A. Veneziani – Linguaggio SQL

Creazione di un DB – istruzione CREATE DATABASE

Per creare un nuovo database (DB), tramite il linguaggio SQL, si utilizza l'istruzione SQL:

CREATE DATABASE <nome del DB>;

è possibile eliminare complessivamente il DB in questione tramite il comando:

DROP DATABASE <nome del DB>;

Il database creato sarà ovviamente completamente vuoto e quindi privo di tabelle e dati.

Selezione di un DB (DB di lavoro)

Siccome il DBMS gestisce di prassi più DB, è necessario indicare allo stesso su quale DB si vada operando, ossia verso quale DB siano inviati i comandi che l'utente dà. Il comando SQL per effettuare la selezione di un DB è:

USE <nome database>;

è possibile passare ad utilizzare un altro qualsivoglia DB, dando di nuovo il comando con il nome del nuovo DB.

Creazione di una tabella – istruzione CREATE TABLE

Dopo aver creato il DB, per popolarlo di dati è essenziale creare i contenitori di tali dati (tabelle). Per creare una tabella si deve utilizzare il comando SQL CREATE TABLE.

Tale comando ha numerose sottoparti e varianti e quindi daremo qui solo un introduzione ai più comuni comandi SQL interni a CREATE TABLE.

Il comando nella sua sintassi generale ha la forma:

```
CREATE TABLE <nome tabella>  (
        <nome campo 1>  <tipo> <NULL / NOT NULL> <altri specificatori>,
        <nome campo 2>  <tipo> <NULL / NOT NULL>,
        <nome campo 3>  <tipo>  <NULL / NOT NULL>,
        ...
        <nome campo n>  <tipo> <NULL / NOT NULL> );
```

I nomi dei campi sono a piacere, e seguono regole del tutto simili a quelle dei nomi di variabili in C++.

Nomi di campi con spazi possono essere ammessi solo se il campo è riquadrato tra apici.

Il tipo del campo è uno dei tipi presenti a pagina 105 del libro di testo, ove sono elencati quasi tutti i tipi disponibili.

Sostanzialmente noi abbiamo usato negli esempi di DB finora sviluppati:

Tipo	**Contenuto / utilizzo**
INT	Intero con segno (4 byte)
TINYINT	Intero con segno breve (1 Byte)
SMALLINT	Intero con segno medio (2 byte)
INT UNSIGNED	Intero senza segno (4 byte)
DECIMAL(m, n)	Decimale con n cifre decimali (spesso usato per euro)
FLOAT	Numero con la virgola
CHAR(n)	Stringa lunghezza fissa
VARCHAR(n)	Stringa di lunghezza variabile
TEXT	Testo lungo
BIT	Flag booleano
ENUM(...,...,...)	Enumerazione di vari elementi (il campo può

	assuemere uno solo dei valori proposti)
TIME	Contiene un tempo
DATE	Contiene una data
DATETIME	Contiene data + tempo

A seguire segue l'importante specifica che determina se per il campo sono ammessi valori NULL o no. Si tenga presente che per la chiave primaria il valore NULL è escluso per definizione.
La possibilità di ammettere il NULL è di default, ossia se non viene indicato nulla (o viene indicato NULL) vuol dire che il valore NULL per il campo è ammesso. Per escludere che il campo possa assumere il valore NULL bisogna specificare NOT NULL.
Di prassi in cima alla lista dei campi va/vanno le/le chiavi primaria/e ed in fondo alla lista le chiavi esterne.
Alcuni specificatori che possono seguire il tipo sono:

Tipo	**Contenuto / utilizzo**
PRIMARY KEY	Il campo che ha questa specifica diviene chiave primaria
AUTO_INCREMENT	Usato nei campi chiave primaria numerici. Impone un avanzamento automatico del valore del campo chiave primaria (numerico, intero)
DEFAULT <valore>	Indica un valore di default da inserire nel campo se non è indicato un valore specifico durante una INSERT

Specificatori che possono definire altre proprietà sui vari campi sono:

Tipo	**Contenuto / utilizzo**
PRIMARY KEY(,...)	Definisce una chiave primaria su più campi
UNIQUE(,...)	Definisce un vincolo di unicità dei valori su uno o più campi
INDEX(....)	Definisce un campo come indice, ossia lo organizza di modo da effettuare operazioni di ricerca in modo più veloce[12]

Che vengono scritti, questi ultimi, su riga a se stante.

