

Progetto in Sql con database

Si ringrazia l'autore: <http://openskill.info/infobox.php?ID=1040>

Dispensa utilizzata dal prof Paolo Latella per la classe IV B e la VA Mercurio ITCG A. Bassi di Lodi

Nei prossimi paragrafi andremo a vedere come lavorare con mySQL per creare database, tabelle ecc... come inserire dati nel database e come interrogare la base dati per estrarre i dati. Per affrontare questa parte il consiglio è quello di utilizzare il client testuale fornito da mySQL in contemporanea allo studio della documentazione fornita. Quindi aprite subito il client testuale di mySQL! In caso contrario rimarrà tutto teorico e in breve tempo dimenticherete tutto.

Durante i vari passaggi che seguiranno ci saranno diversi approfondimenti per l'analisi della sintassi SQL (sintassi da manuale, reference ecc...) che non sono nè fondamentali nè oggetto di test o verifica. Tutto il resto è assolutamente da testare con il proprio client mySQL e da studiare molto attentamente (salvo successive indicazioni specifiche).

Partiamo quindi da un progettino molto semplice ma utile dal punto di vista didattico, Immaginiamo di dover implementare un sistema di gestione degli ordini per una società di informatica. Dopo l'analisi ho deciso l'utilizzo di 4 tabelle:

STRUTTURA PROGETTINO:

Tabelle:

PRODOTTO (gestione prodotti con relazione a categoria prodotto e fornitore)

CATEGORIA PRODOTTO (gestisce le categorie di prodotti)

ORDINE (Gestisce l'ordine dei prodotti)

FORNITORE (Gestisce i fornitori dei prodotti)

INIZIO PROGETTO GUIDATO (Creazione database e utilizzo DB)

Per creare un database bisogna accedere al server mySQL locale con utente root in modo tale da avere i privilegi necessari.

Successivamente vedrete come utilizzerò per gli STATEMENT di mySQL (CREATE, DROP ecc...) il maiuscolo; la scelta di usare il maiuscolo è dovuta al fatto che si tratta di una convenzione e soprattutto perchè agevola la lettura delle query.

SQL creazione DB:

```
mysql> CREATE DATABASE ictad2;
```

A questo punto, tramite il comando USE, selezioniamo il DB ictad2 sul quale andremo ad operare attraverso le query che verranno quindi eseguite sul suddetto DB

SQL selezione DB:

```
mysql> USE ictad2;
```

DEFINIZIONE DEI CAMPI DELLE TABELLE

Dopo avere selezionato il database dobbiamo andare a creare le tabelle definite in principio. Prima di tutto scegliamo quali campi comporranno la struttura delle nostre tabelle (per motivi di tempo faremo tabelle con pochi campi essenziali).

TABELLA PRODOTTO

IDprodotto
IDfornitore
IDcategoriaprodotto
nomeprodotto
prezzo

TABELLA CATEGORIA PRODOTTO

IDcategoriaprodotto
nomecategoria

TABELLA FORNITORE

IDfornitore
nomefornitore

ORDINE

IDordine
IDprodotto
quantita
dataordine

Prima di andare a creare le tabelle con l'istruzione CREATE andiamo ad analizzare nel dettaglio i tipi di dato che possono essere associati ad un campo di una tabella mySQL.

TIPI DI DATO IN mySQL (da leggere)

TIPO	ATTRIBUTI	DESCRIZIONE
TINYINT	UNSIGNED ZEROFILL	Tipo di dato intero, con segno può variare da -128 a +127, se UNSIGNED da 0 a 255 (default è con il segno). Se viene aggiunto l'attributo ZEROFILL i valori saranno visualizzati con gli zeri al posto degli spazi
SMALLINT	UNSIGNED ZEROFILL	Intero che varia da -32768 a 32767, senza segno da 0 a 65535 (default è con il segno)
MEDIUMINT	UNSIGNED ZEROFILL	Con il segno arriva fino a più di 8 milioni e senza segno a quasi 17 milioni
INT	UNSIGNED ZEROFILL	Con il segno arriva fino a più di 2 miliardi, senza arriva a più di 4 miliardi
INTEGER		Alias di INT
BIGINT	UNSIGNED ZEROFILL	Praticamente arriva a gestire un numero che se letto risulterebbe essere praticamente impronunciabile
FLOAT (precision)	ZEROFILL(M,D)	Numero in virgola mobile con segno: precisione singola se il valore è inferiore o pari a 24, precisione doppia

