Database e SQL

A cura del prof. Gennaro Cavazza

Introduzione ai database

La base di qualsiasi esigenza pratica da parte di una qualsiasi organizzazione è quella di gestire dei dati in maniera logica e conservarli in modo permanente. L'informatica ha messo a disposizione un sistema unico di gestione delle informazioni creando dei semplici archivi poco flessibili ed altrettanto poco performanti. In un secondo momento a questi semplici archivi si sono aggiunti sistemi più potenti chiamati banche dati o base dati. Da qui nasce il termine **database**.

Un database non è altro che una struttura di dati composta da una serie di matrici chiamate **tabelle** a loro volta composte da **campi**. Nasce quindi il concetto di **record**, ovvero un insieme di dati composto da tutti i campi di una tabella.

Credo che un esempio servirà a chiarire le idee. Ipotizziamo un sistema di gestione di una libreria dove si rende necessaria un database ed una prima tabella che contiene gli autori dei libri disponibili. Avremo una struttura formata pressocchè in questo modo:

id	autore
1	J.R.R. Tolkien
2	E.A. Poe
3	B. Stocker

Abbiamo una tabella i cui campi sono **id** ed **autore** ed abbiamo tre record con tre nomi di autori differenti.

Chiarito questo concetto chiariamone un altro: un database è una struttura logica di dati, non un programma! I software di gestione dei database si chiamano **DBMS** che è l'acronimo di Data Base Management System. I DBMS più noti sul mercato sono:

- Microsoft Access
- Microsoft SQL Server
- MySQL
- Oracle

L'implementazione di tutti gli esempi della guida verrà fatta tramite Access, essendo il più semplice da comprendere e da utilizzare rispetto agli altri citati. E' quindi richiesta almeno una minima conoscenza del programma, almeno in merito alla creazione di uno script SQL dalla scheda Query/SQL.

Introduzione all'SQL

L'**SQL** (Structured Query Language) è il linguaggio standard per la manipolazione dei database su qualsiasi DBMS. Lo standard si limita alla sintassi ed al nome dei comandi, ma ogni DBMS parla un dialetto che si differenzia sia pur minimamente dagli altri; nel corso della guida vedremo i casi in cui piccole porzioni di codice sono proprietarie di Access piuttosto che di altri DBMS. Alcuni

DBMS, ad esempio, utilizzando dei comandi proprietari di quello stesso DBMS che non esistono su altri programmi.

Con l'SQL è possibile creare una tabella, modificarne la struttura o cancellarla. E' possibile effettuare ricerche più o meno precise tra i record di una tabella, inserire nuovi dati, modificare o cancellare dati esistenti.
E' possibile mettere in relazione due o più tabelle in funzione della struttura delle tabelle e delle esigenze. A pro di questo tengo a fare una precisazione: l'SQL è un linguaggio molto semplice che si avvicina in maniera impressionante al linguaggio umano e lo stesso utilizzo di un DBMS non è di particolare complessità; la difficoltà di essere un buon progettista di database è quella di avere cognizione di cosa vuol dire gestire un archivio.

Per essere un buon progettista di database, quindi, è necessario entrare nell'ottica citata; lo scopo di questa guida è quella di fornire una buona introduzione al lettore neofita e di chiarire alcuni dubbi al lettore più esperto.

Operatori e tipi di dato

Come un qualsiasi linguaggio di programmazione anche l'SQL utilizza una serie di simboli atti a definire uguaglianze, a fare confronti e calcoli; a pro di questo esistono gli **operatori** e si fa riferimento ai **tipi di dato** che può assumere un campo. In questo capitolo affronteremo questi argomenti.

