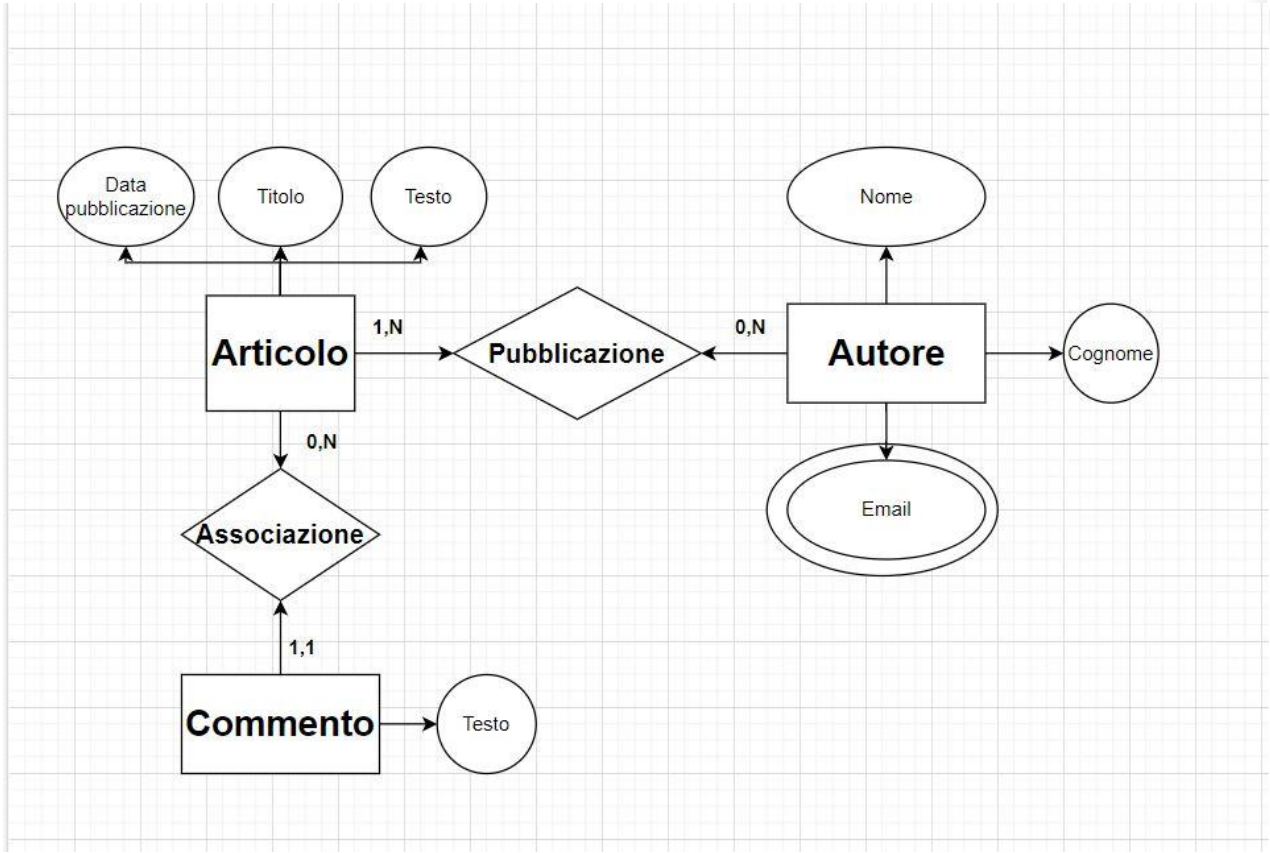
Diagramma ER

Un esempio di diagramma E-R



Caratteristiche del diagramma E-R

Il diagramma E-R (entità relazioni) segue una notazione ben precisa:

- le **entità** del database sono rappresentate all'interno di **rettangoli**;
- ogni entità è definita da una serie di **attributi** rappresentati all'interno di **ellissi** collegate alle entità;
- tra le entità sussistono delle **relazioni**, rappresentate con dei **rombi** collegati alle due (o più) entità coinvolte;
- meno di frequente, anche le relazioni possono avere degli attributi associate.

Entità

Un'**entità** è una classe di oggetti che il database deve gestire, descritti da una serie di attributi, identificabili e distinguibili rispetto ad altre entità.

