ROMA TRE UNIVERSITÀ DEGLI STUDI
Corso di Laurea in Matematica
Dipartimento di Matematica e Fisica

Sistemi per l'elaborazione delle informazioni

5. Database relazionali

Dispense del corso IN530 a.a. 2019/2020

prof. Marco Liverani

Archivi informatici (digitali)

- L'archiviazione dei dati è una delle applicazioni più diffuse ed importanti dei calcolatori
- Tra gli obiettivi dell'archiviazione informatica dei dati evidenziamo i seguenti:
  - Garantire una **elevata capacità di archiviazione**
  - Garantire il **recupero efficiente** delle informazioni archiviate
  - Garantire la possibilità di **selezionare in modo efficiente** le informazioni desiderate
  - Garantire **l'integrità fisica** e **logica** delle informazioni archiviate
  - Garantire la possibilità di utilizzare le informazioni archiviate attraverso **diversi strumenti informatici**
  - Garantire la **protezione delle informazioni** archiviate, impedendo la lettura o l'alterazione delle informazioni da parte di persone non autorizzate a farlo

Archiviazione su file

- Ogni programma applicativo può **archiviare su file** i dati di propria competenza
- Questo tipo di archiviazione tuttavia non tiene conto di alcuni aspetti importanti:
  - Il formato con cui sono memorizzate le informazioni su file è **arbitrario**, **non standard**
  - Altre applicazioni software possono accedere ai dati solo se il **formato dei dati ed il loro significato** vengono resi noti da chi ha progettato l'applicazione di archiviazione (oggi si usa frequentemente una codifica **XML**, che in parte risolve questa problematica)
  - Devono essere gestite situazioni di **accesso concorrente**
  - Chi sviluppa il software di archiviazione **deve farsi carico anche degli aspetti "fisici":** efficienza nella scrittura e nella selezione delle informazioni, protezione delle informazioni riservate, modalità di accesso ai file, ecc.

Database Management System

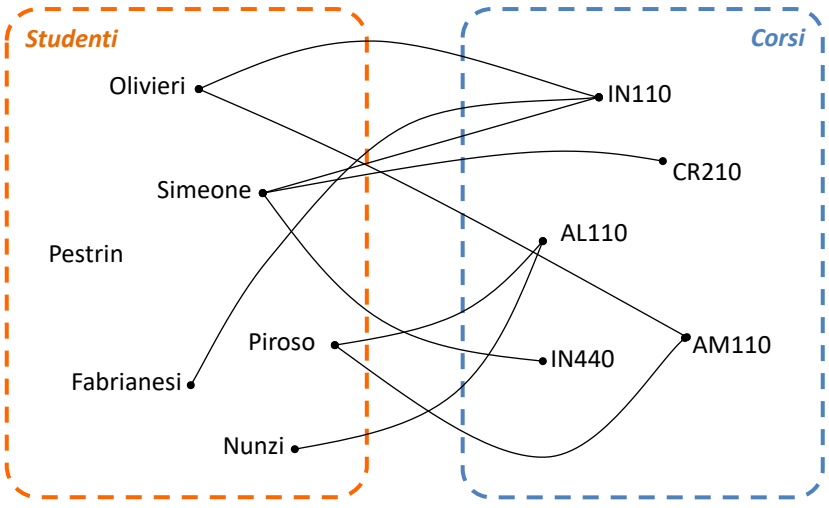
- Per far fronte a questi problemi, a partire dagli anni '70 sono state sviluppate applicazioni dedicate all'archiviazione di informazioni: **DBMS**, *DataBase Management System*
- Caratteristiche di una base dati gestita attraverso un DBMS:
  - Può avere una **grande dimensione** (anche superiore a quella di un singolo disco del computer)
  - Può essere **condivisa** tra più applicazioni
  - È **persistente**
  - È **affidabile**, il DBMS fornisce strumenti per mantenere integro e gestire dei *backup* della base dati
  - È **sicuro e garantisce la riservatezza**, consente il controllo degli accessi, solo gli utenti autorizzati possono accedere ai dati
  - È **efficiente**: le operazioni elementari sull'archivio vengono eseguite molto rapidamente
  - È **standard**: i programmi hanno un'interfaccia astratta e di alto livello per scrivere e leggere informazioni sull'archivio presente sulla memoria secondaria
  - È **disaccoppiato dall'applicazione**: la componente software di gestione dei dati sull'archivio informatico è distinta dalla componente software che implementa la "logica di business" con cui si opera sui dati
  - È **accessibile via rete**: il DBMS può operare su una macchina distinta da quella su cui viene eseguita l'applicazione che utilizza i dati dell'archivio; l'applicazione è inconsapevole della dislocazione fisica del DBMS, si collega al DBMS comunque via rete, anche se il DBMS è sulla stessa macchina su cui gira l'applicazione
- Oggi sul mercato e sui canali del software open source esistono numerosissimi **prodotti DBMS** affidabili: Oracle Database, Microsoft SQL Server, IBM DB/2, Oracle MySQL / MariaDB, PostgreSQL, ecc.

