UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Enrico Lacchin

Enrico Lacchin

Basi di Dati

Appunti

Materia: Basi di Dati
Docente: Andrea De Lorenzo
Pagina corso: `http://delorenzo.inginf.units.it/project/basi-di-dati-2023/`

Indice

1 Introduzione

1.1 Gestione delle informazioni

L'essere umano genera e gestisce tante informazioni:

- Idee informali
- Linguaggio naturale
- Disegni, grafici, schemi
- Numeri
- Codici

e vengono salvate in tanti modi diversi

- Memoria
- Carta
- Pietra
- Scritta sul muro
- Elettronica

Anche le organizzazioni generano informazioni:

- Utenze telefoniche
- Conti correnti
- Studenti iscritti ad un corso di laurea
- Quotazioni di azioni

1.2 Informazione vs Dato

Informazione: notizia, dato o elemento che consente di avere conoscenza più o meno esatta di fatti, situazioni, modi di essere.
Dato: elemento di informazione costituito da simboli che debbono essere elaborati.

1.3 Numeri

Come codifico i numeri?

Numeri naturali: in binario
Numeri interi: devo decidere come rappresentare il segno
Numeri razionali? 10010011110011001111

1.4 Dati e Applicazioni

I **dati** possono variare nel tempo. Le **modalità** con cui i dati sono rappresentati sono di solito stabili. Le **operazioni** sui dati variano spesso.

É importante separare i dati dalle applicazioni che operano su essi

1.5 Database

Genericamente: Collezione di dati, utilizzati per rappresentare le informazioni di interesse per una o più applicazioni di una organizzazione.

- Schede perforate
- File CSV
- Foglio di calcolo
- File XML
- Access

Per noi: Collezione di dati gestita da un DBMS (1.6)

1.6 Database Management System [DBMS]

Un DMBS è un software in grado di gestire collezioni di dati che siano:

- **Grandi**: di dimensioni (molto) maggiori della memoria centrale
- **Persistenti**: con un periodo di vita indipendente dalla singole esecuzioni dei programmi che le utilizzano
- **Condivise**: utilizzate da applicazioni diverse

Un DBMS deve garantire:

- **Affidabilità**: resistenza a malfunzionamenti hardware e software
- **Privatezza**: con una disciplina e un controllo degli accessi
- **Efficienza**: utilizzando al meglio le risorse di spazio e tempo del sistema
- **Efficacia**: rendendo produttive le attività dei suoi utilizzatori

Condivisione

L'integrazione e la condivisione permettono di

- Ridurre la *ridondanza* (evitando ripetizioni)
- Ridurre possibilità di incoerenza (o *inconsistenza*) fra dati

Poiché la condivisione non è mai completa (o comunque non opportuna) i DBMS prevedono meccanismi per

- *Privatezza* dei dati
- Limitazione all'accesso (*autorizzazioni*)

La condivisione richiede coordinamento degli accessi: *controllo della concorrenza*

Efficienza

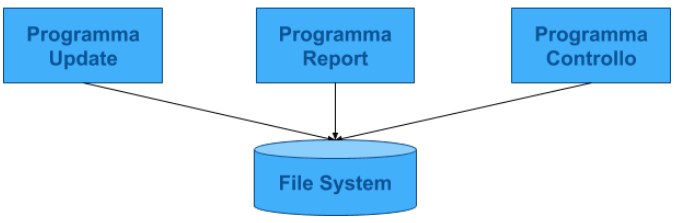
Si misura in termini di tempo di esecuzione e spazio di memoria (principale e secondaria)
I DBMS non sono necessariamente più efficienti dei file system
L'efficienza è il risultato della qualità del DBMS e delle applicazioni che lo utilizzano

1.6.1 DBMS vs FS

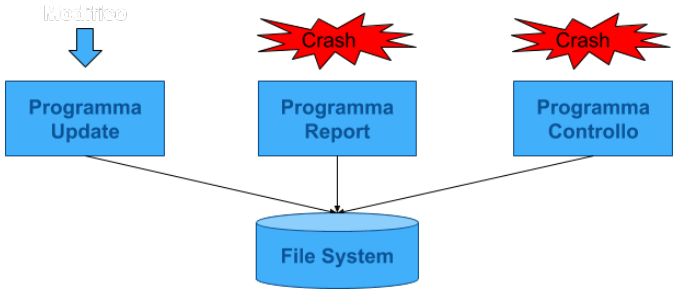
	DBMS	FS
Grandi moli di dati	✓	✓
Persistenti	✓	✓
Condivisi	✓	✓
Affidabile	✓	✓
Privatezza	✓	✓
Efficienza	?	?
Efficacia	X	✓

1.7 File System

Descrizione dei dati contenuta nell'applicazione



In un FS se si va a modificare una delle applicazioni, le altre applicazioni collegate vanno in crash



DBMS e Descrizione dei dati

Il DBMS sa come persistere i dati, per l'applicazione è un atto di fede. I dati sono INDIPENDENTI dalla forma fisica. I programmi parlano con il DBMS per accedere ai dati

1.8 Modello Concettuale

Il modello concettuale non dipende dallo strumento utilizzato

$$\text{Analisi del problema} \implies \text{Modello astratto}$$

1.9 Modello Logico

Il modello logico indica come rappresentare i dati individuati con il modello concettuale.

