# Sui database e altre questioni di analoga importanza (v 0.8.5)

di Emanuele Ferri

## Indice generale

# Introduzione

NB: IL DOCUMENTO POTREBBE SUBIRE VARIAZIONI

Ho scritto queste pagine per chiarire alcuni concetti relativi alle basi di dati (database) e per riassumere in un unico documento tutti i termini usati (evidenziati in neretto). Purtroppo è possibile che io abbia scritto qualcosa di impreciso o addirittura di sbagliato. Se trovate errori o imprecisioni nel testo, per favore comunicatemelo.

# Dati e informazioni

***Dato** e **informazione** sono spesso utilizzati come sinonimi, ma in realtà i due termini, dal punto di vista informatico, possiedono un significato differente; infatti l'informazione è il risultato di una elaborazione dati. Pertanto il dato è un elemento conosciuto, un'informazione grezza o elementare ed è solitamente costituito da simboli che devono essere elaborati.* [1] A differenza del dato l''informazione ha una una connotazione semantica: per esempio, 0 (zero) è un dato, un simbolo privo di significato se slegato da un contesto preciso, mentre 0 inteso come parte di un gruppo sanguigno (AB0) è una informazione. Quindi i dati, per assumere un significato, per fornire una informazione, devono essere collocati in un certo contesto. Una elaborazione più complessa potrebbe consistere nell'estrapolare le quantità di gruppi sanguigni disponibili a partire dalle donazioni e dalle trasfusioni: partendo da dati grezzi in input si ottengono informazioni in output.

# Basi di dati e DBMS

Una **base di dati** (**database** o anche **DB**) è un archivio di dati persistenti e logicamente correlati tra loro, strutturato in modo da permettere la gestione, l'aggiornamento e la ricerca delle informazioni.

Un file di testo libero non è un database perché, pur contenendo dei dati, essi non sono strutturati in modo da permettere di estrapolare facilmente informazioni. Un file di testo dotato di una certa struttura (per esempio tabellare con separatori di colonna (per esempio “tab”) e di riga (per esempio “a capo”)), in un certo senso è una forma primitiva di database, pur con tutti i suoi limiti e difetti. Partendo da file di questo tipo, mediante un linguaggio di programmazione potrei popolare delle strutture dati (per esempio delle tabelle hash) per poi interrogarle.

Da notare che la struttura dati di un programma, che è residente in memoria centrale (purtroppo volatile), è priva della persistenza (se chiudo il programma perdo tutto) ed è inaccessibile al di fuori del programma. Posso ottenere la persistenza e l'accesso da parte di altri programmi salvando la struttura dati in un file (su una memoria di massa (non volatile) ovviamente), sperando che i dati non vadano persi prima del salvataggio a causa di una chiusura anomala del programma (dovuta a un errore software o a un problema hardware come la mancanza di alimentazione) e usare un formato condiviso per renderlo accessibile ad altri programmi. Anche usando un formato condiviso però, difficilmente più programmi potranno accedere contemporaneamente in lettura agli stessi dati aggiornati (devo aspettare che la struttura venga salvata) e molto difficilmente potranno farlo in scrittura.

In certe situazioni dovrei anche considerare la possibilità che più utenti vogliano accedere contemporaneamente agli stessi dati (**multiutenza**).

Realizzare un simile programma, oltre ad essere complesso, quasi certamente porterà a una soluzione specifica del problema. Per risolvere in maniera generale i problemi di gestione delle basi di dati, sono nati i DBMS. Un **DBMS** (**database management system**) è un programma pensato per creare, popolare, modificare, interrogare le basi di dati. I moderni DBMS forniscono nativamente la multiutenza e permettono l'accesso contemporaneo alla base di dati da parte di più programmi.

# Ridondanza e inconsistenza

In un database è presente la **ridondanza** quando le stesse informazioni sono derivabili da dati differenti. Per esempio, in un database che tiene traccia degli esami svolti, è ridondante ripetere, per ogni esame, il nome e il cognome dello studente in quanto è sufficiente la matricola (da cui eventualmente si ricaveranno il nome e il cognome). Altro esempio: in un registro elettronico salvare la media dei voti di ogni studente nel database causa ridondanza perché è una informazione ricavabile dai singoli voti.

