

Appunti sulla Modellazione Concettuale dei Dati

Basato sulle slide del Prof. Danilo Montesi

16 maggio 2025

Indice

1 Perché la Modellazione Concettuale?	2
2 Il Ciclo di Vita del Design del Database	2
2.1 Design Concettuale	2
2.2 Design Logico	2
2.3 Design Fisico	3
3 Modelli di Dati: Costrutti, Schemi e Istanze	4
4 Il Modello Entità-Relazione (ER Model)	4
4.1 Entità (Entity)	4
4.2 Relazione (Relationship)	4
4.3 Promozione di Relazioni a Entità	5
4.4 Attributi (Attribute)	5
4.5 Cardinalità (Cardinality)	5
4.5.1 Cardinalità delle Relazioni	6
4.5.2 Cardinalità degli Attributi	6
4.6 Identificatori (Chiavi - Keys)	6
4.7 Generalizzazione/Specializzazione (Inheritance)	6
5 Documentazione	7
6 UML (Unified Modeling Language) come Alternativa	7

1 Perché la Modellazione Concettuale?

Partire direttamente a definire tabelle SQL (modello logico) è difficile e rischioso. I problemi principali sono:

- Ci si perde nei dettagli troppo presto.
- Il modello relazionale (tabelle, colonne, tipi) è troppo *rigido* per le fasi iniziali di brainstorming e analisi dei requisiti.

La soluzione è il **Modello Concettuale** (ad esempio, il diagramma Entità-Relazione - ERD):

- Permette di ragionare sulla *realtà di interesse* in modo **indipendente dall'implementazione** specifica (quale DBMS useremo, come saranno le tabelle, ecc.).
- Aiuta a definire le **classi di oggetti** (entità) e le loro **relazioni**.
- Fornisce una **rappresentazione visuale** chiara, utile per la documentazione e la comunicazione con gli stakeholder (anche non tecnici).

Esempio Pratico: Immagina di dover creare un sistema per una biblioteca. Invece di pensare subito a `CREATE TABLE Libri (...)`, con il modello concettuale pensi: “Ok, ho bisogno di Libri, Utenti, e una relazione che dice *Un Utente prende in prestito un Libro*”. Questo è più astratto e flessibile.

2 Il Ciclo di Vita del Design del Database

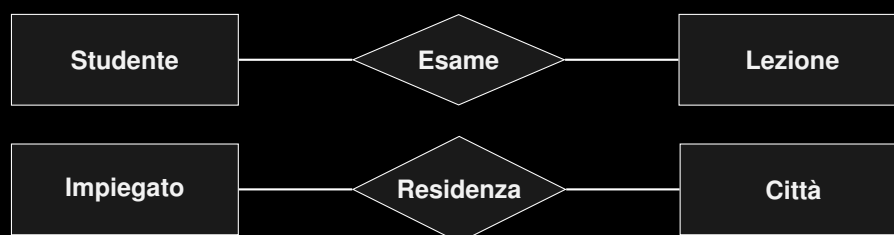
Il design del database è una fase cruciale nello sviluppo di Sistemi Informativi (SI). Le fasi principali del design sono:

2.1 Design Concettuale

- **Input:** Requisiti del database (cosa deve fare il sistema?).
- **Output:** **Schema Concettuale** (es. un diagramma Entità-Relazione - ERD).
- **Focus:** Il “COSA?” – quali informazioni ci servono e come sono collegate, ad alto livello.
- *Esempio Pratico:* “Abbiamo Entità *Studente* e *Corso*. Uno *Studente* si *IscriveA* un *Corso*.”