- Livello intermedio tra utente e implementazione
- Sottintende una specifica rappresentazione dei dati (tabelle, alberi, grafi, oggetti, ...)

1.10 Database System

Un database system è formato da:

- **Software**:
  - DBMS: interposto tra il DB e l'utente
  - Utility di supporto (sviluppo, backup)
- **Utenti**:
  - Progettista

  - Sviluppatore
  - Amministratore
  - Utente finale
- **Schemi** (struttura dei dati)
- **Dati**:
  - Come vengono salvati
  - Condivisione
  - Concorrenza
  - Ridondanza
- **Hardware**

1.10.1 Vantaggi e Svantaggi

Vantaggi:

- Dati sono risorsa in comune
- DB fornisce un modello unificato del business
- Controllo centralizzato dei dati, quindi standardizzazione ed economie di scala
- Riduzione ridondanza ed inconsistenza
- Indipendenza dei dati

Svantaggi:

- Costo e complessità
- Servizi ridondanti/non necessari

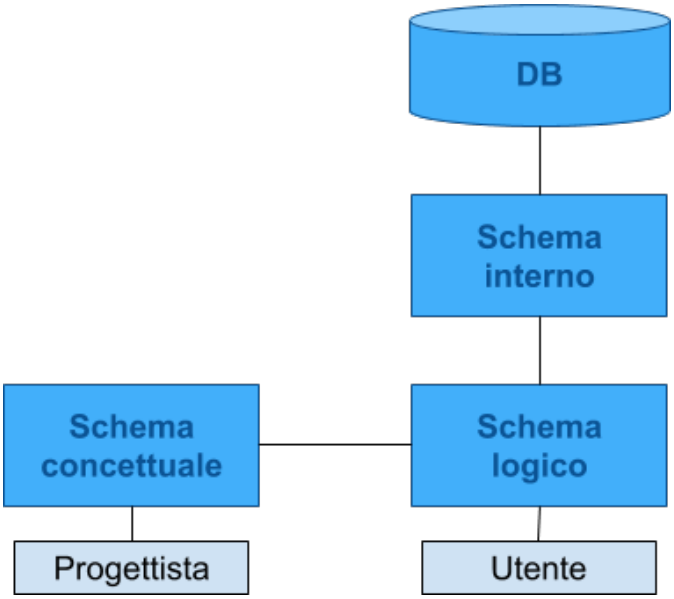
1.11 Schema ed Istanza

In ogni database e esistono:

- **Schema**:
  - Invariante nel tempo
  - Descrive la struttura
  - eg: intestazione tabelle
- **Istanza**:

- I valori attuali
- Possono cambiare
- eg: contenuto delle tabelle

2 Schemi



Schemi concettuali Permettono di rappresentare i dati in modo indipendente da ogni sistema:

- Cercando di descrivere i concetti del mondo reale
- Sono utilizzati nelle fasi preliminari di progettazione

Modello più diffuso: **Entity-Relationship**

2.1 Schemi interni (o fisici)

Rappresentazione dello schema logico per mezzo di strutture di memorizzazione

- File CSV
- File XML
- File binari

2.2 Schemi logici

Com'è organizzato il DB? Diverse soluzioni:

- Gerarchico
- Reticolare
- Relazionale

- Ad oggetti

Indipendenza Lo schema logico è indipendente da quello fisico.
ES: una tabella è utilizzata sempre allo stesso modo qualunque sia la sua realizzazione fisica (che può variare nel tempo)

PROGETTISTA DB $\neq$ SVILUPPATORE SW

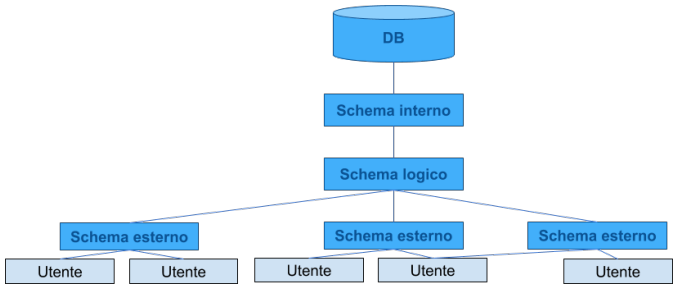
2.3 Vista

L'amministratore del DB può modificare la struttura interna dei dati senza toccarne la visibilità esterna $\Longrightarrow$ Immunità delle applicazioni a modifiche di struttura