Modello di database

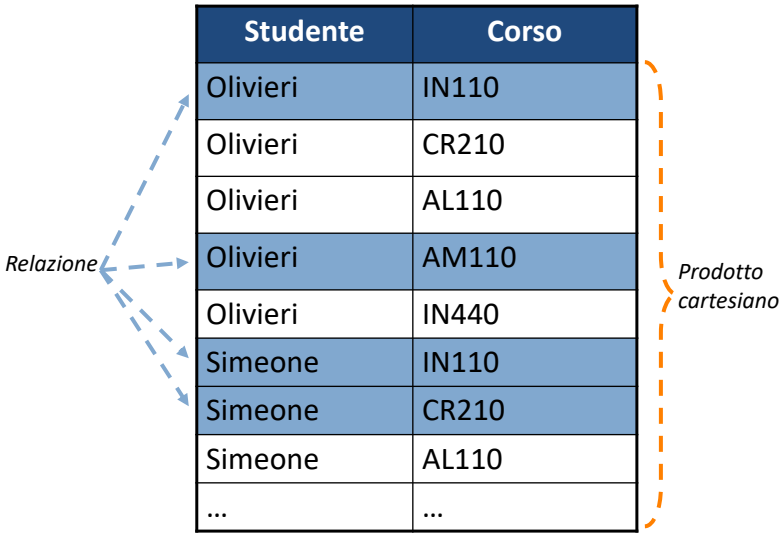
- Ogni DBMS utilizza un **modello astratto** attraverso cui rappresentare le informazioni (e trasformare i singoli dati in vere e proprie informazioni)
- Negli corso degli anni sono stati proposti diversi modelli, tra i quali:
  - Gerarchico, ad albero
  - Reticolare
  - Relazionale
  - Relazionale ad oggetti
- Il modello che ci interessa è il **modello relazionale**, progettato nei laboratori di ricerca di IBM negli anni '70 ed oggi divenuto lo *standard di fatto* per le basi di dati e degli strumenti DBMS: si parla infatti di **RDBMS, *relational database management system***

- Il modello relazionale è basato sul concetto di **relazione** (nel senso algebrico del termine) e di **tabella** (concetto intuitivo)
- Data una collezione di insiemi A_1, A_2,..., A_n, il **prodotto cartesiano** $A_1 \times A_2 \times \ldots \times A_n$ è l'insieme delle n-ple $\{(a_1, a_2, \ldots, a_n)$ tali che $a_i \in A_i \; \forall \; i=1, \ldots, n\}$
- Una **relazione** su A_1, A_2, ..., A_n è un sottoinsieme del prodotto cartesiano $A_1 \times A_2 \times \ldots \times A_n$

Relazioni tra insiemi



Relazioni e prodotto cartesiano



Domini e attributi

- Nell'ambito della teoria relazionale dei dati è utile poter considerare le *n*-ple di una relazione come sequenze **non ordinate** (a differenza di quanto avviene nella teoria degli insiemi)
- Quindi ad esempio $(a_1, a_2, a_3) = (a_3, a_1, a_2)$
- In questo caso, per distinguere i dati presenti in una *n*-pla, è utile assegnare dei **nomi** ai valori delle *n*-ple; tali nomi vengono detti **attributi**
- L'insieme A_i (finito o infinito) dei valori che un certo attributo può assumere è il **dominio** dell'attributo stesso

Relazioni e tabelle

- È possibile rappresentare una relazione mediante una tabella:

Relazione X		
Attributo A	Attributo B	Attributo C
A_1	B_7	C_4
A_9	B_7	C_1

- Il numero di colonne (il numero di attributi) è il **grado** della relazione
- Il numero di righe è la **cardinalità** della relazione
- La relazione dell'esempio ha cardinalità 2 e grado 3

Chiavi primarie

- Consideriamo la seguente relazione rappresentata mediante una tabella:

Esami				
Studente	***Matricola***	***Corso***	Docente	Voto
Bianchi	***102030***	***IN110***	Liverani	24
Verdi	***213243***	***AL210***	Fontana	27
Rossi	***376114***	***IN110***	Liverani	25
Bianchi	***102030***	***AL210***	Fontana	22

- Una **chiave primaria** (*primary key*) è un campo (attributo) o un insieme di campi della tabella che permettono di individuare **univocamente** un record (riga della tabella)
- Nell'esempio la chiave primaria è la coppia "*Matricola*" + "*Corso*"

Chiavi esterne e correlazioni fra tabelle

- Per ridurre la ridondanza dei dati (evitare ripetizioni inutili) è possibile suddividere le informazioni su più tabelle:

Studente	
Matricola	Nome
102030	Bianchi
213243	Verdi
376114	Rossi

Corso	
Nome	Docente
AL110	Fontana
IN110	Liverani

Esame		
Corso	***Matricola***	Voto
IN110	***102030***	24
AL110	***213243***	27
IN110	***376114***	25
AL110	***102030***	22

- Le **chiavi esterne** (*foreign key*) sono attributi di una tabella *X* che consentono di **correlare** il record con un record di una tabella *Y* attraverso la ripetizione del valore della chiave primaria della tabella *Y*

Diagramma Entità/Relazioni

- In questo modo è possibile costruire una rappresentazione dei dati attraverso un diagramma (un grafo) che rappresenta le **relazioni esistenti fra le varie entità**
- Le relazioni possono essere di tre tipi:
  - "**1:1**": ad ogni record dell'entità *A* corrisponde uno ed un solo record dell'entità *B*
  - "**1:M**": ad ogni record dell'entità *A* corrispondono diversi record dell'entità *B*
  - "**M:M**": ad ogni record di *A* corrispondono molti record di *B* e viceversa.