La ridondanza, in genere, è negativa per due motivi: spreca spazio e può causare incompatibilità tra i valori in fase di inserimento o di modifica (**inconsistenza**). Tornando all'esempio degli esami, potrei avere nomi e cognomi differenti in corrispondenza della stessa matricola. Nell'esempio del registro elettronico la media salvata potrebbe differire da quella calcolata.

# Progettazione concettuale

La **progettazione concettuale** consiste nel *rappresentare i dati della realtà d'interesse in termini di un modello (descrizione) formale ad alto livello.* [2] Nella progettazione concettuale si deve tracciare un diagramma scevro da qualunque dettaglio implementativo e avulso da qualunque modello logico per mostrare le entità coi rispettivi attributi e le relazioni che intercorrono tra le entità. Un diagramma a questo livello non deve dipendere da un particolare DBMS o linguaggio di programmazione e neppure da un certo modello o paradigma (per esempio quello ad oggetti, quello relazionale etc.). Domande quali "che linguaggio userò?", "come farò ad implementare questo aspetto?", "quale sarà la chiave primaria?", "meglio un array o una hash table?" non trovano risposte a questo livello.

## Diagrammi entità-relazione

Lo strumento più diffuso per la progettazione concettuale sono i **diagrammi entità-relazione** (**ER**). Diversamente da un testo scritto un diagramma ER rappresenta una certa realtà in modo non ambiguo. Questi diagrammi servono per rappresentare la natura statica della realtà su cui focalizziamo la nostra attenzione.

## Entità

Le **entità** *rappresentano classi di oggetti (fatti, cose, persone, ...) che hanno proprietà comuni ed esistenza autonoma ai fini dell'applicazione di interesse. Un'occorrenza di un'entità è un oggetto o istanza della classe che l'entità rappresenta. Non si parla qui del valore che identifica l'oggetto ma dell'oggetto stesso. Un'interessante conseguenza di questo fatto è che un'occorrenza di entità ha un'esistenza indipendente dalle proprietà ad essa associate. In questo, il modello ER presenta una marcata differenza rispetto al modello relazionale nel quale non possiamo rappresentare un oggetto senza conoscere alcune sue proprietà.* [3]

Una entità è piuttosto simile ad una idea platonica, gli oggetti di quella entità una concretizzazione di quell'idea. Una entità definisce gli **attributi** posseduti dagli oggetti che le appartengono. Per esempio, parlando di moto, l'entità modello ha per attributi un nome, una cilindrata, un peso e tra gli oggetti appartenenti a quella entità c'è un modello di nome RS, cilindrata 125 e peso 121. Anche la marca è un attributo? Potrebbe esserlo, ma è più corretto pensare ad essa come ad una entità separata "collegata" all'entità modello. Una entità viene rappresentata da un rettangolo con all'interno il nome dell'entità (al singolare) che deve essere univoco all'interno del diagramma. Vi sono alcuni insiemi di attributi (anche formati da un solo elemento) che identificano univocamente un oggetto di una certa entità. Questi insiemi di attributi vengono detti **identificatori**.

## Relazioni

Le **relazioni** indicano i "legami" che intercorrono tra le entità. Ne esistono di due tipi: le associazioni e le gerarchie.

## Associazioni

Una relazione resa dal verbo avere (has-a) è detta **associazion**e. Nel modello ER l'associazione viene rappresentata mediante un rombo con all'interno il nome dell'associazione. Anche se, forzando un po', si potrebbe usare il verbo avere come nome delle associazioni, ciò risulta impossibile per l'omonimia che ne deriverebbe quando le associazioni sono più di una. Inoltre è meglio trovare nomi più significativi prediligendo i sostantivi al singolare ed evitando l'uso di verbi, anche all'infinito, che suggeriscono un verso nella lettura (non c'è un verso di lettura del diagramma in quanto rappresenta una realtà statica).

L'associazione che intercorre tra modello e marca ha **cardinalità uno a molti** perché nell'associazione il modello compare una sola volta mentre la marca molte volte (per ogni modello della stessa marca) . La stessa associazione, letta nell'altro verso, ha cardinalità molti ad uno ma in genere si preferisce parlare di uno a molti.

Esistono anche associazioni con cardinalità **uno ad uno** (studente-tesi) ed altre con cardinalità **molti a molti** (specialista-paziente).

