MySQL

MySQL è un DBMS basato sul modello relazionale (*RDBMS*). Supporta gran parte dei costrutti del linguaggio *SQL 2.0* con viste, query annidate, vincoli di chiave, trigger e viste aggiornabili. Supporta anche l'esecuzione di **transazioni** su un tipo particolare di tabelle, dette **INNODB**. Inoltre supporta anche molte tipologie di dati numerici, testuali, temporali e binari. Per concludere, dispone di un proprio **linguaggio di estensione procedurale** per definire le **stored procedures**.

MySQL **non** ha limiti espliciti sulla **dimensione massima** di un database e sul numero di tabelle. Il **numero massimo di righe** in una tabella **dipende dai vincoli imposti dal sistema operativo** sulla dimensione massima di un file.
Non esistono problemi dal punto di vista della concorrenza in termini di **numero massimo di connessioni simultanee** al server MySQL. Tuttavia, i problemi emergono dal punto di vista delle **risorse**, per cui il numero effettivo di connessioni simultanee supportate dipende dalle capacità e dalle risorse hardware dell'elaboratore. Si osservino ora diversi comandi.

Per **creare un nuovo database**:

CREATE DATABASE [IF NOT EXIST] nomedb;

Per **rimuovere un database**:

DROP DATABASE [IF EXIST] nomedb;

Per vedere quali **db sono presenti** nel sistema:

SHOW databases;

Per impostare il **db corrente**:

USE nomedatabase;

Per **creare** una tabella:

CREATE [TEMPORARY] TABLE

nometabella | nomedb.nometabella

[definizione attributi]

[opzioni]

[select]

E' possibile generare una tabella valida solo per la **sessione corrente** con l'opzione *TEMPORARY*.
Inoltre è possibile **popolare** la tabella con il risultato di una query *SELECT* da altre tabelle.

MySQL supporta diversi tipi di **storage engine**, ovvero tipi di tabelle, tra cui i principali sono:

- **INNODB**: supporta il sistema **transazionale**, i vincoli di **chiavi esterne** ed ha una maggiore **robustezza** ai guasti

- **MyISAM**: **non** supporta il sistema transazionale, ha però una **maggiore efficienza** ed un **minore consumo di spazio** su memoria secondaria

Per quanto riguarda il campo non obbligatorio delle **opzioni**, alcuni esempi possono essere:

- *ENGINGE = tipotabella (ISAM|INNODB)*
- *AUTO_INCREMENT = valore*
- *AVG_ROW_LENGTH = valore*
- *CHECKSUM = 0 | 1*
- *COMMENT = stringa*
- *MAX_ROWS = valore*

Per quanto riguarda la sintassi per specificare una **colonna della tabella**:

Nome_colonna TIPO

[NOT NULL | NULL] [DEFAULT valore]

[AUTO_INCREMENT]

[UNIQUE | PRIMARY KEY]

[COMMENT 'commento']

Per definire i vincoli di **integrità referenziale**:

FOREIGN KEY (nome_colonna_interna)

REFERENCES nome_tabella_esterna

(nome_colonna_esterna)

[ON DELETE | ON UPDATE | RESTRICT | CASCADE | SET NULL | NO ACTION]

Nota Bene: funziona solo con tabelle di tipo *INNODB*.

Esaminiamo ora alcuni tipi di dato supportati da MySQL. I tipi di dati **numerici** supportati sono:

BIT

TINYINT [UNSIGNED][ZEROFILL]

SMALLINT [UNSIGNED][ZEROFILL]

MEDIUMINT [UNSIGNED][ZEROFILL]

INT [UNSIGNED][ZEROFILL]

BIGINT [UNSIGNED][ZEROFILL]

FLOAT [UNSIGNED][ZEROFILL]

DOUBLE [UNSIGNED][ZEROFILL]

DECIMAL [UNSIGNED][ZEROFILL]

I tipi di dato **temporali** supportati sono:

DATE

DATETIME

TIMESTAMP [M]

TIME

YEAR [(2, 4)]

