie bisogna collegare le entità nel modello E/R con le relazioni.
Se si è scelto il *mantenimento delle entità associate gerarchicamente* le entità figlie vengono relazionate (0,1) con l'entità padre, nel senso che le istanze delle entità figlie sono relazionate con almeno una istanza dell'entità padre, mentre una istanza dell'entità padre può non essere relazionata con le istanze dell'entità figlie:



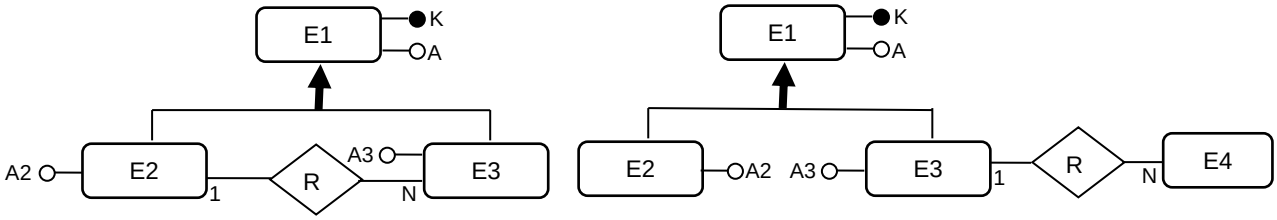
Se si è scelto il *collasso verso il basso* le entità figlie mantengono le stesse relazioni e cardinalità che aveva l'entità padre con le altre entità del modello E/R.



Il *collasso verso l'alto* non avrà provocato alcuna modifica all'entità padre relativamente alle relazioni con le altre entità con le quali era collegato, per cui non è necessaria alcuna azione.

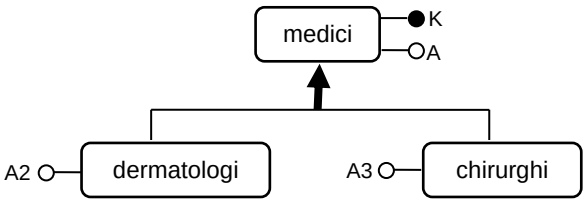
La domanda che di solito viene posta è: quale tra questi modi si deve scegliere per eliminare le gerarchie? La risposta è… dipende. Cominciamo però con l'osservare che entità figlie in relazione tra loro o in relazione con altre entità non appartenenti alla gerarchia sconsiglia il collasso verso l'alto, almeno in modo totale (ossia di tutte le figlie)...

Esempio:



Mentre il collasso verso l'alto è consigliato nel caso delle specializzazioni, sempre se non si verificano le condizioni sopra esposte.

Esempio:

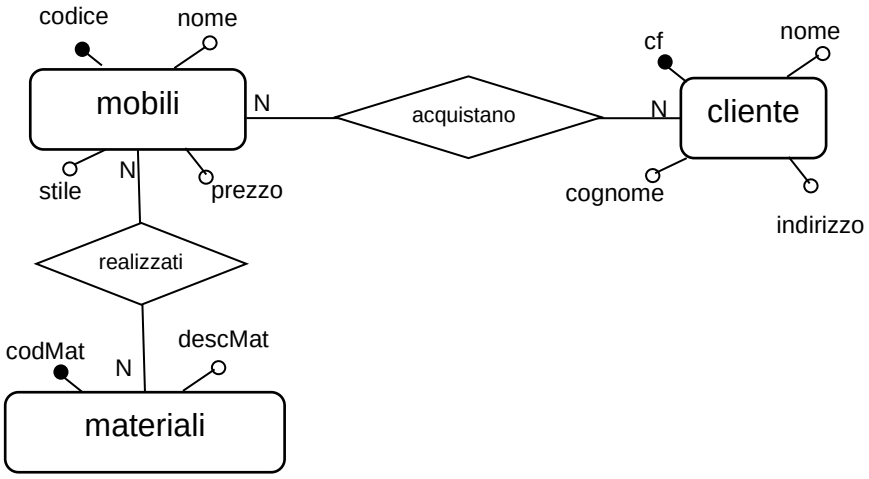


Una volta che si sono eliminate le gerarchie, si procede alla realizzazione del modello logico con le regole di mapping esposte nei capitoli precedenti.

Entità associative.

Supponiamo di avere il seguente problema:
“un Mobilificio produce mobili di cui vogliamo conservare il codice, il nome del mobile, lo stile, il prezzo, i materiali con cui è prodotto. Lo stesso mobile può essere realizzato in materiali diversi. I materiali che usa il mobilificio sono: legno, alluminio, ferro, plastiche… i mobili vengono acquistati da clienti di cui interessa memorizzare i dati anagrafici….”

Ad una prima analisi il modello E/R che deriva dal problema su esposto è:



Il modello logico derivante è:

Tmobili (<u>codice</u>, nome, stile, prezzo)
Tmateriali (<u>codMat</u>, descMat)
Tcliente (<u>cf</u>, nome, cognome, indirizzo)
Trealizzati (<u>codice(*), codMat(*)</u>)
Tacquistano (<u>codice(*), cf(*)</u>)

In cui *codMat* e *descMat* sono rispettivamente la chiave artificiale e l’attributo che descrive il materiale per l’entità ***materiali*** dovuta alla scissione dell’entità ***mobili***: le due entità sono associate tramite la relazione N a N ***realizzati*** (gli asterisco indicano le foreign key con le primary key nominate nello stesso modo nelle altre tabelle).

Il problema è che se voglio conoscere il materiale di un mobile acquistato da un determinato cliente, siccome la relazione consente che lo stesso mobile può essere realizzato in materiali diversi, dalle tabelle risultanti, tale informazione è impossibile da reperire.

Esempio:

supponiamo che le tabelle del modello precedente contengano i dati seguenti:

Tmobili			
Codice	**Nome**	**Stile**	**Prezzo**
1	soggiorno	Liberty	1.500,00
2	libreria	Classico	340,00

Tcliente			
Cf	**Nome**	**Cognome**	**Indirizzo**
RM23	Mario	Rossi	Via Risorgimento
SF12	Francesca	Stoppa	c.so Mazzini

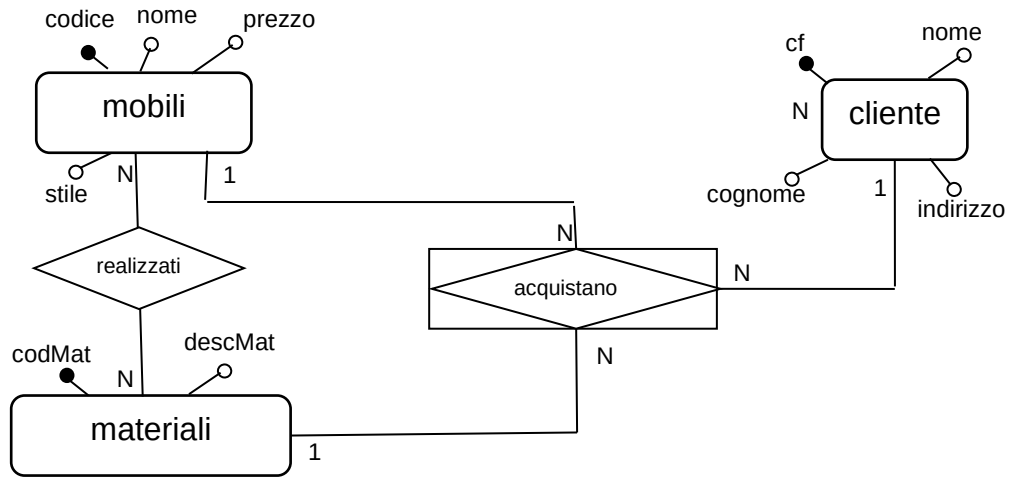
Tmateriali	
CodMat	**descMat**
1m	legno
2m	alluminio

Trealizzati	
Codice	**CodMat**
1	1m
2	1m
2	2m

Tacquistano	
Cf	**Codice**
RM23	2
……	……..

Dalle tabelle, in particolare dalla tabella Tacquistano, risulta che il sig. Mario Rossi ha acquistato una libreria. Sapete, però, indicare, visto che il mobilificio produce la stessa libreria sia in legno che in alluminio, quale tra queste due tipologie di mobile ha acquistato il sig. Rossi?

Bene, penso che il con le tabelle risultanti dal modello E/R, non si riesce a risalire all'informazione cercata. Per risolvere il problema si fa uso delle "entità associative" che ci permettono di specificare meglio le informazioni associando entità che sono relazionate N a N tra loro ma, come risulta dall'esempio precedente, ma che da sole queste relazioni non ci permettono di risalire ad alcuni dati in modo preciso.



Le entità associative vengono indicate con il simbolo che vedete in figura (un rombo racchiuso in un rettangolo), possono avere degli attributi propri (come una qualsiasi entità- anche key) e possono essere relazionate ad altre entità. Nello stesso tempo si comportano come vere e proprie relazioni con le entità ad esse associate ma, a differenza delle relazioni che abbiamo studiato finora, una stessa entità-associativa può essere associata, contemporaneamente a più entità invece che solo a due come normalmente accade alle sole relazioni.

Di solito, le entità associative sostituiscono le relazioni N a N con relazioni N a 1 come mostrato in figura e sarebbe opportuno inserire in queste entità almeno una chiave artificiale se proprio non riusciamo ad individuarne alcuna nelle specifiche che ci sono state fornite. Possiamo, comunque, formare una chiave tramite le foreign key che naturalmente confluiscono nell'entità.

Nel nostro esempio avremo il seguente modello logico:

Tmobili (<u>codice</u>, nome, stile, prezzo)
Tmateriali (<u>codMat</u>, descMat)
Tcliente (<u>cf</u>, nome, cognome, indirizzo)
Trealizzati (<u>codice (*), codMat (*)</u>)
Tacquistano (<u>codice (*), cf (*), codMat (*)</u>) ⟸ la primary key formata dai

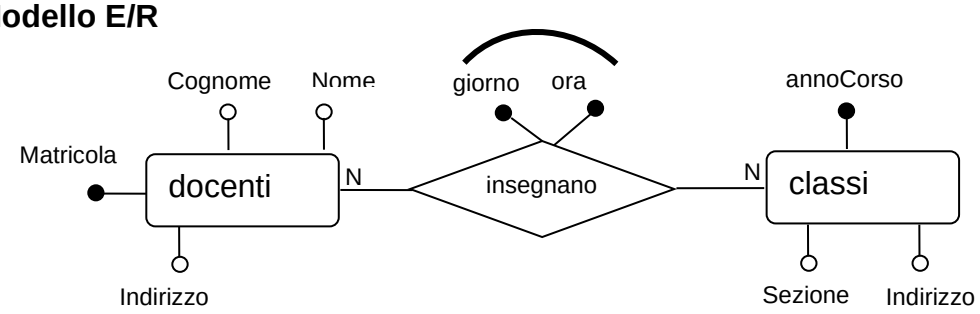
campi *codice, cf, codMat* è stata creata in quanto l'entità associativa era priva di chiave nel modello E/R.

