DB&SQL

una introduzione a Database, SQL e ambiente MySQL

2016.04

Indice

Introduzione

Questi appunti rappresentano una introduzione abbastanza generale, anche se non esaustiva appunto perché introduttiva, ai concetti essenziali sui Database con particolare riferimento al modello relazionale. Gli appunti sono idealmente divisi in due parti: una parte teorica che introduce le motivazioni che hanno portato alla teorizzazione della tecnologia delle basi di dati e le basi teoriche dell'algebra relazionale, una seconda parte rivolta alle applicazioni pratiche incentrata sulla trattazione del linguaggio SQL in ambiente MySQL.

In uno dei primi paragrafi è riportato un frammento di codice sorgente C. Per seguire gli argomenti trattati in questi appunti non è richiesta la conoscenza del linguaggio: il listato è commentato e spiegato nelle parti che interessano e non è indispensabile la cognizione esatta delle istruzioni (semplificate) del programma; tuttavia una certa conoscenza di un linguaggio di programmazione, anche generale, può agevolare la comprensione della tesi presentata.

È richiesta una conoscenza minima del formalismo matematico e delle definizioni della teoria degli insiemi come dei concetti essenziali del modello Client/Server.

Negli esempi di applicazione del linguaggio SQL, spesso, sono riportate le istruzioni inserite dall'utente e le risposte del client mysql. Es:

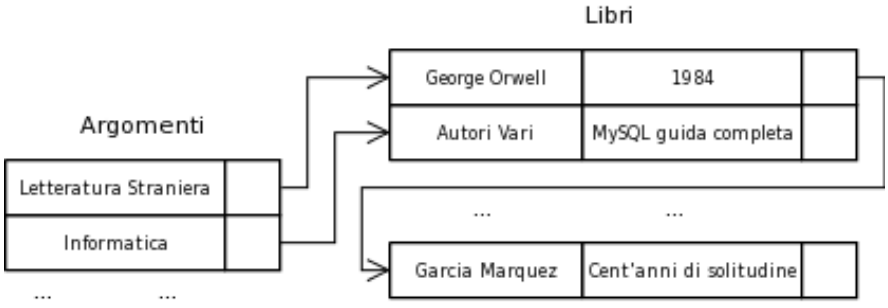
```
~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 47
Server version: 5.5.31-0+wheezy1 (Debian)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

il formalismo adottato in questi appunti prevede, come nell'esempio, la scritta in **`grassetto`** per i comandi introdotti dall'utente e la scritta `normale` per le risposte del programma.

Gli archivi tradizionali

Le applicazioni dell'Informatica gestionale prevedono il trattamento di grandi quantità di dati registrate su memorie di massa. Una applicazione informatica deve essere in grado di consentire la conservazione, l'organizzazione e la ricerca dei dati. In generale si tratta di scegliere quali informazioni conservare, organizzarle in archivi e poi correlare gli archivi fra di loro.

A titolo di esempio si pensi a una gestione semplificata di una biblioteca che registri i libri in suo possesso catalogandoli per argomento. In questo caso si potrebbe decidere di tenere un archivio in cui sono registrati i dati dei libri e un archivio in cui sono registrati gli argomenti per la catalogazione dei libri.



Nell'archivio degli argomenti sono previsti, per ogni registrazione, una descrizione dell'argomento e una indicazione su dove trovare, nell'archivio dei libri, il primo libro dell'argomento. Il libro, a sua volta, registra il nome dell'autore, il titolo e una indicazione sul prossimo libro che viene classificato nello stesso argomento. Le indicazioni sono dei puntatori che specificano la posizione fisica della registrazione puntata. Tramite i puntatori *si legano* gli archivi e le registrazioni tra di loro in modo da restituire informazioni complesse: nell'esempio l'elenco dei libri di letteratura straniera o di informatica.

Quello appena mostrato può essere definito **Database**.

> *Il Database è una rappresentazione della realtà di interesse in termini di dati e collegamenti fra dati.*

La realtà che si vuole rappresentare è quella in cui si classificano i libri in base all'argomento. I dati sono registrati in due **file**: ogni registrazione (**record**) prevede nel file dei libri i **campi** autore, titolo, puntatore al prossimo libro dello stesso argomento. I record del file argomenti prevedono come campi la descrizione e il puntatore al primo libro associabile a quell'argomento. I dati sono collegati fra di loro mediante i puntatori e così: i libri sono libri classificabili negli argomenti registrati, gli argomenti sono argomenti che fanno riferimento a libri registrati.

Il Database rappresentato è ***gerarchico***: per avere l'elenco dei libri bisogna leggere il record dell'argomento che fornisce indicazioni sul primo libro associato e questo, a sua volta, fornisce indicazioni sulla posizione del prossimo libro. Il rapporto fra il record argomento e il record libro è del tipo *padre-figlio*. Il Database è costruito per rispondere in maniera efficiente a richieste (**query**) del tipo: *quali sono i libri dell'argomento X?* Infatti i collegamenti portano a rintracciare immediatamente tutte le registrazioni dei libri associati all'argomento.

In sintesi si parte dall'esigenza da soddisfare (elenco dei libri di quell'argomento), si organizzano i dati in modo da reperirne nel modo più efficiente possibile quelli richiesti, si scrive un programma in un determinato linguaggio di programmazione per recuperare i dati così organizzati.

Sempre a titolo di esempio si riporta una funzione in C che risolve il problema proposto.

```
/* Definizione del record (tracciato record) */

typedef struct {                                                    /*1*/
  char autore[20];
  char titolo[50];
  int prossimo;
}libro;

/*
```

```
    Legge tutti i libri che trattano di un argomento.
    Alla funzione è passata come parametro la posizione
    del primo record del file Libri da leggere:
    il primo libro dell'argomento desiderato
*/

void elenco(int primo){
  int posto;
  long dove;
  FILE *fp;
  libro buflib;

  fp=fopen("DatiLib.dat","r");

  /* La prima volta il posto da cui leggere è
     quello conservato nel parametro */

  posto = primo;

  /* Finchè esiste prossimo libro dello stesso argomento
     ovvero posizione diversa dal valore zero */

  while(posto){

    /* calcola posizione dove spostare puntatore del file */

    dove=(long) sizeof(libro)*(posto-1);                          /*2*/

    /* sposta puntatore del file */

    fseek(fp,dove,SEEK_SET);                                      /*3*/

    /* legge una certa quantità di byte dalla posizione
       in cui è stato spostato il puntatore */

    fread(&buflib,sizeof(libro),1,fp);                            /*2*/

    /* elabora i dati dei campi */

    puts(buflib.autore);
    puts(buflib.titolo);
    printf("%d\n",buflib.prossimo);

