# MySQL

## Cos'è un DBMS

È un sistema software che facilita il processo di **definizione**, **costruzione** e **manipolazione** di una base di dati, garantendone la **persistenza** e consentendo l'**accesso concorrente** ai dati in essa contenuti da parte di **utenti** e **applicazioni**.

È un sistema software che gestisce **grandi quantità** di dati **persistenti** e **condivisi**. Esiste una parte **server** e una parte **client**.

Criticità:

- problemi di efficienza
- meccanosmo per garantire l'affidabilità dei dati (*fault tolerance*), per il controllo degli accessi e per il controllo della concorrenza

## Caratteristiche

- Mantenimento della **correttezza dei dati**:
  - nella struttura
  - nel tempo
- Facilitare l'**accesso delle applicazioni ai dati**
- Gestione degli **accessi concorrenti**
- Controllo delle transazioni: **transaction processing**
- Supporta connessioni SSH e SSL
- Crittografia dei dati e gestione multiutente
- Funzionalità di backup e ripristino
- Funzionalità di progettazione e gestione
- Ricerche fulltext e meccanismi di caching
- Supporto a storage engine distribuiti

Il modello dei dati di SQL è basato su tabelle anziché relazioni:

- Possono essere presenti righe (tuple) duplicate
- SQL adotta la logica a 3 valori dell'Algebra Relazionale
  - Nasce dall'esigenza di gestire i valori null
  - Introduce il terzo valore di verità: unknown U

## Architettura di MySQL

Un singolo **processo Server** si occupa di **ascoltare** su una porta socket **3306** e lanciare i **thread** per le sessioni utente.

I database sono directory che contengono uno o più file per ogni tabella (Engine dependent)

È possibile ospitare più server MySQL sullo stesso host, in ascolto su porte differenti.

**ACID: Atomicity, Consistency, Isolation, Durability**

## Le tabelle

Una tabella è costituita da una collezione ordinata di attributi e da un insieme (eventualmente vuoto) di vincoli.

Parleremo di tabelle e non di relazioni, e di righe e non di tuple, poichè rispetto al modello relazionale possiamo avere duplicazione di righe.

```
CREATE TABLE NomeTabella (
  NomeAttributo Dominio [ ValoreDiDefault ] [ Vincoli ]
  {,NomeAttributo Dominio [ValoreDiDefault ] [Vincoli ] }
  [ AltriVincoli ] )
```

Istruzione **CREATE TABLE**:

- Definisce uno schema di relazione e ne crea un'istanza vuota. Per ogni attributo va specificato il dominio, un eventuale valore di default ed eventuali vincoli
- Possono essere espressi altri vincoli a livello di tabella (tra più attributi).

## I domini

I domini specificano i valori ammissibili per gli attributi di una relazione:

Domini elementari (SQL ha 6 domini predefiniti):

1. Bit SQL-2 poi eliminato e sostituito parzialmente da BOOLEAN in SQL-3:

- **bit**
- **varbit(*lunghezza*)**

2. Carattere:

- **character(*lunghezza fissa*)**
- **varchar(*lunghezza*)**

3. Numerico Esatto:

- **integer** per interi
- **decimal** e **numeric** per decimali

4. Numerico Approssimato:

- real
- float
- double

5. Data/Ora:

- date
- time
- timestamp (data e ora)

6. Intervallo Temporale:

- **interval *PrimaUnitàDiTempo* to *UltimaUnitàDitempo***

Domini definiti dall'utente (semplici, ma riutilizzabili)

## BLOB e CLOB

Binary Large Object (BLOB) e Character Large Object (CLOB).

Il sistema memorizza il loro valore ma non possono essere utilizzati per interrogazioni

## Domini definiti dagli utenti

```
CREATE DOMAIN Voto AS SMALLINT
  DEFAULT 0
  NOT NULL
```

La definizione di “nuovi domini” è utile perché permette di associare dei vincoli a un nome di dominio: questo è importante quando si deve ripetere la stessa definizione di attributo su diverse tabelle: ad esempio, modifiche alla definizione di Voto si ripercuotono in tutte le occorrenze di questo dominio nello schema del Database.