Per conoscere **data/timestamp correnti**:

SELECT NOW();

SELECT CURTIME();

Alcuni tipi di dato **stringa di caratteri o byte**:

CHAR(M) [BINARY | ASCII |UNICODE]

VARCHAR(M) [BINARY]

BINARY(M)

VARBINARY(M)

TINYBLOB

TINYTEXT

BLOB(M)

TEXT(M)

LONGBLOB

Si osservi ora un esempio di creazione di una tabella in MySQL:

CREATE TABLE IMPIEGATI (

Codice smallint auto_ increment primary key,

Nome varchar(200) not null,

Cognome varchar(100) not null,

Salario double default 1000,

Anno date)

engine=innodb;

Il popolamento dei dati viene effettuato attraverso il comando di **INSERT**:

INSERT [LOW_PRIORITY | DELAY | HIGH_PRIORITY]

[INTO] nometabella [(nomecolonne,...)]

VALUES (espressione | DEFAULT,...)

[ON DUPLICATE KEY UPDATE nomecolonna = espressione,...]

E' possibile specificare una **priorità** dell'inserimento dei dati, nel caso in cui la tabella sia usata da altri processi.
Il popolamento dei dati viene può essere effettuato anche attraverso il comando di **REPLACE**:

REPLACE [LOW_PRIORITY | DELAYED]

[INTO] nometabella [(nomecolonne,...)]

VALUES (espressione | DEFAULT,...)

Consente di **rimpiazzare** delle righe preesistenti con delle nuove righe, qualora si verifichi un problema di inserimento con chiave doppia.
Per concludere, un ultimo metodo per il popolamento dei dati è attraverso il comando **LOAD**, il quale permette di popolare la tabella a partire dai dati presenti in un file *.txt*, specificando i separatori delle colonne ed eventualmente le righe da filtrare:

LOAD DATA [LOCAL] INFILE 'file.txt'

[REPLACE | IGNORE]

INTO TABLE nometabella

[FIELDS

[TERMINATED BY 'stringa']

[ENCLOSED BY 'stringa']

[ESCAPED BY 'stringa']]

[LINES

[STARTING BY 'stringa']]

[TERMINATED BY 'stringa']]

[IGNORE numero LINES]

Si osservi la **ricerca di dati**, effettuabile attraverso il comando **SELECT**:

SELECT [ALL | DISTINCT | DISTINCTROW]

lista_colonne

[INTO OUTFILE 'nomefile' | INTO DUMPFILE 'nomefile']

FROM lista_tabelle

[WHERE condizione]

GROUP BY nomecolonna

HAVING condizione

ORDER BY nomecolonna

[LIMIT [offset,] numero_righe]

Si osservi la **cancellazione di dati**, effettuabile attraverso il comando **DELETE**:

DELETE [LOW_PRIORITY][IGNORE][QUICK]

FROM nome_tabella

[WHERE condizione]

[LIMIT numero_righe]

Per rimuovere **tutto il contenuto** attraverso il comando **TRUNCATE**:

TRUNCATE nometabella

Si osservi l'**aggiornamento di dati**, effettuabile attraverso il comando **UPDATE**:

UPDATE [LOW_PRIORITY][IGNORE]

SET nomecolonna = espressione,...

WHERE condizione

Per la creazione di regole attive si utilizza il costrutto **TRIGGER**:

CREATE TRIGGER nome tipo

ON tabella FOR EACH ROW istruzioniSQL

Il **tipo** specifica l'evento che attiva il trigger:

BEFORE INSERT

BEFORE UPDATE

BEFORE DELETE

AFTER INSERT

AFTER UPDATE

AFTER DELETE

Un esempio di definizione di trigger in MySQL:

CREATE TRIGGER upd_check

BEFORE INSERT ON Impiegati

FOR EACH ROW

BEGIN

IF NEW.Salario > 300 THEN

SET NEW.Salario = 300;

END IF;

END;

Per la creazione di **viste** si utilizza il costrutto **VIEW**:

CREATE [OR REPLACE]

[ALGORITHM = (UNDEFINED | MERGE | TEMPTABLE)]

VIEW nome [(lista colonne)]

AS selectSQL

[WITH [CASCADED | LOCAL] CHECK OPTION]

E' possibile definire **viste aggiornabili** attraverso la clausola **WITH CHECK OPTION**.