se il valore va da 25 a 53

FLOAT	ZEROFILL	Numero in virgola mobile con segno e precisione singola; M è la larghezza di visualizzazione, D il numero di decimali
DOUBLE	(M,D) ZEROFILL	Numero in virgola mobile con segno e precisione singola; M è la larghezza di visualizzazione, D il numero di decimali
DECIMAL	(M,D) ZEROFILL	Numero in virgola mobile, con segno e non compresso, memorizza come stringa con un carattere per ogni digit (viene utilizzato di fatto per i valori in €)
DATE		Data compresa tra 1000-01-01 e 9999-12-31. Come potete notare il formato è YYYY-MM-DD
DATETIME		Combinazione di data o ora
TIMESTAMP	(M)	Combinazione di data o ora, compresa tra 1970-01-01 00:00:00 e un giorno dell'anno 2037. Se presente questa colonna imposta automaticamente la data e ora corrente ogni volta che nel record sono eseguiti insert o update. Il valore M definisce in che modo è visualizzata la data e l'ora
TIME		Valore di tempo compreso tra -838:59:59 e 838:59:59
YEAR	(2/4)	Valore in formato a due digit (1970-2069) o come valore di default, con 4 digit (1901-2155)
CHAR(M)	BINARY	Stringa a lunghezza fissa (1-255) riempita a destra di spazi. Impostando l'attributo BINARY le ricerche saranno fatte in modalità case sensitive
VARCHAR(M)	BINARY	Stringa a lunghezza variabile compresa tra 1 e 255 caratteri. Se inferiore a 4 viene convertita automaticamente a CHAR. Se impostiamo BINARY le ricerche saranno case sensitive
TINYBLOB TINYTEXT		Campo di testo max 255 caratteri: BLOB case sensitive TEXT case insensitive

BLOB TEXT	Campo di testo max 65535 caratteri BLOB case sensitive TEXT case insensitive
MEDIUMBLOB MEDIUMTEXT	Campo di testo max 16777215 caratteri BLOB case sensitive TEXT case insensitive
LOB LONGTEXT	Campo di testo con praticamente infiniti caratteri BLOB case sensitive TEXT case insensitive
ENUM('val1','val2','...')	Campo stringa con un set di valori validi Massimo 65535 opzioni
SET('val1','val2','...')	Campo stringa con un set di valori validi Massimo 64 opzioni

ATTRIBUTI DI COLONNA

NULL

Null in SQL è fondamentale ed è differente sia dal valore numerico 0 che da una stringa vuota, in quanto dove è presente un valore nullo significa che non vi è mai stato alcun tipo di inserimento per quel campo in quel record.

Quando si sceglie invece l'opzione NOT NULL si forza il sistema a passare al database almeno un valore.

NULL è il default per ogni colonna.

Valori di DEFAULT

Si può scegliere un valore di default per ogni campo, Es.

```
Active set ('YES', 'NO') DEFAULT 'NO'
```

Chiavi primarie e auto increment

Per indicare quale colonna della tabella rappresenta la chiave primaria della stessa si utilizza l'attributo PRIMARY KEY.

In molti casi è molto comodo utilizzare chiavi primarie di tipo AUTO INCREMENT, ovvero che incrementano di 1 ad ogni record inserito (è possibile stabilire anche il valore dell'incremento). Es.

```
IDprodotto INT NOT NULL AUTO_INCREMENT PRIMARY KEY
```

Per aggiungere chiavi primarie ad una tabella è possibile utilizzare la sintassi: (che vedremo meglio successivamente)

```
PRIMARY KEY (index_col_name, ...)
```

E' naturale che sia possibile utilizzare più campi per definire una chiave primaria

SCELTA TIPI DI DATI NEL PROGETTO

Capiti quali sono i tipi di dato memorizzabili e gli attributi disponibili forniti da MySQL possiamo scegliere tabella per tabella, campo per campo quali tipi di dato scegliere.

E' buona norma cercare di trovare il giusto compromesso tra risparmio risorse (un campo BIGINT

consuma più risorse di un INT) e scalabilità del database senza doverne rivedere la struttura successivamente.