Notare che la cardinalità delle associazioni come l'abbiamo definita poc'anzi (uno a molti, uno a uno e molti a molti) non è molto precisa: ci sarebbero delle precisazioni da fare sulla **cardinalità minima** e sulla **cardinalità massima**.

## Gerarchie

Le **gerarchie** sono quelle relazioni rese dal verbo essere e rappresentate da una freccia che va dalle entità figlie (specializzazioni) all'entità padre (generalizzazione). Quando tra due entità vi è una relazione di questo tipo significa che le entità figlie hanno tutti gli attributi dall'entità padre. Per esempio un imprenditore e un operaio sono persone e quindi possiedono tutti gli attributi tipici di una persona (nome, cognome, data di nascita, ...) oltre ad altri attributi più specifici. Si dice che imprenditore ed operaio sono **specializzazioni** di persona e che persona è una **generalizzazione** di entrambi.

Se ogni oggetto (istanza) dell'entità padre appartiene ad una entità figlia la gerarchia viene detta **totale**, altrimenti viene detta **parziale**.

Se le entità figlie non hanno oggetti (istanze) in comune la gerarchia viene detta **esclusiva**, altrimenti **sovrapposta**.

Le combinazioni possibili sono quattro:

1) Totale esclusiva. Esempio: [(a b c)(d e f)]
2) Parziale esclusiva. Esempio: [a (b c) (d e) f]
3) Totale sovrapposta. Esempio: [(a b (c d) e f)]
4) Parziale sovrapposta. Esempio: [a (b (c d) e) f]

Quando una entità eredita da più entità, si parla di ereditarietà multipla. Nei diagrammi ER non è

possibile rappresentare l'ereditarietà multipla.

# Progettazione logica

La **progettazione logica** di una base di dati dipende dal particolare **modello logico** che si vuole adottare. I principali modelli logici sono tre: gerarchico, reticolare, relazionale a cui si aggiunge il modello ad oggetti (una estensione del relazionale).

## Modello gerarchico

Il modello logico gerarchico ha una struttura ad albero e funziona bene nel tradurre associazioni di tipo uno a uno o uno a molti ma fallisce nel caso di associazioni molti a molti (introduce ridondanza). Per avere una idea di un database basato sul **modello gerarchico** si pensi a un file system o ad un file XML (pur con tutte le loro limitazioni). Importanti e diffusi esempi di database gerarchici sono i server LDAP.

## Modello reticolare

A differenza del modello gerarchico, per il **modello reticolare** è stato raggiunto uno standard condiviso. Il modello reticolare per certi versi somiglia alla struttura dati di un programma. Mediante il modello reticolare si possono tradurre associazioni di tipo molti a molti senza ridondanza. Uno dei principali problemi del modello reticolare è la sua dipendenza dal livello fisico: i dati sono correlati tra loro medianti “puntatori” (riferimenti a locazioni di memoria) che non forniscono alcuna informazione e perdono di significato se trasferiti in un altro contesto.

## Il modello relazionale

*Il **modello relazionale** è oggi quello più diffuso. Esso fu proposto nel 1970 da Edgar F. Codd nel suo articolo A Relational Model for Large Shared Data Banks apparso sulla rivista Communications of the ACM. Codd metteva in risalto i limiti dei modelli utilizzati in quegli anni (modello reticolare e gerarchico), in particolare in fatto che tali modelli non distinguessero il livello logico e quello fisico dei dati. In tali modelli l'accesso ai dati avveniva sfruttando la rappresentazione fisica dei dati, ad esempio l'indirizzo di memoria di un certo dato. Non era dunque possibile cambiare la rappresentazione fisica dei dati senza necessariamente cambiarne i metodi di accesso ai dati.*

*Il modello relazionale risponde al requisito dell'indipendenza fisica dei dati. L'accesso ai dati avviene a livello logico astraendosi dalla loro rappresentazione fisica. L'affermazione del modello relazionale è stata però lenta, in quanto la proprietà di indipendenza fisica dei dati rese difficile una implementazione efficiente delle basi di dati relazionali. Solo verso la metà degli anni 80, quindi dopo 15 anni rispetto alla proposta di Codd, le basi di dati relazionali sono diventate competitive e quindi usate su larga scala.*