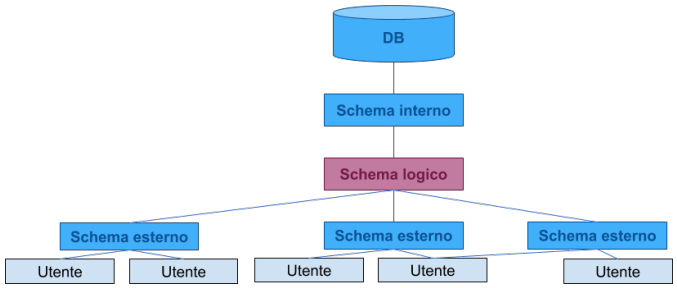
SCHEMA ESTERNO = VISTA

- Descrive parte della base di dati di un modello logico
- NON è una copia dei dati

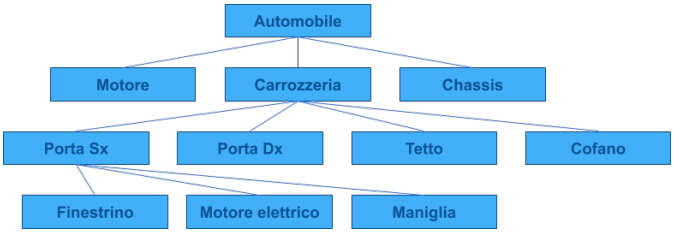
2.3.1 Architettura ANSI/SPARC



2.4 Schemi logici



2.5 Schemi logici gerarchici

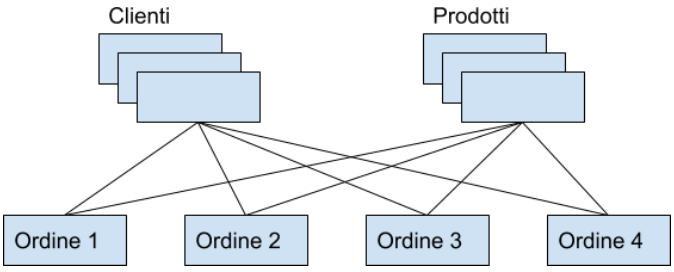


Problemi:

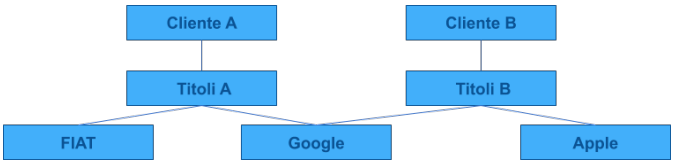
- Accesso sequenziale: per arrivare al figlio devo attraversare tutti i nodi
- Modifica parziale complicata
- Cancellazione gerarchica
- Stretto legame tra programma e struttura del database
- Ridondanza

2.6 Schemi logici gerarchici (ridondanza)

2.7 Schemi logici reticolari



- COBOL, 1970
- Nodi collegati da puntatori
- Navigazione bi-direzionale



2.8 Schemi logici relazionali

[CODD, 1980]
L'obiettivo di questi schemi è quello di liberarsi dai puntatori fisici

- I dati sono organizzati in tabelle di valori
- Le operazioni vengono eseguite sulle tabelle
- I risultati delle operazioni sono tabelle
- I riferimenti tra dati in strutture (tabelle) diverse sono rappresentati con valori

2.8.1 Elementi di un DBR

Tabelle: organizzazione rettangolare di dati

- Record (righe) e campi (colonne) e domini dei dati
- I campi definiscono univocamente il tipo dei dati (dominio)
- I campi hanno un nome ed un ordine, le righe no
- Esistono tabelle vuote

Chiavi Primarie [PK]

- Una (o più) colonne che identificano **UNIVOCAMENTE** il record
- Non possono essere duplicate
- Una tabella in cui ogni riga è diversa dalle altre è detta **RELAZIONE**

Relazioni

- Non esistono relazioni padre-figlio
- Le relazioni sono rappresentate da dati comuni manipolabili

Chiavi Esterne (secondarie, Foreign Key [FK])

- Una colonna in una tabella il cui valore corrisponde ad una chiave primaria
- Sono fondamentali nella creazione delle relazioni

2.8.2 12 Regole di CODD

1. **Informazioni**: Tutte le informazioni in un DBR sono rappresentate esplicitamente da valori in tabelle
2. **Accesso Garantito**: Ciascun valore deve essere raggiunto univocamente da un nome di tabella, chiave primaria e nome di colonna (CHIAVI PRIMARIE)
3. **Valori NULL**: Sono supportati per rappresentare informazioni mancanti indipendentemente dal tipo di dato
4. **System Table**: Un database relazionale deve essere strutturato logicamente come i dati e gestibile con lo stesso linguaggio
5. **Linguaggio di interrogazione standard**: Un DBR può supportare diversi linguaggi, ma deve supportare un linguaggio "English like" dove sia possibile:
   - Definire dati
   - Definire viste
   - Manipolare dati
   - Gestire l'integrità
6. **Viste modificabili**: Le viste che sono modificabili teoricamente dall'utente lo devono essere anche dal sistema (cruciale per campi calcolati). Affinché una vista sia modificabile, il DBMS deve essere in grado di tracciare ciascuna colonna e ciascuna riga UNIVOCAMENTE fino alle tabelle origine
7. **Inserimento e update da linguaggio**: Inserire e aggiornare devono avere la stessa logica "a righe" dell'estrazione (SET ORIENTED)
8. **Indipendenza fisica dei dati**: I programmi applicativi non devono sentire alcuna modifica fatta sul metodo e la locazione fisica dei dati
9. **Indipendenza logica dei dati**: Le modifiche al livello logico non devono richiedere cambiamenti non giustificati alle applicazioni che utilizzano il database (VISTE)
10. **Integrità**: Vincoli di integrità devono essere implementabili sul motore
11. **Indipendenza di localizzazione**: La distribuzione di porzioni del database su una o più allocazione fisiche o geografiche deve essere invisibile agli utenti del sistema
12. **Deve prevenire accessi non desiderati**: Garantisce l'impossibilità di bypassare le regole di integrità