    /* posizione del prossimo libro dello stesso argomento */

    posto = buflib.prossimo;
  }

  fclose(fp);
}
```

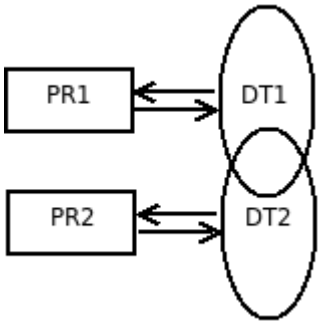
In breve: la funzione legge dal file i record dei libri e ne mostra il contenuto sullo schermo. Al di là della comprensione delle singole righe di codice, sufficientemente commentate, interessa in questa sede fare due osservazioni:

- la struttura del record è specificata nel programma e le istruzioni per il trattamento dei dati tengono conto e sono scritte in funzione di tale struttura (1). La struttura è utile al compilatore per il calcolo della quantità di byte di cui è composto il record e al

programmatore per dare indicazioni su come suddividere logicamente il record. Per fare riferimento a ciò la (1) viene chiamata *tracciato record*

- le istruzioni per il reperimento dei record dal file tengono conto delle dimensioni fisiche, in termini di byte, del record stesso (2). Al sistema operativo viene chiesto di leggere una certa quantità di byte che sono quelli che costituiscono il record. La dimensione del record dipende dall'implementazione del compilatore utilizzato per la traduzione del programma. Nell'esempio, a parte l'autore e il titolo del libro di cui si conosce la dimensione, la quantità di byte necessari per contenere un dato di tipo `int` dipende dall'implementazione che di questo tipo ne fa il compilatore. Per questo motivo il calcolo delle dimensioni del record viene demandato alla funzione `sizeof` e non è espresso come quantità costante. Si tiene inoltre conto dell'organizzazione fisica dei record nel file (3): ci si sposta all'interno del file di blocchi di byte della dimensione del record.

Da quanto esposto dovrebbe essere evidente la correlazione fra procedura di accesso ai dati e organizzazione degli stessi. Tutto ciò è abbastanza normale visto lo scopo di una applicazione informatica che è quello di rendere quanto più efficiente possibile una determinata richiesta. Qualche problema si viene a generare invece quando si cercano di creare procedure per l'automatizzazione di ulteriori richieste.



La procedura PR1 gestisce ed è ottimizzata per un certo insieme di dati DT1. Lo stesso vale per la procedura PR2 e l'insieme di dati DT2. Se le due procedure riguardano due aspetti della stessa realtà i due insiemi DT1 e DT2, in genere, non sono disgiunti: può esserci necessità di conservare parte di dati già esistenti ma non utilizzabili perché inseriti in contesti ottimizzati per altri scopi. Mano a mano che si ha necessità di automatizzare nuove procedure, e quindi gestire nuovi insiemi organizzati di dati, aumentano i duplicati di dati.

Limiti degli archivi tradizionali

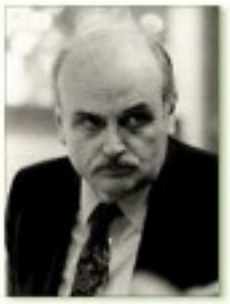
La duplicazione di dati (**ridondanza**) ancorché ineliminabile è causa di alcune conseguenze negative ben conosciute:

- spreco di spazio su cui conservare i dati duplicati. Problema questo sentito fortemente in tempi quando il costo di conservazione per byte dei dati era molto elevato
- problemi di sicurezza. Non è possibile controllare tutte le ricorrenze di un dato anche perché sono gestite da procedure diverse e non ne si conosce realmente la quantità e la locazione
- possibile inconsistenza di dati. Non controllando tutte le ricorrenze quando si aggiorna un dato sarebbe necessario aggiornare tutte le ricorrenze, cosa difficoltosa in ragione delle motivazioni espresse nel punto precedente
- impossibilità di risposte in *real-time*. La risposta ad una query non può essere immediata essendo i dati dispersi in tanti archivi. I tempi di risposta aumentano in ragione dell'aumentare delle procedure e, di conseguenza, degli insiemi di dati gestiti dalle stesse.

Oltre la ridondanza si possono evidenziare altri problemi dei Database gerarchici o reticolari (Database in cui possono esserci più puntatori e i puntatori possono andare anche in altre direzioni oltre che nel verso padre-figlio):

- I sistemi spesso si sviluppano a partire da una situazione iniziale (DT1 nell'esempio precedente) costruita senza un'analisi generale approfondita e con una visione parziale (PR1).
- Non esiste separazione fra l'organizzazione logica e l'implementazione fisica dei dati. L'organizzazione dei libri per argomento dell'esempio è ottenuta mediante l'utilizzo di due file e di una serie di puntatori che rispecchiano la logica che tiene assieme i dati: il puntatore dall'argomento al libro, quello dal libro dell'argomento al prossimo dello stesso argomento.
- Rigidità del modello. Modificare i dati comporta la modifica delle procedure di accesso: come evidenziato in precedenza il codice riporta la struttura dei dati. Una modifica anche lieve della richiesta rispetto a quella iniziale su cui si sono organizzati i dati, porta una modifica totale dell'organizzazione e delle conseguenti procedure di accesso. Anche la risposta a nuove esigenze porterebbe una riorganizzazione generale dei dati e anche delle procedure esistenti, rendendo oggettivamente impensabile, in conseguenza dei costi da sostenere, una espansione.

Il modello relazionale



Per superare i limiti dell'organizzazione tradizionale degli archivi, alla fine degli anni '60 Edgar Codd, matematico inglese che lavorava come ricercatore presso IBM, pubblica la sua teoria sul modello relazionale. Si tratta, sostanzialmente, di un ribaltamento a 360° dei rapporti fino ad allora esistenti fra dati e procedure di accesso ad essi.

Codd rifonda l'argomento partendo da una teorizzazione di tipo matematico sui dati. Il punto di partenza è la teoria degli insiemi. Ora i dati esistono a prescindere dalle procedure di accesso ma in quanto descrizione di una realtà. I dati sono gestiti da uno strato software (**DBMS** Data Base Management System) che si occupa di presentare una interfaccia di alto livello verso gli stessi rendendo indipendente l'organizzazione logica dei dati dalla loro implementazione fisica. Un po' come, in altri ambiti per esempio, il Sistema Operativo presenta una interfaccia di alto livello verso l'hardware permettendo, per esempio, di cambiare la stampante senza che i programmi che accedono ad essa necessitino di alcuna modifica. In un Database si deve poter cambiare l'organizzazione fisica dei dati senza che questo influenzi in alcun modo l'organizzazione logica.

Nel modello relazionale tutti i dati registrati in un Database sono rappresentati come un insieme di tabelle.

Libri

Codice	***Autore***	***Titolo***
L01	George Orwell	1984
L02	Autori Vari	MySQL guida completa
L03	Garcia Marquez	Cent'anni di solitudine

Ogni colonna (***attributo***) contiene una informazione correlata con quelle delle altre colonne della

stessa riga (***tupla***). Ogni attributo è definito in un certo *dominio*: per esempio l'autore ha per dominio tutti i nomi degli autori che hanno scritto libri registrati nella libreria. Ogni colonna è quindi un insieme collegato agli altri insiemi da una ***relazione*** (la tabella). La tabella Libri è una relazione (da qui il nome del modello). Il numero delle colonne della relazione si chiama *grado* e la quantità di tuple della relazione è detta *cardinalità* della relazione. Fra gli attributi ne deve esistere uno, o anche una combinazione di più di uno, che identifica in maniera univoca una tupla. Questo attributo viene chiamato *chiave*. Se esiste più di una chiave, se ne sceglie una fra le chiavi candidate che viene chiamata ***chiave primaria*** (*PK primary key*) e che da questo momento identifica in modo univoco la tupla. Nell'esempio riportato la chiave primaria è il codice del libro: ci possono essere più ricorrenze del nome o del titolo ma il codice è legato in modo univoco ad un unico libro.

Essendo la relazione un insieme, gode di alcune proprietà fondamentali:

- non possono esistere tuple duplicate. In un insieme non ci sono elementi doppioni
- deve esistere necessariamente una chiave primaria
- l'ordine delle righe e delle colonne non è influente
- ogni attributo non può essere scomposto in sotto-elementi.

Nel Database di esempio sono definite due relazioni: Argomenti e Libri.

Argomenti

Codice	Descrizione
A01	Letteratura Straniera
A02	Informatica

Libri

Codice	Autore	Titolo	Codice Argomento
L01	George Orwell	1984	A01
L02	Autori Vari	MySQL guida completa	A02
L03	Garcia Marquez	Cent'anni di solitudine	A01

La relazione Argomenti ha come chiave primaria Codice. Nella relazione Libri è stato aggiunto l'attributo Codice Argomento: in questo modo in ogni libro c'è un riferimento all'argomento di cui tratta. Codice Argomento ha come dominio l'insieme di valori di Codice della relazione Argomenti. Il codice dell'argomento trattato in un libro deve essere uno di quelli già esistenti nella relazione Argomenti. L'attributo Codice Argomento si chiama ***chiave esterna*** (*FK foreign key*).

Il DBMS, lo strato software che gestisce il Database, si deve occupare di garantire l'**integrità referenziale**: la FK di una tupla della relazione Libri o ha valore nullo o ha un valore che compare nella PK di una tupla della relazione Argomenti.

Vantaggi del modello relazionale

Il modello relazionale offre diversi vantaggi a confronto con i modelli di organizzazione di dati precedenti:

- è basato su una solida e consolidata teoria matematica (la teoria degli insiemi) con quello che ne consegue in termini di operazioni
- è flessibile. Non ci sono riferimenti espliciti (non ci sono puntatori) che legano le tabelle fra di loro. Ogni tabella si può collegare a qualsiasi altra definita nel Database utilizzando gli operatori che verranno illustrati nei prossimi paragrafi

- garantisce l'indipendenza dall'organizzazione fisica dei dati. Chi interroga un DBMS non ha necessità di conoscere dove e come sono registrati in memoria di massa i dati: dal suo punto di vista esistono solo le relazioni
- garantisce l'indipendenza dell'organizzazione logica dei dati dalle procedure di accesso ad essi. Le tabelle non nascono per rispondere a determinate query ma per descrivere la realtà di interesse
- garantisce un elevato livello di sicurezza. I dati sono gestiti, in modo centralizzato, dal DBMS e le procedure lo interrogano per ottenere quelli di proprio interesse. Il DBMS può quindi tenere nota di chi accede e a quali dati
- garantisce una bassa e controllata ridondanza. La duplicazione dei dati può essere limitata a solo quelli indispensabili per l'ottimizzazione della gestione (*ridondanza strategica*). Qualora si dovesse presentare una ridondanza la relazione può sempre essere divisa in due sotto relazioni con dati non ridondanti

Questi in definitiva sono stati i motivi della diffusione e del successo del modello al di là di una fase di diffidenza iniziale dovuta, principalmente, alle esose esigenze di risorse hardware richieste da una gestione che si basa su tabelle piuttosto che, come i modelli precedenti, su puntatori che permettono di limitare gli accessi alle memorie di massa solo ai record di interesse e hanno basse esigenze di occupazione di memoria centrale.

Algebra relazionale

Codd, oltre a definire le proprietà del modello relazionale, elencò anche una serie di operazioni applicabili alle relazioni. Le operazioni, essendo il modello relazionale derivato dalla teoria degli insiemi, derivano dalle operazioni definite sugli insiemi.

Prima di continuare è necessario stabilire una convenzione sulla scrittura di una relazione e chiarire un concetto fondamentale:

Libri

Codice	*Autore*	*Titolo*	*Stanza*	*Scaffale*	*Argomento*
L01	George Orwell	1984	S01	SC01	A01
L02	Autori Vari	MySQL guida completa	S02	SC01	A02
L03	Garcia Marquez	Cent'anni di solitudine	S01	SC02	A01

- la relazione Libri si indica: **Libri(<u>codice</u>, autore,titolo,stanza,scaffale,<u>argomento</u>)** dove è evidenziato il nome della relazione e i suoi attributi sono elencati racchiusi fra parentesi. Le chiavi della relazione sono sottolineate: codice è la chiave primaria, argomento è una chiave esterna. Il formalismo adottato si chiama *schema della relazione* e rappresenta la parte costante. I valori contenuti nelle tuple della relazione, che sono la parte variabile (i dati in un Database si aggiungo e tolgono), sono le *istanze della relazione*. Per esempio L01,George Orwell, 1984, S01, SC01, A01 è una istanza della relazione Libri.
- tutte le operazioni definite nell'algebra relazionale sono **operazioni interne** nel senso che applicate a relazioni producono nuove relazioni

A parte le operazioni comunemente applicabili agli insiemi (Unione, Intersezione, Differenza ...)

che sono definite allo stesso modo sulle relazioni (si ricorda che la relazione è un insieme), si evidenziano qui alcune operazioni particolari sulle relazioni che hanno notevole interesse nelle query:

$\pi_{Autore,Titolo}(Libri)$

Autore	*Titolo*
George Orwell	1984
Autori Vari	MySQL guida completa
Garcia Marquez	Cent'anni di solitudine

Proiezione: si chiama proiezione, per esempio, di Libri su Autore e Titolo e si indica con il simbolo: $\pi_{Autore,Titolo}(Libri)$ la relazione che si ottiene prendendo tutte le tuple di Libri ma con solo le colonne degli attributi specificati.

$\sigma_{Stanza="S01"}(Libri)$

Codice	*Autore*	*Titolo*	*Stanza*	*Scaffale*	*Argomento*
L01	George Orwell	1984	S01	SC01	A01
L03	Garcia Marquez	Cent'anni di solitudine	S01	SC02	A01

Selezione: si chiama selezione l'operazione che produce una nuova relazione con gli stessi attributi della relazione originaria ma con solo le tuple che soddisfano alle condizioni specificate. Per esempio $\sigma_{Stanza="S01"}(Libri)$ produce una relazione in cui figurano solo i libri che si trovano nella stanza con quel codice. Si tratta in sostanza di trovare un sottoinsieme. Nella condizione della selezione si possono utilizzare tutti gli operatori di confronto (<, >, >=, <=, =) e, inoltre se ci sono più condizioni possono essere utilizzati gli operatori booleani: AND, OR e NOT.

$Libri \bowtie_{Argomento=Codice} Argomenti$

Argomenti

Codice	*Descrizione*
A01	Letteratura Straniera
A02	Informatica

Libri

Codice	*Autore*	*Titolo*	*Stanza*	*Scaffale*	*Argomento*
L01	George Orwell	1984	S01	SC01	A01
L02	Autori Vari	MySQL guida completa	S02	SC01	A02
L03	Garcia Marquez	Cent'anni di solitudine	S01	SC02	A01

Codice	*Autore*	*Titolo*	*Stanza*	*Scaffale*	*Argomento*	*Descrizione*
L01	George Orwell	1984	S01	SC01	A01	Letteratura Straniera
L02	Autori Vari	MySQL guida completa	S02	SC01	A02	Informatica
L03	Garcia Marquez	Cent'anni di solitudine	S01	SC02	A01	Letteratura Straniera

Join: a differenza delle altre due (operazioni unarie) è definita su due relazioni (operazione binaria).

Libri$\bowtie_{\text{Argomento=Codice}}$**Argomenti** Il join fra le relazioni Libri e Argomenti crea una relazione che ha tutte le tuple che si ottengono combinando le tuple delle due relazioni purché le due tuple che la compongono abbiano lo stesso valore nell'attributo specificato. L'attributo compare una sola volta nella relazione risultato. Se l'attributo comune ha lo stesso nome in entrambe le relazioni l'operazione si chiama *join naturale*, altrimenti *theta join*. Sostanzialmente si tratta di generare un sotto insieme del prodotto cartesiano fra due insiemi (le relazioni dell'operazione). In generale non è necessario che gli attributi su cui si effettua il join abbiano valore uguale ma basta che soddisfino ad una determinata condizione (*Inner Join*).

Combinando fra di loro gli operatori possono essere effettuate query per l'estrazione di qualsiasi genere di informazione si desideri purché, ovviamente, esista nel Database. A titolo di esempio si propongono due query, una più semplice e la seconda in cui si richiedono informazioni più articolate.