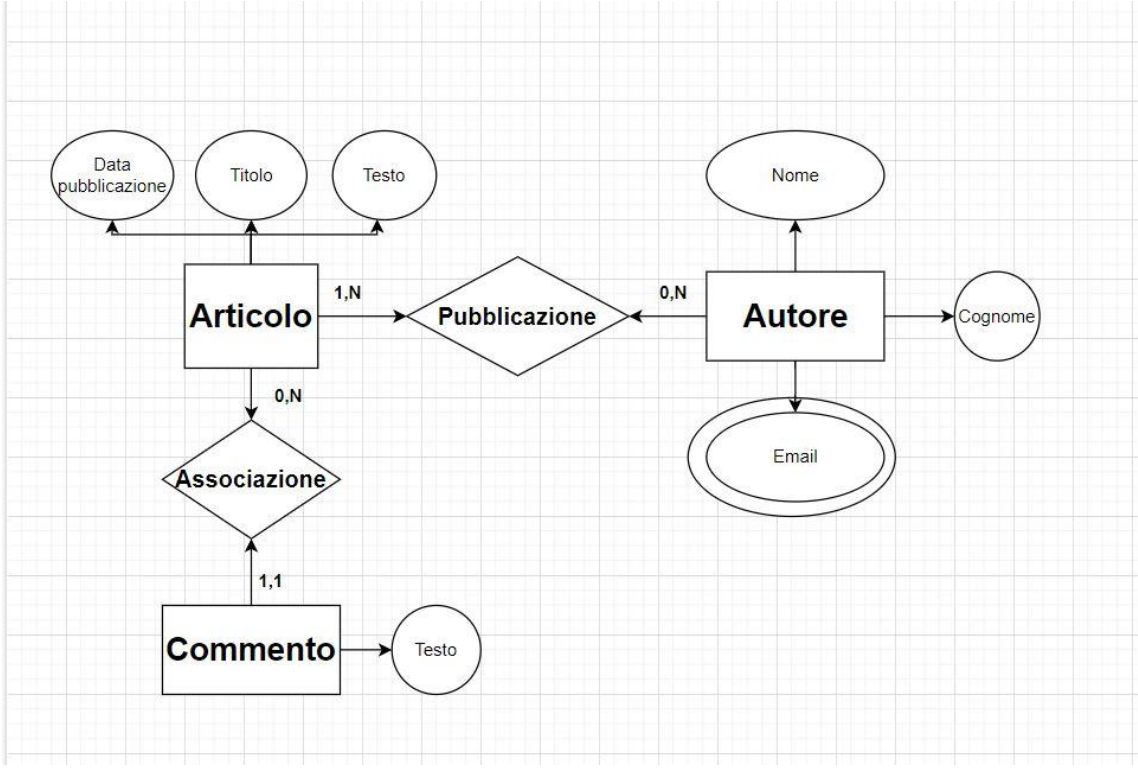
Ecco alcuni esempi:

- database di una banca: ha come entità *il conto corrente, la filiale, il cliente, il dipendente, eccetera*
- database di una scuola: ha come entità *la materia, il professore, lo studente, eccetera*
- database di un blog sul web: ha come entità *l'articolo, l'autore, il commento, eccetera*

Le entità sono rappresentate nel diagramma E-R tramite dei rettangoli.

Tipologie di relazioni

Su ogni freccia tra le entità e le relazioni è presente una coppia di valori (le **cardinalità**): il primo può essere 1 (uno) oppure 0 (zero), mentre il secondo può essere 1 o N.



Tipologia di relazioni

In base al secondo valore di ogni coppia presente sulle frecce del diagramma E-R, possiamo individuare tre tipi di relazione:

- **uno a uno** se per entrambe le entità il valore è 1;
- **uno a molti** se per un'entità il valore è 1 mentre per l'altra è N;
- **molti a molti** se per entrambe le entità il valore è N.

