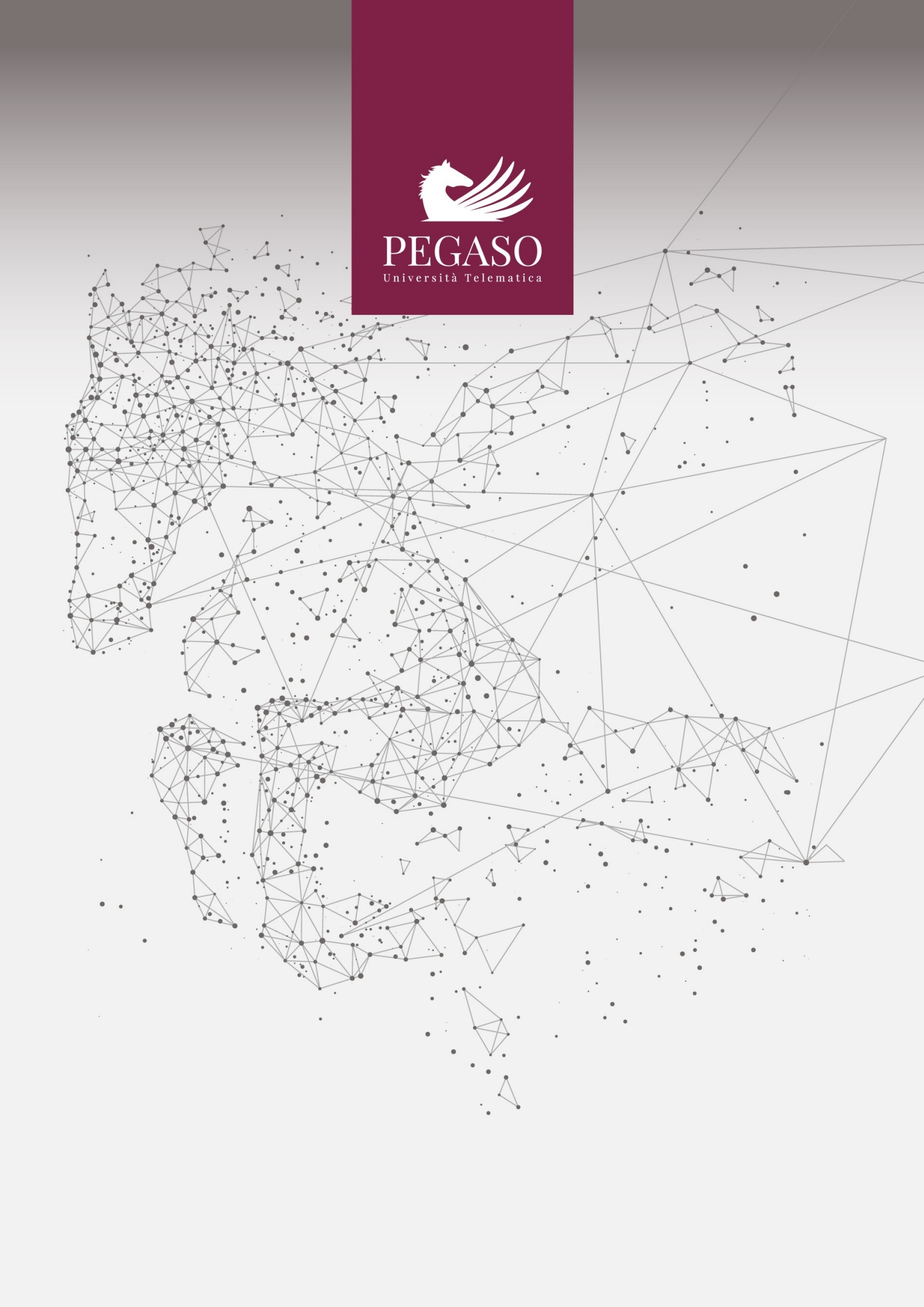
PEGASO
Università Telematica

Indice

1. Introduzione

In questa unità didattica vengono introdotti i modelli dei dati attraverso il concetto fondamentale di transazione, approfondimento sui DBMS e illustrazione dei primi modelli dei dati.