**Valori di default per il dominio**: Definiscono il valore che deve assumere l’attributo quando non viene specificato un valore durante l’inserimento di una tupla

`DEFAULT < ValoreGenerico | user | null >`

- *ValoreGenerico* rappresenta un valore compatibile con il dominio, rappresentato come una costante o come un’espressione.
- *user* è l’identificativo dell’utente che effettua il comando di aggiornamento della tabella.
- *null* è il valore di DEFAULT di base.

## Il significato di NULL

Valore polimorfico.

1. il valore esiste in realtà ma è ignoto al database (es.: data di nascita)
2. il valore è inapplicabile (es.: numero patente per minorenni)
3. non si sa se il valore è inapplicabile o meno (es.: numero patente per un maggiorenne)

# DML

DML: linguaggio di query, modifica, ...

**SELECT** è la parola chiave

## Clausola WHERE

Specifica condizione di selezione.

L’argomento di WHERE è un’espressione booleana:

`[ NOT ] PredicatoSemplice { < AND | OR > [ NOT] PredicatoSemplice }`

**Predicati semplici**: Espressioni di valutazione su attributi mediante operatori di confronto

**Operatori di confronto**: =, <> ,<, >, <=, >=, LIKE, BETWEEN, IS NULL, IS NOT NULL

### Operatore LIKE

L’operatore LIKE permette di esprimere dei “pattern” su stringhe mediante “wildcard”:

- _ (un carattere arbitrario)
- **%** (una stringa arbitraria)

### Operatore BETWEEN

Permette di esprimere condizioni di appartenenza a un intervallo

BETWEEN 30 AND 60

### Operatore IN

L'operatore IN permette di esprimere condizioni di appartenenza a un insieme.

IN (2, 3, 4)

### Valori nulli

1. Un predicato semplice valutato su un attributo a valore nullo dà come risultato della valutazione ?.
2. Una tupla per cui il valore di verità è ? non viene restituita dalla query.
3. Se la valutazione del predicato di un constraint è ? il constraint non viene violato

Per verificare null: IS [NOT] NULL

### SELECT DISTINCT

In algebra relazionale i risultati delle interrogazioni non contengono elementi duplicati.

In SQL, le tabelle prodotte dalle interrogazioni possono contenere più righe identiche tra loro.

I duplicati possono essere rimossi usando la parola chiave `DISTINCT`

### ORDER BY

```
ORDER BY Attr [ ASC | DESC ] { , Attr [ ASC | DESC ] }
```

Ordinamento: ascendente (ASC), discendente (DESC) (ascendente è default e può essere omesso).

## Operatori di aggregazione

Sono un'estensione rispetto all'Algebra Relazionale.

Gli operatori aggregati operano su gruppi di tuple di relazione per:

- Effettuare calcoli sull'insieme di tuple;
- Verificare condizioni relative all'insieme di tuple.

Sono:

- **COUNT**: conta numero di tuple di tabella
- **SUM**: somma valori o espressioni di attributi
- **MAX**: valore massimo di un attributo di tabella
- **MIN**: valore minimo di un attributo di tabella
- **AVG**: valore medio di un attributo di tabella
- **GROUP BY**: raggruppamento delle tuple in sottogruppi gestiti come estensione delle normali interrogazioni

Come vengono applicati?

1. Si esegue l'interrogazione sulla base di clausole FROM e WHERE.
2. Si applica l'operatore aggregato alla tabella risultato dell'interrogazione.

### Operatore COUNT

```
COUNT ( < * | [ DISTINCT | ALL ] ListaAttr > )
```

Restituisce il numero di valori degli attributi in ListaAttr:

- **DISTINCT**: numero di valori distinti diversi da NULL di ListaAttr
- **ALL**: numero di valori diversi da NULL di ListaAttr

Se non è specificato nulla, ALL è l'opzione di default.