SCELTA DEI TIPI DI DATO

TABELLA PRODOTTO

IDprodotto (INT, NOT NULL, AUTO_INCREMENT, PRIMARY KEY)

IDfornitore (SMALLINT, NOT NULL, DEFAULT 0)

IDcategoriaprodotto (SMALLINT, NOT NULL, DEFAULT 0)

nomeprodotto (VARCHAR(200))

prezzo (DECIMAL(10,2))

TABELLA CATEGORIA PRODOTTO

IDcategoriaprodotto (SMALLINT, NOT NULL, AUTO_INCREMENT, PRIMARY KEY)

nomecategoria (VARCHAR(200))

TABELLA FORNITORE

IDfornitore (SMALLINT, NOT NULL, AUTO_INCREMENT, PRIMARY KEY)

nomefornitore (VARCHAR(200))

ORDINE

IDordine (INT, NOT NULL, AUTO_INCREMENT, PRIMARY KEY)

IDprodotto (INT, NOT NULL, DEFAULT 0)

quantita (SMALLINT, NOT NULL, DEFAULT 0)

dataordine (DATETIME)

Ovviamente la scelta dei tipi di dato da utilizzare è abbastanza soggettiva, l'unica indicazione che è buona regola seguire è di utilizzare lo stesso tipo di dato tra chiave primaria e campo che gestisce la chiave esterna in un'altra tabella.

Es. IDfornitore, IDcategoriaprodotto

INDICI E CHIAVI ESTERNE

Gli indici non sono altro che delle strutture che vengono utilizzate da MySQL per velocizzare le query.

A questo punto urge una premessa, ovvero che non vale la regola che più indici utilizzo e più veloce sarà il mio DB!

Gli indici vanno impostati tenendo presenti (almeno) le seguenti norme:

- Più indici ho e più lente saranno le query di INSERT e di UPDATE
- Gli indici vanno messi soprattutto solo su quei campi dove eseguo le ricerche
- Gli indici appesantiscono di molto il peso fisico del database
- Tendenzialmente gli indici vanno aggiunti (vedremo poi come) al termine dello sviluppo dell'applicazione, quando ci sarà più chiaro quali saranno le query critiche

Le chiavi esterne sono quelle chiavi che definiscono a livello logico le relazioni tra le tabelle, sono anche le chiavi sulle quali poi vengono realizzate le JOIN per andare ad unire diverse tabelle per ottenere tutti i dati voluti.

Molto spesso su queste chiavi esterne è molto utile utilizzare un indice al fine di velocizzare le query di ricerca su quella colonna della tabella.

Una particolare nota va fatta per indici particolari quali:

UNIQUE: Quando definisco un indice con attributo UNIQUE forzo la tabella ad evitare che ci siano valori replica su quel/li campo/i.

FULLTEXT: Quando viene creato un indice FULLTEXT sia ha la possibilità di far ricercare a MySQL del testo in campi stringa e di ottenere i risultati con uno score che ne definisce l'importanza all'interno dei risultati ottenuti.

In pratica si può realizzare un motore di ricerca alla Google, Yahoo! Ecc...

CREAZIONE DELLE TABELLE (CREATE TABLE)

Reference da manuale MySQL:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition, ...)]
    [table_options] [select_statement]
```

Or:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [( ) LIKE old_tbl_name ( )];
```

```
create_definition:
    column_definition
    | [CONSTRAINT [symbol]] PRIMARY KEY [index_type]
(index_col_name, ...)
    | KEY [index_name] [index_type] (index_col_name, ...)
    | INDEX [index_name] [index_type] (index_col_name, ...)
    | [CONSTRAINT [symbol]] UNIQUE [INDEX]
(index_name) [index_type] (index_col_name, ...)
    | [FULLTEXT|SPATIAL] [INDEX] [index_name] (index_col_name, ...)
    | [CONSTRAINT [symbol]] FOREIGN KEY
(index_name) (index_col_name, ...) [reference_definition]
    | CHECK (expr)
```

```
column_definition:
    col_name type [NOT NULL | NULL] [DEFAULT default_value]
    [AUTO_INCREMENT] [[PRIMARY] KEY] [COMMENT 'string']
    [reference_definition]
```

Ovviamente come potete notare l'istruzione CREATE TABLE è abbastanza complicata, in ogni caso per semplificare possiamo fare un esempio che riassume di fatto tutte le opzioni principali di CREATE TABLE.

```
CREATE TABLE prodotto (
IDprodotto INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
IDfornitore SMALLINT NOT NULL DEFAULT 0,
IDcategoriaprodotto SMALLINT NOT NULL DEFAULT 0,
nomeprodotto VARCHAR(200) NOT NULL DEFAULT "",
prezzo DECIMAL(10,2) NOT NULL,
INDEX ( `IDfornitore` ),
INDEX ( `IDcategoriaprodotto` )
);
```

L'analisi di questa istruzione è abbastanza semplice, in pratica per convenzione ogni elemento della tabella è su una riga differente.