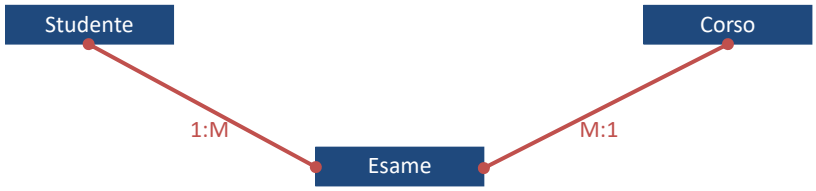


Diagramma Entità/Relazioni

- La relazione viene espressa mediante un verbo o un sostantivo, rappresentato con un rombo collegato a due entità
- Anche le relazioni, come le entità, possono essere caratterizzate da attributi

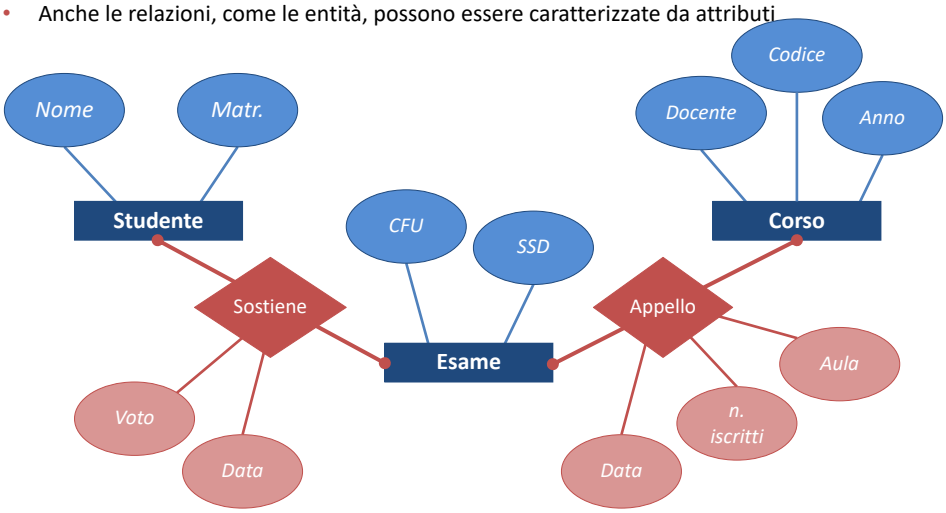
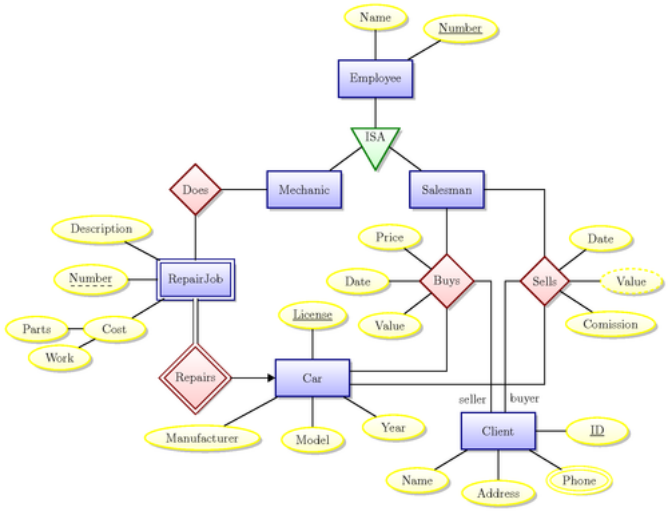


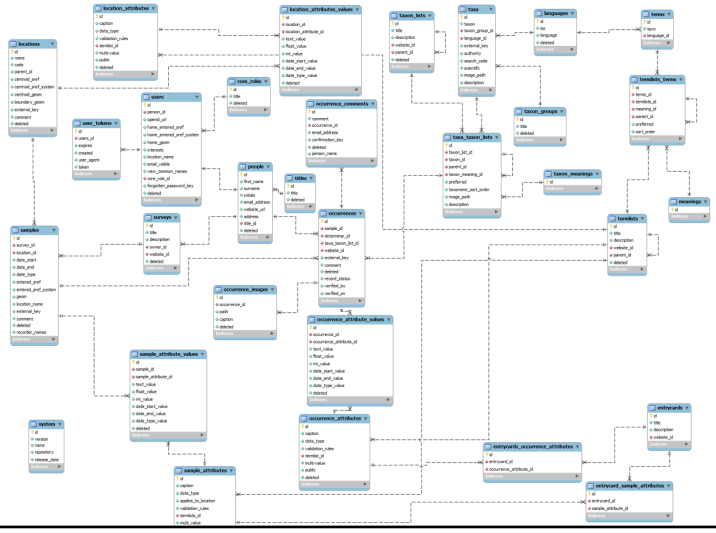
Diagramma Entità/Relazioni

- Di seguito un modello E/R più complesso, parte di un database relazionale di un sistema aziendale



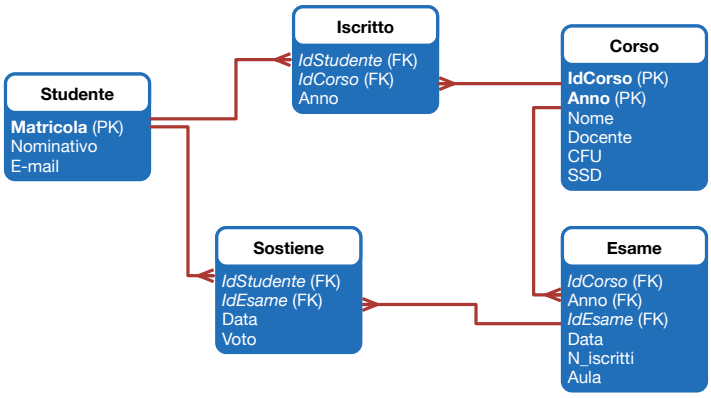
Modello fisico della base dati

- Nel **modello fisico** le *entità* e le *relazioni* sono rappresentate mediante **tabelle** collegate tra loro mediante le **chiavi primarie** e le **chiavi esterne**