Esempio 1: *Elencare autore e titolo di tutti i libri della stanza S01*

$$\pi_{\text{Autore,Titolo}}(\sigma_{\text{Stanza="S01"}}(\textbf{Libri}))$$

prima viene effettuata una selezione sulla stanza generando una relazione che contiene solo le tuple con quel valore nell'attributo *Stanza*. Successivamente l'operatore di proiezione restituisce una relazione con gli attributi di interesse.

Esempio 2: *Elencare titolo, stanza, scaffale e argomento di tutti i libri che ha scritto George Orwell*

$$\pi_{\text{Titolo,Stanza,Scaffale,Descrizione}}(\sigma_{\text{Autore="George Orwell"}}(\textbf{ Libri}\bowtie_{\text{Argomento=Codice}}\textbf{Argomenti}))$$

le informazioni richieste non sono contenute in una solo relazione, è necessario innanzitutto formare una relazione che abbia tutti gli attributi richiesti mettendo assieme (join) le relazioni interessate. Successivamente sulla relazione risultante viene effettuata una selezione sull'autore escludendo tutte le altre tuple. Infine si effettua la proiezione sugli attributi richiesti.

In generale l'ordine delle operazioni da effettuare per ottenere i dati richiesti coincide con quello utilizzato nella risposta alla query dell'esempio 2.

Quando si referenzia un attributo di una relazione possono esserci ambiguità: il nome dell'attributo è univoco nella relazione ma se i dati di una query sono estratti da più relazioni, può accadere che ci siano attributi, in relazioni diverse, con lo stesso nome. Per risolvere le ambiguità i nomi degli attributi è d'uso scriverli referenziandoli completamente anche con il nome della relazione cui appartengono, si avrà pertanto, per esempio: Libri.Autore, Argomenti.Descrizione.

Modello E-R e progettazione delle basi di dati

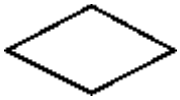
Il Database di esempio è stato presentato già con le relazioni fra i dati, ma quando si prende in esame una realtà le relazioni rappresentano la fase finale della progettazione di una base di dati.

La prima fase della progettazione è costituita dalla **progettazione concettuale**: in questa fase viene costruito lo schema concettuale dove viene rappresentato il contenuto della base di dati prescindendo dal modo come i dati saranno rappresentati nel modello di Database scelto. La **progettazione logica** parte dallo schema generato dalla fase precedente e lo traduce nel modello della base di dati adottata. Per esempio, nel modello relazionale, lo schema viene tradotto in un insieme di relazioni con definite le chiavi primarie ed esterne. Nei modelli precedenti esisteva una

ulteriore fase chiamata **progettazione fisica** in cui viene costruito lo schema fisico dei dati tenendo conto delle specifiche di organizzazione del sistema (file, puntatori, ...).

Il **modello Entità-Relazioni** (*Entity-Relationship*) è il più diffuso modello concettuale dei dati in un Database relazionale. Lo schema E-R è costituito da un diagramma i cui elementi fondamentali sono:

Entità: rappresenta una classe di oggetti di interesse che hanno caratteristiche comuni. Nel Database di esempio della biblioteca ci sono due entità: Libro e Argomento. Dentro il rettangolo viene scritta una etichetta che identifica l'entità.



Relazione: rappresenta un legame logico esistente fra due o più entità. Nel Database di esempio le entità Libro e Argomento potrebbero essere collegate tramite la relazione Classificazione (un libro viene classificato all'interno di un argomento). Nel rombo viene scritta una etichetta che identifica la relazione. Nei rami della relazione viene specificata la *cardinalità* della relazione: descrive numero minimo e massimo di istanze della relazione cui una istanza dell'entità può essere collegata.

In base alle cardinalità le relazioni possono essere di tre tipi:

- 1:1 relazione uno a uno. Ad una istanza della prima relazione può corrispondere una ed una sola istanza della seconda e viceversa. La cardinalità, a seconda se ogni istanza della prima relazione ha un corrispettivo nella seconda o può esisterne una senza, è espressa come (1,1) o (0,1) nei due rami della relazione.
- 1:n relazione uno a molti. Ad una istanza della prima relazione può corrispondere più di una istanza della seconda relazione ma non è vero il viceversa. La cardinalità è espressa come (1,n) e (1,1)
- n:m relazione molti a molti. Ad una istanza della prima relazione può corrispondere una o più istanze della seconda e viceversa. La cardinalità sarà espressa come (1,n) o come (0,n) in entrambi i rami della relazione

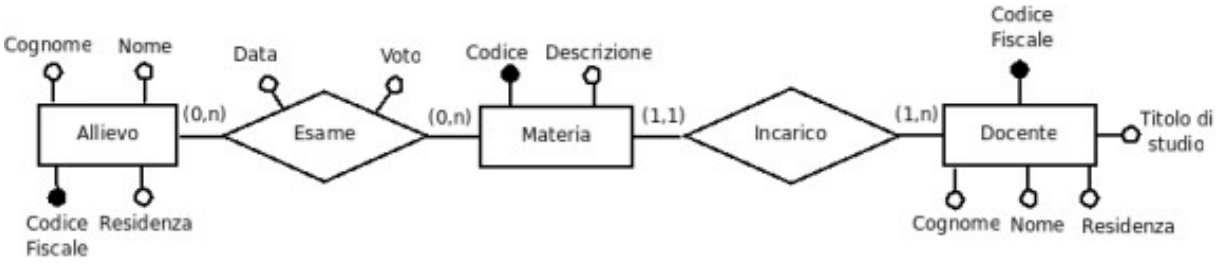
Attributo: descrive una proprietà di interesse dell'entità o della relazione. Può esistere un identificatore che è un attributo particolare che identifica in maniera univoca una istanza dell'entità. Corrisponde al concetto di chiave primaria. In questo caso il simbolo utilizzato è lo stesso ma con il cerchietto finale pieno.

Per chiarire meglio l'utilizzo del diagramma E-R si riportano due esempi. Il primo è la rappresentazione, secondo il modello E-R, del Database della biblioteca utilizzato più volte in precedenza. Il secondo esempio, più complesso, parte dalla analisi dei requisiti e costruisce il diagramma in conseguenza di questi.

Esempio 1: *Una biblioteca organizza i libri in suo possesso in base all'argomento che trattano. Si suppone, per semplicità che l'argomento sia unico per ogni libro. Del libro si vuole conservare autore, titolo, posizione nella stanza e nello scaffale. Dell'argomento si registra una descrizione ed un codice che lo identifica in modo univoco.*

Nella realtà di interesse sono identificabili le due entità Libro e Argomento con gli attributi specificati. L'attributo Codice è un identificatore per Argomento. Le due entità sono collegate dalla relazione Classificazione. La cardinalità della relazione per quanto riguarda il Libro è (1,1) perché, nell'ipotesi effettuata, se c'è un libro allora deve essere classificato in un argomento (primo valore della cardinalità) e, al massimo, può essere classificato sotto un unico argomento (il secondo valore della cardinalità). Per quanto riguarda l'argomento se si fa l'ipotesi che possano essere registrati argomenti a cui non corrisponde alcun libro allora il primo valore della cardinalità sarà 0. Un argomento può classificare più libri (secondo valore n della cardinalità).

Esempio 2: *Un Istituto di formazione vuole gestire la propria base di dati. Le informazioni da conservare prevedono dati anagrafici sugli allievi, le materie oggetto dei corsi e i docenti che hanno incarico di insegnare le materie. Gli allievi sostengono esami, di cui interessano data e voto riportato, sulle materie. Si suppone che un docente possa avere incarico di insegnare più materie ma che una materia può essere insegnata da un unico docente.*



Le entità del Database sono tre: Allievo, Materia e Docente con gli attributi espressi. Fra Allievo e Materia esiste la relazione Esame: un allievo può sostenere più esami, inoltre può essere che esistano allievi che non hanno ancora sostenuto esami in alcuna materia. Una materia può essere oggetto di più esami (secondo valore della cardinalità) ma possono esserci materie per le quali non sono stati sostenuti esami da parte di allievi. Gli attributi Data e Voto sono attributi della relazione Esami in quanto esistono a condizione che esista l'esame. Non possono essere attributi di Allievo o Materia perché, per esempio il voto, nel primo caso sarebbe attribuito all'allievo così come il cognome e, nel secondo, sarebbe una caratteristica della materia così come la descrizione. L'entità Materia ha come identificatore Codice che individua in modo univoco ogni istanza. Se esiste una materia ci dovrà essere un docente incaricato e, per le ipotesi fatte, per una stessa materia non potrà esserci più di un docente incaricato. Sempre dalla descrizione dei requisiti risulta che un docente può essere incaricato sicuramente di una materia, ma anche di più di una.

Progettazione logica

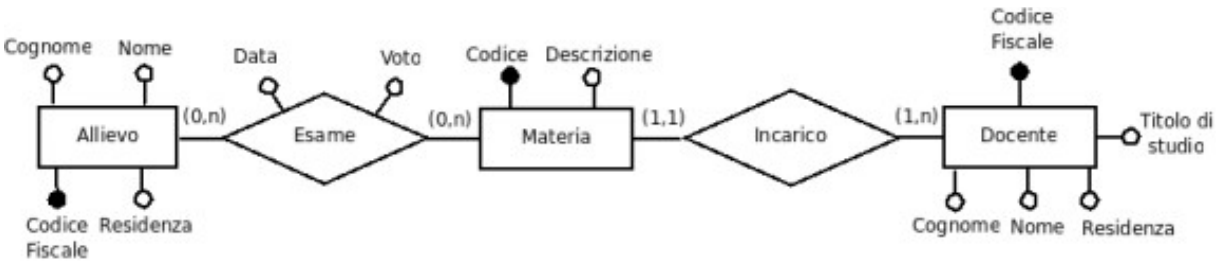
La progettazione logica consiste nella traduzione dello schema concettuale, realizzato utilizzando il modello E-R, nel modello di rappresentazione dei dati così come necessita il sistema di gestione della base di dati. Se si adotta un Database relazionale che rappresenta tutti i dati in modo tabellare, sono necessarie delle **regole di traduzione** che a partire dalle entità e corrispondenze possano permettere di costruire un insieme di tabelle *equivalenti*, soddisfacenti cioè gli stessi vincoli.

La traduzione da uno schema concettuale ad uno schema logico nel modello relazionale si può concretizzare nei seguenti passaggi:

- Ad ogni **entità** si fa corrispondere una relazione avente gli stessi attributi. Se fra gli attributi dell'entità esiste un identificatore, questo sarà una chiave primaria per la relazione. Se non esiste un identificatore, e poiché una relazione richiede necessariamente una chiave primaria, si aggiunge alla relazione un nuovo attributo che possa avere tali caratteristiche. Si può aggiungere un attributo che avrà come valore un numero progressivo che si aggiorna ad ogni istanza: in questo modo ogni riga della tabella avrà un identificatore univoco.
- Per quanto riguarda le **corrispondenze**, possono presentarsi tre casi:

1. Se fra l'entità A e l'entità B esiste una corrispondenza 1:1 con cardinalità (1,1) in entrambi i lati, la corrispondenza si traduce aggiungendo in una relazione (per esempio nella relazione che traduce A), come chiave esterna, la chiave primaria della relazione che traduce l'altra entità (B nell'esempio).
2. Se fra le entità C e D esiste una corrispondenza 1:n (nel diagramma una corrispondenza di questo tipo è evidenziata dalla presenza delle cardinalità (1,1) in un ramo e (1,n) nell'altro), la regola di traduzione comporta l'inserimento nella tabella che esprime l'entità con cardinalità (1,1) della chiave esterna corrispondente alla chiave primaria che, nell'altra relazione, individua l'unica istanza collegata. Infatti la cardinalità dice che può esserci corrispondenza con al massimo un solo elemento dell'altra relazione.
3. Se fra le entità E ed F esiste una corrispondenza n:m (nel diagramma le cardinalità sono espresse come (1,n) in entrambi i lati), si genera una nuova relazione che traduce la corrispondenza. Tale relazione avrà come attributi una chiave primaria eventualmente aggiunta (il numero progressivo delle righe della tabella), gli eventuali attributi presenti nella corrispondenza e come chiavi esterne le chiavi primarie delle due relazioni coinvolte nella corrispondenza. Una istanza di questa nuova relazione rappresenta una corrispondenza, delle n, tra una istanza della relazione E ed una della relazione F.

A titolo di esempio si applicano le regole alla traduzione in schema logico dello schema concettuale già visto della base di dati che descrive l'istituto di formazione:



in una prima approssimazione si può far corrispondere una relazione ad ogni entità. Si avranno quindi: Allievi(CodiceFiscale, Cognome, Nome, Residenza), Materie(Codice, Descrizione), Docenti(CodiceFiscale, Cognome, Nome, Residenza, TitoloDiStudio).

Tutte le relazioni hanno un identificatore e quindi è soddisfatta la condizione che, nel modello relazionale, impone a tutte le tabelle di avere definita una chiave primaria.

Per completare la traduzione dello schema mancano le corrispondenze. Per quanto riguarda Incarico basta aggiungere nella relazione Materie come chiave esterna la chiave primaria di Docenti, quella

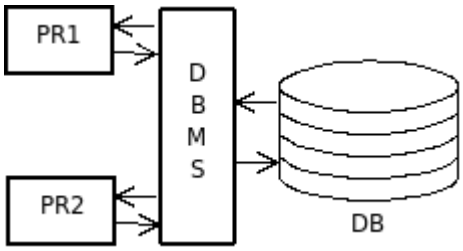
che identifica l'unico docente incaricato per la materia, e quindi sarà: Materie(Codice, Descrizione, CodiceFiscaleDocente). Nella corrispondenza Esame entrambi i rami hanno cardinalità (1,n) e quindi si genera una nuova relazione che completa lo schema logico della Base di dati:

Allievi(CodiceFiscale, Cognome, Nome, Residenza), Materie(Codice, Descrizione, CodiceFiscaleDocente), Esami(ID_Esame, Data, Voto, CodiceFiscaleAllievo, CodiceMateria), Docenti(CodiceFiscale, Cognome, Nome, Residenza, TitoloDiStudio).

Nella relazione Esami è stato aggiunto l'attributo ID_Esame che è un numero progressivo delle righe della tabella Esami. Ogni riga della tabella rappresenta un esame di un allievo in una materia.

Interazioni con i Database e SQL

Le procedure di accesso ai dati conservati in un Database, o le richieste degli utenti del Database, non hanno accesso direttamente ai dati conservati, per via della indipendenza dello schema logico dallo schema fisico, ma interagiscono, utilizzando opportuni linguaggi, con il DBMS che poi si occupa di recuperare i dati richiesti.



I linguaggi per l'interazione con il DBMS possono essere di vario tipo:

- linguaggi interattivi come SQL. Argomento che verrà trattato più avanti
- comandi espressi in linguaggi come il punto precedente ma, *embedded*, inseriti in linguaggi di programmazione tradizionali di cui rappresentano una estensione. Il linguaggio di programmazione si dice *linguaggio ospite*
- linguaggi interattivi sviluppati allo scopo con funzioni specifiche che, però, variano da sistema a sistema
- interfacce che permettono l'interazione senza avere conoscenze di linguaggi specifici. Vengono chiamate *QBE* (Query By Example). L'utente può trovarsi in una interfaccia in cui specifica il tipo di risultato che si aspetta di ottenere. Sarà il DBMS che si occuperà del resto.

I linguaggi per le basi di dati si distinguono in due categorie:

- **DDL** (*Data Definition Language*) utilizzati per la definizione degli schemi del Database
- **DML** (*Data Manipulation Language*) utilizzati per le interrogazioni e gli aggiornamenti delle istanze nella base di dati

Esistono anche linguaggi, come SQL, che integrano funzioni di DDL e DML.

Il linguaggio SQL (originariamente si chiamava SEQUEL) nasce nel 1974 nei laboratori IBM progettato da Donald Chamberlin (nella foto) e Raymond Boyce per operare con Database basati sul modello relazionale.

La caratteristica fondamentale di SQL è quella di essere un linguaggio dichiarativo e, a differenza dei linguaggi imperativi, non richiede di specificare una sequenza di operazioni da compiere ma le proprietà delle informazioni cercate. Nei linguaggi imperativi bisogna specificare *come* ottenere le informazioni richieste, in SQL si specifica *cosa* occorre ottenere.

Nel 1986 iniziò il processo di standardizzazione del linguaggio da parte dell'ANSI. Negli anni successivi sono stati effettuati diversi tentativi di standardizzazione denominate SQL/86, SQL/89, SQL/92, SQL/2003 con l'obiettivo di creare un linguaggio unico per le diverse implementazioni di DBMS, ma i diversi produttori, allo scopo di rendere disponibili proprie variazioni, si orientano, in generale, adottando lo standard ad un livello minimo (definito da ANSI *Entry Level*) e aggiungendo le proprie estensioni.

Caratteristiche generali di MySQL

MySQL è un motore DB molto diffuso in Internet grazie alle sue caratteristiche di leggerezza e velocità. Viene rilasciato con doppia licenza GPL e commerciale. Attua una architettura Client/Server: con l'installazione del software vengono resi disponibili, fra le altre cose, il server mysqld e il client mysql che permette la comunicazione con l'utente collegandosi al server per la manipolazione delle Basi di Dati. Tutti gli esempi riportati in questi appunti fanno riferimento a questo client:

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 47
Server version: 5.5.31-0+wheezy1 (Debian)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> \q
Bye
$
```