2.8.3 Relazione Matematica

- $D_1, \ldots, D_n$ (n insiemi anche distinti) sono i domini
- Prodotto cartesiano $D_1 \times \cdots \times D_n$ è l'insieme di tutte le n-uple $(d_1, \ldots, d_n)$ tali che $d_1 \in D_1, \ldots, d_n \in D_n$
- Relazione matematica su $D_1, \ldots, D_n$: un sottoinsieme di $D_1 \times \cdots \times D_n$

Proprietà

Una relazione matematica è un insieme di n-uple ordinate:

$$(d_1, \ldots, d_n) | d_1 \in D_1, \ldots, d_n \in D_n$$

Una relazione è un insieme:

- Non c'è ordinamento tra le n-uple
- Le n-uple sono distinte
- Ogni n-upla è ordinata: i-esimo valore proviene dall'i-esimo dominio

Struttura non posizionale A ciascun dominio si associa un nome (attributo) che ne descrive il ruolo

Casa	Ospiti	RetiCasa	RetiOspiti
Juve	Lazio	3	1
Lazio	Milan	2	0
Juve	Roma	2	0
Roma	Milan	0	1

Tabelle e Relazioni Una tabella è una relazione se:

- I valori di ogni colonna sono omogenei
- Le righe sono diverse fra di loro
- Le intestazioni delle colonne sono diverse tra di loro

In una tabella che rappresenta una relazione:

- L'ordinamento tra le righe è irrilevante
- L'ordinamento tra le colonne è irrolevante

Relazione

Relation: relazione matematica (teoria degli insiemi)
Relationship: rappresenta una associazione nel modello Entity-Relationship

2.8.4 Modello basato su valori

I riferimenti fra dati in relazioni diverse sono rappresentati per mezzo di valori dei domini che compaiono nelle n-uple

Vantaggi

- Indipendenza dalla struttura fisiche (si potrebbe avere anche con puntatori HL)
- Si rappresenta solo ciò che rilevante dal punto di vista dell'applicazione
- Utente finale vede stessi dati del programmatore
- Portabilità dei dati tra sistemi
- Puntatori direzionali

2.8.5 Gestire i valori NULL

Ogni elemento in una tabella può essere o un valore del dominio oppure il valore nullo NULL

IL MODELLO RELAZIONALE IMPONE UNA STRUTTURA RIGIDA

Le informazioni sono rappresentate per mezzo di n-uple. Solo alcuni formati di n-upla sono ammessi: quelli che corrispondono agli schemi di relazione. I dati disponibili possono non corrispondere al formato previsto

E se usassi il numero 0?

NON CONVIENE, anche se spesso si fa, usare valori del dominio (0, stringa nulla, 99, ...)

- Potrebbero non esistere valori "non utilizzati"
- valori "non utilizzati" potrebbero diventare significativi
- In fase di utilizzo sarebbe necessario tener conto del significato di questi valori

Tipi di valore NULL

Ci sono almeno 3 casi differenti:

- Valore sconosciuto (eg. quanti anni ha?)
- Valore inesistente (eg. non ha il secondo nome)
- Valore non applicabile (eg. anagrafica unica studenti/professori, i professori hanno ufficio)

N.B.: I DBMS NON distinguono i tipi di valore nullo

2.8.6 Vincoli di integrità

Esistono istanze di basi di dati che, pur sintatticamente corrette, non rappresentano informazioni possibili per l'applicazione di interesse.
Un vincolo di integrità è una **proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione**
Un vincolo è una funzione booleana: associa ad ogni istanza il valore vero o falso

Perché

- Descrizione più accurata della realtà
- Contributo alla "qualità dei dati"
- Utili nella progettazione
- Usati dai DBMS nelle interrogazioni

Nota

Alcuni vincoli (ma non tutti) sono supportati dai DBMS

- Possiamo specificare tali vincoli e il DBMS ne impedisce violazione
- Se non supportati, la responsabilità della verifica è dell'utente/programmatore

Tipi di vincoli

- Vincoli **intrarelazionali**
  - vincoli su valori (o di dominio)
  - vincoli di n-upla
- Vincoli **interrelazionali**