Modello fisico della base dati

- Ad esempio il modello E/R relativo agli studenti, ai corsi e agli esami, può essere tradotto nel modello fisico rappresentato in figura



Ottimizzazione di una base dati

- La possibilità di definire correlazioni tra le entità del database ci permette di **ottimizzarne la struttura**
- L'operazione di ottimizzazione consiste nella suddivisione dei dati su tabelle distinte, in modo da ridurre la *ridondanza*; questa attività si chiama **normalizzazione della base dati**
- Esempio di base dati **non** normalizzata

Esami							
Nome	Matricola	AL210	Prof_AL210	IN110	Prof_IN110	GE110	Prof_GE110
Rossi	203040	24	Fontana	22	Liverani	-	Lopez
Verdi	251497	-	Fontana	-	Liverani	27	Lopez
Bianchi	337782	29	Fontana	30	Liverani	26	Lopez

Finalità della normalizzazione

- Perché normalizzare la base dati? Per risolvere i seguenti problemi:
  - **ridondanza**: inutile ripetizione di uno stesso dato
  - **anomalia di aggiornamento**: per mantenere integre e coerenti le informazioni è necessario modificare più volte uno stesso dato
  - **anomalia di cancellazione**: eliminando un'informazione dal database se ne perdono anche altre
  - **anomalia di inserimento**: non è possibile inserire informazioni incomplete, anche quando sarebbe necessario farlo

Esami							
Nome	Matricola	AL210	Prof_AL210	IN110	Prof_IN110	GE110	Prof_GE110
Rossi	203040	24	Fontana	22	Liverani	-	Lopez
Verdi	251497	-	Fontana	-	Liverani	27	Lopez
Bianchi	337782	29	Fontana	30	Liverani	26	Lopez

Forme normali

- Esistono delle regole che devono essere rispettate dalla base dati affinché questa sia correttamente normalizzata; queste regole sono note come **forme normali**
- ***Prima forma normale***
  in una tabella non possono esistere colonne definite per contenere una molteplicità di valori

NO!

Esami				
Matr	Esami sostenuti			
	AL110	IN110	GE110	IN530
102030	25	27	22	30
123987	28	21	29	20
874329	25	26	26	24

OK!

Esame		
Matr	*Corso*	Voto
102030	***AL110***	25
123987	***GE110***	29
102030	***IN110***	27
874329	***IN430***	24

Forme normali

- ***Seconda forma normale***
  *in una tabella in cui la chiave primaria è composta da **più attributi** tutte le colonne devono dipendere dalla chiave primaria*

NO!

Esame			
Matr	*Corso*	Voto	Docente
102030	***AL110***	25	Fontana
123987	***GE110***	29	Lopez
102030	***IN110***	27	Liverani
874329	***AL110***	24	Fontana

OK!

Esame		
Matr	*Corso*	Voto
102030	***AL110***	25
123987	***GE110***	29
102030	***IN110***	27
874329	***IN530***	

Corso	
Nome	Docente
AL110	Fontana
GE110	Lopez
IN110	Liverani

Forme normali

- ***Terza forma normale***

  Non esistono dipendenze tra colonne di una tabella se non basate sulla chiave primaria

NO!

Esame			
Matr	***Corso***	Voto	Crediti
102030	***AL410***	25	6
123987	***GE110***	29	10
102030	***IN110***	27	10
874329	***IN440***	24	7

Corso	
Nome	Docente
AL110	Fontana
GE110	Lopez
IN110	Liverani

OK!

Esame		
Matr	***Corso***	Voto
102030	***AL110***	25
123987	***GE110***	29
102030	***IN110***	27
874329	***IN440***	24

Corso		
Nome	Docente	Crediti
AL410	Fontana	6
IN440	Liverani	7
IN110	Liverani	10

Forme normali

- ***Forma normale di Boyce e Codd***

  *Una tabella è in forma normale se per ogni **dipendenza funzionale** $A \rightarrow B$ definita su di essa, A contiene una chiave della tabella.*

Esame					
Matr	***Corso***	Voto	Docente	Crediti	Studente
102030	***AL110***	25	Fontana	10	Rossi
123987	***GE110***	29	Lopez	10	Bianchi
102030	***IN110***	27	Liverani	10	Rossi
874329	***IN440***	24	Fontana	7	Verdi

Dipendenze funzionali:

$A \rightarrow B$: il valore di A determina il valore di B

Corso → Docente ✗

Corso → Crediti ✗

Matricola, Corso → Voto

Matricola → Studente ✗

NO!

Forme normali

- ***Forma normale di Boyce e Codd***

  *Una tabella è in forma normale se per ogni **dipendenza funzionale** $A \rightarrow B$ definita su di essa, A contiene una chiave della tabella.*

Esame		
Matr	***Corso***	Voto
102030	***AL110***	25
123987	***GE110***	29
102030	***IN110***	27
874329	***IN440***	24

Corso		
Nome	Docente	Crediti
AL110	Fontana	10
GE110	Lopez	10
IN110	Liverani	10

Studente	
Matricola	Nome
102030	Bianchi
213243	Verdi
376114	Rossi

OK!