Nell'esempio ci si è collegati al server, residente nello stesso computer (*localhost*), come utente *root*. Dopo aver verificato la congruenza della password associata all'utente, il client presenta il prompt (`mysql>` è il prompt standard) in attesa di comandi. È bene precisare che si tratta dell'utente root di MySQL non dell'amministratore del sistema operativo. L'utente root di MySQL ha tutti i diritti: può creare/eliminare Database, tabelle. L'installazione di MySQL mette a disposizione anche un programma per la definizione degli utenti (con relativi diritti) che potranno accedere al Database.

In sede di installazione viene chiesto di definire la password da attribuire all'amministratore della Base di Dati (l'utente root).

Nella riga di comando che richiama il client è possibile anche specificare la password associata all'utente e il Database su cui si vuole lavorare.

```
$ mysql –u root –pmiapassword
```

nell'esempio viene richiamato il client con il nome utente `root` specificando la password `miapassword`.

```
$ mysql –u root –p istituto
```

la riga di comando richiama il client per l'utente `root` cui verrà richiesta successivamente la password, e seleziona il Database `istituto`.

Tutti i parametri possono essere utilizzati contemporaneamente:

```
$ mysql –u root –pmiapassword istituto
```

in questo modo si specificano il nome utente, la password e il Database su cui si vuole lavorare.

Il client, quando avviato, presenta il suo prompt (`mysql>`) in attesa di comandi. `\q` permette l'interruzione del client e il ritorno al prompt del sistema operativo.

Il client, come il linguaggio SQL è case-insensitive, non fa differenza fra lettere maiuscole e minuscole. I comandi si possono introdurre in qualsiasi formato:

```
mysql> SELECT USER(),VERSION(),CURRENT_DATE();
+----------------+------------------+----------------+
| USER()         | VERSION()        | CURRENT_DATE() |
+----------------+------------------+----------------+
| root@localhost | 5.5.31-0+wheezy1 | 2013-09-05     |
+----------------+------------------+----------------+
1 row in set (0.00 sec)
```

nell'esempio si chiede di visualizzare l'identificativo dell'utente connesso, la versione del server in esecuzione e la data corrente. Come specificato all'avvio del client, ogni comando finisce con il carattere ;. Il comando da eseguire può essere scritto su più righe: se si preme il tasto Invio viene visualizzata una nuova riga sulla quale si può continuare (il prompt cambia in `->`). Il carattere ; che chiude il comando fa sì che il client invii la riga completa del comando al server che recupera i dati richiesti.

Il file di configurazione di MySQL (`/etc/mysql/my.cnf`) specifica nella variabile *datadir* il luogo del filesystem, su disco, dove sono salvati i database gestiti dal server. La configurazione di default è impostata in modo che tale directory sia `/var/lib/mysql`. I database creati occupano una sotto directory di tale directory. Se per esempio viene definito il database *istituto* il server genererà la directory `/var/lib/mysql/istituto` nella quale verranno conservati tutti i dati per la gestione del database *istituto*. La directory `/var/lib/mysql` è accessibile soltanto da parte dell'utente root di MySQL.

Nel file di configurazione si può anche impostare la lingua.

```
[mysqld]
...
lc_time_name=”it_IT
...
```

con questa configurazione i giorni della settimana o i mesi dell'anno avranno il formato italiano.

L'installazione del pacchetto mysql comprende anche il programma di utilità **`mysqldump`** utilizzabile per effettuare il *dumping* (copia di backup) di un Database con tutte le tabelle, definite al proprio interno, comprensive di dati.

```
$ mysqldump –u root –p istituto > istitutobackup.sql
```

la linea di comando permette, dopo aver specificato la password dell'utente `root` di `mysql`, di generare il file di testo `istitutobackup.sql` che contiene i comandi SQL per ripristinare l'intero Database comprensivo di dati.

La riga di comando inversa:

```
$ mysql –u root –p istituto < istitutobackup.sql
```

permette di ricostruire il Database istituto dal file di backup generato con la riga di comando precedente. Il Database specificato come destinazione nella riga del comando (`istituto` nell'esempio) deve essere esistente.

Tipi di dati e DDL di SQL, comandi di utilità di MySQL

Per definire le tabelle in un Database relazionale è necessario specificare, per ogni colonna, il tipo di dato che verrà conservato. MySQL supporta diversi tipi di dati, in questa sede si esamineranno i principali, rimandando allo studio del Reference (vedi Riferimenti Bibliografici) la trattazione completa. Ogni tipo prevede, dopo la specifica, una coppia di parentesi tonde che, in genere, contengono la quantità di caratteri o cifre da conservare

- **Dati numerici**:
  - `INT` permette l'inserimento nella colonna di numeri interi. Fra parentesi viene specificata la quantità di cifre visualizzate. La quantità può anche non essere specificata se si fa riferimento alla dimensione massima consentita dal tipo
  - `DECIMAL` permette la conservazione di numeri con una parte intera e una decimale. Si usa specificando la quantità di cifre: `DECIMAL(3,2)` specifica che il numero è composto di 3 cifre di cui 2 nella parte decimale.
- **Dati di tipo testo**:
  - `CHAR` permette l'inserimento di dati di tipo testo, a lunghezza fissa, di massimo 255 caratteri. Fra parentesi si specifica la quantità di caratteri: `CHAR(20)` conserva una stringa di testo di 20 caratteri.
  - `TEXT` tipo testuale ma senza la limitazione dei 255 caratteri.
- **Dati di tipo data**:
  - `DATE` permette l'inserimento nella colonna di date di calendario. Le date sono conservate, e si inseriscono, nella forma aaaa-mm-gg. I dati definiti in questo modo permettono le operazioni tipiche fra date come il calcolo della quantità di giorni fra due date o il calcolo della data che si ottiene sommando una certa quantità di giorni ad una determinata data.
  - `YEAR` permette l'inserimento di un numero rappresentante l'anno di una data. Se si specifica 4, come in `YEAR(4)`, l'anno sarà rappresentato con 4 cifre. È possibile anche

specificare `YEAR(2)` e l'anno da 00 a 69 verrà convertito nell'intervallo 2000 2069, l'anno da 70 a 99 viene invece convertito nell'intervallo da 1970 a 1999.