2. La transazione

Una transazione, nel contesto di un database, è un'unità logica che viene eseguita indipendentemente per il recupero dei dati o gli aggiornamenti. Nei database relazionali, le transazioni del database devono essere atomiche, coerenti, isolate e durevoli: sono riassunte come acronimo *ACID*. L'acronimo ACID definisce le proprietà di una transazione di database, come segue:

- **Atomicità**: una transazione deve essere completa, salvata (impegnata) o completamente annullata (rollback). Una vendita in un database di negozi al dettaglio illustra uno scenario che spiega l'atomicità, ad esempio la vendita consiste in una riduzione dell'inventario e un record di contanti in entrata. Entrambe accadono insieme o non accadono - è tutto o niente.
- **Coerenza**: la transazione deve essere pienamente conforme allo stato del database com'era prima della transazione. In altre parole, la transazione non può rompere i vincoli del database. Ad esempio, se la colonna *Numero di telefono* di una tabella di database può contenere solo numeri, la coerenza impone che qualsiasi transazione che tenta di immettere una lettera alfabetica non possa essere svolta.
- **Isolamento**: i dati delle transazioni non devono essere disponibili per altre transazioni fino a quando la transazione originale non viene confermata o ripristinata.
- **Durata**: le modifiche dei dati delle transazioni devono essere disponibili, anche in caso di errore del database.

Le transazioni sono un concetto fondamentale di tutti i sistemi di database. Il punto essenziale di una transazione è che raggruppa più passaggi in un'unica operazione, tutto o niente. Gli stati intermedi tra i passaggi non sono visibili ad altre transazioni concorrenti e, se si verificano alcuni errori che impediscono il completamento della transazione, nessuno dei passaggi interessa il database. Le transazioni sono completate dalle istruzioni SQL COMMIT o ROLLBACK, che indicano l'inizio o la fine di una transazione.

Ad esempio, si consideri un database bancario che contiene i saldi per i vari conti dei clienti, nonché i saldi dei depositi totali per le filiali. Supponiamo di voler registrare un pagamento di € 100,00 dall'account di Rossi all'account di Neri. Semplificando molto, i comandi SQL per questo potrebbero apparire come:

```
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Rossi';
UPDATE branches SET balance = balance - 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name =
'Rossi');
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Neri';
UPDATE branches SET balance = balance + 100.00
    WHERE name = (SELECT branch_name FROM accounts WHERE name =
'Neri');
```

I dettagli di questi comandi non sono importanti qui, anche perché il linguaggio SQL verrà affrontato in seguito; il punto importante è che ci sono diversi aggiornamenti separati coinvolti per realizzare questa operazione piuttosto semplice. I funzionari della nostra banca vorranno essere certi che tutti questi aggiornamenti si verificano, o che nessuno di essi si verifichi. Certamente non farebbe in modo che un errore del sistema provochi che Neri riceve € 100,00 che non sono stati addebitati da Rossi. Rossi non sarebbe rimasta a lungo un cliente felice se fosse stata addebitata senza l'accredito di Neri. Abbiamo bisogno di una garanzia che se qualcosa va storto a metà dell'operazione, nessuno dei passi eseguiti finora avrà effetto. Il raggruppamento degli aggiornamenti in una transazione ci dà questa garanzia. Si dice che una transazione sia atomica: dal punto di vista di altre transazioni, o si verifica completamente o per nulla.

Vogliamo anche garantire che una volta che una transazione è completata e riconosciuta dal sistema di database, è stata effettivamente registrata in modo permanente e non andrà persa, anche se un incidente si verificherà poco dopo. Ad esempio, se stiamo registrando un prelievo di contanti da parte di Neri, non vogliamo alcuna possibilità che il debito sul suo conto scompaia in un incidente subito dopo che esce dalla porta della banca. Un database transazionale garantisce che tutti gli aggiornamenti effettuati da una transazione siano registrati in una memoria permanente (ad esempio, su disco) prima che la transazione venga riportata completa. Un'altra importante proprietà dei database transazionali è strettamente correlata alla nozione di aggiornamenti atomici: quando più transazioni vengono eseguite contemporaneamente, ciascuna di esse non dovrebbe essere in grado di vedere le modifiche incomplete apportate da altri. Ad esempio, se una transazione è occupata a totalizzare tutti i saldi di filiale, non farebbe per includere l'addebito dal ramo di Rossi ma non il credito alla filiale di Neri, né viceversa. Quindi le transazioni devono essere "tutto o niente" non solo in termini del loro effetto permanente sul database,