Dipendenze funzionali:

$A \rightarrow B$: il valore di A determina il valore di B

Corso → Docente

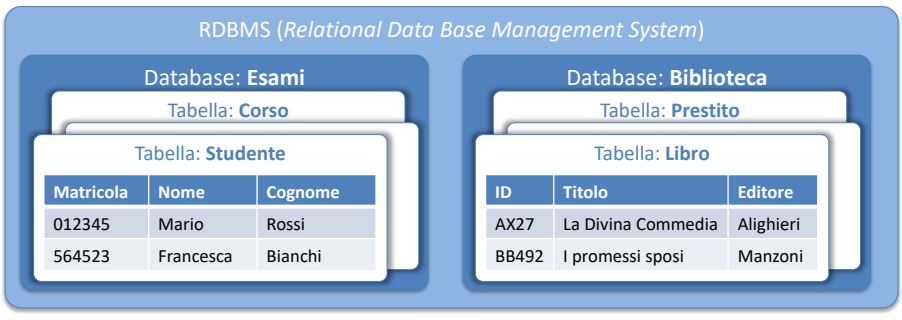
Corso → Crediti

Matricola, Corso → Voto

Matricola → Studente

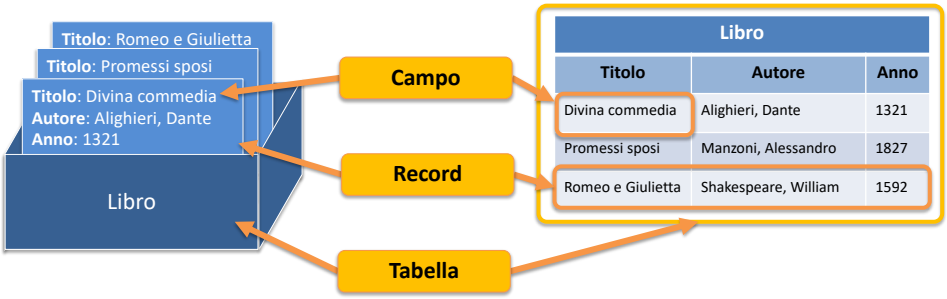
Schema fisico

- Una volta definito il *modello astratto* con cui dovranno essere rappresentate le informazioni, anche attraverso un **diagramma E/R** (entità / relazioni), è necessario trasformare tale modello in una rappresentazione tale da poter essere implementata su un sistema DBMS
- Lo **schema fisico** del database è la rappresentazione del modello delle entità e delle relazioni in una forma gestibile da un prodotto DBMS
- Lo schema fisico di un database è composto da **tabelle** (con cui sono rappresentate le entità e le relazioni e gli attributi che le compongono) e i legami che esistono tra di esse, attraverso l'uso delle **chiavi** (primarie ed esterne)



Schema fisico

- Una **tabella** di database è una struttura dati caratterizzata da
  - un **nome** (univoco nell'ambito del database)
  - un insieme di **attributi**, ognuno identificato da un *nome* (univoco nell'ambito della tabella) e caratterizzati da un *tipo di dato* (numerico o alfanumerico)
  - gli attributi delle tabelle possono avere dei **vincoli** sui valori
  - alcuni attributi possono essere (parte della) **chiave primaria** della tabella o **chiavi esterne** referenziate ad altri attributi di altre tabelle
- Le righe della tabella sono i **record**
- Gli attributi della tabella sono i **campi** del record



Linguaggio SQL: caratteristiche

- Per operare su una base dati relazionale è stato progettato un linguaggio standard:

  SQL (*Structured Query Language*)

- È un linguaggio di **interrogazione e manipolazione** della base dati e delle informazioni in essa contenute
- Non è un linguaggio imperativo/procedurale, è un **linguaggio dichiarativo**: manca dei concetti di *variabile* e di *struttura di controllo algoritmica*, tipica dei linguaggi imperativi e della programmazione strutturata

- È costituito da tre insiemi di istruzioni:
  - **DDL** (*Data Definition Language*): per definire la struttura della base dati
  - **DCL** (*Data Control Language*): per gestire i criteri di protezione e di accesso ai dati
  - **DML** (*Data Manipulation Language*): per operare sui dati

Data Definition Language

- È il set di istruzioni di SQL per la definizione della struttura della base dati.
- Sono tre le istruzioni principali del DDL:
  - `create`: per la creazione di database, tabelle, indici, viste, ecc.
  - `alter`: per la modifica della struttura di una tabella o di altri oggetti interni ad una base dati
  - `drop`: per l'eliminazione di una tabelle, di un intero database o di altri oggetti
- Creazione di un database:

```
create database studenti;
```

- Cancellazione di un database:

```
drop database studenti;
```

Dominio di definizione degli attributi

- Definire una tabella significa definirne gli **attributi** e il **dominio** degli stessi attributi
- I domini sono quelli tipici di ogni linguaggio di programmazione:
  - **numeri interi** (int, integer)
  - **numeri floating point** (float, real, double)
  - **stringhe di caratteri** (char, varchar)
  - **date** (date, time, timestamp)
- Altri tipi possono essere definiti sulla base di specifiche tipiche di un determinato RDBMS
- Ogni attributo può essere anche caratterizzato da un **valore di default** e una serie di **vincoli** (es. "`not null`" per i campi obbligatori di una tabella)
- Tra i vincoli vi è la possibilità di aggiungere un attributo alla **chiave primaria** della tabella o l'attributo di **chiave esterna** referenziando uno o più campi chiave di un'altra tabella