Per introdurre le istruzioni SQL per la definizione dei dati si farà riferimento al Database dell'Istituto di formazione di un esempio precedente, di cui viene riportato, per comodità, lo schema logico:

Allievi(CodiceFiscale, Cognome, Nome, Residenza), Materie(Codice, Descrizione, CodiceFiscaleDocente), Esami(ID_Esame, Data, Voto, CodiceFiscaleAllievo, CodiceMateria), Docenti(CodiceFiscale, Cognome, Nome, Residenza, TitoloDiStudio).

```
mysql> CREATE DATABASE istituto;
Query OK, 1 row affected (0.02 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| istituto           |
| mysql              |
+--------------------+
3 rows in set (0.00 sec)
```

Intanto si crea il Database e si chiede di visualizzare l'elenco dei Database esistenti fra i quali si trova `istituto`.

Per popolare il Database bisogna creare le tabelle. Per esporne alcune caratteristiche si riporta l'istruzione per la creazione della tabella `Allievi`.

```
mysql> USE istituto;
Database changed
mysql> CREATE TABLE Allievi(
    -> CodiceFiscale CHAR(16) NOT NULL PRIMARY KEY,
    -> Cognome CHAR(15),
    -> Nome CHAR(15),
    -> Residenza CHAR(50));
Query OK, 0 rows affected (0.01 sec)
```

La prima istruzione introdotta rende attivo il Database su cui si vuole operare. Le successive istruzioni faranno riferimento a questo.

`CREATE TABLE` crea la struttura di una tabella. Come evidenziato nello schema logico, il primo campo è la chiave primaria. La clausola `NOT NULL` non consente di inserire righe nella tabella che non abbiano un valore in questo campo. Essendo questo infatti la chiave primaria è necessario che il valore ci sia sempre. In genere, al di là della chiave primaria, la clausola va specificata tutte le volte che non si vuole permettere di lasciare vuoto l'inserimento nel campo. In questo caso la clausola `NOT NULL` è ridondante: potrebbe mancare poiché il campo è definito come chiave primaria e, come tale, non può essere lasciato senza valore.

L'istruzione è scritta, per ragioni di comprensibilità, su più linee. Il client mysql finché non viene chiuso il comando con il carattere ; presenta il prompt. Il prompt è diverso da quello mostrato quando si inseriscono i comandi di una sola linea, per evidenziare il fatto che la linea è una continuazione. Se c'è un errore nell'esecuzione della query si può riprendere una immissione con il tasto *Freccia Su*, che scorre le linee introdotte in precedenza, e consente la correzione della linea.

I nomi scelti per i campi soddisfano le stesse regole dei nomi delle variabili nei linguaggi di programmazione (non si usano spazi, caratteri speciali, segni di punteggiatura ...). I nomi del database, delle tabelle, dei campi sono case-sensitive: si fa differenza fra lettere maiuscole e minuscole.

```
mysql> DESCRIBE Allievi;
+---------------+----------+------+-----+---------+-------+
| Field         | Type     | Null | Key | Default | Extra |
+---------------+----------+------+-----+---------+-------+
| CodiceFiscale | char(16) | NO   | PRI | NULL    |       |
| Cognome       | char(15) | YES  |     | NULL    |       |
| Nome          | char(15) | YES  |     | NULL    |       |
| Residenza     | char(50) | YES  |     | NULL    |       |
+---------------+----------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

`DESCRIBE` mostra le caratteristiche della struttura della tabella.

Le colonne mostrano, nell'ordine, il nome dei campi, il tipo di dati conservato nel campo, se il campo accetta valori nulli (nell'esempio `CodiceFiscale` no, tutti gli altri possono essere vuoti. Una riga della tabella può avere valori solo nel primo campo), se il campo è una chiave e, nell'ultima colonna, se sul campo sono definite opzioni extra come nell'esempio successivo relativo alla definizione della tabella Esami(ID_Esame, Data, Voto, CodiceFiscaleAllievo, CodiceMateria):

```
mysql> CREATE TABLE Esami(
    -> ID_Esame INT PRIMARY KEY AUTO_INCREMENT,
    -> Data DATE,
    -> Voto INT(3),
    -> CodFiscAll CHAR(16)NOT NULL,
    -> CodMat CHAR(5)NOT NULL,
    -> FOREIGN KEY (CodFiscAll) REFERENCES Allievi(CodiceFiscale),
    -> FOREIGN KEY (CodMat) REFERENCES Materie(Codice));
Query OK, 0 rows affected (0.01 sec)

mysql> DESCRIBE Esami;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| ID_Esame   | int(11)  | NO   | PRI | NULL    | auto_increment |
| Data       | date     | YES  |     | NULL    |                |
| Voto       | int(3)   | YES  |     | NULL    |                |
| CodFiscAll | char(16) | NO   | MUL | NULL    |                |
| CodMat     | char(5)  | NO   | MUL | NULL    |                |
+------------+----------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Per la tabella il campo `ID_Esame` è una chiave primaria che è stata aggiunta agli altri campi presenti nello schema logico. Sul campo è definita l'opzione `AUTO_INCREMENT` che fa in modo che il server aggiunga in automatico un numero progressivo ad ogni nuova riga. Il tipo `INT` quando è riferito ad un campo `AUTO_INCREMENT` non specifica la quantità di cifre (viene assunta la dimensione massima consentita dal tipo), che invece è specificata in `Voto`. Qui si suppone che il voto non possa avere più di 3 cifre. Nella descrizione del campo l'opzione è evidenziata nella colonna Extra.

I campi `CodFiscAll` e `CodMat` sono chiavi esterne che fanno riferimento, rispettivamente ai campi `CodiceFiscale` di `Allievi` e `Codice` della tabella `Materia`. I due campi non possono essere lasciati vuoti perché rappresentano il collegamento con altre tabelle. Il DBMS verifica, nel caso di

chiavi esterne, l'esistenza delle tabelle e dei campi cui si fa riferimento (`Allievi` e `Materie` nell'esempio) e non permette di generare la tabella `Esami` se prima non sono state generate le altre due cui ci si riferisce nelle chiavi esterne.

Si tralascia la definizione delle tabelle *Materie* e *Docenti* che hanno caratteristiche già discusse.

Nella tabella seguente sono riportate le istruzioni per il DDL di SQL:

Istruzione	***Significato e Uso***
`CREATE DATABASE nome`	Crea il Database con il nome specificato
`DROP DATABASE nome`	Elimina il Database con il nome specificato. Tutte le tabelle contenute nel Database vengono eliminate.
`CREATE TABLE ...`	Crea la struttura di una nuova tabella
`DROP TABLE nome`	Elimina la tabella con tutti i dati in essa contenuti
`ALTER TABLE nome ...`	Modifica la struttura di una tabella: `ALTER TABLE Allievi ADD Telefono CHAR(15);` aggiunge alla tabella la colonna con le caratteristiche specificate. `ALTER TABLE Allievi MODIFY Residenza CHAR(80);` modifica le caratteristiche di una colonna della tabella. `ALTER TABLE Allievi DROP Telefono;` elimina la colonna. `ALTER TABLE Allievi CHANGE Residenza Recapito CHAR(50);` cambia il nome di un campo. `ALTER TABLE Allievi RENAME Alunni;` cambia il nome alla tabella

Già in precedenza sono stati riportati negli esempi alcuni comandi del client mysql che, per esempio, restituiscono informazioni su una tabella. La tabella seguente riporta quelli che potrebbero essere utili:

Comando	***Significato e Uso***
`USE nome;`	Rende attivo il Database specificato. Le istruzioni successive avranno per oggetto tale Database.
`SELECT DATABASE();`	Mostra il Database attivo.
`SHOW DATABASES;`	Mostra tutti i Database gestiti dal server.
`SHOW TABLES;`	Mostra le tabelle definite nel Database in uso.
`DESCRIBE nome;`	Mostra la struttura della tabella specificata.

Il DML di SQL

Oltre alla `SELECT`, istruzione per le interrogazioni su un Database trattata in dettaglio successivamente, il DML di SQL mette a disposizione per la manipolazione di dati le istruzioni della tabella seguente:

Istruzione	***Significato e Uso***
`INSERT INTO tabella (lista campi) VALUES (lista valori)`	Consente di aggiungere una o più righe ad una tabella.

Istruzione	*Significato e Uso*

`REPLACE INTO tabella (lista campi) VALUES (valori)`	Sostituisce una riga della tabella con una riga contenete i valori specificati. Poiché non possono esistere due righe con lo stesso valore nella chiave primaria, la `REPLACE` in sostanza cancella la riga con quella chiave e la sostituisce con una con i valori specificati
	`REPLACE INTO Libri(Codice, Autore, Titolo)`
	`VALUES (“L02”,”Autori Vari”,”MySQL 5”)`
	la query rimpiazza la riga con chiave primaria L02 con la riga i cui valori sono contenuti nella clausola `VALUES`.

`UPDATE tabella SET colonna=valore, ... WHERE condizioni`	Aggiorna valori contenuti in una o più righe.
	Si possono specificare più colonne di cui si vuole modificare il valore, così come è possibile specificare le condizioni che devono soddisfare le righe affinché le colonne specificate siano modificate
	`UPDATE Libri`
	`SET Titolo=”MySQL guida completa”`
	`WHERE Codice=”L02”`
	il valore contenuto nella colonna Titolo della riga con chiave primaria L02 viene modificato con il valore contenuto nella query

`DELETE FROM tabella WHERE condizione`	Cancella dalla tabella le righe che soddisfano la condizione specificata nella clausola `WHERE`

Per popolare di dati una tabella l'istruzione SQL è `INSERT`:

```
mysql> INSERT INTO Esami(Data,Voto,CodFiscAll,Codmat)
    -> VALUES ("2013-09-02",65,"ABCDEF12G34G456H","M0001");
Query OK, 1 row affected (0.00 sec)
```

Nella `INSERT` si specifica la tabella dove si vuole inserire la riga. Fra parentesi va specificato l'elenco dei campi cui si vuole dare un valore. Il riempimento della chiave primaria `ID_Esame` definita con l'opzione `AUTO_INCREMENT` è lasciato in automatico al motore di Database: si potrebbe anche inserire un valore numerico purché non esistente fra quelli già registrati in precedenza. I valori delle rispettive colonne vanno inseriti fra le parentesi che seguono `VALUES`. I valori di tipo testo, comprese le date, vanno inseriti racchiusi fra doppi apici, i valori numerici vanno inseriti così come sono.

Per inserire una riga con il campo Data che contiene la data odierna può essere utilizzata la funzione `CURRENT_DATE()` esaminata in precedenza:

```
mysql> INSERT INTO Esami (Data,Voto,CodFiscAll,CodMat)
    -> VALUES (CURRENT_DATE(),64,"ILMNOP12Q34R567S","M0002");
Query OK, 1 row affected (0.02 sec)
```

Con un unica `INSERT` si possono inserire più righe nella tabella:

```
mysql> INSERT INTO Esami (Data,Voto,CodFiscAll,CodMat)
    -> VALUES (CURRENT_DATE(),66,"TUVZZA12B34B567C","M0002"),
    -> (CURRENT_DATE(),66,"DEFGHJ99K88X678Y",”M0002”);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

Nelle istruzioni che prevedono la possibilità di essere applicate a più righe di una tabella (`UPDATE` o

`DELETE` con la clausola `WHERE`) si può limitare l'effetto dell'azione ad un certo numero di righe specificando la clausola `LIMIT` seguita dal numero di righe.

Integrità referenziale

Un problema di cui si occupa un DBMS è quello della correttezza dei dati. Le relazioni fra i dati impongono dei limiti inerenti operazioni come inserimento, modifica o cancellazione.

Nel Database della libreria con Libri e Argomenti nella tabella Libri dovrà essere inserita una colonna per la chiave esterna che dovrà avere un valore presente nella colonna della chiave primaria di Argomenti: un libro può trattare un argomento fra quelli registrati. Una modifica, per esempio, del codice dell'argomento dovrebbe comportare la corrispondente modifica del codice in una eventuale riga della tabella Libri. Identica osservazione vale per ogni modifica dei dati che vengono utilizzati per i collegamenti. Questa è la proprietà nota come **integrità referenziale**.