Esempio.

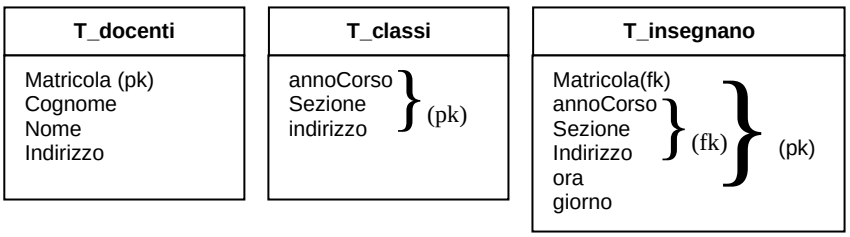
Sia dato il seguente problema: i docenti di una scuola insegnano in diverse classi in determinati orari (1 – 6) durante la settimana (L – S). Dei docenti si vuole memorizzare i dati anagrafici, della classe l'anno di corso, la sezione, l'indirizzo di studio.

Soluzione 1:

Modello E/R

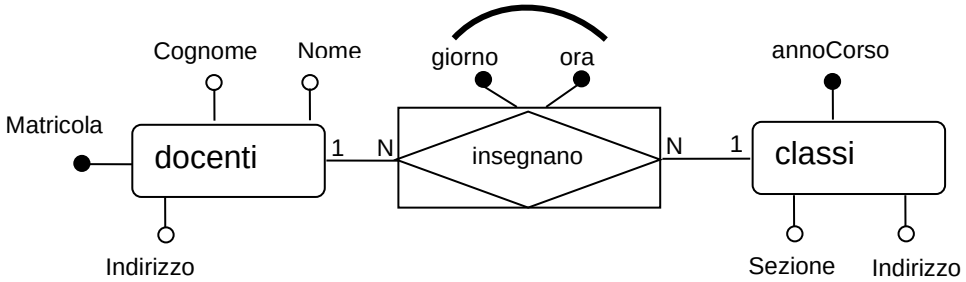


Modello logico

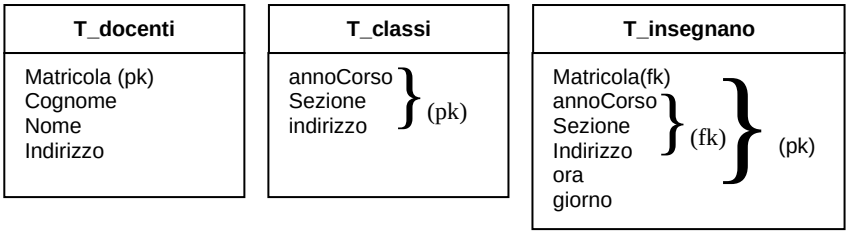


Soluzione 2:

Modello E/R



Modello logico



Come si può notare le due soluzioni sono identiche solo che nella soluzione 2, la chiave primaria va estesa a tutti i campi della tabella *T-insegnano* e non solo a ora e giorno….

Dominio dei campi.

Nel paragrafo ***le entità e gli attributi*** abbiamo evidenziato che è buona norma definire le caratteristiche degli attributi: il formato, la dimensione, l'opzionalità, il range dei valori ammissibili.

Tutte queste informazioni determinano il *dominio* di un campo. A questo punto della progettazione del data base è necessario esplicitare i domini dei campi delle tabelle secondo le seguenti caratteristiche:

CARATTERISTICA	***VALORE ASSUNTO***	***DESCRIZIONE***
formato	Alfanumerico	caratteri e stringhe di caratteri
	Numerico	intero, reale, virgola fissa, virgola mobile
	Booleano	vero/falso
	Data	gg/mm/aaaa, gg.mm.aaaa, aaaa/gg/mm, ecc.
	Valuta	
dimensione		Dimensione del campo in byte
opzionalità	Null	il campo può essere omesso
	Not null	il campo è obbligatorio
range		Insieme di valori che il campo pùò assumere

Esempio:

Nome Tabella				
NOME CAMPO	***FORMATO***	***DIMENSIONE***	***OPZIONALITÀ***	***RANGE***
Codice_fiscale	Alfanumerico	16	Not null	
Cognome	Alfanumerico	100	Not null	
Età	Numerico	3	Null	>0
Sesso	Alfanumerico	1	Not null	f/m

Vincoli d'integrità.

I vincoli d'integrità rappresentano le regole che devono essere rispettate tra i campi di una tabella o tra tabelle che erano relazionate nel modello concettuale.

Questi vincoli si dividono tra vincoli intra-relazionali (vincoli interni - si riferiscono ai campi di una tabella) ed inter-relazionali (vincoli esterni – vincoli tra tabelle).

Vincoli intra-relazionali.

I vincoli itra-relazionali si riferiscono a vincoli che si applicano su un singolo campo (di dominio), su una singola tupla, vincoli su più tuple e sulla primary key.

I vincoli di tupla sono quelle regole che devono essere rispettate tra i campi di una singola tupla.

Esempio. Se abbiamo la tabella con i campi:
VOTI (matricola, esame, data, voto, lode) il campo *lode* è legato al campo *voto* dalla regola che: può essere assegnata la lode se e solo se il voto è 30.

I vincoli di dominio sono quelle regole che si applicano sul singolo campo.

Esempio.

Nella tabella precedente abbiamo che il campo voto può assumere i valori compresi tra 18 e 30. Il range che abbiamo definito nel dominio è un vincolo di dominio.

Esempio.

Sempre nell'esempio della tabella VOTI: il campo *data* ed il campo *voto* non possono assumere valori nulli (ossia devono essere NOT NULL).

Il vincolo su più tuple si riferisce al vincolo che impone l'unicità della chiave primaria: non possono esistere due tuple con valore uguale della chiave primaria.

Il vincolo di primary key impone alla chiave primaria non poter assumere valori *null*.

Vincoli inter-relazionali.

I vincoli inter-relazionali sono i vincoli tra più tabelle che sono relazionate nel modello concettuale.

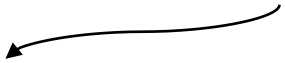
Come visto le entità relazionate generano nelle tabelle, chiave esterne (foreign key). In poche parole le foreign key rappresentano nel modello logico, il legame espresso nel modello concettuale dalle relazioni.

Sulle foreign key posso stabilire un vincolo detto "d'integrità referenziale". Ricordandoci che la chiave esterna è un campo di una tabella che è chiave primaria in un'altra, se applico il vincolo d'integrità referenziale sulla chiave esterna di una tabella, questa non può assumere *valori* che non siano presenti nella chiave primaria della tabella ad essa collegata.

Esempio.

Supponiamo di avere le due tabelle:

IMPIEGATI (matricola, cognome, nome, età, Codice_specializzazione)



SPECIALIZZAZIONE (Codice_specializzazione, descrizione)

IMPIEGATI				
Matricola	Cognome	Nome	Età	Codice_specializzazione
0001	Rossi	Mario	56	S1
0002	Bianchi	Antonio	35	S2

SPECIALIZZAZIONE	
codice_specializzazione	descrizione
S1	Capo area
S2	magazziniere
S3	contabile

In questo caso al campo codice_specializzazione, chiave esterna della tabella IMPIEGATI, è stato applicato il vincolo d'integrità referenziale: non può assumere valori diversi da S1, S2, S3, valori assunti dalla chiave primaria della tabella SPECIALIZZAZIONE.

Dipendenza funzionale.

Siano A e B due insiemi non vuoti di attributi o campi di una tabella (relazione). Se ogni combinazione di *valori* di A, determina un unico insieme di *valori* di B, allora si dice che B ha una dipendenza funzionale da A.
Si scrive:

$$\mathbf{A \longrightarrow B}$$

Esempio.

Supponiamo di avere la tabella ANAGRAFICA(codice_fiscale, cognome, nome, indirizzo) poniamo B {cognome, nome, indirizzo } e A {codice_fiscale} è facile verificare che gli elementi di B dipendono funzionalmente da A. Infatti a codici fiscali diversi (ossia a valori del campo ***codice_fiscale***) sono associati diversi *valori* dei campi ***cognome***, ***nome*** ed ***indirizzo***.

Transitività della dipendenza funzionale.

Siano A, B e C tre insiemi non vuoti di attributi di una tabella. Se:

$$\mathbf{A \longrightarrow B} \text{ e } \mathbf{B \longrightarrow C} \text{ allora } \mathbf{A \longrightarrow C}$$

Esempio.

Supponiamo di avere la tabella MERCE(codice, tipo_merce, collocazione) sia A {codice}, B {tipo_merce} e C {collocazione}

Indicando con ***codice*** il codice numerico della merce, ***tipo_merce*** la tipologia merceologica, ***collocazione*** il ripiano o il posto in cui viene stoccata quella tipologia di merce

È facile dimostrare che il ***tipo_merce*** dipende dal codice, la ***collocazione*** dal ***tipo_merce*** e, di conseguenza, la ***collocazione*** dipende anch'esso dal ***codice***.

La Normalizzazione.

La normalizzazione permette di verificare la correttezza di una base dati. Per correttezza si intende l'assenza di ridondanza ed incoerenza tra i dati presenti in una tabella.

Per verificare ciò, ogni tabella deve rispettare delle 'regole' chiamate *forme normali NF (Normal Form).*

Il procedimento che permette di raffinare le tabelle in modo tale che queste rispettino le forme normali si chiama *normalizzazione*.
questo procedimento permette, in definitiva:

1. di costruire uno schema logico finale privo di dati duplicati;
2. individuare perfettamente le chiavi primarie;
3. definire le associazione tra le diverse tabelle con l'individuazione corretta delle chiave esterne.

Vediamo, quindi, le forme normali.

La prima Forma Normale (1 NF).

La prima forma normale dice che:

> ***1 NF.***
>
> ***Una tabella è in prima forma normale (1 NF) se ogni campo della tabella è atomico.***

Ricordate gli attributi composti del modello concettuale? Bene, quando passiamo nel modello logico, questi attributi, devono essere scomposti in attributi semplici non ulteriormente decomponibili ossia atomici.

Esempio.