Creazione di una tabella

- Nell'esempio riportato di seguito vengono create due tabelle denominate "`corso`" ed "`esame`" (*usando la sintassi di MySQL*):

```
create table corso (
    sigla char(5) not null primary key,
    nome char(20) not null,
    docente char(30),
    crediti integer default 6
) engine=innodb;

create table esame (
    s_corso char(5) not null references corso.sigla,
    matr_studente char(10) not null references studente.matricola,
    voto integer not null,
    primary key (s_corso, matr_studente),
    foreign key (s_corso) references corso(sigla) on update cascade
        on delete restrict,
    foreign key (matr_studente) references studente(matricola)
        on update cascade on delete cascade
) engine=innodb;
```

Indici sulle tabelle

- Per rendere **più efficiente l'operazione di selezione** dei dati presenti su una tabella, è possibile creare uno o più **indici** su una stessa tabella
- Se una tabella è priva di indici, l'unico metodo per cercare dati che corrispondono ad un determinato criterio, è quello di eseguire un "***full table scan***", ossia una scansione dell'intera tabella su cui si esegue la ricerca: il tempo richiesto è lineare nel numero di righe della tabella (complessità $O(n)$)
- L'indice è una struttura dati che consente di migliorare i tempi di ricerca delle informazioni presenti in una tabella
- L'indice è spesso una struttura dati "ad albero" (alberi binari, alberi R-B, ecc.) costruita sui valori di una specifica colonna di una tabella di database (es.: il campo "cognome" di una tabella anagrafica): in questo modo **il tempo di ricerca diventa logaritmico**, anziché lineare: $O(\log_2 n)$
- Su una stessa tabella possono essere definiti più indici su più campi
- Esempio:

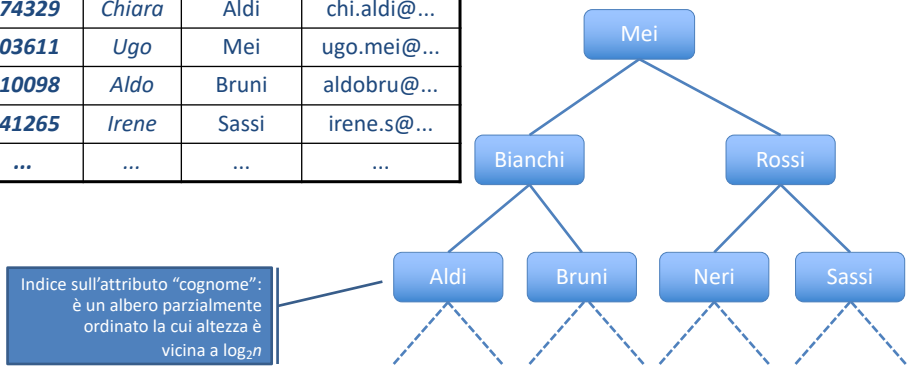
```
create index nominativo on studente (cognome)
```

- Dopo varie operazioni di inserimento e cancellazione sui record di una tabella, l'indice è probabilmente "sbilanciato" e quindi non più efficace: in tali casi è sufficiente eliminare l'indice (`drop index nominativo`) e ricrearlo dopo qualche istante

Indici sulle tabelle

Studente			
Matricola	*Nome*	Cognome	Email
102030	*Mario*	Rossi	m.rossi@...
123987	*Anna*	Bianchi	anna90@...
102030	*Elena*	Neri	nerielena@...
874329	*Chiara*	Aldi	chi.aldi@...
103611	*Ugo*	Mei	ugo.mei@...
510098	*Aldo*	Bruni	aldobru@...
241265	*Irene*	Sassi	irene.s@...
...	...	...	...

Tabella con *n* righe: per cercare un dato devo scorrerla per intero (*n* operazioni di confronto)



Indice sull'attributo "cognome": è un albero parzialmente ordinato la cui altezza è vicina a $\log_2 n$

Data Control Language

- È il set di istruzioni di SQL per la definizione dei **permessi di accesso** sui database e sulle tabelle e per la gestione degli account utente.
- Sono due le istruzioni principali del DCL:
  - **grant**: per assegnare un determinato permesso ad un utente
  - **revoke**: revocare un determinato permesso ad un utente
- Concessione di permessi:

```
grant privilegi on risorsa to utenti with grant option;
```

- Esempio:

```
grant select on studenti to liverani;
```

Data Manipulation Language

- È il set di istruzioni di SQL per la **gestione delle informazioni** presenti nella base dati
- Sono quattro le istruzioni principali del DML:
  - `insert`: per inserire dati in una tabella
  - `select`: per selezionare in base a determinati criteri o condizioni i dati presenti in una o più tabelle
  - `update`: per modificare i dati di una tabella sulla base di un determinato criterio di selezione
  - `delete`: per eliminare i record di una tabella corrispondenti ad una determinata condizione

Inserimento dati

- L'istruzione `insert` consente di inserire un insieme di valori nei campi di un solo record in una tabella: l'ordine con cui vengono forniti i valori ai diversi campi, sono specificati elencando nello stesso ordine i nomi degli attributi corrispondenti
- Le **stringhe di caratteri** devono essere delimitate da apici; se la stringa contiene un apice, questo va riportato due volte
  - esempio: `'Maria Giovanna'`, `'Guido Dell''Acqua'`,
- Le **date** devono essere indicate nel formato specifico del DBMS (es.: *'aaaa-mm-gg'*, come `'2015-06-20'`)
- Sintassi dell'istruzione `insert`:

```
insert into tabella (campo1, …, campok) values (valore1, …, valorek);
```