2.2 Design Logico

- **Input:** Schema Concettuale.
- **Output:** **Schema Logico** (es. definizione di tabelle per un DB relazionale, o collezioni per un DB NoSQL come MongoDB).
- **Focus:** Il “COME?” – come traduciamo il modello concettuale in un modello supportato da un tipo di DBMS (es. relazionale, a documenti, a grafo). È indipendente dal DBMS specifico, ma non dal *tipo* di DBMS.
- *Esempio Pratico (Relazionale):*



- Dallo schema concettuale sopra, un esempio molto *loose* potrebbe essere:

```

-- Table: Studente
CREATE TABLE Studente (
id INT PRIMARY KEY,
nome VARCHAR(100)
);

-- Table: Lezione
CREATE TABLE Lezione (
id INT PRIMARY KEY,
titolo VARCHAR(100)
);

-- Join Table: Esame (between Studente and Lezione)
CREATE TABLE Esame (
studente_id INT,
lezione_id INT,
data DATE,
voto INT,
PRIMARY KEY (studente_id, lezione_id),
FOREIGN KEY (studente_id) REFERENCES Studente(id),
FOREIGN KEY (lezione_id) REFERENCES Lezione(id)
);

-- Table: Impiegato
CREATE TABLE Impiegato (
id INT PRIMARY KEY,
nome VARCHAR(100)
);

-- Table: Città
CREATE TABLE Città (
id INT PRIMARY

```

- *Esempio Pratico (MongoDB/Prisma):* Potresti avere una collezione Studenti con un array di ID_Corso a cui sono iscritti, o una collezione Corsi con un array di ID_Studente iscritti. Prisma ti permette di definire queste relazioni in modo più astratto:

```

model Studente {
  id      Int      @id @default(autoincrement())
  nome    String
  corsi   Corso[]  @relation("Iscrizioni")
}

model Corso {
  id      Int      @id @default(autoincrement())
  titolo  String
  studenti Studente[] @relation("Iscrizioni")
}

```

2.3 Design Fisico

- **Input:** Schema Logico.
- **Output:** **Schema Fisico** (definizioni specifiche per il DBMS scelto: indici, partizionamento, filegroup, ecc.).
- **Focus:** Ottimizzazione delle performance e dello storage.
- *Esempio Pratico:* “Sulla tabella Studenti, creiamo un indice sulla colonna Cognome per velocizzare le ricerche.”

3 Modelli di Dati: Costrutti, Schemi e Istanze

- **Modello di Dati:** Una collezione di “costrutti” (come i tipi di dato in programmazione) per categorizzare i dati e descrivere le operazioni su di essi.
 - Esempio: il modello relazionale usa il costrutto *relazione* (tabella) per insiemi uniformi di tuple (righe).
- **Schema:** La struttura invariante nel tempo dei dati (aspetto *intensionale*).
 - SQL: `CREATE TABLE Users (id INT, name VARCHAR(255));`
 - Prisma: `model User { id Int @id; name String; }`
- **Istanza:** I valori attuali dei dati in un certo momento, che cambiano nel tempo (aspetto *estensionale*).
 - SQL: Le righe effettive nella tabella `Users`: `(1, 'Alice'), (2, 'Bob')`.
 - MongoDB: I documenti effettivi nella collezione `users`.

4 Il Modello Entità-Relazione (ER Model)

È il modello concettuale più usato. Ecco i suoi costrutti principali:

4.1 Entità (Entity)

- Rappresenta una classe di “oggetti” (cose, persone, luoghi) del mondo reale che hanno proprietà comuni e un’esistenza autonoma.
- **Esempi:** *Studente*, *Prodotto*, *Dipartimento*.
- **Rappresentazione Grafica:** Rettangolo.
- **Convenzioni:** Nomi singolari, significativi.
- *Paragone Pratico:* Simile a una classe in OOP, un `model` in Prisma, o una collezione in MongoDB.

4.2 Relazione (Relationship)

- Un legame, un’associazione logica tra due o più tipi di entità.
- **Esempi:** *Studente Frequenta Corso*; *Impiegato LavoraIn Dipartimento*.
- **Rappresentazione Grafica:** Rombo.
- **Convenzioni:** Nomi singolari (se possibile, nomi invece di verbi).
- **Tipi:**
 - **Binarie:** Coinvolgono due entità.
 - **N-arie:** Coinvolgono più di due entità (es. *Fornitore Fornisce Prodotto* a un *Dipartimento*). Spesso si cerca di scomporle in binarie.
 - **Ricorsive:** Un’entità è in relazione con se stessa (es. *Impiegato Supervisiona Impiegato*).
 - * *Paragone Pratico (Ricorsiva):* In SQL, una tabella `Impiegati` con una colonna `ID_Manager` che è una foreign key a `Impiegati.ID`.
- **Ruoli:** Utili nelle relazioni ricorsive per chiarire il significato (es. *Presidente* -(Precedente/Successivo)-> *Successione*).