Supponiamo di avere la seguente entità:

insegnanti
Codice fiscale Cognome Nome *Indirizzo* Numero partita

In cui l'attributo *Indirizzo* è un attributo ***composto*** dagli attributi semplici *Nome_via*, *Cap*, *Città.* La tabella che deriva dall'entità ***Insegnanti***, deve essere:

T_insegnanti
Codice fiscale (pk) Cognome Nome *Nome_via* *CAP* *Città* Numero partita

La seconda Forma Normale (2 NF).

Affinché una tabella sia in seconda forma normale deve rispettare la seguente regola:

2 NF.

Una tabella è in seconda forma normale se è in prima forma normale e se ogni campo non chiave dipende univocamente dalla chiave primaria (e non da parte di essa).

Esempio.

Supponiamo di avere la seguente tabella:

ACQUISTI (codice_merce, p_iva, data_acquisto, quantità_acquistata, descrizione_merce, prezzo_totale)
In cui:
codice_merce è un codice numerico che individua la tipologia di merce,
p_iva è la partita iva dell'acquirente,
data_acquisto è la data in cui è stato effettuato l'acquisto,
quantità_acquistata indica la quantita acquistata di quella tipologia di merce in quella data,
descrizione_merce è la descrizione della tipologia di merce,
prezzo_totale indica il costo pagato per l'acquisto di quella merce.

tale tabella **non rispetta** la 2 NF: infatti il campo *descrizione_merce* dipende solo dal *codice_merce* e non da tutta la chiave (il valore del campo, es. **mortadella**, non cambia né con il variare degli acquirenti né se l'acquisto è fatto in date diverse).
Per portare la tabella in 2 NF bisogna dividere la tabella in due tabelle:

ACQUISTI (codice_merce, p_iva, data_acquisto, quantità_acquistata, prezzo_totale)
↓
MERCE (codice_merce, descrizione)

Abbiamo, quindi, tolto il campo che faceva violare la 2 NF alla tabella *ACQUISTI* ed abbiamo creato una nuova tabella, *MERCE*, formata dai campi *cod_merce* e *descrizione*: siccome *descrizione* dipende da *cod_merce*, nella tabella *MERCE, cod_merce* è la chiave primaria.
Le due tabelle così realizzate rispettano la 2 NF.

La terza Forma Normale (3 NF).

3 NF.

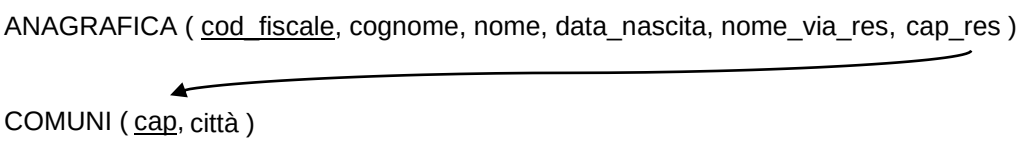
Una tabella è in terza forma normale se è in seconda forma normale e se ogni campo non chiave dipende esclusivamente dalla chiave primaria (e non da altri campi non chiave).

Esempio.

Supponiamo di avere la seguente tabella:

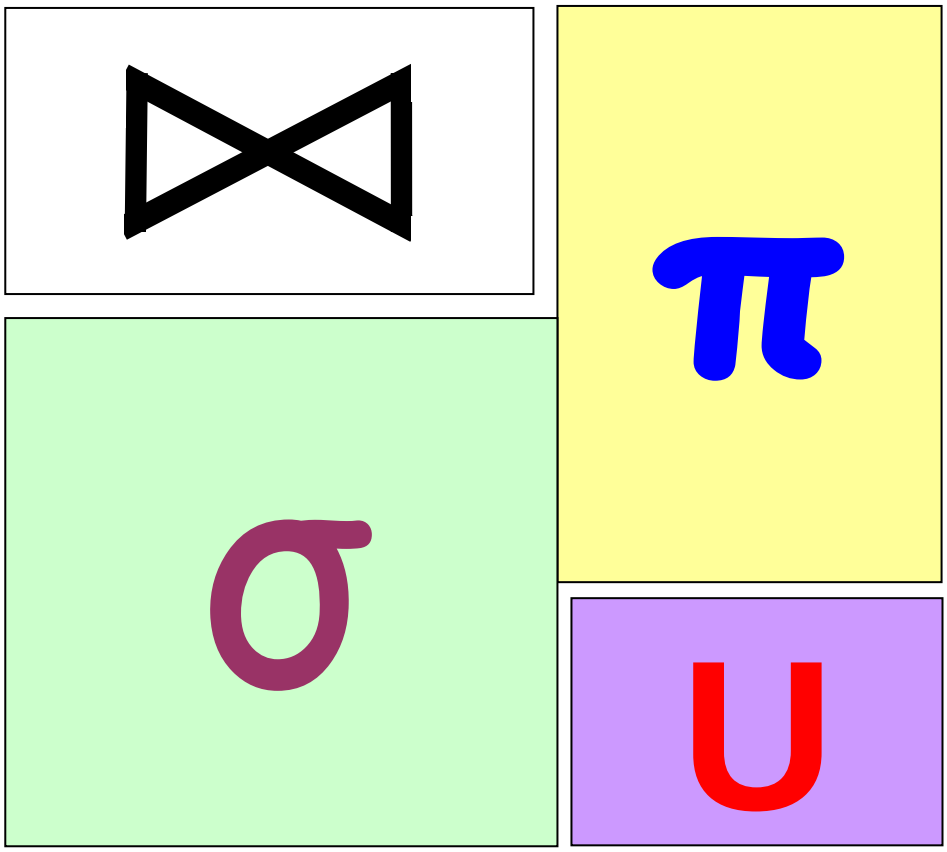
ANAGRAFICA (cod_fiscale, cognome, nome, data_nascita, nome_via_res, cap_res, città_res)

Come, si può notare, tutti i campi non chiave della tabella dipendono dalla chiave primaria, quindi è rispettata la 2 NF, ma il campo *città_res* dipende anche dal *cap_res*.
La tabella *ANAGRAFICA* **non rispetta**, in definitiva, la 3 NF.
Per risolvere il problema dobbiamo procedere come nell'esempio fatto per la 2 NF.



Nella tabella ANAGRAFICA si lascia il campo cap_res come chiave esterna alla nuova tabella COMUNI nella quale cap diventa chiave primaria.

3. L'Algebra Relazionale.

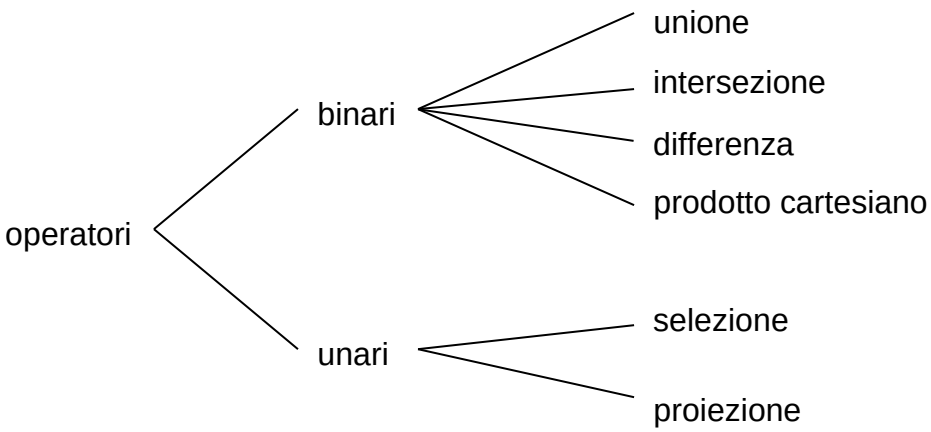


Codd, introdusse, nel 1970, un linguaggio di tipo algebrico per manipolare, tramite operatori, le tabelle relazionali come fossero operandi e, proprio come avviene nell'algebra matematica, il risultato di tale manipolazione è ancora una tabella.

Gli operatori relazionali.

Gli operatori relazionali si dividono in ***operatori binari***, operatori agiscono su due operandi ed ***operatori unari***, operatori che agiscono su un solo operando.

Nello schema sottostante sono indicati gli operatori principali.



Unione

L'unione si applica su due tabelle di ugual grado e le cui colonne abbiano lo stesso dominio.

Siano A e B due tabelle, l'unione A U B da come risultato una tabella di ugual grado di A e B con le colonne con lo stesso dominio di A e B e come righe le righe di A e quelle di B non presenti in A.

Esempio.

A			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
004	Bianchi	22	19
007	Verdi	20	19
010	Neri	19	23

B			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
003	Marroni	28	17
006	Blu	18	24
010	Neri	19	23

A U B			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
004	Bianchi	22	19
007	Verdi	20	19
010	Neri	19	23
003	Marroni	28	17
006	Blu	18	24

Intersezione.

L'intersezione si applica su due tabelle di ugual grado e le cui colonne abbiano lo stesso dominio.

Siano A e B due tabelle, l'intersezione A ∩ B da come risultato una tabella di ugual grado di A e B con le colonne con lo stesso dominio di A e B e come righe le righe comuni di A e B.

Esempio.

A			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
004	Bianchi	22	19
007	Verdi	20	19
010	Neri	19	23

B			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
003	Marroni	28	17
006	Blu	18	24
010	Neri	19	23

A ∩ B			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
010	Neri	19	23

Differenza.

La differenza si applica su due tabelle di ugual grado e le cui colonne abbiano lo stesso dominio.

Siano A e B due tabelle, la differenza A - B da come risultato una tabella di ugual grado di A e B con le colonne con lo stesso dominio di A e B e come righe le righe di A non presenti in B.

Esempio.

A			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
004	Bianchi	22	19
007	Verdi	20	19
010	Neri	19	23

B			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
003	Marroni	28	17
006	Blu	18	24
010	Neri	19	23

A - B			
Codice	Cognome	Età	Punti segnati
004	Bianchi	22	19
007	Verdi	20	19

Prodotto cartesiano.

Il prodotto cartesiano si applica su tabelle di qualsiasi grado e con qualsiasi dominio delle colonne.

Siano A e B due tabelle, il prodotto cartesiano A X B da come risultato una tabella il cui grado è la somma dei gradi delle tabelle, in quanto le colonne della tabella prodotto sono le colonne della tabella A e della tabella B, e la cardinalità e il prodotto delle cardinalità delle tabelle A e B, in quanto ogni riga della tabella A si accoppia con tutte le righe della tabella B.

Esempio.