Per la creazione di **stored procedures** in MySQL:

CREATE PROCEDURE nomeProcedura

([IN|OUT] nomeParametro tipo)

BEGIN

[dichiarazione di variabili locali]

[istruzioniSQL]

END;

Un esempio di definizione di stored procedures in MySQL:

CREATE PROCEDURE nomeImpiegato

(IN cod INT, OUT nomeI VARCHAR(200))

BEGIN

SELECT NOME AS NOMEI

FROM IMPIEGATI

WHERE (CODICE = cod);

END;

CALL *nomeImpiegato(200, @var);*

SELECT *@var;*

Per la dichiarazione di **variabili locali**:

DECLARE a INT DEFAULT 0;

Per i costrutti di **selezione** (IF THEN ELSEIF ELSE):

IF Condizione THEN

IstruzioniSQL

[ELSE IstruzioniSQL]

ENDIF;

Per i costrutti di **iterativi** (WHILE/LOOP/REPEAT):

[nome] WHILE condizione DO

IstruzioniSQL

END WHILE [nome];

Per quanto riguarda la gestione delle **transazioni** per tabelle **INNODB** troviamo due punti principali:

- Di default, la modalità **autocommit** è abilitata, quindi tutti gli aggiornamenti sono effettuati immediatamente sul database
- Nel caso in cui gli autocommit siano disabilitati, è necessario indicare l'inizio della transazione (**START TRANSACTION**) e terminarla con un comando di **COMMIT** o **ROLLBACK**

Si osservi un esempio di transazione in MySQL:

SET AUTOCCOMIT = 0;

START TRANSACTION

INSERT INTO IMPIEGATO (Nome, Cognome, Salario)

VALUES ('Michele', 'Rossi', 1200);

INSERT INTO IMPIEGATO (Nome, Cognome, Salario)

VALUES ('Carlo', 'Bianchi', 1000);

COMMIT

Per quanto riguarda l'**isolamento**, MySQL offre quattro livelli distinti:

- **READ UNCOMMITTED**: sono visibili gli aggiornamenti non consolidati fatti da altri
- **READ COMMITTED**: aggiornamenti visibili solo se consolidati, ossia solo dopo COMMIT
- **REPEATABLE READ**: tutte le letture di un dato operate da una transazione leggono sempre lo stesso valore (comportamento di **default**)
- **SERIALIZABLE**: lettura di un dato blocca gli aggiornamenti fino al termine della transazione stessa che ha letto il dato, lock applicato ad ogni SELECT

Il tool *mysqldump* consente di effettuare **backup** del contenuto di un database, o di tutti i database. Per effettuare il backup di tutti i databse con tabelle INNODB:

mysqldump -single-transaction -all-database > nomeFile

Per effettuare il backup di uno specifico databse con tabelle INNODB:

mysqldump -single-transaction nomedb > nomeFile

Per effettuare il ripristino di un databse, o tutti, da un file di backup:

mysql [nomedb] < nomeFile

Differenze tra MySQL e Oracle

	MySQL	Oracle
Costi	Free (Community Ed.)	Pagamento (Enterprise)
Autenticazione & Sicurezza	Basata su host+ username+password	Meccanismi multipli di autenticazione, ruoli
Gestione della Concorrenza	Lock a livello di tabella	Supporta lock a livello di singola riga
Supporto SQL e stored procedures	SQL base + estensioni procedurali (limitate)	SQL base + estensioni procedurali (PL/SQL)
Backup	Pochi tool di backup	Molti tool di backup
Supporto XML	Limitato	Supporto SQL/XML
Tipi di dati	Solo 2 tipi di dato char	4 Tipi di dato char