- Esempio:

```
insert into corso (sigla, cfu, nome, docente)
values ('IN110', 10, 'Informatica 1', 'Liverani Marco');
```

Selezione di record

- L'istruzione **select** consente di leggere informazioni presenti sul database, selezionando il risultato da una o più tabelle, mediante apposite condizioni che consentono di scegliere tra i record presenti in archivio, quelli da visualizzare in output
- Sintassi dell'istruzione **select**:

```
select tabella1.campo1, …, tabellah.campok
from tabella1, …, tabellah
where condizione
order by tabellai.campoi, …, tabellaj.campoj;
```

- Esempio:

```
select corso.sigla, corso.nome, corso.docente from corso where
ssd='INF/01' order by corso.docente, corso.sigla;
```

- Anche il risultato di una **select** è una tabella: una **select** su una o più tabelle produce una tabella

Join

- È possibile eseguire operazioni di selezione che coinvolgano **più tabelle** grazie alle correlazioni stabilite tra queste attraverso le chiavi primarie e le chiavi esterne
- Questa operazione si chiama ***join***
- Esempio:

```
select studente.nome, studente.cognome, corso.nome, esame.voto
from studente, corso, esame
where studente.matricola=esame.matr_studente and
esame.s_corso=corso.sigla
order by esame.voto;
```

Join e prodotto cartesiano di tabelle

- Ogni operazione di *join* fra più tabelle consiste innanzi tutto in un **prodotto cartesiano** tra le tabelle coinvolte nella **`select`**
- Dalla relazione (tabella) ottenuta come prodotto cartesiano **si selezionano solo le righe** che corrispondono con la "*where condition*"
- Una operazione di *join* può essere quindi **molto onerosa** per la macchina: la base dati deve essere progettata in modo da minimizzare il numero di *join* necessarie per ottenere le informazioni di interesse

Join e prodotto cartesiano di tabelle

Esame		
Matr	Corso	Voto
102030	AL110	25
123987	GE110	29
102030	IN110	27

Studente	
Matricola	Nome
102030	Bianchi
123987	Verdi
376114	Rossi

```
select esame.corso, esame.voto, studente.nome
from esame, studente
where esame.matr=studente.matricola;
```

Join: esame x studente				
Matr	Corso	Voto	Matricola	Nome
102030	AL110	25	102030	Bianchi
102030	AL110	25	123987	Verdi
102030	AL110	25	376114	Rossi
123987	GE110	29	102030	Bianchi
123987	GE1	29	123987	Verdi

Aggiornamento

- L'istruzione **update** consente di modificare alcuni dei valori assegnati ai campi di una tabella dei record che soddisfano determinate condizioni
- Sintassi dell'istruzione **update**:

```
update tabella
set campo1=valore1, …, campok=valorek
where condizione;
```

- Esempio:

```
update studente set nome='Mori' where matricola=102030;
```

- Attenzione: una istruzione **update** priva di una condizione esegue l'aggiornamento su **tutte** le righe (record) della tabella

Cancellazione

- L'istruzione **delete** permette di eliminare i record di una tabella che soddisfano una determinata condizione
- Sintassi dell'istruzione **delete**:

```
delete from tabella
where condizione;
```

- Esempio:

```
delete from studente where nome like 'Mo%' or matricola=213243;
```

- Attenzione: una istruzione **delete** priva di una condizione esegue la **cancellazione di tutti i record** (tutte le righe) della tabella
- L'operatore "**like**" permette di confrontare il valore di un campo con una parte di una stringa, utilizzando il carattere *jolly* "**%**"
  - Nell'esempio elimina i record della tabella studente in cui il campo nome inizia con la stringa "**Mo**"

Operazioni di gruppo

- È possibile ottenere attraverso una **select** anche valori non presenti nei campi della tabella, ma **calcolati raggruppando più record** sulla base di un criterio di aggregazione
- Esempio:

```
select studente.nome, avg(esame.voto) as media
from studente, esame
where studente.matricola=esame.matr
group by studente.nome
order by media desc;

select count(*) from studenti;
```

- Tipiche funzioni di gruppo sono
  - **sum**: somma i valori numerici dei campi
  - **average/avg**: calcola la media aritmetica dei valori numerici dei campi
  - **count**: conta i record che soddisfano determinate condizioni

SQL e altri linguaggi di programmazione

- Spesso il linguaggio SQL viene utilizzato all'interno di programmi scritti con altri linguaggi di programmazione (C, C++, Java, Perl, Python, ecc.)
- Nello sviluppo di un programma è spesso molto utile il concetto di **transazione**
  - una transazione raccoglie una serie di operazioni svolte sul database e solo al termine della transazione stessa le esegue effettivamente (***commit***)
  - in caso si verifichi un errore durante l'esecuzione del programma, è possibile chiudere la transazione annullando tutte le operazioni eseguite fino a quel momento (***rollback***)
- Esistono librerie di funzioni API per l'accesso alle interfacce messe a disposizione dai DBMS
- Esempio in linguaggio Perl utilizzando DBI/DBD per MySQL:

```
#!/usr/bin/perl
use DBI;
$db = DBI->connect("dbi:mysql:dbname=nomeDB",  "user",  "password") or die DBI::errstr;
$query = $db->prepare("select nome, cognome from rubrica order by cognome");
$query->execute();
for ($i=1; $i <= $query->rows(); $i++) {
  ($nome, $cognome) = $query->fetchrow();
  print "Record n. $i\n Nome: $nome\n Cognome: $cognome\n\n";
}
$query->finish();
$db->disconnect();
```