Iniziamo a definire gli operatori messi a disposizione dagli standard dell'SQL. Gli operatori si dividono in quattro categorie:

- Operatori di confronto
- Operatori aritmetici
- Operatori condizionali
- Operatori logici

Gli **operatori di confronto** servono a determinare uguaglianze e disuguaglianze tra valori e ad effettuare ricerche all'interno dei dati. Di seguito uno schema tabellare:

Operatore	Descrizione
=	Esprime uguaglianza tra due valori
LIKE	Esprime somiglianza tra due valori
<	Stabilisce che un valore è minore di un altro
>	Stabilisce che un valore è maggiore di un altro
<=	Stabilisce che un valore è minore o uguale di un altro
>=	Stabilisce che un valore è maggiore o uguale di un altro
<>	Stabilisce che due valori sono diversi tra loro
BETWEEN	Recupera un valore compreso tra due valori

Gli **operatori aritmetici** effettuano calcoli all'interno di una ricerca. Di seguito uno schema tabellare:

Operatore	Descrizione
+	Effettua un'addizione
-	Effettua una sottrazione
*	Effettua una moltiplicazione
/	Effettua una divisione

L'unico **operatore condizionale** di SQL è **WHERE** (*DOVE*) e serve a definire criteri di ricerca mirati. Nei prossimi capitoli lo vedremo spesso in azione.

Gli **operatori logici** di SQL sono **AND** (*E*) e **OR** (*O, OPPURE*) e servono rispettivamente ad indicare alla ricerca di restituire dati in cui due valori devono rispettivamente: essere entrambi trovati oppure uno solo dei due (o più) valori devono essere trovati. Anche in questo caso vedremo esempi chiarificatori nei prossimi capitoli.

Passiamo ai tipi di dato. Dovendo utilizzare Access per l'implementazione degli esempi, passeremo i tipi da dato, e la nomenclatura esatta, messi a disposizione da Access per l'SQL:

Tipo	Access	Descrizione
AutoIncrement	Contatore	Si incrementa automaticamente di una unità quando viene aggiunto un record e non rigenera mai un numero quando un record viene cancellato
Text	Testo	Testuale, accetta fino a 255 caratteri
Memo	Memo	Testuale, accetta diverse migliaia di caratteri ed occupa più memoria rispetto al precedente
Integer	Numerico	Accetta numeri interi
Float	Numerico	Accetta numeri a precisione singola
Double	Numerico	Accetta numeri a precisione doppia
Byte	Numerico	Accetta numeri interi ma occupa meno memoria di un Integer
Currency	Valuta	Formatta un numero nel formato della valuta selezionata (£, €, $, ecc...)
DateTime	Data/ora	Imposta vari formata (numerici e/o alfanumerici) per la data e l'ora
Bit	Si/No	Booleano, restituisce True o False

CREATE - Creazione di una tabella

Conoscendo Access sappiamo benissimo che è possibile creare una tabella in maniera visuale in pochi passaggi, ma è il caso di imparare a creare una tabella anche via codice. La sintassi per la creazione di una tabella è la seguente

```
CREATE TABLE nome_tabella (nome_campo tipo_dato obbligatorio?)
```

Proviamo a creare la tabella **autori** descritta nel primo capitoli:

```
CREATE TABLE autori
(
    id AutoIncrement,
    autore Text (50) NOT NULL
)
```

Questo è un primo semplice esempio di tabella; creiamo adesso un'altra tabella che più avanti, nel corso della guida, relazioneremo alla tabella autori: la tabella **libri** di cui espongo il codice:

```
CREATE TABLE libri
(
    id AutoIncrement,
    id_autore Integer,
    titolo Text (100) NOT NULL,
    descrizione Memo NOT NULL,
    prezzo Currency
)
```

Utilizzeremo queste due tabelle per i nostri esempi. Vi invito comunque, in seguito ed appena acquisita maggiore dimestichezza, ad effettuare delle prove di creazione di altre tabelle.

ALTER - Modifica strutturale di una tabella

Per modifica di una struttura di una tabella si intende l'aggiunta di un campo, la modifica della nomenclatura di un campo esistente, di un tipo di dato o per stabilire che un campo sia NULL (non richiesto) o NOT NULL (obbligatorio).