Potete anche notare la grande facilità con cui si può aggiungere un indice.

Per evitare di dilungarmi non andrò a scrivere la sintassi di CREATE TABLE per le altre tabelle del progetto.

Basterà semplicemente adattare l'attuale query di CREATE in base ai tipi di dato precedentemente scelti per ogni colonna di tutte le tabelle.

ANALISI TABELLA ESISTENTE

Per poter visualizzare una tabella appena esistente (ad esempio la nostra tabella prodotto) bisogna utilizzare l'istruzione DESCRIBE tablename;

image

L'output di DESCRIBE prodotto; come possiamo vedere ha generato una semplice tabellina esemplificativa di tutti i campi presenti nella tabella.

Vediamo insieme le cose che possono apparire dubbie:

- ogni tipo di dato presenta un (x) es. int(11), infatti quando nella CREATE TABLE non viene specificato la lunghezza di un campo, MySQL ne utilizza il DEFAULT.
- nella colonna "Key" possiamo notare come PRI ci indichi chiaramente la chiave primaria, mentre MUL rappresenta che su quei campi sono definite degli indici.

NAMING CONVENTION

Ora trattiamo un argomento molto soggettivo ma anche molto importante quando si lavora sui database server come MySQL.

Come da titolo l'argomento in questione è la naming convention, in sostanza si tratta di stabilire delle regole per assegnare i nomi alle tabelle e ai campi delle tabelle del database.

L'importanza della naming convention è data dal fatto di poter rendere leggibile e facilmente utilizzabile l'applicazione DB anche per altri programmatori e per se stessi, questo perchè quando si lavora con database molto complesso questo aiuta a lavorare senza dover avere sempre davanti la struttura del database.

Di seguito analizziamo la logica della scelta dei nomi dei campi nel progettino su cui stiamo lavorando, è molto importante capire che queste regole non sono assolutamente vincolanti, nel senso che sono le norme che seguo io personalmente (derivanti anche da regole di massima stabilite e personalizzate secondo i miei gusti personali)... la cosa fondamentale è **DEFINIRSI DELLE REGOLE E SEGUIRLE!**

1. Nomi dei campi tutti in minuscolo
2. Chiave primaria: IDnometabella
3. Chiavi esterne: hanno lo stesso nome di IDnometabellediriferimento
4. Per tabelle come "fornitore" il campo che lo descrive, ovvero il campo "fornitore" porta il nome di tabella e non di "nome" o simili per evitare che nelle JOIN si debba fare un rinaming dei campi, operazione abbastanza lunga e noiosa.

POPOLAMENTO DATABASE, SQL DI INSERT

Per popolamento intendiamo l'inserimento di dati nelle tabelle del nostro database, in sostanza ora andiamo ad analizzare le query in INSERT.

Per inserire i dati in MySQL abbiamo sostanzialmente tre possibili differenti sintassi in INSERT. Di seguito vi allego la sintassi come da manuale:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
      [INTO] tbl_name [(col_name,...)]
      VALUES ({expr | DEFAULT},...), (...), ...
      [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
      [INTO] tbl_name
      SET col_name={expr | DEFAULT}, ...
      [ ON DUPLICATE KEY UPDATE col_name=expr, ... ]
```

Or:

```
INSERT [LOW_PRIORITY | DELAYED] [IGNORE]
      [INTO] tbl_name [(col_name,...)]
      SELECT ...
```

Semplificando quanto fornito dal manuale di MySQL vediamo insieme il funzionamento di INSERT con degli esempi pratici che andrete ad eseguire sul nostro progettino:

```
INSERT INTO fornitore VALUES ('', 'Eforhum.it');
```

Sostanzialmente utilizzando questa sintassi possiamo dire che la query “passa” alla tabella “fornitore” i valori da inserire in ordine di campo.

Il vantaggio di usare questa sintassi di INSERT sono:

VANTAGGI:

- 1.Velocità di esecuzione
- 2.SQL molto scarno e rapido da scrivere

SVANTAGGI:

- 1.Bisogna conoscere molto bene i campi della tabella e l'ordine in cui sono messi
- 2.In caso di modifiche al DB bisogna sistemare subito le query
- 3.Lettura della query abbastanza difficile

```
INSERT INTO fornitore (IDfornitore, fornitore) VALUES ('',
'Eforhum.it');
```

Come possiamo notare che l'unica cosa che cambia è la definizione dell'ordine dei campi in cui MySQL inserirà i valori.

VANTAGGI:

- 1.Buona velocità di esecuzione delle query
- 2.In caso di modifiche al DB (anche se è fondamentale aggiornare le query) non si rischia di incorrere in inserimenti errati. Stesso discorso vale per l'ordine dei campi che se cambia non inficia sulla validità della query

SVANTAGGI:

- 1.Non di facile lettura

```
INSERT INTO fornitore SET fornitore = 'eforhum.it';
```

Con questa sintassi possiamo notare che vengono associati campo = valore

VANTAGGI:

1. Se viene cambiato il DB o l'ordine dei campi le query non portano ad errori di inserimento (a meno che non si siano eliminate delle colonne)
2. Facilità di lettura SQL di INSERT

SVANTAGGI:

1. SQL forse un pò ridondante

Approfondite le varie sintassi possiamo facilmente dedurre che la soluzione migliore è di utilizzare l'SQL di INSERT che prevede il setting di campi = valore, questo sia per facilità di lettura che per leggibilità dell'SQL.

Infatti se immaginiamo una tabella di 30/40 campi le prime due opzioni non garantiscono una facile lettura per l'individuazione di eventuali errori.

SELEZIONE ED ESTRAZIONE DEI DATI NEL DATABASE

Avendo visto come popolare il database con le query di INSERT a questo punto andiamo ad analizzare come poter visualizzare i dati presenti nel database server.

E' molto importante sapere che normalmente ci si trova ad eseguire query di visualizzazione molto più frequentemente rispetto a query di insert o di update e che sono le operazioni che richiedono maggiori performance, di conseguenza andremo anche a valutare alcune tecniche per ottimizzare e migliorare la rapidità di esecuzione delle query di estrazione dei dati.

SINTASSI DI SELECT DA MANUALE:

SELECT

```
[ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
select_expr, ...
[INTO OUTFILE 'file_name' export_options
 | INTO DUMPFILE 'file_name']
[FROM table_references
 [WHERE where_definition]
 [GROUP BY {col_name | expr | position}
  [ASC | DESC], ... [WITH ROLLUP]]
 [HAVING where_definition]
 [ORDER BY {col_name | expr | position}
  [ASC | DESC] , ...]
 [LIMIT [offset,] row_count | row_count OFFSET offset]
 [PROCEDURE procedure_name(argument_list)]
 [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT Reference link : <http://dev.mysql.com/doc/mysql/en/SELECT.html>

Come abbiamo già fatto per le query di inserimento dei dati semplifichiamo quanto riportato dal manuale e analizziamo insieme il funzionamento delle query di SELECT in mySQL.

ESTRAZIONE DI TUTTI I DATI DA UNA TABELLA:

```
SELECT * FROM fornitore;
```

ESTRAZIONE DI ALCUNI DATI DA UNA TABELLA:

```
SELECT IDprodotto, nomeprodotto, prezzo FROM prodotto;
```

Possiamo notare come al posto di “*” che significa “tutti i campi” abbiamo specificato campo per campo quali dati volevamo ottenere dalla query.

Con le query viste fino ad ora potremo notare come mySQL estrae i dati con un ordine casuale, per poter ordinare i dati in base ad uno o più campi possiamo impostare l'ordinamento con ORDER BY campo1, campo2 .

Di default quando si effettua un ordinamento con mySQL i dati verranno visualizzati in ordine ascendente. Per poter estrarre i dati ordinati in modalità decrescente bisogna utilizzare ORDER BY campo1 DESC, campo2 DESC .

Ora vediamo in funzione ORDER BY:

```
SELECT * FROM fornitore ORDER BY fornitore;
```

oppure:

```
SELECT * FROM fornitore ORDER BY fornitore DESC;
```

Supponiamo ora di dover ricercare all'interno della tabella fornitore quello con ragione sociale uguale a “eforhum”.