A		
Matricola	Cognome	Età
001	Rossi	18
002	Bianchi	22

B			
Codice	società	matricola	Perc.
A1	Red	001	50%
A3	White	002	40%
B6	Green	001	50%
C1	Black	002	60%

A X B						
Matricola	Cognome	Età	Codice	Società	Matricola	livello
001	Rossi	18	A1	Red	001	50%
001	Rossi	18	A3	White	002	40%
001	Rossi	18	B6	Green	001	50%
001	Rossi	18	C1	Black	002	60%
002	Bianchi	22	A1	Red	001	50%
002	Bianchi	22	A3	White	002	40%
002	Bianchi	22	B6	Green	001	50%
002	Bianchi	22	C1	Black	002	60%

Selezione.

La selezione si applica su una tabella e da come risultato un sottoinsieme di righe della tabella che soddisfano una condizione logica sulle colonne. La selezione si indica: $\sigma_{condizione}$ (*nome tabella*), dove con *condizione* si esprime una condizione logica cui devono soddisfare una o più colonne.

Esempio.

Supponiamo di avere la seguente tabella:

A			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
004	Bianchi	22	19
007	Verdi	20	19
010	Neri	19	23

$\sigma_{Punti\_segnati\ =19}$ (A)

σ			
Codice	Cognome	Età	Punti segnati
004	Bianchi	22	19
007	Verdi	20	19

Come si può notare l'operazione di selezione sulla tabella A, ha selezionato le righe la cui colonna *Punti_segnati* rispetta la condizione di contenere il valore 19

$$\sigma_{\text{Punti\_segnati =19 AND Età =22}}\ (A)$$

σ			
Codice	Cognome	Età	Punti segnati
004	Bianchi	22	19

In questo caso la condizione è espressa sulle colonne *Punti_segnati* ed *Età* che devono essere rispettivamente uguali a 19 e 22.

Per esprimere una condizione logica su più colonne si usano gli operatori logici AND OR e NOT.

Proiezione.

La proiezione si applica su una tabella e dà come risultato un sottoinsieme delle colonne della tabella. L'operatore si indica: $\pi_{\text{Elenco colonne}}$.

In *Elenco colonne* elenchiamo le colonne da selezionare.

Esempio.

Supponiamo di avere la seguente tabella:

A			
Codice	Cognome	Età	Punti segnati
001	Rossi	18	22
004	Bianchi	22	19
007	Verdi	20	19
010	Neri	19	23

$$\pi_{\text{Cognome, Età}}\ (A)$$

π	
Cognome	Età
Rossi	18
Bianchi	22
Verdi	20
Neri	19

In questo caso, come si può notare, la proiezione è intervenuta sulle colonne della tabella A realizzando la tabella con le colonne *Cognome* ed *Età*.

Join.

Operatore Join, in realtà, è un operatore composto nel senso che riunisce gli operatori di prodotto cartesiano e selezione.
In particolare si parla di Join naturale (natural Join) quando la selezione del prodotto cartesiano tra due tabelle, avviene sull'uguaglianza dei valori delle colonne *comuni* tra le due tabelle.

Se A e B sono le due tabelle, l'operazione di Join naturale si indica con A ⋈ B.
Tale operazione corrisponde alla sequenza delle operazioni:

$\sigma_{\text{colonna.A = colonna.B}}$ (A X B) ove colonna.A e colonna.B sono le colonne delle tabelle A e B con lo stesso nome.
In definitiva l'operazione di Join naturale esegue il prodotto cartesiano tra due tabelle A e B, quindi esegue la selezione tra i valori comuni delle colonne che hanno lo stesso nome.

Esempio.

Supponiamo di avere le tabelle:

STUDENTI			
Matricola	Cognome	Nome	Età
1001	Rossi	Mario	22
1004	Bianchi	Giulia	19
1007	Verdi	Antonio	19
1010	Neri	Maria	23

ESAMI		
Matricola	Materia	Voto
1001	Analisi	27
1001	Sistemi	30
1004	Geometria	23
1007	Algebra	25
1007	Sistemi	28
1010	Analisi	21

STUDENTI ⋈ ESAMI					
Matricola	Cognome	Nome	Età	Materia	Voto
1001	Rossi	Mario	22	Analisi	27
1001	Rossi	Mario	22	Sistemi	30
1004	Bianchi	Giulia	19	Geometria	23
1007	Verdi	Antonio	19	Algebra	25
1007	Verdi	Antonio	19	Sistemi	28
1010	Neri	Maria	23	Analisi	21

Come si può notare le due tabelle hanno una colonna con lo stesso nome, la selezione, dopo che si è effettuato il prodotto cartesiano, è avvenuta tra i valori comuni in queste colonne.

Nel caso in cui o il nome delle colonne non coincidono o la selezione è un'operazione più complessa rispetto all'uguaglianza dei valori contenuti nelle colonne, si utilizza un'altra operazione di join chiamata theta-join.
Con la theta-join (da ora θ-join) si devono specificare le colonne delle tabelle su cui deve essere definita l'uguaglianza e, più in generale, operazioni logiche.

Esempio.
Abbiamo le seguenti tabelle:

RICERCATORI			
Matricola	Cognome	Nome	Progetto
0001	Bianchi	Emilio	A0023
0002	Rossi	Francesca	A0021
0003	Verdi	Giuseppe	A0021
0004	Neri	Antonio	A0024

PROGETTI		
Cod_progetto	Denominazione	Sede
A0021	SETI	Bari
A0023	RIFERT	Salerno
A0024	PROFIL	Milano

Adesso se volessimo associare i ricercatori ai progetti:

$RICERCATORI \bowtie_{(Progetto = Cod\_progetto)} PROGETTI$

Ottenendo:

RICERCATORI ⋈ PROGETTI						
Matricola	Cognome	Nome	Progetto	Cod_progetto	Denominazione	Sede
0001	Bianchi	Emilio	A0023	A0023	RIFERT	Salerno
0002	Rossi	Francesca	A0021	A0021	SETI	Bari
0003	Verdi	Giuseppe	A0021	A0021	SETI	Bari
0004	Neri	Antonio	A0024	A0024	PROFIL	Milano

Se adesso volessimo selezionare i soli ricercatori che lavorano per il progetto SETI:

$RICERCATORI \bowtie_{(Progetto = Cod\_progetto) \text{ AND } (Denominazione = \text{“SETI”})} PROGETTI$

Ottenendo, come volevamo:

RICERCATORI ⋈ PROGETTI						
Matricola	Cognome	Nome	Progetto	Cod_progetto	Denominazione	Sede
0002	Rossi	Francesca	A0021	A0021	SETI	Bari
0003	Verdi	Giuseppe	A0021	A0021	SETI	Bari

4. Il Modello Fisico.

Prima di passare al modello fisico, il progettista, deve aver scelto quale DBMS utilizzare. Nel nostro caso la scelta è obbligata in quanto, come già detto, questo testo si occupa di un particolare tipo di DBMS: RDBMS, i data base relazionali.

Il modello fisico si ottiene dal modello logico definendo le tabelle ed i relativi vincoli tramite un linguaggio formale: SQL.

SQL (Structured Query Language).

L'SQL è la parte del DBMS che si occupa della gestione dei dati del database, intendendo per gestione la creazione, modifica e cancellazione dei "contenitori" dei dati (tabelle) e l'inserimento, modifica e cancellazione dei dati nei "contenitori".

L'SQL è un linguaggio suddiviso logicamente in tre gruppi d'istruzioni:

DDL: Data Definition Language – istruzioni per la gestione dei "contenitori" (tabelle);
DML: Data Manipulation Language – istruzioni per la gestione dei dati;
DCL: Data Control Language – istruzione per il controllo dei dati.

Inoltre esiste un altro gruppo d'istruzioni, TCL (Transaction Control Language), che serve per gestire i cambiamenti (confermare/annullare) fatti sui dati con le istruzioni DML; tale gruppo d'istruzioni verrà trattato superficialmente.

In più l'SQL fornisce un ambiente di lavoro ove far eseguire le istruzioni (da ora query) che chiameremo genericamente QL (Query Language).

Prima, di elencare le istruzioni SQL diamo qualche regola di lettura della sintassi. Le parole ed i simboli obbligatori (quando vanno scritti) delle istruzioni sono scritti in **GRASSETTO MAIUSCOLO;** i nomi delle colonne e delle tabelle sono scritte in *corsivo minuscolo* ciò che è tra <….> è opzionale ma deve contenerne almeno un valore, una espressione, il nome di una colonna, ecc.. Ciò che è scritto tra […] è opzionale e nell'istruzione può anche mancare; la barra verticale | indica oppure.

DDL (Data Definition Language).

Come abbiamo detto, le istruzioni che appartengono a questo gruppo sono quelle che servono per definire e creare i contenitori (le tabelle) dei dati ed il 'contenitore' dei contenitori, ossia il data base.
La prima cosa da fare è quindi creare il Data Base. Questo individuerà l'insieme delle tabelle che lo compongono. In definitiva creare un data base consiste creare uno 'spazio' definito da un nome che contiene un insieme di dati suddiviso in tabelle. Il DBMS può, in questo modo, gestire più data base diversi (quindi diverse tabelle) contemporaneamente, senza che si creino ambiguità.

CREATE.

L'istruzione che crea un database è:

CREATE DATABASE <nome_database>.

Fatta questa operazione, possiamo creare le tabelle.
L'istruzione che crea una tabella e:

CREATE TABLE <nome_tabella>
(*<nome_campo1> < tipo> [vincolo di colonna]***,**
*<nome_campo2> <tipo> [vincolo di colonna]***,**
……..
*<nome_campoN> <tipo>[vincolo di colonna]***,**
[vincoli di tabella] **)**

In cui:
nome_campo è il nome con cui chiamiamo la colonna;

tipo indica il tipo di dato che può contenere la colonna. Questi possono essere:

- **CHARACTER(n)**
  Una stringa a lunghezza fissa di esattamente n caratteri. CHARACTER puo' essere abbreviato con CHAR
- **CHARACTER VARYING(n)**
  Una stringa a lunghezza variabile di al massimo n caratteri. CHARACTER VARYING puo' essere abbreviato con VARCHAR o CHAR VARYING.
- **INTEGER**
  Un numero intero con segno. Puo' essere abbreviato con INT. La precisione, cioe' la grandezza del numero intero che puo' essere memorizzato in una colonna di questo tipo, dipende dall'implementazione del particolare DBMS.
- **SMALLINT**
  Un numero intero con segno con precisione non superiore a INTEGER.
- **FLOAT(p)**
  Un numero a virgola mobile, con precisione p. Il valore massimo di p dipende dall'implementazione del DBMS. E' possibile usare FLOAT senza indicazione della precisione, utilizzando quindi la precisione di default, anch'essa dipendente dall'implementazione. REAL e DOUBLE PRECISION sono dei sinonimi per un FLOAT con una particolare precisione. Anche in questo caso le precisioni dipendono dall'implementazione, con il vincolo che la precisione del primo non sia superiore a quella del secondo.
- **DECIMAL(p,q)**
  Un numero a virgola fissa di almeno p cifre e segno, con q cifre dopo la virgola. DEC e' un'abbreviazione per DECIMAL. DECIMAL(p) e' un'abbreviazione per DECIMAL(p,0). Il valore massimo di p dipende dall'implementazione.
- **INTERVAL**
  Un periodo di tempo (anni, mesi, giorni, ore, minuti, secondi e frazioni di secondo).
- **DATE, TIME e TIMESTAMP**
  Un preciso istante temporale. DATE permette di indicare l'anno, il mese e il giorno. Con TIME si possono specificare l'ora, i minuti e i secondi. TIMESTAMP e' la combinazione dei due precedenti. I secondi sono un numero con la virgola, permettendo cosi' di specificare anche frazioni di secondo.

vincolo di colonna può assumere uno o più dei seguenti valori:

- **NOT NULL** – indica che il campo è obbligatorio e non può essere omesso
- **PRIMARY KEY** – indica che il campo è la chiave primaria
- **DEFAULT** { NULL | VALORE } – permette di assegnare valori di default al campo
- **UNIQUE** – il campo non può avere duplicati
- **CHECK** (<condizioni>) – permette al campo di assumere un range di valori che rispetti le condizioni.
- **CONSTRAINT** nome_vincolo VINCOLO – permette di assegnare un nome ad un vincolo definito su una o più colonne ed usarlo come se fosse una colonna.