*Il modello relazionale può essere suddiviso in due componenti principali:*

*- le strutture che permettono di organizzare i dati;*

- *i vincoli di integrità che permettono di inserire solo dati corretti per la realtà modellata.* [4]

*Il modello relazionale è un modello logico di rappresentazione dei dati implementato su sistemi di gestione di basi di dati (DBMS* - database management system*), detti perciò sistemi di gestione di basi di dati relazionali (RDBMS* - relational database management system*). Esso si basa sull'algebra relazionale e sulla teoria degli insiemi ed è strutturato attorno al concetto di relazione (detta anche tabella). Venne proposto da Edgar F. Codd nel 1970 per semplificare la scrittura di interrogazioni sui database e per favorire l'indipendenza dei dati; venne reso disponibile come modello logico in DBMS reali nel 1981. Oggi è uno dei modelli logici più utilizzati, implementato su moltissimi DBMS sia commerciali che open source. La tabella è la rappresentazione grafica normalmente accettata per rappresentare la relazione.* [5]

Sostanzialmente un **database relazionale** è un insieme di **relazioni** (rappresentate mediante tabelle) e di vincoli.

Solitamente la progettazione logica della base di dati deriva dalla traduzione di uno schema concettuale, anche se nulla vieta di partire direttamente dalla progettazione logica e magari realizzare successivamente uno schema ER con finalità esplicativa. La scelta del modello dei dati (gerarchico, reticolare, relazionale) dipende da molti fattori. In certi casi potrebbe essere più semplice utilizzare il file system, un file XML o una struttura dati serializzata e salvata in un file di testo ma i database relazionali garantiscono l'indipendenza logica dei dati dall'implementazione fisica e implementano associazioni di cardinalità arbitraria (anche le molti a molti) senza ridondanza dei dati.

# Relazione (matematica)

Una **relazione in matematica** è definita come un sottoinsieme del prodotto cartesiano tra n insiemi, ossia un insieme di **ennuple** (dette anche **tuple**) con lo stesso numero di elementi aventi il primo elemento appartenente al primo insieme, il secondo elemento appartenente al secondo insieme, …

Ci sono molti modi per rappresentare una relazione matematica:

### Elencazione

Nella rappresentazione per elencazione le parentesi graffe sono usate per racchiudere gli elementi dell'insieme, le parentesi tonde per racchiudere gli elementi delle ennuple. Gli elementi sono separati da virgole. Chiaramente, essendo un insieme, bisogna stare attenti a non scrivere due volte la stessa ennupla e l'ordine delle ennupe non ha importanza mentre l'ordine degli elementi all'interno delle ennuple è fondamentale. Le ennuple devono avere lo stesso numero di elementi.

Esempio:

{(1,2),(2,1),(3,0)}

### Tabella

La rappresentazione tramite tabella concettualmente è simile a quella per elencazione. Anche qui bisogna stare attenti a non scrivere due righe uguali. Ovviamente l'ordine delle righe è irrilevante

mentre quello delle colonne è fondamentale.

Esempio:

| | | | | |
|---|---|---|---|---|
| 1 | 2 | | 3 | 0 |
| 2 | 1 | uguale a | 2 | 1 |
| 3 | 0 | | 1 | 2 |

La rappresentazione tabellare può essere “migliorata” se forniamo ogni colonna di una intestazione. Da notare che ora anche l'ordine delle colonne risulta irrilevante perché quello che conta non è più la posizione ma l'intestazione.

Esempio:

| X | Y | | Y | X |
|---|---|---|---|---|
| 1 | 2 | uguale a | 2 | 1 |
| 2 | 1 | | 1 | 2 |
| 3 | 0 | | 0 | 3 |

### Diagramma di Eulero-Venn

E' una rappresentazione grafica che fa uso di pallini, cerchi ed archi.

### Piano cartesiano

E' una rappresentazione per relazioni binarie o al massimo ternarie definite su sottoinsiemi di $R^2$ o $R^3$. Per definizione, esiste una corrispondenza biunivoca fra i punti del piano (spazio) cartesiano e le coppie (terne) di numeri reali. Chiaramente non è quasi mai possibile ricavare tutte le ennuple appartenenti alla relazione partendo dal solo disegno (per i limiti della carta e dell'inchiostro) o dalla rappresentazione grafica al computer (per i limiti della memoria video e dei pixel).