4.3 Promozione di Relazioni a Entità

Quando?

- Se una relazione ha attributi propri (es. la relazione Iscrizione tra Studente e Corso ha attributi come DataIscrizione e VotoEsame).
- Se uno studente può sostenere lo stesso esame più volte (es. per migliorare il voto). La semplice relazione Studente-Esame-Corso non cattura i tentativi multipli.

Come? La relazione diventa un'entità "associativa".

- *Esempio Pratico:* La relazione Studente-Iscrizione-Corso diventa: Entità Studente — Relazione HaSostenuto — Entità IstanzaEsame — Relazione Riguarda — Entità Corso. L'entità IstanzaEsame avrà attributi come Data, Voto.
- *SQL:* Questo si traduce in una "join table" o "tabella associativa":

```
CREATE TABLE EsamiSostenuti (  
    ID_Studente INT,  
    ID_Corso INT,  
    Data DATE,  
    Voto INT,  
    PRIMARY KEY (ID_Studente, ID_Corso, Data) -- Data inclusa per tentativi multipli  
);
```

4.4 Attributi (Attribute)

- Una proprietà o caratteristica di un'entità o di una relazione.
- Collega ogni istanza dell'entità/relazione a un valore da un "dominio" (insieme di valori possibili).
- **Esempi:** Nome dell'entità Studente; Data della relazione Esame.
- **Rappresentazione Grafica:** Ovale.
- **Tipi:**
 - **Semplici:** Atomici (es. Età).
 - **Composti:** Possono essere scomposti in sotto-attributi (es. Indirizzo composto da Via, NumeroCivico, Città).
 - * *Paragone Pratico (Composto):* In MongoDB è naturale: `address: { street: "...", city: "..."}` . In SQL, spesso si "appiattiscono" in colonne separate (Via, NumeroCivico, Città) o, se complesso, si mette in una tabella separata.

4.5 Cardinalità (Cardinality)

Specifica il numero minimo e massimo di istanze di un'entità che possono partecipare a una relazione, o il numero di valori che un attributo può assumere.

- **Notazione comune:** (min, max)
 - min = 0: partecipazione opzionale.
 - min = 1 (o più): partecipazione obbligatoria.
 - max = 1: al massimo una.
 - max = N (o *): molte.

4.5.1 Cardinalità delle Relazioni

- **Esempio:** Impiegato (1,1) — LavoraPer — (0,N) Dipartimento
 - Un Impiegato deve lavorare per **esattamente un** Dipartimento.
 - Un Dipartimento può avere **da zero a molti** Impiegati.
- **Tipi comuni (basati su max):**
 - **Uno-a-Uno (1:1):** Es. Persona (0,1) — Possiede — (0,1) Pacemaker.
 - **Uno-a-Molti (1:N):** Es. Cliente (1,1) — Effettua — (0,N) Ordine.
 - **Molti-a-Molti (M:N):** Es. Studente (0,N) — Frequenta — (0,N) Corso.
 - * *Paragone Pratico (M:N):* In SQL, le relazioni M:N si implementano sempre con una tabella associativa intermedia. Prisma gestisce questo in modo più astratto.

4.5.2 Cardinalità degli Attributi

- (0,1): Attributo opzionale (può essere NULL). Es. NumeroTelefonoSecondario.
- (1,1): Attributo obbligatorio, singolo valore (default). Es. CodiceFiscale.
- (0,N) o (1,N): Attributo multivalore (un'entità può avere più valori per quell'attributo). Es. NumeroTelefono (una persona può avere più numeri).
 - *Paragone Pratico (Multivalore):* In SQL, si usa una tabella separata: Persona(ID_Persona), NumeriTelefono(ID_Persona_FK, Numero). In MongoDB, si usa un array: telefoni: ["123", "456"].