Altre rappresentazioni del diagramma E-R

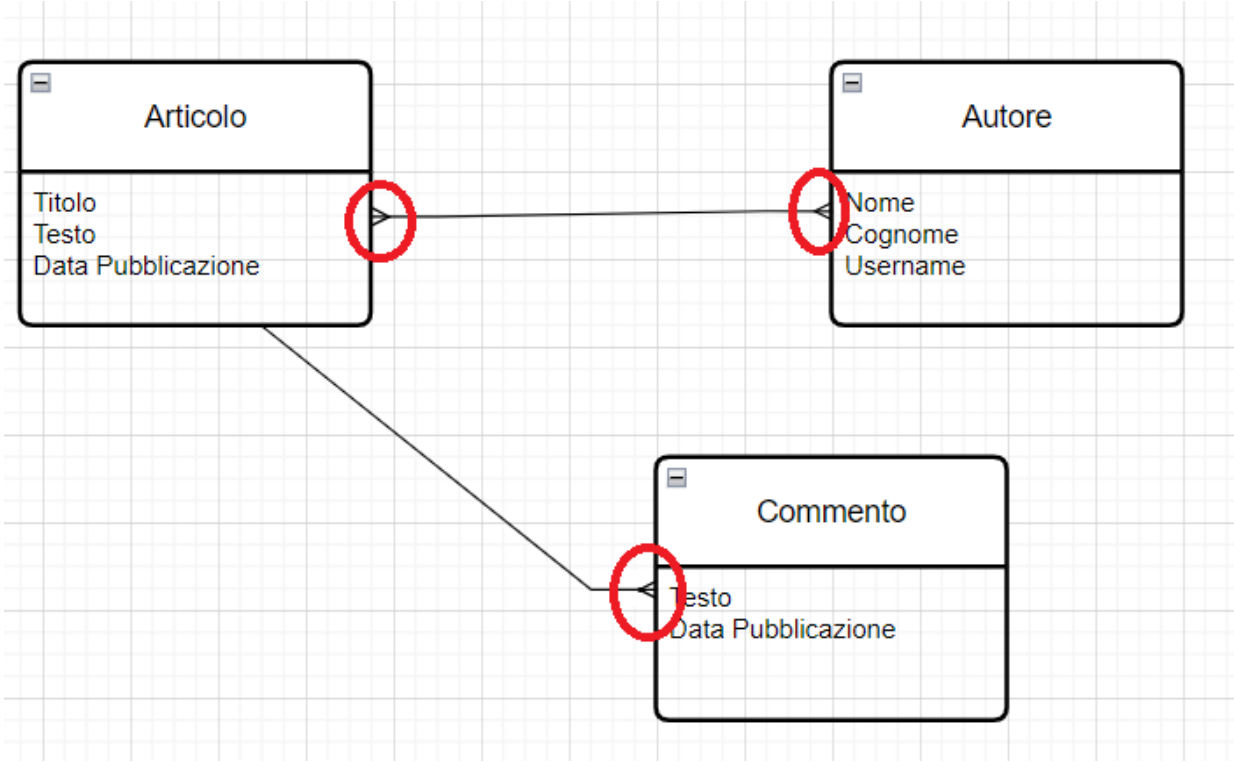
In generale non ci sono delle regole rigide sulla modalità di rappresentazione del diagramma E-R.

In alcuni casi si può usare una rappresentazione più compatta come quella della slide seguente:

- gli attributi sono inseriti in modo da formare un elenco, sotto l'entità
- scompaiono i rombi per le relazioni
- la punta della freccia indica la cardinalità delle relazioni

Diagramma E-R "compatto"

Un *articolo* può avere più *commenti* perché la punta della freccia è *"a tridente"*, mentre un *commento* può essere associato ad un solo *articolo* perché la punta della freccia è *singola*.



Progettazione logica

La progettazione logica dei dati parte dall'output della progettazione concettuale per costruire un database.

La prima scelta da fare è stabilire la tipologia di Database. Spesso la scelta cade sui Database **relazionali**.

A questo punto, a partire dalle entità e dalle relazioni, andrò a definire un insieme di **tabelle** (con i rispettivi **vincoli**).

Esempio di progettazione logica

Traduzione di entità e attributi

Vediamo come passare da un diagramma E-R a un insieme di tabelle:

- ogni **entità** diventa una **tabella**;
- gli **attributi semplici** dell'entità diventeranno le rispettive **colonne**;
- identifichiamo le **chiavi primarie** per ogni tabella (attributo o insieme di attributi che identificano un'istanza di un'entità);
- per ogni **attributo multi-valore** creiamo **un'ulteriore tabella** contenente in ogni riga la chiave primaria dell'entità di partenza e un valore dell'attributo.