SQL e altri linguaggi di programmazione

- Un esempio analogo in linguaggio C

```
#include <my_global.h>
#include <mysql.h>

int main(void) {
  MYSQL *connessione = mysql_init(NULL);

  mysql_real_connect(connessione,"127.0.0.1", "user", "passwd", "nomeDB", 0,NULL,0);

  mysql_query(connessione, "select nome, cognome from rubrica order by cognome");

  MYSQL_RES *risultato = mysql_store_result(connessione);

  MYSQL_ROW riga;

  while (riga = mysql_fetch_row(risultato)) {
    printf("Nome: %s\nCognome: %s\n\n", row[0], row[1]);
  }
  mysql_free_result(risultato);
  mysql_close(connessione);
  return(0);
}
```

Object-Relational Mapping

- Spesso per lo sviluppo di programmi che fanno uso di una base dati relazionale si utilizzano linguaggi di programmazione *object oriented* (es.: Java, C#, C++, ecc.)
- In tali casi una buona ingegnerizzazione del codice la si può raggiungere **mappando le classi di oggetti implementate dal programma con entità presenti sul database relazionale**, che offrono funzioni di persistenza per gli oggetti ottenuti istanziando tali classi
- Un prodotto **ORM** (*Object-Relational Mapping*) offre delle funzioni di alto livello per realizzare il collegamento tra le classi del programma e le entità del database
- In questo modo il codice sorgente si semplifica, visto che è la libreria ORM ad occuparsi della maggior parte delle questioni tecniche di interazione tra il programma e il DBMS
- **Hibernate** è un software middleware open source che implementa il servizio ORM per applicazioni scritte in Java:
  - mediante file di configurazione si definisce la mappatura tra oggetti del programma Java ed entità presenti sul DBMS (o che Hibernate deve creare sul DBMS nella fase di inizializzazione)
  - con alcuni metodi messi a disposizione dal middleware si eseguono operazioni di scrittura e lettura dei dati, operando direttamente sugli oggetti del programma, senza la necessità di utilizzare direttamente il linguaggio SQL
- **NHibernate** è un prodotto equivalente ad Hibernate, per il framework Microsoft .net

NO-SQL Database Management Systems

- Il modello relazionale è potente, flessibile e assai diffuso, ma ha dei limiti, soprattutto quando è necessario trattare moli enormi di dati o quando è difficile strutturare a priori il dato nelle entità del database
- Esistono numerosi modelli per la rappresentazione dei dati, **diversi dal modello relazionale**
- Questi modelli sono implementati in prodotti software disponibili sul mercato e più spesso sui canali del software open source: si tratta di prodotti sperimentali o assai solidi e ben strutturati, utilizzati per la gestione affidabili di servizi on-line estremamente onerosi (Google, Facebook, ecc.)
- Tali prodotti utilizzano linguaggi di interrogazione del database e di manipolazione dei dati diversi da SQL: per questo si parla di **NO-SQL database**, *Not Only SQL Database*

NO-SQL Database Management Systems

- **Document Oriented Database**
  - Non memorizzano i dati in tabelle con campi uniformi per ogni record come nei DB relazionali: ogni record è memorizzato come un "documento" (una struttura dati XML/JSON) che possiede determinate caratteristiche; qualsiasi numero di campi con qualsiasi lunghezza può essere aggiunto al documento
  - Prodotti software: IBM Lotus Notes, OrientDB, MongoDB, Apache Solr, ecc.
- **Graph Database**
  - Rappresenta i dati come un grafo ed utilizza vertici e spigoli del grafo per memorizzare informazioni; gli elementi del grafo (vertici e spigoli) possono rappresentare informazioni con struttura e significato differente
  - Prodotti software: Neo4j, OrientDB, ecc.
- **Database basati su coppie chiave/valore**
  - Sono basati su array associativi, mappe o dizionari: sono costituiti da coppie chiave/valore (es.: "nome/Marco", "cognome/Liverani", ecc.)
  - Prodotti software: BigTable (Google), Berkeley DB, ecc.
- **Database object oriented**
  - Implementano un modello ad oggetti, tipicamente integrato con il modello relazionale; consentono l'implementazione di metodi per operare su specifici tipi di dato (immagini fotografiche, dati spaziali georeferenziati, dati con uno specifico significato fisico, ecc.)
  - Prodotti software: Informix Illustra, IBM DB/2 Universal Database, ecc.

Bibliografia essenziale

1. Atzeni, Ceri, Paraboschi, Torlone, *Basi di dati – Concetti, linguaggi e architetture*, McGraw-Hill, 1996
2. Guidi, Dorbolò, *Guida a SQL*, McGraw-Hill, 1996
3. Codd, *A relational model for large shared data banks*, Communication of the ACM, 13(6), June 1986
4. MySQL, *MySQL Reference Manual*, http://dev.mysql.com/doc/refman/5.6/en/
5. PostgreSQL, *PostgreSQL Documentation*, http://www.postgresql.org/docs/9.4/static/

Oracle Exadata Database Machine
Oracle, leader mondiale nel settore dei prodotti RDBMS, produce delle macchine server di classe enterprise specificamente progettate ed ingegnerizzate per svolgere con la massima efficienza il ruolo di "database machine"