```
mysql> CREATE DATABASE Libreria;
Query OK, 1 row affected (0.01 sec)

mysql> USE Libreria;
Database changed
mysql> CREATE TABLE Argomenti(
    -> Codice CHAR(5) PRIMARY KEY,
    -> Descrizione CHAR(25));
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE Libri(
    -> Codice CHAR(5) PRIMARY KEY,
    -> Autore CHAR(20),
    -> Titolo CHAR(50),
    -> Stanza CHAR(5),
    -> Scaffale CHAR(5),
    -> CodArg CHAR(5) NOT NULL,
    -> FOREIGN KEY (CodArg) REFERENCES Argomenti(Codice)
    -> ON UPDATE CASCADE);
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW TABLES;
+--------------------+
| Tables_in_Libreria |
+--------------------+
| Argomenti          |
| Libri              |
+--------------------+
2 rows in set (0.00 sec)
```

Dopo aver creato il Database vengono create le tabelle in accordo con quanto esposto in precedenza. Nella definizione della tabella Libri è presente la clausola `ON UPDATE CASCADE`. Questa clausola garantisce, da parte del server, che una modifica del dato conservato in Codice di

Argomenti comporta, *a cascata*, la modifica dello stesso dato inserito in qualche riga di Libri. CodArg è un vincolo di integrità referenziale (***constraint***).

```
mysql> INSERT INTO Argomenti VALUES
    -> ("A01","Letteratura Straniera"),
    -> ("A02","Informatica");
Query OK, 2 rows affected, 1 warning (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 1

mysql> SELECT * FROM Argomenti;
+--------+----------------------+
| Codice | Descrizione          |
+--------+----------------------+
| A01    | Letteratura Straniera|
| A02    | Informatica          |
+--------+----------------------+
2 rows in set (0.00 sec)
```

Nel codice si è usata `INSERT` per effettuare l'inserimento di due righe nella tabella. Non si sono specificate le colonne perché l'inserimento le riguarda tutte.

L'istruzione `SELECT` della seconda query verrà trattata in maniera estesa nei prossimi paragrafi. Per il momento basta sapere che, utilizzata nel modo descritto, permette di visualizzare il contenuto di una tabella.

```
mysql> INSERT INTO Libri VALUES
    -> ("L01","George Orwell","1984","S01","SC01","A01");
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Libri VALUES
    -> ("L02","Autori Vari","MySQL Guida Completa","S02","SC01","A05");
ERROR 1452 (23000): Cannot add or update a child row: a foreign key constraint
fails (`Libreria/Libri`, CONSTRAINT `Libri_ibfk_1` FOREIGN KEY (`CodArg`)
REFERENCES `Argomenti` (`Codice`) ON UPDATE CASCADE)

mysql> INSERT INTO Libri VALUES
    -> ("L02","Autori Vari","MySQL Guida Completa","S02","SC01","A02");
Query OK, 1 row affected (0.00 sec)
```

Con la prima `INSERT` viene inserita una riga nella tabella Libri. Anche la seconda `INSERT` tenta di inserire una riga nella tabella ma, poiché si tenta di inserire nella colonna CodArg un dato incongruente (A05 non esiste come valore nella colonna Codice di Argomenti), il server restituisce un codice di errore e non inserisce la riga nella tabella. Se invece si immette un codice valido (A02) la riga è inserita con successo.

```
mysql> SELECT * FROM Libri;
+--------+--------------+----------------------+--------+----------+--------+
| Codice | Autore       | Titolo               | Stanza | Scaffale | CodArg |
+--------+--------------+----------------------+--------+----------+--------+
| L01    | George Orwell | 1984                | S01    | SC01     | A01    |
| L02    | Autori Vari   | MySQL Guida Completa | S02    | SC01     | A02    |
+--------+--------------+----------------------+--------+----------+--------+
2 rows in set (0.00 sec)
```

Se si modifica la chiave primaria di un argomento, vengono modificati, di conseguenza, anche i valori conservati nelle chiavi esterne che fanno riferimento ad essa (in accordo alla `ON UPDATE` …

della definizione):

```
mysql> UPDATE Argomenti
    -> SET Codice="A10"
    -> WHERE Descrizione="Informatica";
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Argomenti;
+--------+-----------------------+
| Codice | Descrizione           |
+--------+-----------------------+
| A01    | Letteratura Straniera |
| A10    | Informatica           |
+--------+-----------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM Libri;
+--------+---------------+----------------------+--------+----------+--------+
| Codice | Autore        | Titolo               | Stanza | Scaffale | CodArg |
+--------+---------------+----------------------+--------+----------+--------+
| L01    | George Orwell | 1984                 | S01    | SC01     | A01    |
| L02    | Autori Vari   | MySQL Guida Completa | S02    | SC01     | A10    |
+--------+---------------+----------------------+--------+----------+--------+
2 rows in set (0.00 sec)
```

Come ci si attendeva modificando il codice dell'argomento, la chiave esterna presente nelle righe dei libri è stata aggiornata di conseguenza.

Le opzioni per l'integrità referenziale possono essere:

- `ON UPDATE CASCADE`: se viene modificato il record padre, si garantisce la conseguente modifica dei record figli
- `ON DELETE CASCADE`: se viene cancellato un record padre vengono cancellati anche i record figli che fanno riferimento ad esso. Per esempio cancellando la Descrizione Informatica, cui corrisponde la chiave A10, vengono cancellati tutti i libri che hanno nella colonna CodArg lo stesso valore.
- `ON DELETE SET NULL`: cancellando una riga da Argomenti verrà inserito il valore NULL (stringa vuota) nel campo CodArg dei libri, con il codice dell'argomento eliminato.

Introduzione all'istruzione *SELECT*

L'istruzione `SELECT` permette la ricerca dei dati all'interno di un Database. È l'istruzione più nota e rappresentativa di SQL. Nella sua forma più elementare l'istruzione assume la forma:

```
SELECT lista campi
  FROM lista tabelle
  WHERE condizioni
```

della lista fanno parte i nomi dei campi o delle tabelle separati da virgole. Nelle condizioni si possono usare, in una prima approssimazione, gli operatori di condizione (<, >, <=, >=, =) e gli operatori booleani (AND, OR, NOT) per connettere più condizioni.

Per mostrare l'uso di base dell'istruzione si riprendono gli esempi svolti in precedenza sul Database Biblioteca.

Nel Database sono definite due tabelle: Argomenti (Codice, Descrizione), Libri (Codice, Autore, Titolo, Stanza, Scaffale, Argomento). Il campo Argomento di Libri è una chiave esterna che fa riferimento alla chiave primaria (Codice) di Argomenti.

Esempio 1: *Elencare autore e titolo di tutti i libri della stanza S01*

$$\pi_{Autore,Titolo}(\sigma_{Stanza=”S01”}(\mathbf{Libri}))$$

la query in SQL sarà:

```
SELECT Autore, Titolo
  FROM Libri
  WHERE Stanza=”S01”
```

I campi oggetto della proiezione vengono elencati immediatamente dopo la `SELECT`. Nella clausola `FROM` è indicata la tabella da dove prendere i dati che formeranno la nuova tabella oggetto della query. Nella `WHERE` è specificata la condizione di selezione.

Esempio 2: *Elencare titolo, stanza, scaffale e argomento di tutti i libri che ha scritto George Orwell*

$$\pi_{Titolo,Stanza,Scaffale,Descrizione}(\sigma_{Autore=”George\ Orwell”}(\mathbf{Libri} \bowtie_{Argomento=Codice} \mathbf{Argomenti}))$$

in SQL:

```
SELECT Libri.Titolo, Libri.Stanza, Libri.Scaffale, Argomenti.Descrizione
  FROM Libri, Argomenti
  WHERE Libri.Argomento = Argomenti.Codice AND
        Libri.Autore = “George Orwell”
```

In questa query si referenziano i campi con il nome completo che comprende anche la tabella cui appartengono. Nella `WHERE` è espressa la condizione del Join e la selezione sull'Autore.

La ricerca, in base alle condizioni specificate nella `WHERE`, viene effettuata prescindendo dall'uso di caratteri minuscoli e maiuscoli: in ogni caso se c'è corrispondenza la riga farà parte della selezione. Solamente nei nomi di tabelle o campi si fa distinzione fra maiuscole e minuscole: è necessario rispettare le stesse regole utilizzate nel momento della generazione delle tabelle.

Con la `SELECT` è possibile ordinare l'output secondo una o più colonne o limitare la visualizzazione della tabella risultato nel numero di righe:

```
SELECT Autore, Titolo
  FROM Libri
  WHERE Stanza=”S01”
  ORDER BY Autore
  LIMIT 10
```

La query mostra i primi 10 libri, in ordine alfabetico per autore, presenti nella stanza S01.

Nella `ORDER BY` si possono specificare più colonne separate da virgole e, in questi casi, l'ordinamento seguirà l'ordine delle colonne: prima si ordina in base alla prima colonna specificata, poi, a parità di valore, le righe sono ordinate in accordo con la seconda colonna ecc...

A volte può essere conveniente effettuare un ordinamento inverso:

```
SELECT *
  FROM Libri
  WHERE Autore=”George Orwell”
```

```
  ORDER BY Titolo DESC
```

La clausola `DESC` elenca le righe della tabella in ordine alfabetico inverso (dalla Z alla A se si tratta di campo alfanumerico, dal numero più grande al numero più piccolo se invece il campo è di tipo numerico).

Se come risultato della query si vogliono tutte le colonne della tabella risultato si può usare *:

```
SELECT *
  FROM Libri
  WHERE Autore=”George Orwell”
```

Produce una tabella con tutte le colonne della tabella Libri e le righe che soddisfano la condizione.

È possibile cercare dati anche se non si conosce una corrispondenza esatta:

```
mysql> SELECT *
    -> FROM Libri
    -> WHERE Titolo LIKE "%Guida%";
+--------+-------------+----------------------+--------+----------+--------+
| Codice | Autore      | Titolo               | Stanza | Scaffale | CodArg |
+--------+-------------+----------------------+--------+----------+--------+
| L02    | Autori Vari | MySQL Guida Completa | S02    | SC01     | A10    |
+--------+-------------+----------------------+--------+----------+--------+
1 row in set (0.00 sec)
```

La query cerca i libri che hanno nel titolo la parola Guida.

L'operatore `LIKE` permette di specificare una maschera cui deve soddisfare il campo affinché la riga sia di interesse. Il carattere `%` presente nella maschera indica una quantità qualsiasi di caratteri. La maschera specifica che la parola Guida può essere preceduta e seguita da un numero qualsiasi di caratteri.

Nella maschera si può anche specificare una quantità esatta di caratteri:

```
mysql> SELECT *
    -> FROM Libri
    -> WHERE Titolo LIKE "_y%";
+--------+-------------+----------------------+--------+----------+--------+
| Codice | Autore      | Titolo               | Stanza | Scaffale | CodArg |
+--------+-------------+----------------------+--------+----------+--------+
| L02    | Autori Vari | MySQL Guida Completa | S02    | SC01     | A10    |
+--------+-------------+----------------------+--------+----------+--------+
1 row in set (0.00 sec)
```

Nell'esempio si chiede l'elenco dei libri che hanno un titolo la cui seconda lettera è una y seguita da un numero qualsiasi di caratteri. Il carattere _ è il sostituto di un singolo carattere qualsiasi: se si vogliono specificare più caratteri è necessario inserire più ricorrenze.