Traduzione delle relazioni

Nel nostro caso *Articoli-Autori* sarà una relazione **molti a molti**, mentre *Commenti-Articoli* sarà una relazione **molti a uno**. Procederemo in questo modo:

- creeremo una **nuova tabella** per la relazione molti a molti, contenente le chiavi primarie delle entità coinvolte;

- **aggiungeremo una colonna nella tabella corretta** per gestire la relazione molti a uno. Nel caso che stiamo seguendo occorrerà aggiungere alla tabella *Commenti* (in cui è presente un 1 come secondo numero) la chiave primaria della tabella *Articoli* (in cui è presente un N come secondo numero).

- sulle chiavi primarie aggiunte nelle nuove tabelle inseriremo il vincolo di **chiave esterna**

Progettazione fisica

Durante la progettazione fisica progettiamo l'effettiva implementazione delle tabelle sul particolare RDBS scelto. Ragioniamo in particolare anche su:

- i tipi delle colonne disponibili nel particolare RDBS scelto
- gli indici clustered e non clustered da aggiungere sulle tabelle
- l'implementazione nello specifico linguaggio dei vincoli sui dati (anche tramite oggetti particolari come i trigger e le stored prodecure)
- la creazione di viste e viste materializzate

Implementazione in SQL

Tramite il linguaggio SQL posso creare concretamente il Database su un particolare R-DBMS (Relational-DBMS).

Eseguirò in particolare il comandi CREATE che fa parte delle istruzioni DDL (Data Definition Language)

Tipi delle colonne

Dopo ogni colonna è indicato il tipo. I tipi utilizzati più frequentemente su SQL Server sono

- **Int:** numeri interi
- **Decimal(18,4):** numeri decimali con 18 cifre totali di cui al più 4 decimali
- **Varchar(250):** per stringhe lunghe al più 250 caratteri
- **Nvarchar(250**): per gestire anche caratteri particolari (Unicode data)
- **Date:** date senza orario (su Oracle sarebbe diverso)
- **Datetime:** date con indicazione dell'orario

Il tipo come vincolo

Il tipo di una colonna è in ultima analisi un **Vincolo**: vincoliamo le colonne ad ammettere solo alcune tipologie di dati.

Ad esempio in una colonna di tipo **int** non possiamo inserire un valore che non sia un numero intero. La scelta del tipo di una colonna è dunque un aspetto cruciale per garantire che i dati del database abbiano una qualità minima necessaria per poter svolgere attività di analisi dei dati.

Null / Not Null

Indicare di fianco ad una colonna l'opzione NOT NULL significa garantire che la colonna non potrà mai contenere NULL.

Viceversa, specificando NULL teniamo aperta questa possibilità.

Non specificare nessuna delle due opzioni equivale a specificare NULL.

Chiave primaria

Inserire un vincolo **di chiave primaria** su una colonna significa garantire che in quella colonna non potrà mai essere presente lo stesso valore in più di una riga della tabella.

Inoltre in una colonna chiave primaria non può essere presente NULL.

La chiave primaria può essere definita anche su un insieme di colonne: in questo caso non potrà ripetersi la combinazione di valori presenti in quelle colonne.

Vincolo di chiave esterna

Spieghiamo i vincoli generati dalla definizione di una **chiave esterna** con un esempio.

Nella tabella *Commenti* abbiamo inserito la chiave esterna *IdArticolo* che fa riferimento alla chiave primaria *IdArticolo* della tabella *Articoli.* Sono generati così tre vincoli:

- **Primo vincolo:**

all'interno della tabella *Commenti* non è possibile inserire un valore di *IdArticolo* che non sia **già presente** all'interno della chiave primaria *IdArticolo* della tabella *Articoli*;

Vincolo di chiave esterna

- **Secondo vincolo:**

nel momento in cui è presente una riga nella tabella *Commenti* con un determinato *IdArticolo*, non è possibile cancellare la riga con il medesimo *IdArticolo* nella tabella *Articoli.*

- **Terzo vincolo:**

nel momento in cui è presente una riga nella tabella *Commenti* con un determinato *IdArticolo*, non è possibile modificare il valore di quell'*IdArticolo* all'interno della tabella *Articoli*.

Identity

Se in una colonna indichiamo la proprietà **IDENTITY(1,1)** stiamo chiedendo al database di valorizzare automaticamente quella colonna tramite un numero progressivo che parte da 1 e si incrementa di 1 ad ogni **tentativo** di inserimento.