Per poter ricercare dobbiamo utilizziamo la clausola WHERE che va inserita dopo il nome della tabella e prima di ORDER BY

```
SELECT * FROM fornitore WHERE fornitore = 'eforhum' ORDER BY  
fornitore;
```

Se non ci ricordassimo esattamente la ragione sociale di eforhum ma solamente forh con la query precedente non otterremmo il risultato voluto.

Per ovviare al problema bisogna utilizzare LIKE '%forh%' .

```
SELECT * FROM fornitore WHERE fornitore LIKE '%forh%' ORDER BY  
fornitore;
```

Nella clausola WHERE oltre a = e LIKE esistono altri tipi di operatori:

1.>

2.<

3.<> Diverso da 'valore'

4.>= Maggiore o uguale a

5.<= Minore o uguale a

Ovviamente è anche possibile ricercare su più campi utilizzando gli operatori AND, OR ecc... vediamo di seguito un semplice utilizzo di questi operatori:

```
SELECT * FROM prodotto WHERE prezzo > '33.5' AND IDfornitore <>  
'55'
```

Estraiamo i record con prezzo minore di 33.5 e IDfornitore diverso da 55

Supponiamo ora di avere 100.000 record nella tabella prodotto se andassimo a visualizzare tutti i record solo per vedere ad esempio gli ultimi 100 prodotti inseriti il risultato sarà di caricare inutilmente il DB server ma soprattutto di ritrovarci con dati inutili ai nostri fini e difficilmente leggibili. Per evitare queste problematiche bisogna utilizzare il comando LIMIT 100 .

Vediamo rapidamente l'utilizzo pratico di LIMIT:

```
SELECT * FROM prodotto ORDER BY IDprodotto DESC LIMIT 100;
```

In pratica abbiamo preso tutti i campi della tabella prodotto, li abbiamo ordinati per IDprodotto (INT e auto incrementante) in modalità DESC e abbiamo limitato l'output dei dati a 100 record.

OTTIMIZZAZIONE QUERY DI INTERROGAZIONE

Quando ci si trova a lavorare con database di notevoli dimensioni (almeno sopra i 100.000 record) bisogna stare molto attenti alle performance delle query di visualizzazione per evitare inutili rallentamenti all'applicazione.

Di seguito andiamo ad analizzare alcuni accorgimenti per ottenere maggiori performance dal MySQL server (queste regole valgono sostanzialmente per tutti i DB server... Oracle, SQL server, Access ecc...):

1. SELEZIONE DEI CAMPI

Per quanto la scelta di utilizzare * sia comoda per velocizzare la scrittura dell'SQL questo comporta che in moltissimi casi MySQL estragga valori che noi non utilizzeremmo nemmeno, perciò è consigliabile scegliere sempre i campi desiderati.

2. UTILIZZO DI LIMIT

Come già detto se abbiamo bisogno solo di 100 record è inutile far estrarre a MySQL tutte le righe di una tabella

3. INDICI SUI CAMPI DI RICERCA

Un'altra buona soluzione per migliorare la velocità di esecuzione delle query di SELECT è di impostare degli indici sui campi dove avvengono ricerche frequenti (ricordatevi come sempre di non abusarne).

Per approfondimento in merito all'ottimizzazione delle query di MySQL vi rimando alla documentazione ufficiale di MySQL (Lettura assolutamente facoltativa):

Ottimizzazione query di MySQL :

(http://dev.mysql.com/doc/mysql/en/Optimise_Overview.html)

UNIONE DI PIU' TABELLE (JOIN)

Come avete già visto nel modulo database I per ottenere i dati su più tabelle bisogna utilizzare le JOIN. In MySQL sono supportate solo le JOIN interne e la JOIN a sinistra o esterna. Vediamo ora un esempio pratico. Assumiamo di avere i seguenti dati nelle tabelle prodotto e fornitore:

IDprodotto	IDfornitore	IDcategoriaprodotto	Nomeprodotto	Prezzo
1	1	1	Computer	10
2	1	1	Monitor	100
3	2	4	Tastiera	1

IDfornitore	fornitore
1	eForHum
2	Coresis S.r.l.

Se eseguiamo una normale query di SELECT sulla tabella prodotto otterremo per IDfornitore solo gli ID di riferimento, il che rende di fatto inutilizzabile il valore trovato.