N.B. prima di chiudere l'ultima parentesi della CREATE non deve essere scritta la virgola.

Vincoli di tabella – determinano quei vincoli che coinvolgono uno o più campi della tabella:

vincolo di chiave esterna:

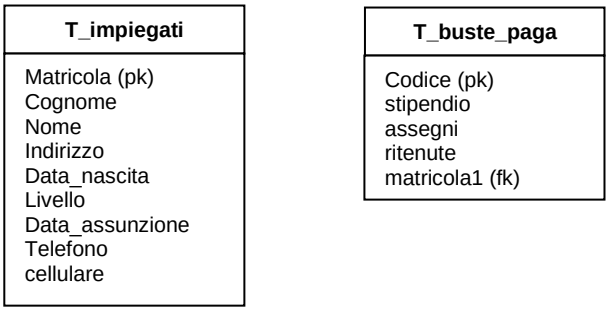
FOREIGN KEY (<*nome_campo*> **) REFERENCES** <*nome_altra_tabella*> **(**<*nome_campo_altra_tabella*>**)**

In cui *nome_campo* è la chiave esterna nella tabella, *nome_campo_altra_tabella* è la chiave primaria cui la chiave esterna si riferisce nella tabella *nome_altra_tabella*

Vincolo di chiave primaria:

PRIMARY KEY (<*nome_campo1*>,< *nome_campo2*>,….) rispetto all'analogo vincolo di colonna, il vincolo di PRIMARY KEY di tabella deve essere usato quando la chiave primaria della tabella è composta da più campi (anche se può essere usato al posto del vincolo di colonna anche se la primary key è composta da un solo campo).

Esempio.
Dato il seguente schema logico:



si vuole realizzare la definizione delle relazioni in linguaggio SQL (creare le tabelle):

```
 CREATE TABLE T-impiegati
(
    matricola INTEGER(5) PRIMARY KEY,
    cognome CHAR (20) NOT NULL,
    nome CHAR (20) NOT NULL,
```

indirizzo **CHAR** (50) **NOT NULL**,
data_nascita **DATE NOT NULL**,
livello **INT**(1) **NOT NULL**,
data_assunzione **DATE NOT NULL**,
telefono **CHAR** (12),
cellulare **CHAR** (12)
)

N.B. i numeri tra parentesi tonde indicano la lunghezza (in Byte) del campo. Inoltre i campi in cui non è esplicitato il vincolo NOT NULL, per default sono NULL.

CREATE TABLE *T-buste-paga*
(
codice **INTEGER**(5) **NOT NULL**,
stipendio **DECIMAL** (5,2) **NOT NULL**,
assegni **DECIMAL** (5,2),
ritenute **DECIMAL** (5,2) **NOT NULL**,
matricola1 **INTEGER**(5) **NOT NULL**,
PRIMARY KEY(*codice*),
FOREIGN KEY (*matricola1*) **REFERENCES** *T-impiegati* (*matricola*)
)

N.B. nelle situazioni reali, ossia quando siamo davanti al computer e stiamo realizzando un database, è necessario creare prima le tabelle che non hanno il vincolo FOREIGN KEY, poi quest'ultime.

ALTER.

Per modificare la struttura di una tabella l'istruzione da usare è:

ALTER TABLE <nome_tabella> **ADD | ALTER COLUMN (CHANGE, ALTER, MODIFY, RENAME) | DROP** <*nome_campo* [*tipo*] [*vincoli di colonna*]>

ADD <*nome_campo tipo* [*vincoli di colonna*]> è la clausola che serve per aggiungere una nuova colonna alla tabella

ALTER COLUMN <*nome_campo tipo*> è la clausola che permette di modificare il *tipo* alla colonna *nome_campo* (al posto di ALTER COLUMN, a secondo i DBMS, ad esempio in ACCESS, si usa **ALTER**, mentre in altri DBMS si utilizzano le clausole **CHANGE**, **MODIFY** e **RENAME** per modificare la tipologia o rinominare il campo);

DROP <*nome_campo*> è la clausola che consente di eliminare la colonna *nome_campo*

CONSTRAINT <nome_vincolo> deve essere usata insieme a **ADD** o **DROP** per aggiungere nuovi vincoli o eliminarli.

Esempio.

Supponiamo di voler inserire la colonna data_busta nella tabella T-buste-paga:

ALTER TABLE *T-buste-paga* **ADD** *data_busta* **DATE NOT NULL**

La seguente istruzione cambia il tipo della colonna *assegni* da decimal a float:

ALTER TABLE *T-buste-paga* **ALTER COLUMN** *assegni* **FLOAT**

CREATE INDEX.

Per creare gli indici si usa l'istruzione:

CREATE [UNIQUE] INDEX *nome_indice* **ON** *nome_tabella* (*nome_campo<,nome_campo1>*)

UNIQUE impone all'indice di non avere duplicati.

Attraverso gli indici è possibile velocizzare la ricerca su uno o più campi di una tabella individuandone celermente i contenuti.

> Esempio.
> Supponiamo di effettuare molte ricerche sulla tabella *T-impiegati* per *cognome* e *nome*. In questo caso per velocizzare la ricerca è necessario creare un indice:
>
> **CREATE INDEX** *idx-nominativo* **ON** *T-impiegati* (*cognome, nome*)
>
> È ovvio che per ottimizzare le queries, nel caso in cui un indice è composto da più colonne, devono essere usate le colonne che compongono l'indice.(nel nostro esempio se volessimo effettuare una ricerca con l'istruzione SELECT nella tabella T-impiegati, la WHERE deve contenere cognome e nome).

DROP.

Per eliminare indici, tabelle e database si una l'istruzione **DROP:**

DROP INDEX <*nome_indice*> **ON** *nome_tabella* (in MySQL **ALTER TABLE** *nome_tabella* **DROP INDEX** *nome_indice*)
L'istruzione elimina l'indice *nome_indice* dalla tabella *nome_tabella*

DROP TABLE <*nome_tabella*>
Elimina la tabella *nome_tabella* dal database

Infine l'istruzione **DROP DATABASE** <*nome_database*> elimina il database dalla memoria ove era allocato.

DML (Data Manipulation Linguage).

Le istruzioni di questo gruppo servono per manipolare i dati delle tabelle. Queste istruzioni possiamo definirle l'aspetto esecutivo dell'algebra relazionale vista nel capitolo precedente. Le istruzioni principale di questo gruppo sono **INSERT**, **UPDATE**, **DELETE**, **SELECT**.

INSERT

L'istruzione **INSERT** serve per inserire una riga di dati in una tabella. la sintassi dell'istruzione è:

INSERT INTO <*nome_tabella*>
[**(**<*nome_campo1*>, <*nome_campo2*>,….,<*nome_campoN*>**)**]

VALUES (<valore_campo1>,< valore_campo2>,……, <valore_campoN>**)**

ATTENZIONE! L'ordine con cui inseriamo i valori dopo la keyword VALUES deve rispettare l'ordine con cui abbiamo scritto i nome_campo1, nome_campo2….

Se non desideriamo assegnare nessun valore ad un campo che non ha il vincolo NOT NULL, allora non dobbiamo inserire il *nome_campoX* ed ovviamente il corrispondente valore.

Se, viceversa, vogliamo inserire la riga intera senza tralasciare alcun valore, allora possiamo evitare di scrivere dopo *nome_tabella* i nomi delle colonne (nome_campo1,….) a patto, però, che i valori che scriviamo dopo VALUES, li scriviamo nello stesso ordine con cui abbiamo scritto i nomi delle colonne nell'istruzione CREATE che ha creato la tabella.

Esempio.

INSERT INTO *T-impiegati*
(*matricola*, *cognome* ,*nome*,*indirizzo*,*data_nascita*,*livello*,*data_assunzione*,*telefono*)
VALUES
(321, 'Rossi', 'Mario', 'via Tal dei Tali', '31/01/1990', 2, '21/12/2012', '069988774')

Nota: manca il campo cellulare (campo NULL), quindi ho dovuto inserire il nome delle colonne

Se avessi inserito tutti i campi:

INSERT INTO *T-impiegati*
VALUES
(321, 'Rossi', 'Mario', 'via Tal dei Tali', '31/01/1990', 2, '21/12/2012', '069988774', '347009988')

SELECT.

Per estrarre dati da database o eseguire interrogazioni (query) si usa l'istruzione:

SELECT [**DISTINCT**] <*nome_campo1*>, <*nome_campo2*>,…..,<*nome_campoN*>
FROM <*nome_tabella*>[,<*nome_tabella1*>,……,<*nome_tabellaX*>]
[**WHERE** <condizione>]
[**JOIN** <*nome_tabella*> **ON** <condizione>]
[**GROUP BY** <*nome_campo*>]
[**HAVING** <condizione>]
[**ORDER BY** <*nome_campo*>] [**ASC**] | [**DESC**]

L'istruzione restituisce una tabella con le cui colonne sono quelle indicate nell'istruzione, in pratica effettua l'operazione di proiezione dell'algebra relazionale, (se vogliamo che ci vengano restituite tutte le colonne allora invece di elencarle tutte possiamo utilizzare il simbolo asterisco, *, al loro posto) della tabella specificata (se sono presenti più di una tabella allora l'estrazione avverrà dalla tabella 'prodotto cartesiano' di quelle indicate).