COUNT (*) conta numero di righe di una tabella

SQL impedisce di includere in una stessa target list funzioni aggregate e espressioni al livello di riga. Una query del tipo `SELECT Cognome, Nome, MAX(Stipendio)` NON SI PUÒ FARE!

### Operatore GROUP BY

La clausola GROUP BY serve a definire gruppi omogenei di tuple, specificando una o più colonne (di raggruppamento) sulla base della/e quale/i le tuple sono raggruppate per valori uguali.

Importante restrizione: una clausola di proiezione di una query contenente la clausola GROUP BY può solo includere:

1. Una o più colonne se compaiono nella clausola GROUP BY;
2. Operatori di aggregazione (COUNT, SUM, AVG, ...)

### Condizioni sui gruppi

Verificano condizioni su gruppi di tuple aggregate con GROUP BY.

Tipi di clausole di selezione:

- **WHERE**: verifica condizioni su tuple individuali
- **HAVING**: verifica condizioni (espressioni booleane) su gruppi di tuple (opera su raggruppamenti)

Le condizioni in clausola HAVING sono verificate dopo la creazione dei sottogruppi di tuple specificati da GROUP BY.

**WHERE**: predicati che operano su tuple individuali (selezione di tuple prima di GROUP BY)

**HAVING**: predicati che operano su gruppi di tuple creati da GROUP BY (selezione raggruppamenti)

**Osservazioni su HAVING**:

- La sintassi permette di definire la clausola having anche senza la clausola group by.
- In questo caso l'intero insieme di righe è trattato come un unico raggruppamento.
- Per la clausola having devono essere usati solo predicati in cui compaiono operatori aggregati.

## Operatori insiemistici

Permettono di costruire query concatenando due query SQL

- union
- intersect
- except (differenza) Sintassi:

```
SelectSQL { < union | intersect | except > [ all ] SelectSQL }
```

I duplicati vengono eliminati (a meno che si usi **all**)

### Union

L'operatore UNION impone alcune restrizioni sulle interrogazioni su cui opera:

- le interrogazioni devono restituire lo stesso numero di colonne, e le colonne corrispondenti devono avere lo stesso dominio (non è richiesto che abbiano la stessa lunghezza) o domini compatibili
- la corrispondenza si basa sulla posizione delle colonne, indipendentemente dal loro nome
- se si usa una clausola di ORDER BY questa deve essere usata una sola volta alla fine dell'interrogazione e non alla fine di ogni SELECT

### Intersezione e differenza

Gli operatori INTERSECT ed EXCEPT (o MINUS) eseguono l'intersezione e la differenza.

Stessi requisiti della UNION.

## Query nidificate

Una delle ragioni che rendono SQL un linguaggio potente è la possibilità di esprimere interrogazioni più complesse in termini di interrogazioni più semplici, tramite il meccanismo delle subqueries (sottointerrogazioni).

La clausola WHERE di una query (detta query esterna) può infatti contenere un'altra query (detta subquery)

La subquery viene usata per determinare uno o più valori da usare come valori di confronto in un predicato della query esterna

Nella clausola WHERE possono comparire predicati che confrontano un attributo (o un'espressione sugli attributi) con il risultato di una query SQL;

Subquery scalare: restituisce non più di una tupla

Sintassi: `ScalarValue Operator < any | all > SelectSQL`

- **ANY**: il predicato è vero se almeno una riga restituita dalla query SelectSQL soddisfa il confronto
- **ALL**: il predicato è vero se tutte le righe restituite dalla query SelectSQL soddisfano il confronto

Operator: uno qualsiasi tra =, <>, <, <=, >, >=

La query che appare nella clausola where è chiamata query nidificata

È inoltre possibile selezionare più di una colonna tramite una sottointerrogazione; in tal caso è necessario apporre delle parentesi alla lista delle colonne a sinistra dell'operatore di confronto (Costruttore di tupla)

- In una subquery non si possono usare operatori insiemistici (UNION, INTERSECT e EXCEPT)
- una subquery può comparire solo come operando destro in un predicato
- in una subquery scalare non ci possono essere le clausole group by e having.
- le subquery possono comparire anche nelle espressioni della clausola select (non solo nella clausola where) e nelle clausole having

### Query correlate

**L'interrogazione interna viene eseguita una volta per ciascuna tupla dell'interrogazione esterna**

Questo tipo di interrogazioni è chiamato correlato, perchè ogni esecuzione della subquery è correlata al valore di uno o più attributi delle tuple candidate nella interrogazione principale.