La sintassi per aggiungere un campo ad una tabella è la seguente:

```
ALTER TABLE nome_tabella ADD COLUMN nome_campo tipo_dato
```

Proviamo ad aggiungere un campo booleano alla tabella libri per verificare se in un dato momento ci sono scorte del libro richiesto o meno. Il nome del campo sarà **disponibile** di tipo Bit (Si/No):

```
ALTER TABLE libri ADD COLUMN disponibile Bit
```

Vediamo come modificare o cancellare un campo da una tabella. Non eseguite queste operazioni ma limitatevi, per ora, a prendere atto della sintassi, dato che gli esempi che seguono si basano sui campi attualmente esistenti delle due tabelle non modificati.

Modifica di un campo:

```
ALTER TABLE libri MODIFY COLUMN titolo Memo
```

Ho impostato il tipo Memo al campo titolo al posto del tipo Testo.

Cancellazione di un campo:

```
ALTER TABLE libri DROP COLUMN id_autore
```

Questa modifica cancella il campo id_autore e renderebbe impossibile l'associazione del libro al suo autore.

DROP - Cancellazione di una tabella

La cancellazione di una tabella è un'operazione molto delicata da eseguire con estrema attenzione e non tanto per prova: la cancellazione è fisica e con la tabella saranno irrimediabilmente persi tutti i dati ivi contenuti.

Si utilizza il comando **DROP**. La sintassi è la seguente:

```
DROP TABLE nome_tabella
```

Provate a cancellare la tabella utenti con questo comando

```
DROP TABLE utenti
```

e ricreatela seguendo le specifiche del quarto capitolo di questa guida.

INSERT - Inserimento di dati in una tabella

Fino a questo momento abbiamo visto come creare e modificare la struttura di una tabella ed eventualmente come cancellarla fisicamente dal database, utilizzando i comandi della categoria **DDL** (Data Definition Language). Da questo capitolo inizia il percorso con cui impareremo a gestire i dati di una tabella utilizzando i comandi della categoria **DML** (Data Manipulation Language).

Nel quarto capitolo della guida abbiamo creato due tabelle allo scopo di metterle in relazione tra loro negli esempi dell'undicesimo capitolo. Utilizziamo adesso l'istruzione **INSERT** per inserire dati in entrambe le tabelle. La sintassi di INSERT è la seguente:

```
INSERT INTO nome_tanella (elenco_campi) VALUES ('elenco_valori')
```

Facciamo un esempio di inserimento nella tabella utenti:

```
INSERT INTO utenti (autore) VALUES ('J.R.R. Tolkien')
```

Se ben ricordate la tabella utenti è formata dal campi id e dal campo autore ma abbiamo valorizzato solo il campo autore. Il campo id è un contatore (AutoIncrement) e quindi si incrementa da se senza dover andare a specificare alcun valore.

Facciamo adesso un inserimento nella tabella libri:

```
INSERT INTO libri
(
    id_autore,
    titolo,
    descrizione,
    prezzo,
    disponibile
)
VALUES
(
    1,
    'Il Signore degli Anelli',
    'Fantastico capolavoro di genere epico fantastico',
    50,
    True
)
```

Nel campo id_autore ho inserito l'id della tabella autori corrispondente a Tolkien, ovvero l'autore del libro Il Signore degli Anelli. Il valore True passato al campo disponibili non è una stringa ma un comando; in alternativa accetta False.

Lascio a voi il compito di popolare le due tabelle con altri dati in funzione di quanto spiegato in merito all'istruzione INSERT.

Suggerimenti: Provate ad inserire altri due autori i quali, come detto nel primo capitolo della guida, avranno id 2 e 3. Inserite poi una serie di libri scritti da questi autori e nel campo id_autore della tabelle libri inserite l'id corrispondente all'autore del libro definito dal campo id della tabella autori.

SELECT - Interrogazione di una tabella

Eccoci arrivati al capitolo più lungo ed articolato ma se vogliamo anche più semplice ed intuitivo della guida. COn l'istruzione **SELECT** possiamo creare degli script di interrogazione al database, interrogazioni dette **query**.