### Equazione

La rappresentazione mediante equazione di una relazione si colloca su un piano completamente differente. Con le equazioni si possono rappresentare relazioni di cardinalità anche più che numerabile, cosa di fatto impossibile con le precedenti rappresentazioni. Notare che le equazioni sono una infinità numerabile, mentre l'insieme delle relazioni definite su sottoinsiemi di $R^n$ è più che numerabile, motivo per cui, mediante equazioni, possiamo rappresentare solo alcune relazioni.

Esempio:

x^2+y^2=1

che rappresenta la circonferenza goniometrica.

## Relazione (modello relazionale)

Le **relazioni del modello relazionale** NON HANNO NULLA A CHE FARE con le relazioni del modello concettuale (associazioni e gerarchie) mentre somigliano alle (sono le) relazioni della matematica: la differenza, se di differenza si può parlare, è che vengono rappresentate mediante tabelle fornite di intestazioni dove l'ordine delle colonne è chiaramente ininfluente. Si tratta di una rappresentazione delle relazioni che comunque si usa anche in matematica. Da sottolineare che, ovviamente, anche l'ordine delle righe è irrilevante.

Esempio:

| nome | console |
|---|---|
| Super Mario 64 | Nintendo 64 |
| Super Mario Sunshine | Gamecube |
| Devil May Cry | Playstation 2 |

Prima si è accennato a relazioni definite su sottoinsiemi di $R^n$ con cardinalità anche più che numerabile. Chiaramente simili relazioni non ci interessano minimamente: le relazioni che tratteremo avranno sempre un numero finito di ennuple.

Un database relazionale in sostanza è un insieme di tabelle e di vincoli. Ribadisco che il termine relazionale si riferisce al fatto che le tabelle cercano di rappresentare vere e proprie relazioni e non ai possibili legami tra tabelle. Ho scritto “cercano di rappresentare” perché per rappresentare una relazione con una tabella, le ennuple (righe) non dovrebbero ripetersi. I DBMS relazionali tuttavia lo permettono. Chiaramente non è una situazione desiderabile. Da un punto di vista pratico ennuple uguali occupano spazio inutilmente e, cosa ancora peggiore, non è possibile distinguere una ennupla da un'altra. Comunque, se la tabella è fatta bene (senza ennuple che si ripetono) essa rappresenta effettivamente una relazione.

# Un po' di terminologia

In informatica con **record** si intende una struttura contenente dati non omogenei. E' un termine usato comunemente quando si parla di basi di dati (non solo relazionali). Quando si parla di database relazionali la riga di una tabella è una ennupla ma spesso ci si riferisce ad essa col termine di tupla o record. Il termine ennupla in genere si usa in matematica, tupla mi sembra più usato nell'algebra relazionale, record è un termine più generale dell'informatica (usato anche nei modelli non relazionali).

**Campo** è un termine specifico che viene usato solo quando si parla di tabelle mentre **attributo** è più generale (si parla di attributo nei database relazionali, nei diagrammi ER, nella programmazione ad oggetti, ...).

## Chiavi

Una **superchiave** di una relazione è un insieme di attributi che identifica univocamente una ennupla. Le ennuple di valori della superchiave non devono ripetersi. Notare che i singoli valori possono anche ripetersi, sono le ennuple di valori che formano la superchiave che non si devono ripetere. Le superchiavi possono contenere valori **null** ossia mancare di alcuni valori. La presenza di valori null non impedisce di identificare univocamente una ennupla ma esclude alcune ennuple dall'identificazione.

Una **chiave candidata**, detta più semplicemente **chiave**, è una **superchiave minimale** ossia una superchiave che, privata anche di un singolo attributo, non è più più superchiave. Anche una chiave candidata può contenere valori null.

Tra le chiavi candidate se ne può scegliere una, in qualche modo privilegiata, che viene detta **chiave primaria** (**primary key**). A differenza delle chiavi candidate, la chiave primaria NON PUO' CONTENERE VALORI NULL.

La chiave primaria dunque è un insieme privilegiato di attributi della relazione che identifica univocamente TUTTE le ennuple (o, se preferite, un insieme privilegiato di campi della tabella che identifica univocamente TUTTI i record) scelta tra le candidate perché più significativa, più corta, meno esosa in termini di spazio o per altri motivi ancora.

Notare che la scelta della chiave primaria dipende dal libero arbitrio di chi progetta la base di dati.