2.8.7 Identificare le n-uple

- Non ci sono due ennuple con lo stesso valore sull'attributo Matricola
- Non ci sono due ennuple uguali su tutti e tre gli attributi Cognome, Nome e Data di Nascita

2.8.8 Chiave

Definiamo **Chiave** l'insieme di attributi che identificano le n-uple di una relazione

Formalmente

Un insieme di K attributi è **superchiave** per r se non contiene due n-uple distinte t_1 e t_2 con $t_1^K = t_2^K$
K è **chiave** per r se è una **superchiave minimale** per r
superchiave minimale = non contiene un'altra superchiave

Esistenza

- Una relazione non può contenere n-uple distinte ma uguali
- Ogni relazione ha come superchiave l'insieme degli attributi su cui è definita
- Quindi ha (almeno) una chiave

Importanza

- L'esistenza delle chiavi garantisce l'accessibilità a ciascun dato della base di dati
- Le chiavi permettono di correlare i dati in relazioni diverse: il modello relazionale è basato su valori.

Chiavi e valori NULL

- In presenza di valori nulli, i valori della chiave non permetteranno
  - Di identificare le n-uple
  - Di realizzare facilmente i riferimenti da altre relazioni
- La presenza di valori nulli nelle chiavi deve essere limitata
- Sulla **Chiave primaria** non sono ammessi valori NULL

2.8.9 Integrità referenziale

Esami				**Studenti**		
Studente	**Voto**	**Lode**	**Corso**	**Matricola**	**Cognome**	**Nome**
276545	32		01	276545	Rossi	Mario
276545	30	e lode	02	787643	Neri	Piero
787643	27	e lode	03	787642	Bianchi	Luca
787643	24		04			

- Informazioni in relazioni diverse sono correlate attraverso valori comuni
- In particolare, valori delle chiavi (primarie)
- Le correlazioni debbono essere "coerenti"

Vincolo di integrità referenziale

Un vincolo di integrità referenziale ("foreing key") fra attributi X di una relazione r_1 e un'altra relazione r_2 impone ai valori su X in r_1 di comparire come valori della chiave primaria di r_2

Integrità referenziale e valori NULL

Impiegati			Progetti			
Matricola	**Cognome**	**Progetto**	**Codice**	**Inizio**	**Durata**	**Corso**
34321	Rossi	IDEA	IDEA	01/2000	36	200
53524	Neri	XYZ	XYZ	07/2001	24	120
64521	Verdi	NULL	BOH	09/2001	24	150
73321	Bianchi	IDEA				

Viene eliminata una n-upla causando una violazione

Comportamento standard: Rifiuto dell'operazione Azioni compensative: Eliminazione in casata o introduzione di valori nulli

Eliminazione in casata

Impiegati			Progetti			
Matricola	**Cognome**	**Progetto**	**Codice**	**Inizio**	**Durata**	**Corso**
34321	Rossi	IDEA	IDEA	01/2000	36	200
64521	Verdi	NULL	BOH	09/2001	24	150
73321	Bianchi	IDEA				

Introduzione di valori null

Impiegati			Progetti			
Matricola	**Cognome**	**Progetto**	**Codice**	**Inizio**	**Durata**	**Corso**
34321	Rossi	IDEA	IDEA	01/2000	36	200
53524	Neri	NULL	XYZ	07/2001	24	120
64521	Verdi	NULL	BOH	09/2001	24	150
73321	Bianchi	IDEA				

3 SQL

3.1 Benefici SQL

- Indipendenza dai venditori di HW e SW
- Portabilità attraverso varia piattaforme HW
- Coperto da standard internazionali SQL1, SQL2 e SQL3
- Strategico per IBM, Oracle, Microsoft, ...
- Linguaggio per data base relazionali (unico)
- Strutturato ad alto livello (English-like)
- Linguaggio programmazione (Statico/Dinamico/API)
- In grado di fornire viste diverse del data base
- Linguaggio completo (IF, triggers, ...) con T-SQL e PL-SQL
- Definizione dinamica dei dati
- Client/Server

Portabilità: Davvero?

Non si può fare tutto

- Codici di errore non standard
- Tipi di dati non sempre supportati
- Tabelle di sistema non sono uguali
- Definisce solo linguaggio statico, non dinamico
- Sorting

3.2 SQL Basics

3.2.1 Linguaggi

Data Definition Language [DDL]

```
CREATE / DROP / ALTER
TABLE / VIEW / INDEX
```

Data Manipulation Language [DML]

```
SELECT / INSERT / DELETE / UPDATE
```

Data Control Language [DCL]

```
GRANT / REVOKE
```

Transaction Control Language [TCL o T-SQL]

```
COMMIT / ROLLBACK
```

Programming Language [PL]

```
DECLARE / OPEN / FETCH / CLOSE
```

3.2.2 Comandi

Elencare i database

```
SHOW DATABASES;
```

Ritorna l'elenco dei Database presenti nel DBMS
I comandi possono occupare anche più righe e terminano con il ';'

Creare un database

```
CREATE DATABASE [IF NOT EXISTS] nomeDatabase;
```

Crea un nuovo DataBase con il nome specificato e lo rende accessibile all'utente root.
La condizione 'IF NOT EXISTS' crea il database solo se non esiste già

Eliminare un database

```
DROP DATABASE [IF EXISTS] nomeDatabase;
```

La condizione 'IF EXISTS' elimina il database solo se esiste, altrimenti non fa nulla

Eliminare un database

```
USE nomeDatabase;
```

Tutti i comandi ora saranno riferiti a questo DB.