ma anche in termini di visibilità nel momento in cui si verificano. Gli aggiornamenti fatti fino ad ora da una transazione aperta sono invisibili ad altre transazioni fino al completamento della transazione, dopo di che tutti gli aggiornamenti diventano visibili contemporaneamente. Nel DBMS PostgreSQL, una transazione viene impostata circondando i comandi SQL della transazione con i comandi BEGIN e COMMIT. Quindi la nostra transazione bancaria sembrerebbe in realtà:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Rossi';
-- etc etc
COMMIT;
```

Se, a metà della transazione, decidiamo di non voler eseguire il commit (forse abbiamo appena notato che il saldo di Rossi è negativo), possiamo impartire il comando ROLLBACK invece di COMMIT, e tutti i nostri aggiornamenti finora saranno annullati. PostgreSQL tratta effettivamente ogni istruzione SQL come eseguita all'interno di una transazione. Se non si emette un comando BEGIN, ogni singola istruzione ha un COMMIT implicito e un COMMIT (se riuscito) generale. Un gruppo di istruzioni circondate da BEGIN e COMMIT viene talvolta chiamato *blocco di transazione*. È possibile controllare le dichiarazioni in una transazione in modo più granulare attraverso l'uso di punti di salvataggio. I punti di salvataggio consentono di scartare le parti della transazione in modo selettivo, mentre il resto è impegnato. Dopo aver definito un punto di salvataggio con SAVEPOINT, si può, se necessario, tornare al punto di salvataggio con ROLLBACK TO. Tutte le modifiche del database della transazione tra la definizione del punto di salvataggio e il rollback su di esso vengono eliminate, ma vengono mantenute le modifiche precedenti al punto di salvataggio.

Dopo essere tornato su un punto di salvataggio, continua a essere definito, in modo da poter tornare indietro più volte. Viceversa, se si è certi che non sarà necessario eseguire nuovamente il rollback su un particolare punto di salvataggio, sarà possibile rilasciarlo, in modo che il sistema possa liberare alcune risorse. Il rilascio o il rollback su un punto di salvataggio rilascerà automaticamente tutti i punti di salvataggio definiti successivamente. Tutto ciò sta accadendo all'interno del blocco della transazione, quindi nessuno di essi è visibile ad altre sessioni del database. Quando e se si esegue il commit del blocco di transazione, le azioni confermate diventano visibili come unità ad altre sessioni, mentre le azioni di cui è stato eseguito il rollback non diventano mai visibili.

Ricordando il database bancario, supponiamo di addebitare € 100,00 dall'account di Rossi e di accreditare l'account di Neri, per poi scoprire che avremmo dovuto accreditare l'account di Verdi. Potremmo farlo usando punti di salvataggio come questo:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Rossi';
SAVEPOINT my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Neri';
-- oops ... forget that and use Verdi's account
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Verdi';
COMMIT;
```

Questo esempio è, ovviamente, semplicistico, ma c'è un sacco di controllo possibile in un blocco di transazione attraverso l'uso di punti di salvataggio. Inoltre, ROLLBACK TO è l'unico modo per riprendere il controllo di un blocco di transazione che è stato messo in stato di sospensione dal sistema a causa di un errore, a meno che non sia stato ripristinato completamente e riavviato.

3. DBMS

Una gestione efficiente dei dati, nel mercato competitivo odierno, è una dei compiti importanti di un'azienda per essere competitiva. È un compito difficile ottenere le informazioni giuste al momento giusto per prendere la decisione giusta. Pertanto, il successo di un'organizzazione dipende ora più che mai dalla sua capacità di acquisire dati accurati, affidabili e tempestivi sulla sua attività o operazione per un processo decisionale efficace. Per quanto riguarda l'organizzazione dei dati, i DBMS che li gestiscono devono avere anche le due caratteristiche di **efficienza** ed **efficacia**:

I DBMS debbono essere...**efficienti**

- Cercano di utilizzare al meglio le risorse di spazio di memoria (principale e secondaria) e tempo (di esecuzione e di risposta);
- I DBMS, con tante funzioni, rischiano l'inefficienza e per questo ci sono grandi investimenti e competizione;
- L'efficienza è anche il risultato della qualità delle applicazioni.

I DBMS debbono essere...**efficaci**

- Cercano di rendere produttive le attività dei loro utilizzatori, offrendo funzionalità articolate, potenti e flessibili.

3.1 DBMS vs. File Systems

La gestione di insiemi di dati grandi e persistenti è possibile anche attraverso sistemi più semplici — gli ordinari file system dei sistemi operativi. Il File System è un sistema generale e di facile utilizzo per archiviare file generali che richiedono meno sicurezza e vincoli. Il DBMS viene utilizzato quando i vincoli di sicurezza sono elevati. Infatti, la ridondanza dei dati è inferiore nel sistema di gestione dei database. Inoltre:

- I file system prevedono forme rudimentali di condivisione: "tutto o niente".
- I DBMS estendono le funzionalità dei file system, fornendo più servizi ed in maniera integrata.
- Nei programmi tradizionali che accedono a file, ogni programma contiene una descrizione della struttura del file stesso, con i conseguenti rischi di incoerenza fra le descrizioni (ripetute in ciascun programma) e i file stessi.
- Nei DBMS, esiste una porzione della base di dati (il catalogo o dizionario) che contiene una descrizione centralizzata dei dati, che può essere utilizzata dai vari programmi.