La sintassi base di una SELECT SQL è la seguente:

```
SELECT * FROM nome_tabella
```

dove ***** sta per *tutti i campi*. La query

```
SELECT id, autore FROM autori
```

estrae tutti i campi dalla tabella autori. Al posto di ***** posso usare i nomi dei campi che mi interessa estrarre. Inseriamo adesso un filtro nella query utilizzando l'operatore WHERE:

```
SELECT autore FROM autori WHERE id = 1
```

Il risultato sarà *J.R.R. Tolkien*. Possiamo anche decidere di conoscere l'id di una tabella in funzione di un altro parametro di ricerca. Ad esempio:

```
SELECT id FROM autori WHERE autore = 'J.R.R. Tolkien'
```

Attenzione: effettuando una ricerca in funzione di un dato di tipo numerico non dobbiamo utilizzare gli apici per racchiudere il valore; in SQL i singoli apici delimitano una stringa.

Creiamo adesso una serie di query sulla tabella libri che, essendo più ampia, ci permette di giocare un po di più.

Estraiamo tutti i titoli dalla tabella libri dove l'id dell'autore è 1 ed il titolo inizia per I utilizzando gli operatori AND per stabilire due condizioni entrambe vere e LIKE per effettuare una ricerca generica:

```
SELECT titolo FROM libri WHERE id_autore = 1 AND titolo LIKE 'I*'
```

L'operatore LIKE necessita del sotto-operatore ***** per identificare *tutto il resto della stringa*. '**I***' vuol dire *tutto ciò che inizia per I.*

LIKE permette di effettuare ricerche su stringhe a partire dall'inizio della stringa, dalla fine della stringa o dalla fine. Rispettivamente potremmo avere:

```
SELECT * FROM libri WHERE titolo LIKE '*Anelli'
```

e/o

```
SELECT * FROM libri WHERE titolo LIKE '*Signore*'
```

impostando semplicemente ***** come conviene. In altri DBMS il simbolo ***** per il LIKE viene sostituito da **%**.

Proviamo ad effettuare quattro ricerche in cui il prezzo è: inferiore a 50 euro; superiore a 50 euro; diverso da 50 euro; compreso tra 30 e 60 euro. Avremo rispettivamente:

```
SELECT * FROM libri WHERE prezzo < 50
SELECT * FROM libri WHERE prezzo > 50
SELECT * FROM libri WHERE prezzo <> 50
SELECT * FROM libri WHERE prezzo BETWEEN 30 AND 60
```

Esiste poi il modo di unire i risultati di due tabelle in un unico risultato con le query di unione, grazie alla clausola **UNION**. Vediamo un esempio:

```
SELECT * FROM autori UNION SELECT * FROM libri
```

Per fare altri esempi dobbiamo immaginare un diverso caso di studio. Create la tabella **utenti** composta dai campi **id** (AutoIncrement), **nome** (Text) e **cognome** (Text). In un simile caso possiamo avere molti nomi o cognomi uguali; se ad esempio avessimo una tabella dove ci sono molti utenti di nome Luca e volessimo estrarre solo una volta il dato Luca in una ricerca, dovremmo utilizzare la clausola **DISTINCT** come segue:

```
SELECT DISTINCT nome FROM utenti WHERE nome = 'Luca'
```

Riprenderemo l'istruzione SELECT nei capitoli 11 e 12, parlando rispettivamente di relazioni e di funzioni di aggregazione dei dati.

UPDATE - Aggiornamento dei dati

I dati di una tabella sono spesso soggetti a cambiamenti ed a modifiche. Un esempio di modifica che potremmo dover effettuare all'interno della nostra tabella dei libri è la variazione del prezzo o della descrizione del libro, trovando una forma pubblicitaria più o meno adatta alle circostanze.