A questo punto è lecito porsi due domande:

In cosa la chiave primaria differisce da una qualunque chiave candidata con valori diversi da null?

Perché la chiave primaria deve essere una sola?

Per esempio: se avessi una tabella studenteUniversitario, perché dovrei scegliere UNA chiave primaria tra le due chiavi candidate matricola e codiceFiscale? Entrambe le chiavi identificano univocamente i record della tabella, entrambe le chiavi si presume che non contengano valori null, entrambe hanno la stessa lunghezza: perché non usare una o l'altra a seconda delle esigenze?

La verità è che nessuno mi impone di scegliere una chiave primaria: le chiavi candidate, se prive di valori null, vanno altrettanto bene. Potrei accedere ai dati di uno studente mediante la matricola se sto facendo una ricerca sugli iscritti ad un certo esame e attraverso il codice fiscale se mi servono informazioni sul reddito per l'assegnazione di una borsa di studio. Da un punto di vista strettamente semantico (di significato) non c'è differenza tra una chiave primaria e una qualunque chiave candidata priva di valori null. L'esistenza della chiave primaria probabilmente ha più che altro motivazioni storiche (compatibilità con determinati DBMS).

Una relazione ha sempre almeno una superchiave che è l'insieme di tutti gli attributi.

| matricola | nome | cognome |
|---|---|---|
| 1 | Pino | Scotto |
| 2 | Pino | Daniele |
| 3 | Gianni | Scotto |

Nell'esempio {matricola,nome,cognome}, {matricola,nome}, {matricola,cognome}, {nome,cognome}, {matricola} sembrerebbero superchiavi mentre {nome} e {cognome} no. Per esempio {matricola,nome} è superchiave perché (1,Pino), (2,Pino) e (3,Gianni) sono diverse mentre {nome} non è superchiave perché (Pino) compare due volte. Tra le superchiavi, {matricola} e {nome,cognome} sono anche minimali. Le superchiavi dovrebbero servire per identificare univocamente una ennupla usando solo alcuni attributi (es: {matricola,nome}), le chiavi a farlo con il minor numero di attributi possibili (es: {matricola}). Da notare però che {nome,cognome} è superchiave non in virtù della natura dei suoi attributi ma perché, allo stato attuale, quella relazione non contiene omonimi (che però nella realtà ci sono). Se inserissimo nuovi valori (come accade normalmente) non è detto che l'univocità si preservi. {matricola} invece, anche se modificassimo i valori delle ennuple presenti o ne aggiungessimo altre, evitando di fare errori, resterà sempre una superchiave perché non ci sono due matricole uguali.

Purtroppo nei database relazionali, alcuni valori potrebbero non essere disponibili (null). Due valori mancanti sono considerati diversi, anche se, secondo me, l'approccio più corretto sarebbe astenersi dal dare un giudizio (però così facendo non si potrebbero più fare confronti). Dunque le superchiavi, pur identificando univocamente le ennuple, non sono in grado di identificarle tutte.

| matricola | nome | cognome |
|---|---|---|
| null | Pino | null |
| null | Pino | null |
| 3 | Gianni | Scotto |

Contrariamente a quello che si potrebbe pensare, la prima e la seconda ennupla sono considerate diverse.

Altro esempio:

| matricola | nome | cognome | codiceFiscale |
|---|---|---|---|
| in116 | Han | Solo | slhnxx85r20f132x |
| in117 | Pino | Scotto | sctpni60a01h620i |
| in234 | Luke | Skywalker | skylku00a01f205a |

In questa anagrafica di studenti universitari, {matricola} e {codiceFiscale}, sono entrambe chiavi candidate in quanto sono superchiavi perché i valori non si ripetono e sono minimali (più corte di così non si può).

Più avanti, quando parleremo dei vincoli, vedremo che è possibile IMPORRE l'inserimento e l'unicità dei valori o, in alternativa, generare automaticamente valori sempre diversi (mediante contatori).

Nell'esempio di prima potremmo scegliere {matricola} come chiave primaria perché in un database di studenti sembra la scelta migliore.

Identificare univocamente le ennuple con un insieme di attributi ci serve anche per collegare tra loro le tabelle. Usare il minor numero possibile di attributi per farlo semplifica le interrogazioni. Pretendere che un valore sia sempre presente garantisce che tutti i record vengano presi in considerazione.