Osservazione sulle Join

Quando scriviamo una Join nella condizione ON inseriamo nella maggior parte dei casi:

- l'uguaglianza tra la chiave primaria di una tabella e la chiave esterna di un'altra

- l'uguaglianza tra le due chiavi primarie (ma solo se contengono la stessa informazione)

```
SELECT *
FROM   query.Articoli AS A
LEFT JOIN query.Commenti AS C
   ON A.IdArticolo = C.IdArticolo;
```

Operazioni DDL

Creazione di un nuovo schema o Database

In base al particolare R-DBMS potremmo pensare di creare gli oggetti:

- in uno schema già esistente di un Database già esistente
- In un nuovo Schema all'interno di un database già esistente, con l'istruzione CREATE Schema
- in un nuovo Database con l'istruzione CREATE DATABASE (non su SQL A

In base alla scelta avremo ovviamente possibilità diverse di segregare i dati.

ATTENZIONE: su Azure SQL Database l'istruzione CREATE DATABASE **non** è disponibile perché la creazione di un Database richiede logiche differenti

Vincolo di chiave primaria

Creiamo le tabelle relative all'entità *Autore* e all'entità *Commenti* specificando diversamente la chiave primaria. Solo per facilità, continuiamo sullo schema query.

```
CREATE TABLE query.Autori(
    IdAutore INT NOT NULL,
    Nome VARCHAR(100) NOT NULL,
    Cognome VARCHAR(100) NOT NULL,
    PRIMARY KEY (IdAutore));

CREATE TABLE query.Commenti(
    IdCommento INT NOT NULL,
    Testo VARCHAR(1000) NOT NULL,
    PRIMARY KEY (IdCommento));
```

Aggiungere una colonna a una tabella

Gestiamo la relazione tra l'entità *Articolo* e l'entità *Commento* aggiungendo una colonna alla tabella *Commenti*

```
ALTER TABLE query.Commenti ADD IdArticolo INT NOT NULL;
```

Aggiungere un vincolo di Chiave esterna

Aggiungiamo alla colonna *IdArticolo* della tabella *dbo.Commenti* un vincolo di chiave esterna che fa riferimento alla chiave primaria *IdArticolo* della tabella query.*Articoli*

```
ALTER TABLE query.Commenti
ADD FOREIGN KEY (IdArticolo)
    REFERENCES query.Articoli(IdArticolo);
```

Vincoli di chiave nella Create

Creiamo la tabella per gestire l'attributo multi-valore *Email* dell'entità *Autore*

```
CREATE TABLE query.AutoriEmail(
    IdAutore INT NOT NULL,
    Email VARCHAR(100) NOT NULL,
    PRIMARY KEY (IdAutore,Email),
    FOREIGN KEY (IdAutore) REFERENCES query.Autori(IdAutore) );
```

Opzioni ON DELETE e ON UPDATE

Molti R-DBMS permettono di modificare il comportamento del secondo e terzo vincolo con le opzioni ON DELETE e ON UPDATE, definite in fase di creazione della chiave esterna.

Attenzione: utilizziamo queste opzioni solo se siamo davvero sicuri che sia la scelta giusta!

Consideriamo ad esempio questa istruzione

ALTER TABLE dbo.Commenti
ADD FOREIGN KEY (IdArticolo)
REFERENCES dbo.Articoli(IdArticolo)
ON DELETE CASCADE
ON UPDATE CASCADE;

CASCADE

- L'opzione **ON DELETE CASCADE** definisce questo comportamento:

è permesso cancellare una riga della tabella *Articoli*. Ne conseguirà anche la **cancellazione automatica** delle righe referenziate nella tabella *Commenti.*

- L'opzione **ON UPDATE CASCADE** definisce questo comportamento:

è permesso aggiornare il valore di *IdArticolo* all'interno della tabella *Articoli*. Ne conseguirà l'**aggiornamento automatico con lo stesso valore** anche delle righe referenziate nella tabella *Commenti.*

SET NULL

Vediamo come agisce SET NULL al posto di CASCADE.