3.3 Definizione di dati

Istruzione *CREATE TABLE*

- Definisce uno schema di relazione e ne crea un'istanza vuota
- Specifica attributi, domini e vincoli

```
CREATE TABLE [IF NOT EXISTS] nomeTabella(
  nomeAttributo1 tipo,
  nomeAttributo2 tipo,
  ...
  nomeAttributoN tipo
)
```

3.4 Domini

Numeri interi

Tipo	Byte	Minimo	Massimo
TINYINT	1	-128	127
SMALLINT	2	-32768	32767
MEDIUM	3	-8388608	8388607
INT	4	-2147483648	2147483647
BIGINT	8	-2^{63}	$-2^{63}-1$

$INT(N)$: suggeriamo al motore di usare N caratteri per mostrare il dato.

Numeri razionali

Virgola Mobile

- float - 4 bytes
- double - 8 bytes

Virgola Fissa

- $numeric(i, n)$ salva esattamente n cifre decimali
- $decimal(i, n)$ salva almeno n cifre decimali

Testo

Tipo	Descrizione
CHAR	Stringa di lunghezza fissa non binaria
VARCHAR	Stringa di lunghezza variabile non binaria
BINARY	Sequenza binaria a lunghezza fissa
VARBINARY	Sequenza binaria a lunghezza variabile

SALVATI IN TABELLA

Tipo	Descrizione
TINYTEXT	Stringa non binaria piccola
TEXT	Stringa non binaria
MEDIUMTEXT	Stringa non binaria media
LONGTEXT	Stringa non binaria grande

SALVATAGGIO DEDICATO

Generico

Tipo	Descrizione
TINYBLOB	Binary Large OBject piccolo
BLOB	Binary Large OBject
MEDIUMBLOB	Binary Large OBject medio
LONGBLOB	Binary Large OBject grande

SALVATAGGIO DEDICATO

Tempo

- YEAR - anno nel formato $YYYY$
- DATE - data nel formato $YYYY - MM - DD$
- TIME - tempo nel formato $hh : mm : ss$
- DATETIME - tempo nel formato $YYYY - MM - DD\ hh : mm : ss$
- TIMESTAMP - come DATETIME, ma si aggiorna da solo

Spazio

Tipo	Descrizione
GEOMETRY	Valore spaziale di qualsiasi tipo
POINT	Coordinate X, Y
LINESTRING	Curva (uno o più POINT)
POLYGON	Un poligono
	...e molti altri

Stringhe

Sto salvando testo o sequenze di byte?

- Testo: devo convertire la stringa in sequenza di byte (charset)
- Sequenza e basta: posso salvarla così com'è

Dimensione fissa o variabile?

- Fissa: devo indicare una dimensione (max 255)
- Variabile: occupa lunghezza + 1; posso indicare una lunghezza massima

Valore	CHAR(4)	Spazio	VARCHAR(4)	Spazio
' '	'____'	4 byte	' '	1 byte
' ab '	' ab__'	4 byte	' ab '	3 byte
' abcd '	' abcd '	4 byte	' abcd '	5 byte
' abcdef '	' abcd '	4 byte	' abcd '	5 byte

Dove salvo il dato?

- Nella tabella: più rapido accedere al dato per interrogazioni
- Storage dedicato: anche se ho molti dati la tabella resta piccola

3.5 Tabelle

Creare una Tabella

Studenti

Matricola	Cognome	Nome
276545	Rossi	Mario
787643	Neri	Piero
787642	Bianchi	Luca

```sql
CREATE TABLE Studenti(
  matricola int(11),
  cognome varchar(45),
  nome varchar(45)
);
```

Cancellare una tabella

```
DROP TABLE nomeTabella;
```

3.6 Vincoli

Posso definire dei vincoli:

- PRIMARY KEY: chiave primaria (una sola, implica NOT NULL)
- NOT NULL
- UNIQUE: definisce chiavi
- CHECK: vedremo più avanti

Chiave Primaria

```
CREATE TABLE Studenti(
  matricola int(11) PRIMARY KEY,
  cognome varchar(45),
  nome varchar(45)
);
```

analogo a

```
CREATE TABLE Studenti(
  matricola int(11),
  cognome varchar(45),
  nome varchar(45),
  PRIMARY KEY (matricola)
);
```

Proibire i NULL

```
CREATE TABLE Studenti(
  matricola int(11) PRIMARY KEY,
  cognome varchar(45) NOT NULL,
  nome varchar(45) NOT NULL
);
```