4.6 Identificatori (Chiavi - Keys)

- Un attributo o un insieme di attributi che identificano univocamente ogni istanza di un'entità.
- **Rappresentazione Grafica:** Attributo sottolineato.
- **Tipi:**
 - **Identificatore Interno:** Formato da attributi della stessa entità.
 - * Es. CodiceFiscale per l'entità Persona.
 - * *Paragone Pratico:* PRIMARY KEY in SQL; _id in MongoDB; @id in Prisma.
 - **Identificatore Esterno:** Formato da attributi dell'entità più l'identificatore di un'entità esterna a cui è collegata tramite una relazione con cardinalità (1,1) dal lato dell'entità da identificare. Usato per "entità deboli" che non possono esistere o essere identificate senza l'entità "forte".
 - * Es. NumeroRiga (attributo di RigaOrdine) + ID_Ordine (dall'entità Ordine) identifica univocamente una RigaOrdine. RigaOrdine è un'entità debole rispetto a Ordine.
- Ogni entità deve avere almeno un identificatore.
- Le relazioni di solito non hanno identificatori (se ne hanno bisogno, si promuovono a entità).

4.7 Generalizzazione/Specializzazione (Inheritance)

- Una relazione tra un'entità genitore (superclasse, es. Veicolo) e una o più entità figlie (sottoclassi, es. Automobile, Motocicletta).
- Le figlie sono "tipi di" genitore: ereditano attributi e relazioni del genitore e possono averne di propri.
- **Rappresentazione Grafica:** Freccia (triangolo vuoto) dalle figlie al genitore.
- **Proprietà:**
 - **Ereditarietà:** Le proprietà del genitore sono implicitamente presenti nelle figlie.

- **Copertura (Total/Partial):**
 - * **Totale:** Ogni istanza del genitore DEVE essere un'istanza di (almeno) una delle figlie. (Es. Persona -> Maschio, Femmina).
 - * **Parziale:** Un'istanza del genitore PUÒ essere un'istanza di una figlia (o solo del tipo genitore). (Es. Veicolo -> Automobile, Motocicletta).
- **Disgiunzione (Disjoint/Overlapping):**
 - * **Disgiunta:** Un'istanza del genitore può essere al massimo un tipo di figlia. (Es. Persona è Maschio O Femmina).
 - * **Sovrapposta:** Un'istanza del genitore può essere più tipi di figlia contemporaneamente (raro e più complesso da modellare).
- Di solito ci si concentra su generalizzazioni **Disgiunte (Totali o Parziali)**.
- *Paragone Pratico:*
 - OOP: `class Veicolo {}, class Automobile extends Veicolo {}.`
 - SQL: Ci sono diversi pattern per implementare l'ereditarietà.
 - Prisma: Può essere modellato con campi discriminatori o modelli separati con relazioni.

5 Documentazione

- **Dizionario dei Dati:** Descrive in dettaglio ogni entità, relazione e attributo.
- **Vincoli Non Esprimibili:** Alcuni vincoli di business non possono essere rappresentati graficamente nell'ERD (es. "Lo stipendio di un impiegato non può superare quello del suo manager"). Vanno documentati a parte.
 - *Paragone Pratico:* Questi vincoli si implementano spesso con CHECK constraints in SQL, triggers, o a livello applicativo.

6 UML (Unified Modeling Language) come Alternativa

- UML è un linguaggio di modellazione più ampio, usato per vari aspetti dello sviluppo software.
- Per la modellazione dei dati, si usano principalmente i **Diagrammi delle Classi (Class Diagrams)**.
- Molti concetti ER hanno un equivalente in UML:
 - **Entità -> Classe**
 - **Relazione -> Associazione**
 - **Relazione con attributi -> Classe di Associazione**
 - **Cardinalità:** 1, 0..1, *, 1..*
 - **Identificatori:** {id} accanto all'attributo.
 - **Generalizzazione/Specializzazione:** Freccia con triangolo vuoto verso la superclasse.
 - **Concetti specifici UML:** Aggregazione (rombo vuoto), Composizione (rombo pieno).