Se è presente la clausola **WHERE**, che verrà studiata approfonditamente più tardi, l'istruzione restituisce una tabella le cui colonne sono quelle elencate e le righe sono quelle che verificano la condizione.
La clausola **DISTINCT** consente di visualizzare esclusivamente righe non duplicate.
La clausola **JOIN** si suddivide in diverse tipologie:
INNER JOIN, **LEFT OUTER JOIN**, **RIGHT OUTER JOIN**, **FULL OUTER JOIN**, **CROSS JOIN**, ed effettua il prodotto cartesiano tra la tabella specificata dopo la **FROM** e quella specificata dopo la **JOIN** selezionando le righe che rispettano la condizione specificata dopo **ON**.
La clausola **GROUP BY** raggruppa il/i campi specificati, aggregandoli. La clausola **HAVING** rappresenta un vincolo sulle colonne, mentre la clausola **ORDER BY** effettua un ordinamento crescente (**ASC**) o decrescente (**DESC**) sulle colonne specificate.

Quando la SELECT tratta più tabelle per determinare quale colonna appartiene a quale tabella (soprattutto nel caso in cui le colonne di tabelle diverse hanno nomi uguali) si deve utilizzare la *dot-notation* che consiste nell'individuare la colonna anteponendo al nome della colonna il nome della tabella con un punto: *nome_tabella.nome_campo.*

Esaminiamo le varie clausole in modo approfondito.

Clausola WHERE.

La **WHERE** effettua la selezione vera e propria, intesa come sottoinsieme di righe che verificano la condizione. **WHERE** *nome_campo1* = 'a' , ad esempio, seleziona tutte le righe dalla tabella in cui il valore di *nome_campo1* sia 'a'.
Ovviamente possiamo usare gli operatori logici per realizzare espressioni logiche più complesse:
WHERE *nome_campo1* **=** 'a ' **AND** *nome_campo2* **>** 10;
WHERE *nome_campo1* **=** 'a' **OR** *nome_campo2* **=** 'b';
WHERE *nome_campo1* **<>** 'c' (**<>** corrisponde all'operatore NOT EQUAL);
nel primo caso vogliamo tutte quelle righe in cui la colonna *nome_campo1* sia uguale ad 'a' E (contemporaneamente) la colonna *nome_campo2* sia maggiore di 10;
nel secondo vogliamo tutte quelle righe in cui la colonna *nome_campo1* sia uguale ad 'a' OPPURE la colonna *nome_campo2* sia uguale a 'b';
nel terzo vogliamo tutte quelle righe in cui la colonna *nome_campo1* siano diverse da 'c'.

Opzione LIKE.

Questa opzione permette di selezionare parti di stringhe contenute in una colonna:

WHERE *nome_campo1* **LIKE** ('mas%')
WHERE *nome_campo1* **LIKE** ('%mo')
WHERE *nome_campo1* **LIKE** ('%tra%')

Nel primo caso si selezionano tutte le righe in cui la colonna *nome_campo1* ha stringhe che comincino con 'mas' (es. '**mas**simo', '**mas**sa', **mas**tercard',ecc.);

nel secondo caso si selezionano tutte le righe in cui la colonna *nome_campo1* ha stringhe che finiscano con 'mo' (es. 'massi**mo**', 'ti**mo**', 'paler**mo**', ecc.);
nel terzo caso si selezionano tutte le righe in cui la colonna *nome_campo1* ha stringhe che contengano 'tra' (es. '**tra**nsazione', 'at**tra**zione', 'al**tra**', ecc.)

il carattere % (che in ACCESS è sostituito dal carattere *) viene detto carattere Jolly.

Opzione BETWEEN.

Questa opzione permette di verificare l'appartenenza del valore di una colonna all'interno di un intervallo [primo_valore, ultimo_valore] (in questo caso le parentesi quadre indicano che gli estremi dell'intervallo appartengono all'intervallo).

WHERE *nome_campo1* **BETWEEN** primo_valore **AND** ultimo_valore.

Esempio.

WHERE *stipendio* **BETWEEN** 2000,00 **AND** 4000,00

In questo caso desideriamo tutte quelle righe in cui i valori della colonna *stipendio* siano compresi tra 2000,00 e 4000,00 €.

Opzione IN.

Questa opzione permette di verificare l'appartenenza del valore di una colonna all'interno di un insieme di valori esplicitati dopo l'opzione IN.

WHERE *nome_campo1* **IN** (valore1, valore2, valore3,…., valoreK)

Esempio.

WHERE *nazione* **IN** ('Italia', 'Francia', 'Germania', 'Spagna')

In questo caso desideriamo tutte quelle righe in cui i valori della colonna *nazione* siano 'Italia' o 'Francia' o 'Germania' o 'Spagna'.

Opzione IS [NOT] NULL.

Questa opzione consente di verificare se un campo contiene o meno valore nullo (NULL) campo vuoto, da non confondere con lo spazio che è un carattere.

WHERE *nome_campo1* **IS [NOT] NULL**

Esempio.

WHERE *nazione* **IS NULL**

Restituisce vero se il campo è vuoto

WHERE *nazione* **IS NOT NULL**

Restituisce vero se il campo non è vuoto

Clausola DISTINCT

I valori di una colonna possono essere ripetuti. La clausola **DISTINCT** della SELECT, elimina i valori ripetuti di una colonna evidenziandole solo uno. DISTINCT, quindi, restituisce solo valori distinti di una colonna senza ripetizioni.

Esempio.

STUDENTI			
Matricola	Cognome	Nome	Provincia
1001	Rossi	Mario	Roma
1004	Bianchi	Giulia	Milano
1007	Verdi	Antonio	Roma
1010	Neri	Maria	Milano

SELECT DISTINCT *Provincia* **FROM** *studenti*

Restituisce

Roma
Milano

Clausola JOIN.

Come abbiamo detto la JOIN effettua il prodotto cartesiano tra tabelle; vi sono però modi diversi di effettuare tale prodotto. Vediamo quali.

INNER JOIN

La **INNER JOIN** effettua il prodotto cartesiano tra le tabelle specificate dopo la **FROM** e **INNER JOIN** selezionando le righe che rispettano la condizione specificata dopo **ON**.

Esempio.

STUDENTI			
Matricola	Cognome	Nome	Provincia
1001	Rossi	Mario	Roma
1004	Bianchi	Giulia	Milano
1007	Verdi	Antonio	Roma
1010	Neri	Maria	Milano

FREQUENTA		
Matricola	Corso	Durata
1001	Ingegneria	5
1004	Farmacia	5
1007	Matematica	3
1008	Fisica	5
1010	Chimica	3
1011	Ingegneria	5

SELECT *studenti.matricola, studenti.cognome, studenti.nome, studenti.provincia, frequenta.corso, frequenta.durata* **FROM** *studenti* **INNER JOIN** *frequenta* **ON** *studenti.matricola* **=** *frequenta.matricola*

Studenti.Matricola	Studenti.Cognome	Studenti.Nome	Studenti.Provincia	frequenta.corso	frequenta.durata
1001	Rossi	Mario	Roma	Ingegneria	5
1004	Bianchi	Giulia	Milano	Farmacia	5
1007	Verdi	Antonio	Roma	Matematica	3
1010	Neri	Maria	Milano	Chimica	3

LEFT OUTER JOIN.

In questo caso il prodotto cartesiano 'aggiunge' tutte le righe della tabella di 'sinistra' (quella della FROM, per intenderci) anche se non rispettano la condizione della ON.

Esempio.

STUDENTI			
Matricola	Cognome	Nome	Provincia
1001	Rossi	Mario	Roma
1004	Bianchi	Giulia	Milano
1007	Verdi	Antonio	Roma
1009	Blue	Ivo	Milano
1010	Neri	Maria	Milano

FREQUENTA		
Matricola	Corso	Durata
1001	Ingegneria	5
1004	Farmacia	5
1007	Matematica	3
1008	Fisica	5
1010	Chimica	3
1011	Ingegneria	5

SELECT *studenti.matricola, studenti.cognome, studenti.nome, studenti.provincia, frequenta.corso, frequenta.durata* **FROM** *studenti* **LEFT OUTER JOIN** *frequenta* **ON** *studenti.matricola* **=** *frequenta.matricola*

Studenti.Matricola	Studenti.Cognome	Studenti.Nome	Studenti.Provincia	frequenta.corso	frequenta.durata
1001	Rossi	Mario	Roma	Ingegneria	5
1004	Bianchi	Giulia	Milano	Farmacia	5
1007	Verdi	Antonio	Roma	Matematica	3
1009	Blue	Ivo	Milano	-null-	-null-
1010	Neri	Maria	Milano	Chimica	3

RIGHT OUTER JOIN.

In questo caso il prodotto cartesiano 'aggiunge' tutte le righe della tabella di 'destra' (quella della JOIN, per intenderci) anche se non rispettano la condizione della **ON**.

Esempio.

STUDENTI			
Matricola	Cognome	Nome	Provincia
1001	Rossi	Mario	Roma
1004	Bianchi	Giulia	Milano
1007	Verdi	Antonio	Roma
1009	Blue	Ivo	Milano
1010	Neri	Maria	Milano

FREQUENTA		
Matricola	Corso	Durata
1001	Ingegneria	5
1004	Farmacia	5
1007	Matematica	3
1008	Fisica	5
1010	Chimica	3
1011	Ingegneria	5

SELECT *studenti.matricola, studenti.cognome, studenti.nome, studenti.provincia, frequenta.corso, frequenta.durata* **FROM** *studenti* **RIGHT OUTER JOIN** *frequenta* **ON** *studenti.matricola* **=** *frequenta.matricola*

Studenti.Matricola	Studenti.Cognome	Studenti.Nome	Studenti.Provincia	frequenta.corso	frequenta.durata
1001	Rossi	Mario	Roma	Ingegneria	5
1004	Bianchi	Giulia	Milano	Farmacia	5
1007	Verdi	Antonio	Roma	Matematica	3
-null-	-null-	-null-	-null-	Fisica	5
1010	Neri	Maria	Milano	Chimica	3
-null-	-null-	-null-	-null-	Ingegneria	5

FULL OUTER JOIN.

In questo caso il prodotto cartesiano ‘aggiunge’ tutte le righe delle tabelle anche se non rispettano la condizione della ON.

Esempio.