Infatti, il database e il sistema di gestione del database (DBMS) sono essenziali per la gestione della nostra attività, governi, banche, commercio elettronico, ingegneria, diritto, istruzione, medicina e ogni altro tipo di sforzo umano. Il sistema di elaborazione dei file invece è una raccolta di programmi che memorizzano e gestiscono i file nel disco rigido del computer. D'altra parte, un sistema di gestione del database è una raccolta di programmi che consente di creare e gestire un database. In Tabella 1 è illustrato un riepilogo delle suddette differenze.

Tabella 1: *DBMS Vs. File System*

DBMS	**File Processing System**
Minimal data redundancy problem in DBMS	Data Redundancy problem exits
Data Inconsistency does not exist	Data Inconsistency exist here
Accessing database is easier	Accessing is comparatively difficult
The problem of data isolation is not found in database	Data is scattered in various files and files may be of different format, so data isolation problem exists
Transactions like insert, delete, view, updating, etc are possible in database	In file system, transactions are not possible
Concurrent access and recovery is possible in database	Concurrent access and recovery is not possible
Security of data	Security of data is not good
A database manager (administrator) stores the relationship in form of structural tables	A file manager is used to store all relationships in directories in file systems

4. Il modello dei dati

In questo capitolo introduciamo il concetto di modello di dati, iniziando dalle definizioni dei costrutti principali.

4.1 Schema e istanza di un database

Uno schema di database è la struttura dello scheletro che rappresenta la vista logica dell'intero database. Definisce come sono organizzati i dati e come sono associate le relazioni tra loro. Uno schema di database non contiene dati o informazioni. Formula tutti i vincoli che devono essere applicati sui dati. Esso si coniuga con il concetto di istanza:

- **Lo schema**, sostanzialmente invariante nel tempo, che ne descrive la struttura (aspetto intensionale)
    - es.: le intestazioni delle tabelle.
- **L'istanza**, i valori attuali, che possono cambiare anche molto rapidamente (aspetto estensionale)
    - es.: il "corpo" di ciascuna tabella.

In un determinato stato del database, ogni costrutto di schema ha il proprio insieme corrente di istanze. In particolare, Il DBMS memorizza le descrizioni dei costrutti e dei vincoli dello schema chiamati anche metadati nel catalogo DMBS in modo che il software DBMS possa fare riferimento allo schema ogni volta che è necessario. In Figura 1 abbiamo un esempio di schema (la tabella) e di istanza dei dati (i dati nella tabella).

Insegnamento	Docente	Aula	Ora
Analisi matem. I	Luigi Neri	N1	8:00
Basi di dati	Piero Rossi	N2	9:45
Chimica	Nicola Mori	N1	9:45
Fisica I	Mario Bruni	N1	11:45
Fisica II	Mario Bruni	N3	9:45
Sistemi inform.	Piero Rossi	N3	8:00

Figura 1: *Aspetti estensionali e intensionali di una tabella.*

4.2 I costrutti

I modelli di dati si basano su un insieme di costrutti utilizzati per organizzare i dati di interesse e descriverne la dinamica. Le componenti fondamentali di un costrutto sono le seguenti:

- Meccanismi di strutturazione (o **costruttori di tipo**);
- Come nei linguaggi di programmazione esistono meccanismi che permettono di definire nuovi tipi, così ogni modello dei dati prevede alcuni costruttori.
    - Esempio: il modello relazionale prevede il costruttore relazione, che permette di definire insiemi di record omogenei.

Bibliografia

- Curtin, P., D., Foley, K., Sen, K., Morin, C. (2016). Informatica di base. McGraw-Hill Education.
- Mezzalama, M. and Piccolo, E. (2010). Capire l'informatica. CittàStudi Edizioni.
- Atzeni, P., Ceri, S., Fraternali. P., Paraboschi, S., Torlone, R. (2018). Basi di Dati. McGraw-Hill Education.
- Batini, C., Lenzerini, M. (1988). Basi di Dati. In Cioffi, G. and Falzone, V. (Eds). Calderini. Seconda Edizione.