Per poter riferire le colonne delle tuple candidate nella query esterna si fa uso degli alias di relazione; un alias di relazione è definito nella query esterna e riferito nella query interna

Due sono i concetti principali che sono alla base della correlazione: a) una subquery correlata fa riferimento ad un attributo selezionato dalla query esterna b) se una subquery seleziona tuple dalla stessa relazione riferita dalla query esterna, è necessario definire un alias per tale relazione della query esterna

La subquery deve usare l'alias per riferire i valori di attributo nelle tuple candidate nella query principale.

### EXISTS

L'operatore logico EXISTS (sq) restituisce il valore Booleano True se la subquery restituisce almeno una tupla; restituisce il valore Booleano False altrimenti

## Vincoli intrarelazionali

- NOT NULL: Il valore deve essere non nullo
- UNIQUE: I valore devono essere non ripetuti
- PRIMARY KEY: Chiave primaria
- CHECK: Condizioni complesse

```
CREATE TABLE Impiegato (
  Matricola CHAR(6) PRIMARY KEY,
  Nome CHAR(20) NOT NULL,
  Cognome CHAR(20) NOT NULL,
  Stipendio NUMERIC(9) DEFAULT 0,
  UNIQUE (Cognome, Nome)
)
```

L'attributo o la n-pla di attributi su cui è definito il vincolo **UNIQUE** non possono avere istanze uguali ripetute.

**L'attributo o attributi sono (super)chiave**!

Eccezione: valore **null** può comparire su diverse righe senza violare il vincolo: si assume che i valori null siano tutti diversi fra loro.

Il vincolo **PRIMARY KEY** può essere definito una sola volta all'interno della relazione.

```
CREATE DOMAIN Voto AS SMALLINT
  DEFAULT 0
  CHECK (Voto>=18 AND VOTO<=30)
```

## Vincoli interrelazionali

### Integrità referenziale

Esprime un legame gerarchico (padre / figlio) fra tabelle.

Alcuni attributi della tabella figlio sono definiti **FOREIGN KEY** e si devono riferire (**REFERENCES**) ad alcuni **attributi della tabella padre che costituiscono una chiave** (**devono essere UNIQUE e NOT NULL oppure PRIMARY KEY**).

I valori contenuti nella FOREIGN KEY devono essere sempre presenti nella tabella padre.

```
CREATE TABLE Impiegato (
   Matricola   CHAR(6)       PRIMARY KEY,
   Nome        CHAR(20)      NOT NULL,
   Cognome     CHAR(20)      NOT NULL,
   Dipart      CHAR(15)      REFERENCES Dipartimento(NomeDip),
   Stipendio   NUMERIC(9) DEFAULT 0,
   UNIQUE (Cognome,Nome)
)
```

- Per tutti gli altri vincoli visti fino ad ora, a seguito di una violazione, il comando di aggiornamento viene rifiutato segnalando l'errore all'utente.
- Per i vincoli di integrità referenziale invece SQL permette di scegliere delle reazioni da adottare in caso di violazioni.