Per aggiornare, o modificare che dir si voglia, i dati di una tabella, si utilizza l'istruzione **UPDATE** come segue:

```
UPDATE nome_tabella SET nome_campo = 'valore'
```

In questo modo verrebbero aggiornati tutti i record di una tabella; per aggiornare un record specifico si utilizza l'operatore condizionale WHERE. Ad esempio:

```
UPDATE nome_tabella
SET
nome_campo = 'valore'
WHERE
campo_di_condizione = 'valoredi_condizione'
```

Proviamo a modificare il prezzo di un libro; portiamo *Il Signore degli Anelli* da 50 1 100 euro:

```
UPDATE libri SET prezzo = 100 WHERE id = 1
```

Abbassiamo di nuovo il prezzo del libro e portiamolo a 60 euro; cambiamo anche la descrizione del libro:

```
UPDATE libri
SET
    prezzo = 60,
    descrizione = 'Un bel libro anche se un po troppo lungo'
WHERE id = 1
```

Utilizziamo la virgola per separare i valori da aggiornare; utilizzeremo invece AND per specificare più parametri di condizione. Proviamo ad esempio ad aggiornare il prezzo di tutti i libri il cui autore è Tolkien ed il prezzo è compreso tra 50 e 100 euro:

```
UPDATE libri
SET
    prezzo = 40
WHERE
    id_autore = 1
AND
    prezzo BETWEEN 50 AND 100
```

DELETE - Cancellazione di dati

I dati di una tabella possono anche diventare obsoleti e si rende necessaria la cancellazione fisica, anche se in gran parte dei casi è sconsigliata, ma dipende dalla sensibilità del dato trattato.

La sintassi dell'istruzione **DELETE** è la seguente:

```
DELETE * FROM nome_tabella
```

In questo modo si cancellano indiscriminatamente tutti i dati dalla tabella specificata e saranno irrimediabilmente persi: prego quindi di prestare attenzione ad eseguire una simile operazione.

Ipotizziamo di cancellare tutti i libri dalla relativa tabella che sono non disponibili, dato che, ad esempio, non saranno mai più disponibili. Vediamo il codice da utilizzare:

```
DELETE * FROM libri WHERE disponibile = False
```

Anche in questo caso è possibile utilizzare più parametri condizionali con l'utilizzo di AND e/o di OR. Fate delle prove.

JOIN - Relazioni tra più tabelle

Lo scopo di una database è quello di conservare i dati in maniera stabile, ma anche quello di organizzarli in forma **normalizzata**, evitando la **ridondanza** dei dati.

Normalizzare un database significa creare una struttura tale in cui i dati sono fisicamente separati tra loro ma possono essere messi insieme con le **relazioni**.

I casi di studio sono molteplici e di diversa natura, in cui si può arrivare a diverse soluzioni, magari altrettanto valide. Prendiamo il caso delle nostre due tabelle, **autori** e **libri**; la tabella libri contiene un campo di riferimento all'autore del libro specificato, il cui nome si trova fisicamente all'interno della tabella autori. Avremmo potuto inserire ogni volta il nome dell'autore nel record in cui sono specificati i dati del libro ed avremmo potuto definire una query del tipo

```
SELECT DISTINCT autore FROM libri
```

per ottenere un report descrittivo di tutti gli autori presenti, elencandoli una singola volta.

La scelta di due tabelle per un'esigenza come quella della nostra libreria informatizzata ha dei pro e dei contro. Il pro è che si ottiene un database normalizzato che evita la ridondanza dei dati; il contro è che l'implementazione di una relazione tra due o più tabelle determina un maggior dispendio di memoria in fase di esecuzione della ricerca. A seconda delle esigenze e delle tecnologie a disposizione potremmo effettuare scelte differenti per ottenere le soluzioni ottimali in funzione delle esigenze pratiche: da qui si determinerà la nostra bravura di progettisti di database.

Veniamo alla pratica.

Esistono due sistemi differenti per implementare una relazione, ovvero utilizzando una semplice clausola WHERE, oppure utilizzando il comando **INNER JOIN**; la differenza tra i due è nel rapporto potenza/dispendio di memoria: il comando JOIN è più performante in termini di potenza ma richiede un maggior impiego di memoria in fase di esecuzione della ricerca.

Facciamo un esempio con la modalità tradizionale in cui ricerchiamo tutti i libri scritti da Tolkien, visualizzando anche il nome dell'autore al posto dell'id di riferimento.

