La strutturazione dei database: la normalizzazione

Nell'articolo precedente abbiamo trattato l'importante concetto della chiave primaria. Oggi vedremo quali sono nella teoria i principali accorgimenti (che prendono il nome di *forme normali*) da attuare nei database per renderli più efficienti e mantenibili e soprattutto per garantire la consistenza dei dati. Nell'ultimo articolo di questa serie vedremo invece come applicare queste norme nella pratica.

Che cos'è la normalizzazione?

Prima di iniziare vediamo una definizione di normalizzazione:

*La **normalizzazione** è un procedimento volto all'eliminazione della ridondanza e del rischio di incoerenza dal database. Esistono vari livelli di normalizzazione (forme normali) che certificano la qualità dello schema del database.*
Questo processo si fonda su un semplice criterio: se una relazione presenta più concetti tra loro indipendenti, la si decompone in relazioni più piccole, una per ogni concetto. Questo tipo di processo non è sempre applicabile in tutte le tabelle, dato che in taluni casi potrebbe comportare una perdita d'informazioni. (fonte: Wikipedia)

Che cosa s'intende con ridondanza?

La ridondanza è una nemica giurata dei database e dobbiamo evitarla come la peste.
Immagina se la tabella che contiene gli articoli di questo blog fosse così:

Id_post	contenuto	Nome_autore	email
1	Testo testo	Maurizio	mau@...
2	Testo testo ...	Maurizio	mau@...

Ora, quando *Maurizio* avrà scritto 1000 articoli (manca poco) ci saranno 2000 dati inutili in questo database.
E' chiaro che i dati dell'autore (il nome, l'email, ecc...) figureranno nella tabella *autori*. Per mettere in relazione un articolo al suo autore, sarà sufficiente riportare nella tabella *articoli* la chiave primaria che fa capo all'autore.

Id_autore	nome	Email
4	Maurizio	mau@...
6	Nando	nando@...

Id_post	contenuto	Id_autore
1	Testo testo	4
2	Testo testo ...	4

La prima forma normale (1NF)

Un database si dice in prima forma normale quando:

- Ogni colonna fa riferimento a dei dati atomici

Questo significa che ogni colonna deve contenere dei valori non divisibili.
Dunque in pratica una colonna *nome_e_cognome* non permetterebbe alla tabella di essere in prima forma normale. Le conseguenze di questo errore sono evidenti. Non potrei fare ad esempio una ricerca mirata per cognome o per nome. Oppure non potrei ordinare i dati per nome o per cognome.
La non atomicità dei dati può prendere forme anche più complesse: considera ad esempio un database contenente l'elenco degli studenti e dei corsi ai quali sono iscritti.

id	Studente	corsi
7	Maurizio	Html,js,php
22	Luigi	Php,mysql

Ho utilizzato solo il nome dello studente, ma considera che la tabella conterrebbe tutti i dati anagrafici.
Ora, questa tabella non è in prima forma normale. I corsi ai quali lo studente è iscritto non sono espressi in forma atomica.
Per ovviare al problema, la tentazione che potrebbe venire sarebbe quella mostrata nell'immagine

sotto:

Id	Studente	Html	Js	Php	mysql
7	Maurizio	1	1	1	0
22	Luigi	0	0	1	1

Ma anche questa non sarebbe una buona soluzione. Ogni nuovo corso introdotto ci obbligherebbe ad aggiungere una colonna alla tabella con **effetti disastrosi sulla mantenibilità**. Inoltre questo è un esempio di *tabella grassa* (fat table), questo termine descrive le tendenza (sbagliata) a voler creare un'unica grande tabella che contenga tutto e mescolando dati che non hanno un legame logico.
La logica invece ci dice che i dati anagrafici degli studenti andranno in una tabella, mentre l'elenco dei corsi in un'altra tabella.
Per descrivere a quali corsi è iscritto uno studente andremo a creare un'altra tabella che metterà in relazione studenti e corsi. Vedremo nel prossimo articolo come.

La seconda forma normale (2NF)

Un database si dice in seconda forma normale quando:

- è in prima forma normale;
- i campi non chiave dipendono dall'intera chiave primaria e non da una parte di essa.

Da una parte di essa? Sì, la seconda forma normale va applicata a tabelle con chiavi composte, quindi potremmo dire che i campi non chiave non possono dipendere da solo una parte dei campi che costituiscono la chiave primaria (una tabella con chiave primaria singola, la possiamo considerare implicitamente in seconda forma normale, in quanto è impossibile che possa violarne i principi).

Facciamo subito un esempio. Considera un evento sportivo dove gli atleti svolgono tre corse. Nel nostro database potremmo dare la seguente rappresentazione:

Codice_atleta	Numero_gara	Nome_atleta	tempo
3455	1	Maurizio	36
3455	2	Maurizio	32
3320	1	Luigi	38
3320	2	Luigi	39
3320	3	Luigi	33

In questa tabella possiamo utilizzare la chiave primaria composta *codice_atleta/numero_gara*.
Ma non è in seconda forma normale.
Infatti il campo non chiave *nome_atleta* dipende da una sola parte della chiave primaria (*codice_atleta*) e non dall'intera chiave.
Per portare questa tabella in seconda forma normale dovremo scomporla in due tabelle

Codice_atleta	Numero_gara	tempo
3455	1	36
3455	2	32
3320	1	38
3320	2	39
3320	3	33

Codice atleta	Nome atleta
3455	Maurizio
3320	Luigi

La terza forma normale (3NF)

Un database si dice in terza forma normale quando:

- è in seconda forma normale (quindi implicitamente anche in prima);
- i campi non chiave non dipendono da altri campi non chiave.

E' molto più semplice di quello che può sembrare. In pratica significa che se un dato può essere calcolato/dedotto/ricostruito a partire da un'altro dato diventa superfluo inserirlo nella tabella.
Facciamo un esempio di un database di ordinazioni di prodotti:

Codice_prodotto	Costo_unitario	Quantità	Prezzo
1040	23	2	46
567	100	6	600
99854	55	1	55

Il campo *prezzo* (non chiave) dipende dai campi *costo_unitario* e *quantità* (non chiave) ed è calcolabile moltiplicando questi ultimi due campi. **Dunque il campo prezzo è inutile e deve essere eliminato.**
Un altro semplice esempio potrebbe essere il database di un e-commerce. Nel nostro store vogliamo indicare i prezzi in euro e in dollari per ogni nostro prodotto. Ora, sarebbe un errore una tabella del genere:

Id_prodotto	Prezzo_euro	Prezzo_dollari
23	100	136
44	120	163
45	90	122

Il prezzo in dollari dipende infatti dal prezzo in euro (o viceversa). **Dunque una colonna va eliminata**. Questo renderà il database più efficiente e razionale. Diversamente dovremmo modificare periodicamente tutti i valori per aggiornarli al cambio.
Il prezzo in dollari lo potremo calcolare al momento dell'estrazione dei dati, magari utilizzando un API di qualche sito di quotazioni on-line; in questo modo il cambio sarà sempre aggiornato in tempo reale.

Conclusione

A questo punto disponi delle basi teoriche per realizzare un database correttamente impostato. Nel prossimo articolo vedremo nella pratica come stabilire delle relazioni logiche tra le tabelle allo scopo di ottenere la migliore efficienza dal database in termini di affidabilità, mantenibilità e razionalità.