Le violazioni possono essere introdotte:

1. da modifica (update) dell'attributo cui si fa riferimento
2. da cancellazioni di tuple

Reazioni previste:

- CASCADE: propaga la modifica
- SET NULL: annulla l'attributo che fa riferimento
- SET DEFAULT: assegna il valore di default all'attributo
- NO ACTION: impedisce che la modifica possa avvenire

```
CREATE TABLE Esame (
Matr          CHAR(6),
CodCorso      CHAR(6),
Data          DATE          NOT NULL,
Voto          Voto,
PRIMARY KEY (Matr, CodCorso)
FOREIGN KEY (Matr) REFERENCES Studente
        ON DELETE CASCADE
        ON UPDATE CASCADE
FOREIGN KEY (CodCorso) REFERENCES Corso
        ON DELETE NO ACTION
        ON UPDATE CASCADE
 )
```

È possibile associare dei nomi ai vincoli, ad esempio:

```
Stipendio INTEGER CONSTRAINT StipendioPositivo
CHECK (Stipendio > 0),

CONSTRAINT ForeignKeySedi
```

```
FOREIGN KEY (Sede) REFERENCES Sedi
```

Check

La clausola check può essere usata per esprimere vincoli arbitrari nella definizione dello schema

`CHECK (condizione)`

Le condizioni che si possono usare sono quelle che si possono usare per la clausola WHERE: cioè un predicato o una combinazione booleana di predicati e sottointerrogazioni

Tale condizione può contenere sottointerrogazioni che fanno riferimento ad altre tabelle, ma il vincolo viene controllato solo quando viene modificato il valore dell'attributo a cui è associato.

# SQL in MySQL

MySQL prevede 3 sottoparti SQL:

- DCL
- DDL
- DML

## DCL: Data Control Language

Esempi:

```
use univ;
show tables;
show schemas;
```

## DDL: Data Definition Language

DDL: definizione di domini, tabelle, autorizzazioni, vincoli, procedure, ecc.

Uno schema di base di dati è una collezione di oggetti: domini, tabelle, asserzioni, viste, privilegi...

Uno schema ha un nome e un proprietario

```
CREATE SCHEMA
  [ NomeSchema ]
  [ [ authorization ] Autorizzazione ]
  { DefinizioneElementoSchema }
```

**Creare/eliminare database:**

N.B.: CREATE SCHEMA is a synonym for CREATE DATABASE as of MySQL 5.0.2.

```
create schema univ;
drop schema univ;
create database if not exists univ charset='utf8' COLLATE='latin1_bin';
drop database univ;
```

**Creare/eliminare tabelle:**

```
CREATE TABLE IF NOT EXISTS univ.studente (
  matricola_st INT NOT NULL ,
  cognome VARCHAR(45) NOT NULL ,
  nome VARCHAR(45) NOT NULL ,
```

```
  citta_di_nascita VARCHAR(45) NOT NULL ,
  citta_residenza VARCHAR(45) NOT NULL ,
  corso_di_laurea VARCHAR(45) NOT NULL ,
  CONSTRAINT uc_PersonID UNIQUE (cognome,nome),
  PRIMARY KEY (matricola_st) )
ENGINE = InnoDB;

drop table univ.studente;
```

```
CREATE TABLE IF NOT EXISTS univ.esame (
  codice_e INT NOT NULL ,
  matricola_studente INT NOT NULL ,
  data DATETIME NULL ,
  voto INT NOT NULL ,
  PRIMARY KEY (Codice_e) ,
  INDEX matricola_idx (matricola_studente ASC) ,
  INDEX corso_idx (codice_e ASC) ,
  CONSTRAINT matricola_key
    FOREIGN KEY (matricola_studente)
    REFERENCES univ.studente (matricola_st)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

## ALTER

```
ALTER DOMAIN NomeDominio <
  SET   DEFAULT ValoreDefault |
  DROP  DEFAULT |
  ADD CONSTRAINT DefVincolo |
  DROP CONSTRAINT NomeVincolo >
```

```
ALTER TABLE NomeTabella <
  ALTER COLUMN NomeAttributo <
    SET DEFAULT NuovoDefault |
    DROP DEFAULT > |
  DROP COLUMN NomeAttributo |
  ADD COLUMN DefAttributo |
  DROP CONSTRAINT NomeVincolo
  ADD CONSTRAINT DefVincolo >
```

```
ALTER TABLE studente DROP PRIMARY KEY;
ALTER TABLE studente ADD PRIMARY KEY(matricola_st);
ALTER TABLE corso CHANGE idcorso corso_id VARCHAR(50);
ALTER TABLE studente ADD COLUMN cod_fisc VARCHAR (16) AFTER nome;
ALTER TABLE studente DROP COLUMN cod_fisc;
ALTER TABLE studente MODIFY nome VARCHAR(45);
```

## DROP

Cancella oggetti DDL, si applica su domini, tabelle, indici, view, asserzioni, procedure,...