Premetto che per indicare un campo contenuto in una determinata tabella, in SQL si usa la forma *nome_tabella.nome_campo*, ad esempio **libri.titolo**.

Ecco il codice:

```
SELECT
    autori.autore,
    libri.titolo,
    libri.prezzo
FROM
    autori,
    libri
WHERE
    autori.id = libri.id_autore
AND
    autori.id = 1
```

Specifico i nomi dei campi che voglio visualizzare associandoli alla tabella di appartenenza; specifico le tabelle in cui sono contenuti i dati che mi interessano; nel WHERE indico per prima cosa i campi di relazione (detti comunque anche *campi di JOIN*), ovvero effettuo l'associazione tra il'id dell'autore con l'id di riferimento all'autore nel record del libro; con una clausola AND specifico che voglio estrarre i libri scritti dall'autore che ha id = 1, ovvero Tolkien.

Scriviamo la stessa query utilizzando l'istruzione INNER JOIN:

```
SELECT
    autori.autore,
    libri.titolo,
    libri.prezzo
FROM
    autori
INNER JOIN
    libri
ON
    autori.id = libri.id_autore
WHERE
    autori.id = 1
```

Con questo codice: specifico i campi; specifico la prima tabella e poi la seconda, associandole i campi di relazione con la clausola **ON** ed utilizzo il WHERE per stabilire il criterio (o filtro, che dir si voglia) di ricerca.

Divertitevi ad effettuare ricerche in base a dei criteri relazionando le due tabelle.

Funzioni di aggregazione

Le funzioni di aggregazioni sono funzioni standard native di SQL che permettono di ottenere valori numerici e/o effettuare calcoli in funzione di query specifiche. Di seguito l'elenco delle funzioni di aggregazione di SQL in schema tabellare:

Funzione	Descrizione
AVG()	Restituisce la media tra due valori specificati
COUNT()	Restituisce un intero che indica il numero di record trovati
MAX()	Restituisce il valore massimo tra due valori
MIN()	Restituisce il valore minimo tra due valori

SUM()	Restituisce la somma tra più record dello stesso campo

Nell'utilizzo di una funzione di aggregazione è importante (consigliato, anche se non obbligatorio) specificare un *alias* per il risultato, con l'utilizzo della clausola **AS**. I nostri alias si chiameranno, per convenzione, **temp** (che utilizzo in genere per definire un valore temporaneo).

Facciamo qualche esempio.

Restituisce la media del prezzo di tutti i libri trovati:
```
SELECT AVG(prezzo) AS temp FROM libri
```

Restituisce il numero di libri trovati:
```
SELECT COUNT(id) AS temp FROM libri
```

Restituisce il prezzo del libro più costoso:
```
SELECT MAX(prezzo) AS temp FROM libri
```

Restituisce il prezzo del libro meno costoso:
```
SELECT MIN(prezzo) AS temp FROM libri
```

Restituisce la somma dei prezzi di tutti i libri:
```
SELECT SUM(prezzo) AS temp FROM libri
```

ovviamente in funzione di una singola unità. Per sapere il guadagno totale bisognerebbe creare una tabella **vendite**, ad esempio, ed inserirci i libri venduti al relativo prezzo.

Altre funzioni di SQL

Concludiamo questa breve, ma credo utile, guida all'SQL citando una serie di funzioni che i vari DBMS hanno implementato all'SQL per la manipolazione delle stringhe. I DBMS di casa Microsoft, ad esempio, utilizzano le funzioni **LEFT()** e **RIGHT()** per definire delle sottostringhe, ovvero porzioni di una stringa a partire, rispettivamente, dalla sinistra e dalla destra di una stringa.

Vediamo un esempio:

```
SELECT LEFT(titolo, 10) FROM libri WHERE id = 1
```

Il risultato sarà *Il Signore*. Invece:

```
SELECT RIGHT(titolo, 6) FROM libri WHERE id = 1
```

restituirà *Anelli*.