```
SELECT DISTINCT Autore
  FROM Libri
```

La clausola `DISTINCT` permette, nel caso di duplicati, la visualizzazione di una sola copia dei dati richiesti. La query dell'esempio costruisce una tabella con i nomi di tutti gli autori che hanno scritto libri presenti nella biblioteca. Anche se nella biblioteca sono presenti più libri scritti dallo stesso autore, il nome figurerà una sola volta.

Fra le condizioni di una `SELECT` è possibile utilizzare inoltre due clausole che permettono la scrittura semplificata di alcune query. Per mostrarne l'utilizzo si fa riferimento al database dell'istituto di formazione:

- `BETWEEN...AND` ricerca tutte le tuple che contengono nella colonna un valore fra i limiti specificati:

```
SELECT Allievi.Cognome, Allievi.Nome, Esami.Voto
  FROM Allievi, Esami
  WHERE Allievi.CodiceFiscale = Esami.CodFiscAll AND
        Esami.Voto BETWEEN 60 AND 65
```

  la query elenca i dati degli allievi che hanno ottenuto un voto compreso tra 60 e 65 negli esami registrati. La clausola prevede anche l'operatore logico `NOT` (`NOT BETWEEN...`) per negare l'appartenenza all'intervallo specificato.

- `IN` specifica un sottoinsieme di ricerca:

```
SELECT *
  FROM Allievi
  WHERE Allievi.Nome IN (“Aldo”, “Giovanni”, “Mario”)
```

  la query elenca gli allievi il cui nome rientra nel sottoinsieme specificato. Anche in questo caso è possibile utilizzare l'operatore logico `NOT` (`NOT IN...`).

La `SELECT` permette di generare tabelle con colonne che presentino risultati di operazioni effettuate su colonne delle tabelle del Database, presenti nella clausola `FROM`, o risultati di operazioni qualsiasi:

```
SELECT CodFiscAll, CodMat, Voto*10/100 AS “Voto in decimi”
  FROM Esami
```

Come risultato della query si vuole una tabella dove, oltre al codice fiscale dell'allievo e al codice della materia, venga riportata una colonna con il voto in decimi. Si suppone che la tabella Esami riporti il voto in centesimi. La clausola `AS` che può essere utilizzata per qualunque colonna, nell'esempio permette di cambiare il nome della colonna calcolata del risultato della query. Qualora la clausola non fosse specificata il nome della colonna riporterebbe come intestazione la formula (poco comprensibile) specificata nella `SELECT`. Se nella nuova intestazione di colonna si vogliono inserire caratteri particolari, come lo spazio, è necessario racchiudere la stringa fra doppi apici altrimenti si possono omettere.

Elaborazioni su campi di tipo *data*

MySQL rende disponibili una serie di funzioni che facilitano le operazioni con campi definiti di tipo data, facilitazioni particolarmente utili in query applicate in ambiti gestionali dove è frequente avere interesse, per esempio, a conoscere il giorno della settimana in cui è stata effettuata una determinata vendita ecc …

Funzione	***Significato***
`DATE_ADD()`	Permette di ottenere una nuova data da una precedente aggiungendo o togliendo giorni. In parantesi si specifica la data su cui effettuare il calcolo e, separato da una virgola, `INTERVAL` seguito da un numero che, se positivo indica i giorni da aggiungere, se negativo quelli da togliere.

Funzione	*Significato*
`DATEDIFF()`	Calcola la quantità di giorni fra due date. In parentesi vanno specificate le due date: prima quella più recente e subito dopo, separata da una virgola, quella più vecchia.
`DATE_FORMAT()`	Permette di ottenere un formato diverso per la data specificata. In parentesi vanno specificate, separate da una virgola, la data e la maschera, racchiusa fra apici, all'interno della quale si possono usare: `%d` per visualizzare i giorni `%m %M` la prima combinazione visualizza il mese come numero, la seconda come stringa `%y %Y` la prima visualizza l'anno della data utilizzando 2 cifre, la seconda utilizzandone 4.

`YEAR()`
`MONTH()`
`DAY()`
	Permettono di estrarre da una data, specificata in parentesi, rispettivamente, l'anno, il mese e il giorno espressi in forma numerica.

`MONTHNAME()` `DAYNAME()`	Permettono l'estrazione, da una data specificata in parentesi, del mese e del giorno come stringhe con il nome.

Di seguito alcuni esempi di applicazione delle funzioni, per questioni di semplicità, applicate alla data odierna.

```
mysql> SELECT CURRENT_DATE();
+----------------+
| CURRENT_DATE() |
+----------------+
| 2011-10-11     |
+----------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT(CURRENT_DATE(),'%d-%M-%Y');
+---------------------------------------+
| DATE_FORMAT(CURRENT_DATE(),'%d-%M-%Y') |
+---------------------------------------+
| 11-ottobre-2011                       |
+---------------------------------------+
1 row in set (0.00 sec)
```

Dopo aver richiesto la data odierna così come come fornita di default, nella seconda query si applica una maschera per una visualizzazione più vicina alla comune modalità italiana.

```
mysql> SELECT DAYNAME(CURRENT_DATE()) AS "Giorno della settimana",
    -> MONTHNAME(CURRENT_DATE()) AS "Mese";
+------------------------+---------+
| Giorno della settimana | Mese    |
+------------------------+---------+
| martedì                | ottobre |
+------------------------+---------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME(DATE_ADD(CURRENT_DATE(),INTERVAL 1 DAY)) AS "Domani",
    -> DAYNAME(DATE_ADD(CURRENT_DATE(),INTERVAL 5 DAY)) AS "Fra 5 giorni";
+------------+--------------+
| Domani     | Fra 5 giorni |
+------------+--------------+
```

```
| mercoledì  | domenica     |
+------------+--------------+
1 row in set (0.00 sec)
```

Per concludere il paragrafo si propone una query sulla tabella degli esami inclusa nel Database istituto:

```
SELECT *
  FROM Esami
  WHERE MONTHNAME(Data)="ottobre" AND
  YEAR(Data)=YEAR(CURRENT_DATE())
```

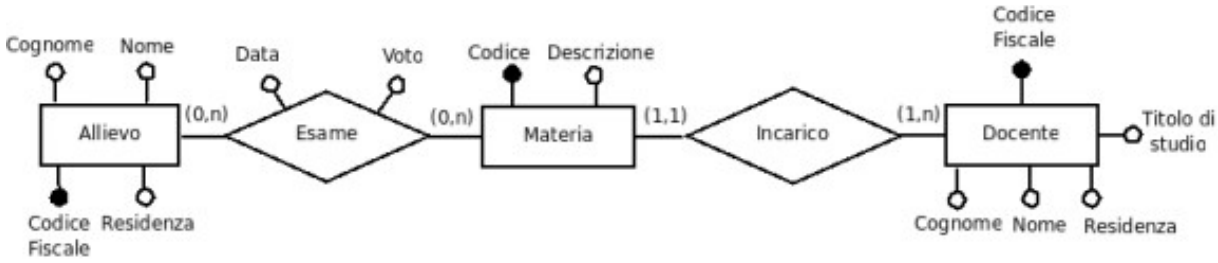
La query richiede l'elenco degli esami che si sono svolti nel mese di ottobre dell'anno corrente.

Funzioni di aggregazione

Sotto il nome di funzioni di aggregazione si raggruppano un insieme di funzioni, inseribili nella `SELECT` di una query, che si occupano di fornire informazioni sommarie sulle righe della tabella: restituiscono un risultato calcolato sui valori oggetto della funzione.

Funzione	***Significato***
`COUNT()`	Calcola la quantità di righe di una tabella. Se fra le parentesi si specifica il nome di una colonna, la funzione restituisce la quantità di righe che hanno valore non nullo nella colonna specificata, se si specifica il carattere * calcola la quantità totale di righe della tabella.
`SUM() AVG()`	Agiscono su campi numerici. Calcolano, rispettivamente, la somma e la media aritmetica dei valori contenuti nella colonna specificata fra le parentesi.
`MIN() MAX()`	Si applicano a colonne qualsiasi. Restituiscono, rispettivamente, il valore minimo e il valore massimo contenuti nella colonna specificata fra le parentesi. Se la colonna è di tipo carattere valore minimo e massimo vanno intesi come valore alfabetico.

Si riportano di seguito alcuni esempi di applicazione delle funzioni riferentesi al Database dell'Istituto di formazione:




```
mysql> USE istituto;
Database changed
mysql>  SELECT * FROM Allievi;
+------------------+---------+---------+-----------+
| CodiceFiscale    | Cognome | Nome    | Residenza |
+------------------+---------+---------+-----------+
| ABCDEF12G34G456H | Aldi    | Aldo    | NULL      |
| BCDEFG23H45H567I | Carli   | Carlo   | NULL      |
| CDEFGH34I56I678L | Bruni   | Bruno   | NULL      |
| DEFGHI45L67L789M | Danieli | Daniele | NULL      |
```

```
| EFGHIJ12K34L567N | Enei     | Enea     | NULL      |
+------------------+----------+----------+-----------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM Esami;
+----------+------------+------+------------------+---------+
| ID_Esame | Data       | Voto | CodFiscAll       | CodMat  |
+----------+------------+------+------------------+---------+
|        1 | 2013-09-02 |   65 | ABCDEF12G34G456H | M0001   |
|        2 | 2013-09-05 |   63 | ABCDEF12G34G456H | M0002   |
|        3 | 2013-09-05 |   61 | BCDEFG23H45H567I | M0001   |
|        4 | 2013-09-05 |   60 | DEFGHI45L67L789M | M0001   |
|        5 | 2013-09-05 |   62 | CDEFGH34I56I678L | M0001   |
|        6 | 2013-09-05 |   61 | BCDEFG23H45H567I | M0002   |
+----------+------------+------+------------------+---------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM Materie;
+--------+----------------------+------------------+
| Codice | Descrizione          | CodFiscDoc       |
+--------+----------------------+------------------+
| M0001  | Informatica Generale | XYZXYZ99M11X111Y |
| M0002  | Basi di Dati         | ZXYZXY88H22S222X |
+--------+----------------------+------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM Docenti;
+------------------+-------------+----------+-----------+-----------+
| CodFisc          | Cognome     | Nome     | Residenza | TitStudio |
+------------------+-------------+----------+-----------+-----------+
| XYZXYZ99M11X111Y | De Giovanni | Giovanni | NULL      | NULL      |
| ZXYZXY88H22S222X | Luci        | Lucio    | NULL      | NULL      |
+------------------+-------------+----------+-----------+-----------+
2 rows in set (0.00 sec)
```

Esempio 1: *La quantità degli Allievi*

```
mysql> SELECT COUNT(*) AS Quantita FROM Allievi;
+----------+
| Quantita |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

Esempio 2: *La quantità di esami sostenuti il 2013-09-05*

```
 mysql> SELECT COUNT(*) AS Quantita
     -> FROM Esami
     -> WHERE Esami.Data="2013-09-05";
+----------+
| Quantita |
+----------+
|        5 |
+----------+
1 row in set (0.00 sec)
```

Esempio 3: *La media dei voti ottenuti negli esami dall'allievo Aldi Aldo*

```
mysql> SELECT AVG(Voto) AS “Media Voti”
    -> FROM Allievi,Esami
    -> WHERE Allievi.CodiceFiscale=Esami.CodFiscAll AND
    -> Allievi.Cognome="Aldi" AND
    -> Allievi.Nome="Aldo";
+------------+
| Media Voti |
+------------+
|    64.0000 |
+------------+
1 row in set (0.00 sec)
```

Esempio 4: *Il voto minimo e il voto massimo ottenuti dagli allievi negli esami di Informatica Generale*

```
mysql> SELECT MIN(Voto) AS "Voto Minimo", MAX(Voto) AS "Voto Massimo"
    -> FROM Esami,Materie
    -> WHERE Esami.CodMat=Materie.Codice AND
    -> Materie.Descrizione="Informatica Generale";
+-------------+--------------+
| Voto Minimo | Voto Massimo |
+-------------+--------------+
|          60 |           65 |
+-------------+--------------+
1 row in set (0.01 sec)
```

Raggruppamenti

Le funzioni di aggregazione associate ad una `SELECT` effettuano le proprie elaborazioni su tutte le righe della tabella risultato. È possibile tuttavia, utilizzando la clausola `GROUP BY`, fare in modo che le funzioni si applichino a sottoinsiemi definiti nella tabella risultato. Dal punto di vista matematico la clausola effettua un partizionamento dell'insieme cui si applica.