- L'opzione **ON DELETE SET NULL** definisce questo comportamento:

è permesso cancellare una riga della tabella *Articoli*. Ne conseguirà anche **l'aggiornamento a NULL** dell'*IdArticolo* delle righe referenziate nella tabella *Commenti.*

- L'opzione **ON UPDATE SET NULL** definisce questo comportamento:

è permesso aggiornare il valore di *IdArticolo* all'interno della tabella *Articoli*. Ne conseguirà anche l'**aggiornamento a NULL** delle righe referenziate nella tabella *Commenti.*

Su SQL Server il comportamento di default è equivalente a specificare ON DELETE NO ACTION e ON UPDATE NO ACTION.

Tabella per relazione molti a molti

Gestiamo la relazione molti a molti tra l'entità *Articolo* e l'entità *Autore*

```
CREATE TABLE query.AutoriEmail(
    IdAutore INT NOT NULL,
    Email VARCHAR(100) NOT NULL,
    PRIMARY KEY (IdAutore,Email),
    FOREIGN KEY (IdAutore) REFERENCES query.Autori(IdAutore) );
```

Eliminare definitivamente una tabella

Per eliminare **definitivamente** e **permanentemente** una tabella posso eseguire l'istruzione *Drop Table*

```
DROP TABLE CorsoSQL.query.Articoli;
```

ATTENZIONE: si tratta di un'istruzione che **non lanceremo praticamente MAI in un ambiente di produzione**, se non in casi molto specifici.

Anteponendo anche il nome del Database, aggiungiamo un ulteriore livello di sicurezza.

Eliminare definitivamente il contenuto di una tabella

Per eliminare **definitivamente** e **permanentemente** il contenuto di una tabella (senza eliminare la struttura) posso eseguire l'istruzione *Truncate Table*

```
TRUNCATE TABLE CorsoSQL.query.Articoli;
```

ATTENZIONE: si tratta di un'istruzione che **non lanceremo praticamente MAI in un ambiente di produzione**, se non in casi molto specifici.

Anteponendo anche il nome del Database, aggiungiamo un ulteriore livello di sicurezza.

Normalizzazione

Cos'è la normalizzazione

- La normalizzazione è il processo di ottimizzazione dello schema logico ottenuto dopo (o contestualmente) il processo di progettazione logica.

- Essa riguarda principalmente i **database operazionali** e quindi gli schemi logici ottenuti traducendo diagramma E-R

- La normalizzazione è eseguita al fine di evitare ridondanze dei dati e garantire una corretta interazione con il database, sia per le operazioni di lettura e sia per le operazioni di scrittura.

- In generale la normalizzazione è considerata come un ulteriore step della progettazione logica.

Prima forma normale: definizione

Una tabella è in prima forma normale se ogni colonna contiene valori atomici (cioè indivisibili).

Vediamo qualche esempio di tabelle che **non** sono in prima forma normale.

In questo caso la colonna *Denominazione* non contiene valori atomici.

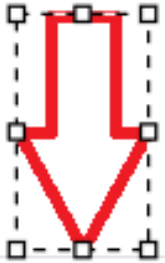
Di conseguenza sarà molto complicato estrarre dalla tabella i clienti con un determinato cognome.

IdCliente	Denominazione
1	Nicola Iantomasi

Prima forma normale: correzione

In questo caso basta creare due colonne separate per dividere l'informazione.

IdCliente	Denominazione
1	Nicola Iantomasi



IdCliente	Nome	Cognome
1	Nicola	Iantomasi

Prima forma normale: altri esempi

Ecco altri due casi di tabelle **non normalizzate.**

L'attributo multi-valore *mail* viene:

- a sinistra tradotto in una singola colonna, dove i valori sono separati da una virgola.
- a destra "spalmato" su più colonne

In entrambi i casi sarà complicato sia effettuare una ricerca tramite mail e sia aggiungere una nuova mail per uno specifico cliente.

IdCliente	Mail
1	nicola.iantomasi@mail.it, nicola.iantomasi@mail2.it

IdCliente	Mail1	Mail2
1	nicola.iantomasi@mail.it	nicola.iantomasi@mail2.it

Prima forma normale: soluzioni

Possiamo risolvere la situazione creando una nuova tabella contenente in ogni riga:

- la chiave primaria della tabella di partenza
- un singolo valore dell'attributo.

In questo modo sarà immediato aggiungere, modificare, cancellare delle mail ed effettuare delle ricerche (usando le JOIN SQL).