Per poter ottenere il nome del fornitore dovremmo unire (joinare) le due tabelle.

JOIN INTERNA o EQUIJOIN

```
SELECT prodotto.*, fornitore.fornitore
FROM prodotto, fornitore
WHERE prodotto.IDfornitore = fornitore.IDfornitore
```

Sostanzialmente con questa query diciamo a mySQL di restituirci tutti i record delle tabelle prodotto e fornitore dove la chiave primaria di fornitore è uguale alla chiave esterna IDfornitore presente nella tabella prodotto.

Otterremo quindi anche il campo fornitore e i dati saranno perfettamente leggibili ed utilizzabili con facilità.

Quando si lavora su più tabelle (a volte si può arrivare ad unire anche 10 tabelle) si incorre nel rischio di avere campi che creano collisioni di nomi e quindi si sceglie di utilizzare l'espressione tabella.nomecampo .

Tutto ciò però può risultare lungo e ridondante, per questo motivo in mySQL e in generale nell'SQL è possibile rinominare le tabelle all'interno della query.

Vediamo l'esempio pratico:

```
SELECT p.*, f.fornitore
FROM prodotto AS p, fornitore AS f
WHERE p.IDfornitore = f.IDfornitore
```

Questo rinaming delle tabelle prende il nome di aliasing delle tabelle.

JOIN SINISTRA o ESTERNA

Un'altra possibilità di JOIN è data dal concatenamento delle tabelle, viene chiamata o JOIN sinistra o JOIN esterna. Questo tipo di JOIN è formata dall'istruzione LEFT JOIN.

```
SELECT p.*, f.fornitore
FROM prodotto AS p LEFT JOIN fornitore AS f
ON p.IDfornitore = f.IDfornitore
```

Questo tipo di JOIN anzichè visualizzare solo i record con dati corrispondenti di entrambe le tabelle inserisce valori NULL dove non esiste relazione.

Quando si utilizza questo tipo di concatenamento se si invertono i nomi delle tabelle il risultato potrebbe cambiare perchè la selezione viene eseguita nella tabella fornitore e i record corrispondenti vengono cercati nella tabella prodotto.

Quale scegliere?

Solitamente va benissimo la JOIN interna, solo in alcuni casi è necessario ottenere tutte le righe di una tabella ed eventualmente i dati relativi alle chiavi esterne, in tal caso si deve usare la LEFT JOIN.

MODIFICA DEI RECORD (UPDATE QUERY)

Dopo aver visto come inserire e visualizzare i dati nelle tabelle di mySQL vediamo come eseguire modifiche ai record in mySQL.

Per eseguire modifiche si utilizza il comando UPDATE

```
UPDATE fornitore SET fornitore = 'Coresis S.r.l.' WHERE
IDfornitore = '1';
```

Come possiamo notare si tratta di settare i campi da modificare esattamente come per la query di INSERT (la terza che avevamo visto).

BISOGNA FARE ESTREMA ATTENZIONE A NON DIMENTICARSI LA CLAUSOLA WHERE ALTRIMENTI LA MODIFICA DIVENTA VALIDA PER TUTTI I RECORD!

Ovviamente è anche possibile fare una UPDATE su più record semplicemente lavorando sulla condizione della query (WHERE).

Qualora si modifichi una chiave primaria di una tabella che ha chiavi esterne in altre tabelle bisogna ASSOLUTAMENTE aggiornare anche le foreign key.

CANCELLAZIONE DEI RECORD (DELETE QUERY)

Per cancellare record da una tabella di un database MySQL bisogna utilizzare il comando DELETE.

```
DELETE fornitore WHERE IDfornitore = '1';
```

Anche in questo caso bisogna prestare la massima attenzione per evitare di cancellare tutti i record della tabella.

Per quanto concerne le foreign key bisogna fare un discorso a parte, in pratica bisogna stabilire i comportamenti da tenere quando si vanno ad eliminare da una tabella dei record che hanno la chiave primaria presente come foreign key in altre tabelle del database.

La scelta sarà lato applicazione e sarà del tipo:

Eliminazione a cascata dei record correlati

Negazione alla cancellazione in quanto salterebbero le relazioni

La cosa peggiore è senza dubbio lasciare in altre tabelle del database server dei record orfani della propria precedente relazione.