Queste due funzioni, ripeto, sono proprietarie dei DBMS di casa Microsoft, ovvero Access e SQL Server, in quanto nascono da Visual Basic.

Oracle, ad esempio, dispone di simili funzioni chiamate però **SUBSTR()** e **SUBSTRING()**.

Possiamo calcolare la lunghezza della stringa restituita da una query con la funzione **LEN()**. Ad esempio:

```
SELECT LEN(titolo) FROM libri WHERE id = 1
```

Possiamo estrarre i primi N dati da una tabella con la funzione **TOP** come segue

```
SELECT TOP 3 * FROM libri
```

Utilizzo degli Alias

In particolar modo nel capitolo 12, parlando delle funzioni di aggregazione, abbiamo incontrato gli **alias**, ovvero dei nomi temporanei assegnati a delle operazioni; per definire un alias si utilizza la parola chiave **AS** che effettua l'assegnazione di un valore al nome dell'alias.

Riprendiamo un esempio del capitolo 12, quello in cui contiamo i record della tabella dei libri:

```
SELECT COUNT(id) AS temp FROM libri
```

Il nostro alias è *temp*. Se avessimo scritto

```
SELECT COUNT(id) FROM libri
```

il DBMS avrebbe impostato un alias al volo, assegnandogli un nome definito che nella maggior parte dei casi sarebbe **Expr1** (espressione 1) ed il risultato, nel buffer di memoria del DBMS sarebbe:

```
SELECT COUNT(id) AS Expr1 FROM libri
```

Possiamo scegliere di assegnare un alias anche ad un campo definito; supponiamo di avere una tabella chiamata **acquisti** facente parte di un complesso database dedicato ad un'attività di compravendita, il cui scopo è quello di effettuare relazioni a catena con diverse altre tabelle, ad esempio quella dei prodotti e quella degli utenti, contenendo quindi campi di riferimento agli ID di queste altre tabelle.

Possiamo scrivere una query del genere per assegnare un nome (temporaneo) più significativo ad un campo, ad esempio:

```
SELECT id_utente AS acquirente FROM acquisti
```

Relazionando la tabella degli acquisti a quella degli utenti (Rif. capitolo 11) potremo visualizzare il nome dell' utente acquirente in un campo temporaneo chiamato **acquirente**.

Ho tenuto a sottolineare la funzionalità e l'importanza degli alias, concetto che di per se è semplice, per far fronte alle esigenze didattiche del prossimo capitolo.

Nuove istruzioni SQL

Gli standard dell'SQL, come gli standard di tutti i linguaggi, sono sempre oggetto di studio da parte delle softwarehouse preposte all'implementazione del linguaggio. Da un po di tempo sono state introdotte diverse nuove istruzioni atte ad implementare funzionalità più potenti e versatili. Quelle che menzioneremo in questo capitolo sono:

- **GROUP BY** - simile ad ORDER BY con la differenza che non ordina i dati in funzione di un campo ma li raggruppa in funzione del campo specificato, permettendo di effettuare operazioni di conteggio dei risultati all'interno di una stessa query
- **HAVING** - simile alla clausola condizionale WHERE ma serve ad effettuare operazioni utilizzando come clausole condizionali funzioni di aggregazione (Rif. capitolo 12) piuttosto che valori definiti staticamente

In funzione del database e delle tabelle create, facciamo qualche esempio.

Iniziamo con GROUP BY. Contiamo tutti i campi dalla tabella libri in funzione dell'autore:

```
SELECT COUNT(*) AS quanti FROM libri GROUP BY id_autore
```

Passiamo ad HAVING. Contiamo il numero di record della tabella dei libri in cui il prezzo è inferiore a 10 #

```
SELECT COUNT(*) AS quanti FROM libri HAVING MAX(prezzo) < 10
```

che equivale a

```
SELECT COUNT(*) AS quanti FROM libri WHERE prezzo < 10
```

con la differenza che nel primo caso abbiamo usato come parametro condizionale una funzione di aggregazione, e nel secondo abbiamo solo impostato una disuguaglianza.