Tabella Primaria

IdCliente	Denominazione
1	Nicola Iantomasi

Nuova tabella aggiuntiva

IdCliente	Mail2
1	nicola.iantomasi@mail.it
1	nicola.iantomasi@mail2.it

Seconda forma normale: definizione

Le attività legate alla seconda forma normale coinvolgono le tabelle dove la chiave primaria è composta da una combinazione di colonne (come ad esempio le colonne derivanti da una relazione molti a molti).

Una tabella è in seconda forma normale se:

- è in prima forma normale
- ogni colonna che non appartiene a una chiave dipende da tutte le colonne della chiave, e non soltanto da alcune di esse.

Seconda forma normale: esempio

Vediamo un esempio di tabella che **non** è in seconda forma normale.

Nella tabella *ArticoliAutori* in basso, la chiave primaria è data dalla coppia *(IdArticolo, IdAutore).* In questo esempio la colonna *Titolo* dipende soltanto dalla colonna *IdArticolo.* Di conseguenza l'informazione sarà **ripetuta** per ogni autore che ha partecipato all'articolo.

Tabella ArticoliAutori		
IdArticolo	**IdAutore**	**Titolo**
1	1	I Database
1	2	I Database

La soluzione prevede semplicemente di spostare la colonna *Titolo* nella tabella degli *Articoli*. In effetti, ripensando al modello concettuale, *Titolo* è un attributo dell'entità *Articolo*, non della relazione tra *Articolo* e *Autore*.

Terza forma normale: definizione

Una tabella è in terza forma normale se:

- è in seconda forma normale
- ogni colonna che non appartiene a una chiave dipende dalla chiave e non da altre colonne che non sono chiave

Vediamo un esempio che **non** rispetta la terza forma normale

IdArticolo	Categoria	SottoCategoria
1	Programmazione	SQL
2	Programmazione	Python
3	Office	Excel
4	Programmazione	SQL
5	Office	Word
6	Office	Excel

Terza forma normale: esempio

Le colonne *SottoCategoria* e *Categoria* non sono chiave.

La terza forma normale dunque è violata perché la colonna *Categoria* dipende direttamente dalla colonna *SottoCategoria*.

Questo porta a delle ridondanze. Ad esempio che la *SottoCategoria "SQL"* è legata alla *Categoria "Programmazione"* sarà ripetuto per tutti gli articoli di *SQL*.

IdArticolo	Categoria	SottoCategoria
1	Programmazione	SQL
2	Programmazione	Python
3	Office	Excel
4	Programmazione	SQL
5	Office	Word
6	Office	Excel

Terza forma normale: soluzione

Posso risolvere creando una seconda tabella che riporti l'associazione tra categoria e sotto-categoria.

IdArticolo	Categoria	SottoCategoria
1	Programmazione	SQL
2	Programmazione	Python
3	Office	Excel
4	Programmazione	SQL
5	Office	Word
6	Office	Excel

IdArticolo	SottoCategoria
1	SQL
2	Python
3	Excel
4	SQL
5	Word
6	Excel

SottoCategoria	Categoria
SQL	Programmazione
Python	Programmazione
Excel	Office
Word	Office

Terza forma normale: soluzione

La nuova tabella dovrà avere una chiave primaria. Posso decidere di scegliere la colonna *SottoCategoria* (come nella slide precedente) o di creare una chiave artificiale *IdSottoCategoria*

IdArticolo	Categoria	SottoCategoria
1	Programmazione	SQL
2	Programmazione	Python
3	Office	Excel
4	Programmazione	SQL
5	Office	Word
6	Office	Excel

IdArticolo	SottoCategoria
1	1
2	2
3	3
4	1
5	4
6	3

SottoCategoria	Categoria
1	Programmazione
2	Programmazione
3	Office
4	Office

Denormalizzazione dei database

In alcuni casi particolari si può decidere di fare un'eccezione a quest'ultima regola.

Infatti, aumentare il numero di tabelle può in alcuni casi rendere meno efficienti le operazioni di lettura.

Ad esempio, se nel caso precedente volessi estrarre tutti gli articoli di una determinata *categoria*, dopo la normalizzazione dovrei ricorrere ad una JOIN SQL che interroghi entrambe le tabelle.

Processi di denormalizzazione riguardano spesso i DataWarehouse: infatti, il modello a stella prevede che le tabelle provenienti dalle dimensioni non siano normalizzate.