STUDENTI			
Matricola	Cognome	Nome	Provincia
1001	Rossi	Mario	Roma
1004	Bianchi	Giulia	Milano
1007	Verdi	Antonio	Roma
1009	Blue	Ivo	Milano
1010	Neri	Maria	Milano

FREQUENTA		
Matricola	Corso	Durata
1001	Ingegneria	5
1004	Farmacia	5
1007	Matematica	3
1008	Fisica	5
1010	Chimica	3
1011	Ingegneria	5

SELECT *studenti.matricola, studenti.cognome, studenti.nome, studenti.provincia, frequenta.corso, frequenta.durata* **FROM** *studenti* **FULL OUTER JOIN** *frequenta* **ON** *studenti.matricola* **=** *frequenta.matricola*

Studenti.Matricola	Studenti.Cognome	Studenti.Nome	Studenti.Provincia	frequenta.corso	frequenta.durata
1001	Rossi	Mario	Roma	Ingegneria	5
1004	Bianchi	Giulia	Milano	Farmacia	5
1007	Verdi	Antonio	Roma	Matematica	3
1009	Blue	Ivo	Milano	-null-	-null-
1010	Neri	Maria	Milano	Chimica	3
-null-	-null-	-null-	-null-	Fisica	5
-null-	-null-	-null-	-null-	Ingegneria	5

CROSS JOIN.

L'istruzione, di cui non verrà prodotto alcun esempio, in assenza di WHERE effettua il prodotto cartesiano (vedi capitolo 3. anche per il relativo esempio) tra le tabelle specificate.

Clausola GROUP BY.

La clausola raggruppa le righe in cui le colonne specificate nella **GROUP BY** contengono gli stessi valori. La clausola, detta così, somiglia alla DISTINCT, in realtà le due clausole sono differenti. La DISTINCT evidenzia solo i differenti valori, GROUP BY raggruppa le righe, in questo modo, come vedremo, possiamo compiere delle operazione sulle righe raggruppate. Attenzione le colonne presenti nella GROUP BY devono essere tra quelle elencate nella SELECT.

Clausola HAVING.

Associata alla GROUP BY la HAVING consente di stabilire vincoli sui raggruppamenti (funziona in modo molto simile alla WHERE, solo che quest'ultima agisce solo sulle colonne delle tabelle).

Clausola ORDER BY.

La clausola consente di effettuare l'ordinamento crescente (**ASC**) o decrescente (**DESC**) sulle colonne specificate.

Esempio.

FREQUENTA		
Matricola	Corso	Durata
1001	Ingegneria	5
1004	Farmacia	5
1007	Matematica	3
1008	Fisica	5
1010	Chimica	3
1011	Ingegneria	5

SELECT * **FROM** *frequenta*
ORDER BY *durata* **DESC**

FREQUENTA		
Matricola	Corso	Durata
1001	Ingegneria	5
1004	Farmacia	5
1008	Fisica	5
1011	Ingegneria	5
1007	Matematica	3
1010	Chimica	3

SELECT * **FROM** *frequenta*
ORDER BY *durata* **DESC**, *corso* **ASC**

FREQUENTA		
Matricola	Corso	Durata
1004	Farmacia	5
1008	Fisica	5
1001	Ingegneria	5
1011	Ingegneria	5
1010	Chimica	3
1007	Matematica	3

Clausola ALIAS.

L'alias consente di usare un nome (di colonna o di una tabella) definito dall'utente al posto del nome definito nel database nella visualizzazione dei dati. L'istruzione che consente di assegnare un nome ad una colonna o ad una tabella è **AS**.

Esempio

ANAGRAFICA		
c_f	cog	nome

SELECT *c_f* **AS** 'codice fiscale', *cog* **AS** 'cognome', *nome* **FROM** anagrafica

Visualizzeremo:

ANAGRAFICA		
codice fiscale	cognome	nome

Nello stesso modo possiamo dare un nome diverso alle tabelle. Questo ci risulterà estremamente utile quando si usa la dot notation.

Esempio.

SELECT *studenti.matricola, studenti.cognome, studenti.nome, studenti.provincia, frequenta.corso, frequenta.durata* **FROM** *studenti* **FULL OUTER JOIN** *frequenta* **ON** *studenti.matricola* **=** *frequenta.matricola*

Con l'uso degli alias diventa:

SELECT *s.matricola, s.cognome, s.nome, s.provincia, c.corso, c.durata* **FROM** *studenti* **AS** *s* **FULL OUTER JOIN** *frequenta* **AS** c **ON** *s.matricola* **=** *c.matricola*

Funzioni della SELECT.

La select contiene un insieme di funzioni che restituiscono risultati su operazioni effettuate dalla select sulle righe di una o più tabelle

Funzione COUNT().

La funzione **COUNT** restituisce in output il numero di righe selezionate dalla SELECT.

SELECT <*nome_campo*,> **COUNT(*) FROM** <*nome_tabella*> [**WHERE** <condizione> | ecc.]

Esempi.

Consideriamo le tabelle *frequenta* e *studenti* degli esempi precedenti:

SELECT COUNT(*) FROM *frequenta* **WHERE** *durata* = 5
Il risultato che otteniamo

FREQUENTA
COUNT(*)
4

SELECT COUNT(*) AS 'nr corsi' **FROM** *frequenta* **WHERE** *durata* **=** 5
Il risultato che otteniamo

FREQUENTA
nr corsi
4

SELECT *provincia***, COUNT(*) FROM** *studenti* **GROUP BY** *provincia*
Il risultato che otteniamo

studenti	
provincia	**COUNT(*)**
Milano	3
Roma	2

Funzioni statistiche e matematiche AVG, SUM, MIN, MAX.

Le funzioni calcolano la media, somma, valore minimo, valore massimo di una o più colonne delle righe selezionate dalla SELECT.
La funzione **AVG(**<*nome_campo*>**)** esegue la media sulla colonna *nome_campo* sulle righe selezionate dalla SELECT

SELECT <*nome_campo*,> **AVG(**<*nome_campo1*>**) FROM** <*nome_tabella*> [**WHERE** <condizione> | ecc.]

Esempio.

Consideriamo la tabella

stipendi		
Matricola	**Livello**	**Stipendio**
10001	3	1200
10003	2	1500
10004	3	1300
10005	1	2000
10007	4	1000
10008	2	1600

SELECT AVG(*stipendio***) FROM** *stipendi*

Stipendi
AVG(stipendio)
1433,333

SELECT *livello*, **AVG(***stipendio***) FROM** *stipendi* **GROUP BY** *livello*

Stipendi	
livello	**AVG(stipendio)**
1	2000
2	1550
3	1250
4	1000

SELECT AVG(*stipendio***) AS** *media* **FROM** *stipendi* **WHERE** *livello* **=** 2

Stipendi
media
1550

Analogamente alla funzione AVG, la funzione SUM (<nome_campo>) esegue la somma sulle colonne nome_campo sulle righe selezionate dalla WHERE o nei raggruppamenti definiti dalla GROUP BY.

Esempio.

Consideriamo la tabella dell'esempio precedente.

SELECT SUM (*stipendio***) FROM** *stipendi*

Stipendi
SUM(stipendio)
8600

SELECT *livello*, **SUM (***stipendio***) FROM** *stipendi* **GROUP BY** *livello*

Stipendi	
livello	**SUM (stipendio)**
1	2000
2	3100
3	2300
4	1000

SELECT SUM (*stipendio***) AS** totale **FROM** *stipendi* **WHERE** *livello* **=** 3

Stipendi
totale
2300

Le funzioni **MAX (**<*nome_campo*>**)** e **MIN (**<*nome_campo*>**)** determinano l'elemento massimo o minimo delle colonne *nome_campo* sulle righe selezionate dalla WHERE o nei raggruppamenti definiti dalla GROUP BY.

Esempio.

Sempre con la tabella stipendi degli esempi precedenti.

SELECT MAX (*stipendio***) FROM** *stipendi*

Stipendi
MAX(stipendio)
2000

Funzioni di stringa UPPER(), LOWER(), TRIM(),SUBSTRING().

Le funzioni UPPER, LOWER, TRIM, SUBSTRING eseguono:

- **LOWER(<str>):** converte la stringa <str> in caratteri tutti minuscoli.
- **UPPER(<str>)**: converte la stringa <str> in caratteri tutti maiuscoli.
- **TRIM(<str>)**: elimina eventuali spazi iniziali e finali dalla stringa <str>
- **SUBSTRING(<str>, <pos>[, <len>])**: restituisce la sottostringa di <str> a partire dalla posizione <pos>; opzionale il numero di caratteri len (in ACCESS devono essere usate le funzioni LEFT(<str>, pos, len), RIGHT(<str>, pos, len), MID(<str>, pos, len) che selezionano le stringhe a partire rispettivamente da sinistra, destra, da una posizione intermedia)

<str> è, ovviamente una colonna della tabella di tipo stringa.

Esempio.

SELECT SUBSTRING (*corso*,1,3) **FROM** *frequenta*
WHERE *matricola* =1001

Risultato: 'Ing'

SELECT SUBSTRING (*corso*,4) **FROM** *frequenta*
WHERE *matricola* =1001

Risultato: 'egneria'

SUBSELECT E UNION.

Per subselect intendiamo una SELECT annidata, in qualche modo, in un'altra SELECT.

Esempio.

Supponiamo di avere le tabelle attori e film e vogliamo sapere i film in cui ha recitato George Clooney nel 1997

ATTORI				
Codice_attore	Cognome	Nome	Nazionalità	Data_nascita
1	Clooney	George	USA	06/05/1961
2	……..	……..	……	……

FILM				
Codice_film	Codice_attore	Titolo	Anno	Codice_regista
1	1	La scuola degli orrori	1987	1
2	1	Grizzly II: The Predator	1987	2
3	1	The Harvest	1993	3
4	1	Batman & Robin	1997	4
5	1	The Peacemaker	1997	3
6	……	……..	…….	……

SELECT *titolo* **FROM** *film* **AS** *a*
WHERE *a.anno* = 1997 **AND** *a.codice_attore* =(**SELECT** *b.codice_attore* **FROM** *attori* **AS** *b*
WHERE *b.cognome* = 'Clooney' **AND** *b.nome* = 'George')

Avremo come risultato:

FILM
Titolo
Batman & Robin
The Peacemaker

L'opzione **UNION** unisce i risultati di due SELECT che vengono effettuate su tabelle che hanno gli stessi campi.

Esempio.

Supponiamo di avere le due tabelle:

PARTENZE		
Treno	Stazione	Ora
Er500	Paola	15.32
Er500	Salerno	18.34
Er500	Napoli	19.31

ARRIVI		
Treno	Stazione	Ora
Er500	Paola	15.30
Er500	Salerno	18.31
Er500	Napoli	18.25

SELECT *treno*, *stazione*, *ora* **FROM** *partenze*
WHERE *treno* = 'er500' **AND** *stazione* = 'Paola'
UNION
SELECT *treno,stazione, ora* **FROM** *partenze*
WHERE *treno* = 'er500' **AND** *stazione* = 'Salerno'

Avremo come risultato:

SELECT …….		
Treno	stazione	Ora
Er500	Paola	15.32
Er500	Salerno	18.31

UPDATE.