MODIFICA STRUTTURA DI UNA TABELLA (Alter table)

Sintassi dal manuale di MySQL:

```
ALTER [IGNORE] TABLE tbl_name  
    alter_specification [, alter_specification] ...
```

alter_specification:

```
    ADD [COLUMN] column_definition [FIRST | AFTER col_name ]  
    | ADD [COLUMN] (column_definition,...)  
    | ADD INDEX [index_name] [index_type] (index_col_name,...)  
    | ADD [CONSTRAINT [symbol]]  
        PRIMARY KEY [index_type] (index_col_name,...)  
    | ADD [CONSTRAINT [symbol]]  
        UNIQUE [index_name] [index_type] (index_col_name,...)  
    | ADD [FULLTEXT|SPATIAL] [index_name] (index_col_name,...)  
    | ADD [CONSTRAINT [symbol]]  
        FOREIGN KEY [index_name] (index_col_name,...)  
        [reference_definition]  
    | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}  
    | CHANGE [COLUMN] old_col_name column_definition  
        [FIRST|AFTER col_name]  
    | MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]  
    | DROP [COLUMN] col_name  
    | DROP PRIMARY KEY  
    | DROP INDEX index_name  
    | DROP FOREIGN KEY fk_symbol  
    | DISABLE KEYS
```

```
| ENABLE KEYS
| RENAME [TO] new_tbl_name
| ORDER BY col_name
| CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
| [DEFAULT] CHARACTER SET charset_name [COLLATE collation_name]
| DISCARD TABLESPACE
| IMPORT TABLESPACE
| table_options
```

Vediamo ora alcuni degli utilizzi più comuni di ALTER TABLE:

```
ALTER TABLE prodotto
  CHANGE COLUMN nomeprodotto nomeprod varchar(200) NOT NULL;
```

Con la precedente query abbiamo cambiato il campo 'nomeprodotto' della tabella 'prodotto' in 'nomeprod' con lo stesso prima tipo di campo precedente (varchar(200)) NOT NULL

```
ALTER TABLE prodotto RENAME TO prodotti;
```

Con questa istruzione SQL abbiamo rinominato la tabella 'prodotto' in 'prodotti'

```
ALTER TABLE prodotto ADD COLUMN pippo text NULL;
```

Come è facile intuire ora abbiamo aggiunto il campo 'pippo' con tipo di dato text alla tabella prodotto. Di default viene aggiunto come ultimo campo, se si vuole inserire ad esempio dopo IDprodotto:

```
ALTER TABLE prodotto ADD COLUMN pippo text NULL AFTER IDprodotto;
```

```
ALTER TABLE prodotto DROP COLUMN pippo;
```

Con l'SQL precedente abbiamo eliminato il campo 'pippo' dalla tabella prodotto.

Vediamo ora come aggiungere indici su uno o più campi della tabella prodotto:

```
ALTER TABLE prodotto ADD INDEX ( prezzo )
```

oppure per aggiungere un indice su più colonne:

```
ALTER TABLE prodotto ADD INDEX prezzonome ( prezzo , nomeprodotto
 )
```

Nella prima query viene aggiunto un indice nella tabella prodotto sul singolo campo "prezzo", se non specificato il nome dell'indice sarà "prezzo".

Nella seconda espressione viene aggiunto un indice di nome "prezzonome" alla tabella prodotto e che è generato su 2 colonne.

Da quanto visto se ne deduce che è possibile creare indici su infinite colonne.

Ovviamente l'utilizzo di indici su più colonne è consigliato quando nelle query di ricerca si eseguono delle where su due campi. Es:

```
SELECT * FROM prodotto WHERE prezzo = '33' AND nomeprodotto =
'computer';
```

CONCLUSIONE

Dopo aver studiato la documentazione del corso l'obiettivo di avere solide basi sull'utilizzo di mySQL l'abbiamo ottenuta.

Naturalmente non possiamo pretendere di conoscere mySQL a memoria, anche perchè su questo software si potrebbe scrivere un libro di mille pagine (basti vedere la documentazione ufficiale), in

ogni caso i punti principali sono stati trattati tutti unendo la pratica alla teoria.

Il consiglio che vi do per apprendere al meglio il tutto è di fare molta pratica seguendo sempre passo passo la documentazione stessa in quanto sarà lo scopo delle esercitazioni che vi verranno proposte.