Il collegamento tra due tabelle può avvenire utilizzando una qualunque superchiave ma chiaramente è meglio usare una chiave candidata, possibilmente con tutti i valori presenti. Notare che, per quanto solitamente si usi la chiave primaria, non è sbagliato usare una qualunque chiave candidata.

L'insieme di attributi che corrisponde, nell'altra tabella, ad una chiave candidata prende il nome di **chiave esterna** (**foreign key**).

## Vincoli d'integrità

Un **vincolo di integrità** è una proprietà che deve essere soddisfatta. Anche i vincoli di integrità si possono assegnare tramite **SQL** (**structured query language**).

Facciamo un esempio: l'altezza di una guardia svizzera non deve essere inferiore al metro e settantaquattro. Se cerco di inserire nella tabella una aspirante guardia svizzera alta 170 cm, il DBMS me lo impedisce, segnala un errore e gli dice di trovarsi un altro lavoro. Se inseriamo un record che viola il vincolo, il record non viene inserito e il DBMS genera un errore.

Quella che segue è la classificazione dei vincoli d'integrità:

**Vincoli intrarelazionali o interni** (coinvolgono una sola tabella)

- **Su una singola ennupla**
  - **Su un attributo** (una restrizione del dominio di un singolo attributo)

**Su più attributi**

**Su più ennuple**

**Vincoli interrelazionali o esterni** (coinvolgono più tabelle)

Un importante vincolo su un solo attributo è **not null**. Con not null imponimo che il valore di un attributo sia diverso da null. Significa che quel valore ci deve essere, è importante, non può mancare!

Anche l'altezza dell'aspirante guardia svizzera è un vincolo su un solo attributo. La scelta delle guardie svizzere però coinvolge anche altri fattori, oltre all'altezza (età, religione, nazionalità). Il peso in effetti non è un vincolo però potrebbe interessare l'indice di massa corporea (IMC) per evitare guardie troppo magre o troppo grasse. L'organizzazione mondiale della sanità considera regolare un IMC compreso tra 18,5 e 24,9. Poiché IMC=massa/altezza2 chiaramente un vincolo di questo tipo coinvolge più attributi (altezza e peso) della stessa ennupla.

**Unique** è invece un vincolo intrarelazionale su più ennuple. Con unique si impone che un insieme attributi di una certa tabella non si ripeta, ossia che un certo insieme di attributi formi una chiave. Attenzione: in una tabella anagrafica, è diverso dire che nome e cognome devono essere unici rispetto a dire che il nome deve essere unico e il cognome deve essere unico. Nel primo caso il vincolo unique viene applicato una volta sola ad un insieme di due attributi. Nel secondo, più restrittivo, viene applicato due volte, a due attributi diversi (meglio, a due insiemi contententi un solo attributo ciascuno). Se riteniamo un insieme di attributi formi una chiave, possimo attribuirgli il vincolo unique. Se imponiamo che quegli attributi siano anche diversi da null ancora meglio.

Un altro vincolo intrarelazionale è il vincolo di **chiave primaria** (**primary key**). Si tratta probabilmente del vincolo più importante ed utilizzato. Questo vincolo, che si può definire al massimo una volta per ogni tabella, impone che un certo insieme di attributi formi una chiave primaria.

Un importante vincolo interrelazionale è quello di **chiave esterna** (**foreign key**). Il vincolo di chiave esterna impone che un certo insieme di attributi sia una chiave esterna. Va fatto notare che non è il vincolo di chiave esterna a legare tra loro due tabelle! I legami tra tabelle non sono salvati ed esistono indipendentemente da questi vincoli. Sono nella testa di chi utilizza il database. Sono nella scelta dei nomi degli attributi. Sono nel legame che, implicitamente o esplicitamente, esiste tra gli attributi di tabelle diverse.

# Sitografia:


[1] http://www.differenzatra.it/differenza-tra-dato-e-informazione

[2] http://www.di.unipi.it/~levifran/4-progettazioneconcettuale.pdf

[3] http://it.wikipedia.org/wiki/Modello_ER#Entit.C3.A0

[4] http://sole.dimi.uniud.it/~massimo.franceschet/teatro-sql/logica.html

[5] http://it.wikipedia.org/wiki/Modello_relazionale