Esempio 1: *Le quantità, per data, di tutti gli esami sostenuti*

```
mysql> SELECT COUNT(*) AS "Quantita' Esami", Data
    -> FROM Esami
    -> GROUP BY Data;
+-----------------+------------+
| Quantita' Esami | Data       |
+-----------------+------------+
|               1 | 2013-09-02 |
|               5 | 2013-09-05 |
+-----------------+------------+
2 rows in set (0.00 sec)
```

Effettuando un raggruppamento per data, la funzione `COUNT` ha effetto su ogni valore diverso della data.

Esempio 2: *La quantità di esami effettuati da ciascun insegnante con la media dei voti assegnati agli allievi*

```
mysql> SELECT COUNT(*) AS "Quantita' Esami",AVG(Voto) AS "Media Voti",
    -> Docenti.Cognome,Docenti.Nome
    -> FROM Esami,Materie,Docenti
    -> WHERE Esami.CodMat=Materie.Codice AND
    -> Materie.CodFiscDoc=Docenti.CodFisc
```

```
    -> GROUP BY Docenti.CodFisc;
+-----------------+------------+-------------+----------+
| Quantita' Esami | Media Voti | Cognome     | Nome     |
+-----------------+------------+-------------+----------+
|               4 |    62.0000 | De Giovanni | Giovanni |
|               2 |    62.0000 | Luci        | Lucio    |
+-----------------+------------+-------------+----------+
2 rows in set (0.00 sec)
```

Esempio 3: *Voto minimo, voto massimo e quantità esami sostenuti da ciascun allievo*

```
mysql> SELECT COUT(*) AS "Quantita' Esami", MIN(Voto) AS "Voto Minimo",
    -> MAX(Voto) AS "Voto Massimo", Allievi.Cognome, Allievi.Nome
    -> FROM Allievi,Esami
    -> WHERE Allievi.CodiceFiscale=Esami.CodFiscAll
    -> GROUP BY Allievi.CodiceFiscale;
+-----------------+-------------+--------------+---------+---------+
| Quantita' Esami | Voto Minimo | Voto Massimo | Cognome | Nome    |
+-----------------+-------------+--------------+---------+---------+
|               2 |          63 |           65 | Aldi    | Aldo    |
|               2 |          61 |           61 | Carli   | Carlo   |
|               1 |          62 |           62 | Bruni   | Bruno   |
|               1 |          60 |           60 | Danieli | Daniele |
+-----------------+-------------+--------------+---------+---------+
4 rows in set (0.00 sec)
```

Negli esempi proposti le `SELECT` oltre le funzioni di aggregazione, comprendono anche alcuni campi che sono quelli che hanno senso per ogni gruppo. Nell'ultimo caso per esempio si richiede il cognome e il nome dell'allievo che è unico per il gruppo: si è infatti effettuato un partizionamento per codice fiscale e quindi tutti i record della partizione hanno in comune i dati dell'allievo.

Anche sui gruppi si possono effettuare selezioni. La clausola `HAVING` permette di selezionare solo i gruppi che soddisfano alle condizioni specificate.

Esempio 4: *Voto minimo, voto massimo e quantità esami di ciascun allievo che ha sostenuto più di un esame*

```
mysql> SELECT COUNT(*) AS Sostenuti, MIN(Voto) AS "Voto Minimo",
    -> MAX(Voto) AS "Voto Massimo", Allievi.Cognome, Allievi.Nome
    -> FROM Allievi,Esami
    -> WHERE Allievi.CodiceFiscale=Esami.CodFiscAll
    -> GROUP BY Allievi.CodiceFiscale
    -> HAVING Sostenuti > 1;
+-----------+-------------+--------------+---------+--------+
| Sostenuti | Voto Minimo | Voto Massimo | Cognome | Nome   |
+-----------+-------------+--------------+---------+--------+
|         2 |          63 |           65 | Aldi    | Aldo   |
|         2 |          61 |           61 | Carli   | Carlo  |
+-----------+-------------+--------------+---------+--------+
2 rows in set (0.00 sec)
```

Il risultato della `COUNT` viene conservato nella variabile `Sostenuti` e il valore contenuto in essa serve per la selezione dei gruppi interessati. È opportuno notare che, in questo caso, `Sostenuti` non è solo l'intestazione di una colonna ma il nome di una variabile su cui si effettua un controllo e, quindi, valgono le regole generali per i nomi di variabili. Non va racchiuso fra doppi apici perché non è una stringa e non può contenere caratteri particolari.

Esempio 5: *Cognome, Nome e Media dell'allievo con la media più alta*

```
mysql> SELECT Cognome,Nome,AVG(Voto) AS Media
    -> FROM Esami,Allievi
    -> WHERE Esami.CodFiscAll=Allievi.CodiceFiscale
    -> GROUP BY CodiceFiscale
    -> ORDER BY Media DESC
    -> LIMIT 1 ;
+---------+------+---------+
| Cognome | Nome | Media   |
+---------+------+---------+
| Aldi    | Aldo | 64.0000 |
+---------+------+---------+
1 row in set (0.00 sec)
```

In conseguenza del raggruppamento la media dei voti riportati viene calcolata per ogni allievo che ha sostenuto esami. La tabella risultante è ordinata in maniera decrescente e, quindi, basta selezionare la prima riga per avere le informazioni richieste.

Query annidate

"*Le query annidate ... rappresentano uno strumento sintattico molto importante per effettuare interrogazioni complesse sui database. Un'interrogazione nidificata (o subquery), è una query che sta all'interno di un'altra interrogazione. La query interna, cioè la subquery, passa i risultati alla query esterna*" (Wikipedia). Il DBMS esegue prima le query più interne e, una volta completate, quelle che si trovano ai livelli superiori.

La subquery può trovarsi nell'elenco delle tabelle della clausola `FROM` di una `SELECT` e, in questo caso, la sua esecuzione produce una tabella che viene aggiunta alle altre presenti nella `FROM` della query esterna. La subquery può essere inserita in una delle condizioni specificate nella clausola `WHERE` e, in questo caso, produce valori che possono essere verificati nelle condizioni.

Esempio 1: *Quantità di allievi esaminati e quantità di esami effettuati nel giorno 2013-09-05*

```
mysql> SELECT COUNT(*) AS "Esaminati", SUM(temp.es) AS "Quant.Esami"
    -> FROM (SELECT COUNT(*) AS es
    ->       FROM Esami
    ->       WHERE Data="2013-09-05"
    ->       GROUP BY CodFiscAll) AS temp;
+-----------+-------------+
| Esaminati | Quant.Esami |
+-----------+-------------+
|         4 |           5 |
+-----------+-------------+
1 row in set (0.00 sec)
```

la query interna conta per ogni allievo gli esami effettuati. La tabella prodotta è necessario che abbia un nome (temp nell'esempio). Contando le righe di temp, nella query esterna, si può sapere quanti sono gli allievi che hanno sostenuto almeno un esame alla data stabilita e, inoltre, sommando i dati della colonna es della temp si può conoscere la quantità totale di esami effettuati.

Esempio 2: *Elenco (cognome e nome) degli allievi che ancora non hanno sostenuto alcun esame*

```
mysql> SELECT Cognome,Nome
    -> FROM Allievi
    -> WHERE CodiceFiscale NOT IN
```

```
    ->       (SELECT DISTINCT CodFiscAll FROM Esami);
+---------+------+
| Cognome | Nome |
+---------+------+
| Enei    | Enea |
+---------+------+
1 row in set (0.00 sec)
```

la query interna produce un insieme di valori che rappresentano i codici fiscali degli allievi che hanno sostenuto almeno un esame considerati una sola volta (clausola `DISTINCT`). La query esterna produce una tabella con l'elenco di coloro il cui codice fiscale non risulta (clausola `NOT IN`) nell'insieme di valori calcolato in precedenza.

I seguenti due esempi, molto simili, intendono chiarire la differenza concettuale fra l'inserire la query interna nella clausola `FROM` o nella `WHERE`.

Esempio 3: *Allievi che hanno ottenuto voti di esame superiori rispetto alla media, con informazioni sugli esami*

```
mysql> SELECT Cognome, Nome, Data, Voto
    -> FROM Allievi,Esami
    -> WHERE Allievi.CodiceFiscale=Esami.CodFiscAll AND
    -> Voto > (SELECT AVG(Voto) FROM Esami);
+---------+------+------------+------+
| Cognome | Nome | Data       | Voto |
+---------+------+------------+------+
| Aldi    | Aldo | 2013-09-02 |   65 |
| Aldi    | Aldo | 2013-09-05 |   63 |
+---------+------+------------+------+
2 rows in set (0.00 sec)
```

Esempio 3bis: *Stesse informazioni dell'esempio 3 ma in più la visualizzazione della media*

```
mysql> SELECT Cognome,Nome,Data,Voto,Media
    -> FROM Allievi,Esami,
    ->      (SELECT AVG(Voto) AS Media FROM Esami) AS temp
    -> WHERE Allievi.CodiceFiscale=Esami.CodFiscAll AND
    ->       Voto > Media;
+---------+------+------------+------+---------+
| Cognome | Nome | Data       | Voto | Media   |
+---------+------+------------+------+---------+
| Aldi    | Aldo | 2013-09-02 |   65 | 62.0000 |
| Aldi    | Aldo | 2013-09-05 |   63 | 62.0000 |
+---------+------+------------+------+---------+
2 rows in set (0.00 sec)
```

nel caso dell'esempio 3 la query interna calcola il valore della media che viene utilizzato, nella `WHERE`, per selezionare gli esami con voto maggiore di tale media. Nell'esempio 3bis la query interna produce una tabella la cui colonna viene utilizzata sia per la selezione che per la visualizzazione. Operazione, questa ultima, non possibile con la query dell'esempio 3 perché in quel caso si tratta di valore.

Riferimenti bibliografici

Materiali consultati per la stesura di questi appunti:

- Atzeni, Ceri, Paraboschi, Torlone - *Basi di dati*
- Manuali di Linux pro – *MySQL: dal setup all'uso professionale*
- Il manuale di riferimento di MySQL *http://dev.mysql.com/doc/refman/5.0/en/index.html* e, in particolare, il tutorial: *http://dev.mysql.com/doc/refman/5.0/en/tutorial.html*