Chiavi composte

Esami

Studente	Voto	Lode	Corso
276545	32		01
276545	30	e lode	02
787643	27	e lode	03
787643	24		04

```sql
CREATE TABLE Esami(
  studente int(11),
  voto smallint NOT NULL,
  lode bool,
  corso int(11),
  PRIMARY KEY (studente, corso)
);
```

NOT NULL + UNIQUE = PRIMARY KEY

```sql
CREATE TABLE Corsi(
  codice int(11) NOT NULL UNIQUE,
  titolo varchar(45) NOT NULL,
  docente varchar(45)
);

CREATE TABLE Esami(
  studente int(11) NOT NULL UNIQUE,
  voto smallint NOT NULL,
  lode bool,
  corso int(11) NOT NULL UNIQUE
);
```

UNIQUE su più colonne

```sql
CREATE TABLE nomeTabella (
        id int(11) PRIMARY KEY,
        campo1 int(19),
        campo2 int(12),
        CONSTRAINT [nome] UNIQUE(campo1, campo2)
    );
```

AUTO INCREMENT

- Il motore si occupa di incrementare il contatore numerico
- Identifico in modo chiaro una n-upla
- Ottimo come chiave primaria

```
CREATE TABLE Studenti(
  matricola int(11) PRIMARY KEY AUTO_INCREMENT,
  cognome varchar(45),
  nome varchar(45)
);
```

CHECK

Serve per specificare vincoli complessi (eg NETTO = LORDO - TRATTENUTE)

- Non supportato in MySQL
- MySQL ignora il comando dato alla creazione della tabella

Dettagli:
Cosa succede se cancello una riga?

- Non riciclo
- Successivo = inserito automaticamente + 1

Cosa succede se imposto un valore?

- Se non è duplicato viene accettato
- Successivo = inserito manualmente + 1

Cosa succede se modifico un valore?

- Se non è duplicato viene accettato
- Successivo = inserito automaticamente + 1

Valori predefiniti

```
CREATE TABLE nomeTabella(
        nomeAttributo tipo DEFAULT valore
    );
```

Commenti

```
CREATE TABLE nomeTabella(
        nomeAttributo tipo COMMENT 'commento'
    );
```

3.6.1 Vincoli interrelazionali

- *FOREIGN KEY* e *REFERENCES* permettono di definire vincoli di integrità referenziale
- Abbiamo due sintassi:
  - Per singoli attributi (non in MySQL)
  - Su più attributi
- É possibile definire azioni compensative

FOREIGN KEY: Colonne che sono FK
REFERENCES: colonne nella relazione (tabella) esterna

```
CREATE TABLE Esami(
        studente int(11),
        voto smallint NOT NULL,
        lode bool,
        corso int(11),
        PRIMARY KEY (studente, corso),
        FOREIGN KEY (studente) REFERENCES Studenti(
   matricola)
);
```

Definizione compatta:

```
CREATE TABLE Esami(
        studente int(11) REFERENCES Studenti(matricola),
        voto smallint NOT NULL,
        lode bool,
        corso int(11),
        PRIMARY KEY (studente, corso)
);
```

Vincoli interrelazionali con nome

```
CREATE TABLE Esami(
        studente int(11),
        voto smallint NOT NULL,
        lode bool,
        corso int(11),
        PRIMARY KEY (studente, corso),
        CONSTRAINT FK_Studente FOREIGN KEY (studente)
   REFERENCES Studenti(matricola)
        CONSTRAINT FK_Corso FOREIGN KEY (corso) REFERENCES
   Corsi(codice)
);
```

Disattivare i vincoli interrelazionali

- Sto caricando i dati: ordine importante, altrimenti ho un errore
- Voglio disattivare temporaneamente i vincoli
- disattivazione: SET foreign_key_checks = 0
- riattivazione: SET foreign_key_checks = 1

3.7 Integrità referenziale

Tabella Vigili:

```
CREATE TABLE Vigili(
        matricola int(11) PRIMARY KEY AUTO_INCREMENT,
        cognome varchar(45) NOT NULL,
        nome varchar(45) NOT NULL
);
```

Vigili

Matricola	Cognome	Nome
3987	Rossi	Luca
3295	Neri	Piero
9345	Neri	Mario
7543	Mori	Gino

Tabella Automobili:

```
CREATE TABLE Automobile(
        prov char(2),
        targa char(6),
        cognone varchar(45) NOT NULL,
        nome varchar(45) NOT NULL,
        PRIMARY KEY (prov, targa)
 );
```

Automobile

Prov	**Targa**	Cognome	Nome
MI	39548K	Rossi	Mario
TO	E39548	Rossi	Mario
PR	839548	Neri	Luca

Tabella infrazioni:

```
CREATE TABLE Infrazioni(
        codice int(11) PRIMARY KEY AUTO_INCREMENT,
        data datetime,
        vigile int(11),
        prov char(2),
        targa char(6),
        FOREIGN KEY (vigile) REFERENCES Vigili(matricola),
        FOREIGN KEY (prov, targa)
        REFERENCES Automobile(prov, targa)
);
```

Infrazioni

Codice	Data	Vigile	Prov	Targa
34321	1/1/95	3987	MI	39548K
53524	4/3/95	3295	TO	E39548
64521	5/4/96	3295	PR	839548
73321	5/2/98	9345	PR	839548

3.8 Cambiare lo schema

3.8.1 Modificare Tabelle

```
ALTER TABLE nomeTabella
    azione1 [, azione2, ...]
```

- Aggiungere / togliere colonne

- Cambiare il tipo di dato
- Rinominare la tabella
- Definire chiavi primarie, esterne, etc.