```
DROP < schema, domain, table, view, ...> NomeElemento
  [ RESTRICT | CASCADE ]
```

- **RESTRICT**: Impedisce drop se gli oggetti comprendono istanze non vuote
- **CASCADE**: Applica drop agli oggetti collegati. Potenziale pericolosa reazione a catena

# Popolamento in MySQL

## Insert

- L'**ordinamento** degli attributi (se presenti) e dei valori è **significativo**
- Le due **liste debbono avere lo stesso numero di elementi**
- Se la lista di attributi è omessa, si fa riferimento a tutti gli attributi della relazione, secondo l'ordine con cui sono stati definiti
- Se la lista di attributi non contiene tutti gli attributi della relazione, per gli altri viene inserito un valore nullo (che deve essere permesso) o un valore di default

**Inserimento singolo record**

```
INSERT INTO studente
  SET matricola = "39201", cognome = "Bianchi", nome = "Laura";
```

**Inserimento di record multipli**

```
INSERT INTO studente
  (matricola, cognome, nome)
  VALUES
  ("39201", "Bianchi", "Paolo"),
  ("39202", "Rossi", "Alberto");
```

## Update

```
UPDATE NomeTabella
SET Attributo = < Espressione |SELECTSQL|NULL | DEFAULT >
{,Attributo = < Espressione |SELECTSQL |NULL | DEFAULT >} [ WHERE Condizione ]
```

se non è presente clausola where, si suppone WHERE TRUE e quindi si opera la modifica su tutte le righe

```
UPDATE studente
  SET nome= "Federica" WHERE matricola = "39201"
  LIMIT 1;
```

Le modifiche in SQL sono eseguite in modo set-oriented: la clausola WHERE e il valore dell'espressione SET vengono valutati un'unica volta, poi gli aggiornamenti vengono effettuati su tutte le tuple "contemporaneamente"

## Delete

**Cancellazione record**

```
DELETE FROM studente
  WHERE matricola = "39201"
  LIMIT 1;
```

- Elimina le ennuple che soddisfano la condizione
- Può causare (se i vincoli di integrità referenziale sono definiti con politiche di reazione cascade) eliminazioni da altre relazioni
- Ricordare: se la clausola where viene omessa, si intende where true

## Azioni referenziali

```
> ON UPDATE RESTRICT
> ON UPDATE CASCADE
```

```
> ON UPDATE SET NULL
> ON UPDATE NO ACTION
> ON DELETE RESTRICT
> ON DELETE CASCADE
> ON DELETE SET NULL
> ON DELETE NO ACTION
```

## Importazione CSV

```
LOAD DATA INFILE file.csv INTO TABLE studente;
```

## Viste (views)

In SQL è possibile definire viste alternative degli stessi dati.

- una vista (view) è una relazione virtuale attraverso cui è possibile vedere i dati memorizzati nelle relazioni reali (dette di base)
- una vista non contiene tuple, ma può essere usata quasi a tutti gli effetti come una relazione di base
- una vista è definita da una interrogazione su una o più relazioni di base o altre viste
- una vista è materializzata eseguendo l'interrogazione che la definisce

Il meccanismo delle viste è utile per

- semplificare l'accesso ai dati
- garantire la privatezza dei dati

Si definiscono associando un nome ed una lista di attributi al risultato di un'interrogazione.

Il comando di creazione di viste ha il seguente formato

```
create view V [ ( ListaAttributi ) ] as query [ with [ local | cascaded ] check option ]
```