Per modificare i valori dei campi di una tabella si usa l'istruzione:

UPDATE <*nome_tabella*>
SET <*nome_campo1*> = <valore_campo1> | <espressione1>[,
<*nome_campo2*> = <valore_campo2> | <espressione2>,
……..
<*nome_campoN*> = <valore_campoN>| <espressioneN>]
[**WHERE** <condizione>]

L'istruzione modifica le colonne indicate (*nome_campo*) con un valore che può essere il risultato di un'espressione.
Se si usa la clausola WHERE verranno modificati solo le colonne delle righe della tabella che verificano la condizione. Altrimenti i cambiamenti verranno effettuati su tutte le colonne elencate dopo la keyword SET di tutte le righe della tabella.

Esempio.

Nella tabella impiegati vogliamo aumentare il livello dell'impiegato Rossi da 2 a 3.

UPDATE T-impiegati **SET** *livello* = 3
WHERE cognome = 'Rossi'

N.B. Se nella tabella esistono altri impiegati il cui cognome è Rossi, beh il livello di tutti i Sig. Rossi sarebbe, dopo l'esecuzione dell'istruzione, 3 (ciò sarebbe un regalo per coloro che hanno il livello minore, per gli altri…… un guaio!)

UPDATE T-impiegati **SET** *livello* = 3

Questa istruzione avrebbe posto i livelli di tutti gli impiegati a 3.

DELETE.

Per cancellare una o più righe (al limite tutte) di una tabella si usa l'istruzione:

DELETE FROM <nome_tabella>
[**WHERE** <condizione>]

Se la clausola **WHERE** è utilizzata, verranno eliminate solo le righe che verificano la condizione, se la clausola **WHERE** non è usata la tabella viene 'svuotata', nel senso che verranno cancellate tutte le righe in essa contenute.

Esempio.

L'impiegato Rossi si è licenziato (dopo tutti i guai che ha combinato con i livelli….), lo devo eliminare dalla tabella.

DELETE FROM *T-impiegati* **WHERE** *matricola* = 321

In questo caso l'istruzione elimina la sola riga (tupla) relativa a Rossi.

Considerazioni sui vincoli d'integrità.

La tabella *T-buste-paga* contiene le informazioni per calcolare la busta paga del sig. Rossi. È ovvio che se elimino dalla tabella *T-impiegati* il sig. Rossi, le informazioni sulla tabella *T-buste-paga* relative alle sue buste paga rimangono 'orfane'.

Se tra le tabelle *T-impiegati* e *T-buste-paga* era stato stabilito un vincolo d'integrità referenziale, cosa sarebbe successo tentando di cancellare il Sig. Rossi? Il DBMS non lo avrebbe permesso!! Ci avrebbe avvisato che c'erano tabelle collegate con *T-impiegati* e che quindi si doveva prima cancellare gli elementi che si 'riferiscono' alla tabella *T-impiegati* (le tabelle che hanno una chiave esterna con *T-impiegati*) e che riguardano il Sig. Rossi e solo dopo cancellare l'elemento dalla tabella *T.impiegati*.

Alcuni DBMS permettono la cosiddetta cancellazione a 'cascata'. Nel caso ci siano vincoli di integrità referenziale, se tentiamo di cancellare una riga dalla tabella padre (quella che non ha chiavi esterne) il DBMS ci chiede se vogliamo cancellare tutti gli elementi correlati (ossia tutte le righe nelle altre tabelle referenziate all'elemento della tabella padre).

La stessa cosa accade se vogliamo inserire elementi nelle tabelle referenziate. Ossia, prendendo l'esempio precedente, se volessimo inserire i dati relativi alla busta paga del Sig. Bianchi nella tabella *T-buste-paga*, il sig. Bianchi deve esistere già nella tabella padre (*T-impiegati*); il DBMS ci comunicherebbe, altrimenti, che l'operazione di inserimento dei dati della busta paga non è possibile.

DCL (Data Control Language).

Le istruzioni di questo gruppo permettono di assegnare o revocare agli utenti che utilizzano il database le autorizzazioni ad usare le istruzioni DML.

Innanzitutto diciamo che i vari DBMS hanno istruzioni diverse per creare gli utenti, quindi riconoscerli e permettere loro di operare sul database. Ovviamente questo ha senso quando i database sono multiutenza, ossia permettono di far accedere contemporaneamente più utenti ad essi (es. architetture client-server ove il database è collocato sul server e gli utenti si collegano simultaneamente da più client).

La maggior parte di questi database utilizzano le istruzioni **CREATE USER**, per creare un utente (o un gruppo di utenti), meglio, un account d'accesso che identifica l'utente, **DROP USER**, per cancellare l'utente o il gruppo di utenti, **ALTER USER** per modificare le impostazioni. Queste operazioni possono essere usate da una figura mitica il *Database Administrator* , il guru del database che lo gestisce, lo cura….. e forse gli vuole anche un po' di bene!

Vediamo le sintassi, con le dovute attenzioni in quanto, come detto queste istruzioni possono avere sintassi diverse nei vari 'dialetti' SQL

CREATE USER nome_utente **IDENTIFIED BY** password;

DROP USER nome_utente;

ALTER USER nome:utente **IDENTIFIED BY** new-password.

Il database administrator, una volta creato l'account d'accesso per un determinato utente (gruppo di utenti), deve assegnare le autorizzazioni, detti privilegi, su come l'account può operare sulle tabelle del database.

L'istruzioni per assegnare le autorizzazioni è l'istruzione GRANT

GRANT.

La sintassi dell'istruzione è:

GRANT <elenco privilegi> **ON** <oggetto> **TO** <nome utente> | <account d'accesso> | <public>
[**WITH GRANT OPTION**]

Dove:
elenco privilegi indica una o più istruzioni DML, eventualmente separate da virgola, che l'utente può usare ossia:

SELECT
INSERT
UPDATE
DELETE
ALL (ALL PRIVILEGES) (privilegi su tutte le istruzioni)

Oggetto

Indica la tabella (tabelle) su cui assegnare i privilegi di solito è della forma *nome_db.nome_tabella*; con *nome_db.** indichiamo tutte le tabelle di un determinato database (voglio ricordare che sulle memorie di massa di un elaboratore possono essere presenti più database usati da più applicazioni), mentre con *.* diamo all'utente i privilegi su tutte le tabelle di tutti i database.

nome utente | account d'accesso identifica l'utente o il gruppo di utenti a cui vengono assegnati i privilegi se viene specificato PUBLIC al database può accedere chiunque.

La clausola **WITH GRANT OPTION** consente all'utente (gruppo di utenti) a cui è stato assegnato uno o più privilegi di assegnarli ad altri utenti.

Esempio.

Supponiamo di avere i database db1 e db2 e all'interno di questi un certo numero di tabelle.

Creiamo l'utente 'ciccio':

CREATE USER 'ciccio' **IDENTIFIED BY** 'ciccio9112309881'

Assegnamo all'utente dei privilegi

GRANT SELECT ON db1.* **TO** 'ciccio'

In questo caso permettiamo all'utente 'ciccio' di eseguire delle select su tutte le tabelle del DB1.

GRANT SELECT, INSERT ON db1.* **TO** ‘ciccio’

In questo caso permettiamo all’utente ‘ciccio’ di eseguire select e insert su tutte le tabelle del DB1.

GRANT ALL ON db1.* **TO** ‘ciccio’

In questo caso permettiamo all’utente ‘ciccio’ di eseguire tutte le operazioni su tutte le tabelle del DB1.
GRANT ALL ON *.* **TO** ‘ciccio’

In questo caso permettiamo all’utente ‘ciccio’ di eseguire tutte le operazioni su tutte le tabelle dei due database.

GRANT ALL ON db1.anagrafica **TO** ‘ciccio’

In questo caso permettiamo all’utente ‘ciccio’ di eseguire tutte le operazioni sulla sola tabella anagrafica di db1.

Per revocare i privilegi assegnati ad un utente si usa l’istruzione REVOKE.

REVOKE.

La sintassi dell’istruzione REVOKE è:

REVOKE [**GRANT OPTION FOR**] <privilegi> **ON** <oggetto> **FROM** <nome utente> | <account d’accesso> | <public>

In cui privilegi, oggetto, nome utente | account d’accesso | public, hanno lo stesso significato dell’istruzione GRANT

L’opzione **GRANT OPTION FOR,** revoca all’utente solo il privilegio di assegnare ad altri utenti gli stessi suoi privilegi, privilegio, questo, concessigli con l’opzione WITH GRANT OPTION nella GRANT; restano invariati gli altri privilegi.

TCL (Transaction Control Language).

In gergo tecnico un utente che opera con un programma o che effettua delle queries su un database distribuito, si dice che effettua un transazione.

Quando un database distribuito viene usato da più utenti che effettuano ‘transazioni’, le operazioni sulle tabelle avvengono non direttamente su di esse ma su una loro ‘copia’; ogni utente ha una copia ‘personale’ della/delle tabelle che sta manipolando.

Questo modo di operare consente di ‘sincronizzare’ le operazioni sul database che possono avvenire contemporaneamente, ad esempio un utente effettua una modifica su una tabella mentre sulla stessa tabella, nello stesso istante, un altro utente effettua un’altra modifica: il primo utente crede di modificare la tabella con certi dati, mentre questa, in realtà è stata modificata dal secondo utente che ha cambiato i dati.

Il DBMS, in questo caso, lavorando su ‘copia’ delle tabelle, può effettuare operazioni di sincronizzazione come ad esempio ‘avvisare’ uno dei due utenti che la

tabella è in uso, bloccarne l'uso, o verificare che l'azione dei due utenti sono tali da non compromettere l'integrità dei dati con le loro modifiche.

Le tabelle 'effettive' vengono aggiornate, di solito, quando l'utente ha terminato di effettuare le operazioni sulle tabelle (la transazione è terminata).

L'utente che effettua una transazione può, forzare l'aggiornamento del database ossia l'aggiornamento delle tabelle 'effettive' durante la transazione stessa tramite l'istruzione **COMMIT**.

Come ha la possibilità di forzare l'aggiornamento delle tabelle, ha altresì la possibilità di annullare le operazioni effettuate dalla transazione (a patto che non sia stata effettuata già un'operazione COMMIT) tramite l'istruzione **ROLLBACK**. La **ROLLBACK** non fa aggiornare le tabelle 'effettive' dalle operazioni effettuate dall'utente sulle tabelle copia.