Aggiungere colonne

```
ALTER TABLE nomeTabella  ADD COLUMN definizioneColonna
  [ FIRST | AFTER nomeColonna ]
```

Rimuovere colonne

```
ALTER TABLE nomeTabella  DROP COLUMN nomeColonna
```

Modificare colonne

```
ALTER TABLE nomeTabella CHANGE COLUMN nomeOriginale
   nomeNuovo tipo
```

Rinominare tabelle

```
ALTER TABLE nomeTabella RENAME TO nuovoNome
```

Aggiungere / Rimuovere Foreign Key

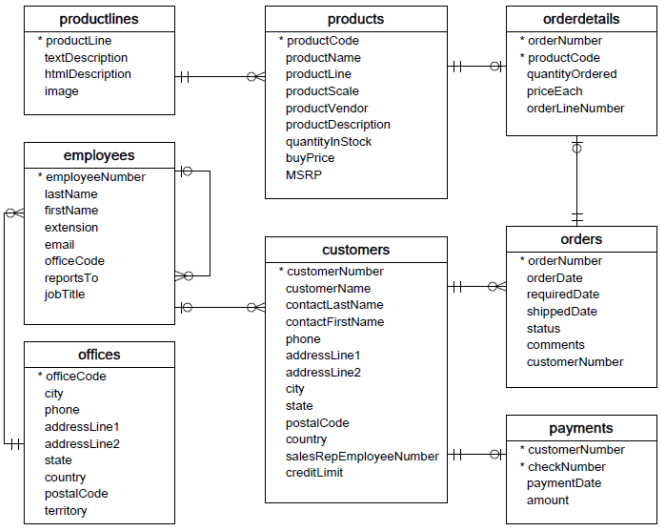
```
ALTER TABLE nomeTabella
    ADD CONSTRAINT nome
    FOREIGN KEY (...)
    REFERENCES tabella(...)
```

Per rimuoverle:

```
ALTER TABLE nomeTabella
    DROP FOREIGN KEY nome
```

4 Manipolazione di Database

Per tutti i comandi andremo ad utilizzare il database di prova 'classicmodels' strutturato nel modo seguente:



L'url per scaricarlo: `https://www.mysqltutorial.org/getting-started-with-mysql/mysql-sample-database-aspx/`

4.1 Istruzione SELECT

```
SELECT attributo1 [, attributo2, ...]
FROM tabella1 [, tabella2, ...]
[WHERE condizione]
```

- $*$: vogliamo indicare tutti gli attributi
- $FROM$: da che tabella
- $WHERE$: quali ennuple, solo quelle dove è soddisfatta la condizione

Esempio: mostra nome, prezzo di acquisto e di vendita dei modellini che costano meno di 75$

```
SELECT productName, buyPrice, MSRP FROM products
WHERE MSRP < 75;
```

4.2 Istruzione AS

L'istruzione AS ha lo scopo di rinominare gli attributi

```
SELECT  productName AS nomeProdotto,
        productVendor AS nomeVenditore
FROM products;
```

4.3 Condizioni: Testo esatto

Esempio: Mostra tutti i dipendenti di nome 'Leslie'

```
SELECT * FROM employees
WHERE firstName = 'Leslie';
```

Virgolette singole o doppie? è indifferente! Anche se lo standard ANSI dice singole
Come posso inserire una virgoletta in una stringa? Basta raddoppiarla.
Esempio: Ci'ao lo scrivo come 'Ci''ao'

4.4 Condizioni: Testo incompleto

Esempio: Mostra tutti i dipendenti il cui cognome finisce per 'son'

```
SELECT * FROM employees
WHERE lastName LIKE  '%son';
```

- % : zero o più caratteri
- _ : esattamente un carattere
- Per cercare il carattere % uso \%
- Per cercare il carattere _ uso _

4.5 Intervalli

Seleziona i valori compresi tra x e Y (inclusi)

```
SELECT ... FROM ...
WHERE colonna BETWEEN x AND y;
```

4.6 Liste

Controlla se il valore è presente in una lista di valori

```
SELECT ... FROM ...
WHERE colonna IN (val1, val2, ...);
```

4.7 Gestire i NULL

```
SELECT ... FROM ...
WHERE colonna IS NULL;
```

4.8 Funzioni

Per le **stringhe**:

- length()
- reverse()
- right()
- trim()
- ...

Per **Data e Ora**:

- day()
- year()
- now()
- month()
- monthname()
- ...

4.9 Ordinamento

#249