Modificare una tabella – Istruzione ALTER TABLE

Per modificare una tabella senza ricostruirla da zero è possibile utilizzare l'istruzione ALTER TABLE.
Anche di questa istruzione esistono diverse varianti.
L'istruzione per inserire una colonna (come ultima) ad esempio è:

```
ALTER TABLE utenti ADD COLUMN Professione VARCHAR(50) NOT NULL;
```

La sintassi generale di questa forma sarà quindi:

ALTER TABLE <tabella> ADD COLUMN <nome colonna> <tipo> <altre specifiche>;

Volendo inserire la colonna Professione in un ordine ben preciso nella tabella è possibile far ricorso alla sintassi:

```
ALTER TABLE utenti ADD COLUMN Professione VARCHAR(50) NOT NULL AFTER Cognome;
```

la specifica AFTER indica il nome della colonna dopo la quale sarà inserita quella di cui si stà parlando.
Per eliminare la stessa colonna si opererà con:

```
ALTER TABLE utenti DROP COLUMN Professione;
```

in questo caso è ovvio che non è necessario, trattandosi di una cancellazione, indicare la tipologia e neppure la posizione della colonna, ma solo il nome.
Se invece si vuole variare il nome, il tipo di campo o le specifiche di una colonna, si potrà usare il comando ALTER TABLE del tipo in modalità modifica:

```
ALTER TABLE Utenti CHANGE COLUMN Professione Professione TEXT;
```

[1] In compenso diventano più onerosi i tempi di inserimento di un elemento e lo spazio su disco consumato.
[2] Definire vincoli di integrità converte sempre automaticamente in campi indice i campi chiave esterna coinvolti in esso.

ossia:

ALTER TABLE <nome tabella> CHANGE COLUMN <nome attuale colonna> <nuovo nome colonna> <nuovo tipo> <nuove specifiche colonna>;

Eliminazione di un DB o di una tabella

Per eliminare un intero DB o una singola tabella è possibile utilizzare le apposite istruzioni:

DROP <nome database>;

DROP <nome tabella>;

per eliminare invece un campo da una tabella, si userà il comando DROP COLUMN, visto in precedenza, nella istruzione ALTER TABLE.

Istruzioni SHOW

Serie di istruzioni che servono ad analizzare la struttura del DBMS. Permette di conoscere il nome dei database, delle tabelle in un DB e delle colonne in una tabella.

Ad esempio:

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| accessi            |
| biblioteca         |
| cartelle           |
| cartelle2          |
| cdcol              |
| compravendite      |
| concerti           |
...
```

Elenca i nomi di tutti i DB presenti nel DBMS, o comunque visibili all'utente corrente.

In modo analogo, selezionato uno specifico DB, è possibile con:

```
mysql>  USE elezioni;
Database changed
mysql>  SHOW TABLES;
+--------------------+
| Tables_in_elezioni |
+--------------------+
| circoscrizioni     |
| elettori           |
| partiti            |
| presenze           |
+--------------------+
4 rows in set (0.00 sec)
```

Ed infine un altro comando spesso usato è quello capace di elencare i campi di una certa tabella e le loro carratteristiche (tipo, lunghezza, ammissibilità valore NULL, ecc.):

```
mysql> SHOW COLUMNS FROM elettori;
+-------------+--------------+------+-----+---------+-------+
| Field       | Type         | Null | Key | Default | Extra |
+-------------+--------------+------+-----+---------+-------+
| Carta_Ident | char(10)     | NO   | PRI | NULL    |       |
| Sesso       | enum('M','F') | NO   |     | NULL    |       |
| DataNascita | date         | NO   |     | NULL    |       |
| NomeCircoscr | varchar(50)  | NO   | MUL | NULL    |       |
| NomePartito | varchar(50)  | YES  | MUL | NULL    |       |
+-------------+--------------+------+-----+---------+-------+
5 rows in set (0.01 sec)
```

Istruzione INSERT INTO

Serve per inserire un record o riga in una tabella, ossia per inserire dati nel DB.

Esistono alcune varianti della sua sintassi. Consideriamo la tabella Utenti definita da:

```
CREATE TABLE Utenti  (
Id_Utente    INT UNSIGNED PRIMARY KEY,
Nome         VARCHAR(30) NOT NULL,
Cognome      VARCHAR(30) NOT NULL,
DataNascita DATE NOT NULL,
Sesso        ENUM('M','F')  NOT NULL  );
```

Per inserire un nuovo utente effettueremo l'istruzione:

```
INSERT INTO utenti VALUES(1,'Mario','Rossi','1990-03-29','M');
```

Ovviamente ai successivi inserimenti il primo dato che è chiave primaria dovrà variare opportunamente, ossia eventualmente tramite un contatore che opera in modo indipendente o altro. Questo dato comunque non potrà essere ripetuto. Il dato và comunque specificato.

L'iserimento di un dato NULL, in questo caso, ossia campo chiave primaria non auto increment, porta ad un errore. Una sintassi alternativa è:

```
INSERT INTO utenti(Id_Utente,Nome,Cognome,DataNascita,Sesso)
VALUES(2,'Gianni','Gialli','1995-04-22','M');
```

Ove si specificano I nomi di tutti i campi in cui si vanno ad inserire dati. Quindi, si può indicare solo alcuni campi e lasciare gli altri al loro valore di default:

```
INSERT INTO utenti(Id_Utente,Nome,Cognome)
VALUES(3,'Carlo','Bianchi');
```

in questo caso i campi DataNascita e Sesso rimangono al loro valore di default ('0000-00-00' per DataNascita e 'M' per Sesso – è il primo valore enumerato del campo ENUM).

Se il campo chiave primaria è del tipo auto_increment (ossia un numerico con autoincremento automatico), allora le sintassi per inserire dati saranno:

```
INSERT INTO utenti VALUES(NULL,'Nino','Neri','2001-03-20','M');
```

L'indicazione NULL, qui segnala che non viene imposto un valore al campo (ma in realtà esso non assume il valore NULL), oppure l'equivalente:

```
INSERT INTO utenti(Nome,Cognome,DataNascita,Sesso)
 VALUES(NULL,'Nino','Neri','2001-03-20','M');
```

Dove si omette il campo chiave primaria, lasciandolo libero di autodeterminare il suo valore.

Infine si deve tener conto che è possibile anche effettuare inserimenti multipli di dati tramite una sola INSERT INTO, tramite la sintassi:

```
INSERT INTO utenti
VALUES
(6,'Carla','Bruni','1980-03-02','F'),
(7,'Nina','Zilli','1985-10-12','F');
```

Si noti anche che non avendo senso in un inserimento effettuare una operazione di selezione, la clausola WHERE, nella INSERT INTO non è presente.

Vincoli di integrità referenziale possono impedire le operazioni di inserimento non valide.

Istruzione DELETE

L'istruzione DELETE serve a cancellare record o righe dale tabelle di un DB. L'istruzione di solito opera su righe multiple e quindi sono possibili cancellazioni di interi gruppi di righe.

L'istruzione DELETE è conformata come:

```
DELETE FROM utenti WHERE Id_Utente = 7;
```

In questo caso la DELETE cancella un solo utente dalla tabella, quello relativo al record 7. Una DELETE senza clausola WHERE cancella tutti i record della tabella Utenti.

```
DELETE FROM utenti
```

Vincoli di integrità referenziale possono impedire le operazioni di cancellazione non valide.

Istruzione UPDATE

Permette l'aggiornamento di uno o alcuni campi di una tabella, su uno, alcuni o tutti i record.

L'UPDATE ha la sintassi:

```
UPDATE utenti SET Nome='Valeria' WHERE Id_Utente = 6;
```

ovviamente più o tutti i campi della tabella possono essere coinvolti:

```
UPDATE utenti SET Nome='Valerio', Cognome='Bianchi',
                  DataNascita='1984-03-05', Sesso='M'
                  WHERE Id_Utente = 6;
```

Ovviamente la UPDATE può operare su record multipli, effettuando aggiornamenti multipli.
Senza la clausola WHERE, l'UPDATE effettua le sue modifiche su tutti i record della tabella.

```
UPDATE utenti SET Sesso = 'M';
```

Istruzione SELECT

L'istruzione SELECT è una delle più complesse del linguaggio SQL, e quindi vedremo via via sue diverse applicazioni e varianti durante l'anno. Ad ogni modo iniziamo qui a dare una panoramica del comando:
La forma più semplice della SELECT è quella che opera su una singola tabella senza effettuare selezioni (ossia dare condizioni di filtro dei record) e senza indicare campi specifici da mostrare / non mostrare.

SELECT * FROM <tabella>

In pratica questa istruzione estrae tutti i dati di una tabella in quanto non ha filtri applicati (parte WHERE) e indica che vanno estratte tutte le colonne di cui la tabella è dotata.
Varianti semplici di questa istruzione sono quelle che inseriscono qualche criterio di selezione (dare una condizione che faccia apparire solo alcuni record) o proiezione (produrre come risultato, un altra tabella, con solo alcune colonne della tabella originaria):
Riferendosi sempre alla solita tabella Utenti:

```
SELECT * FROM utenti;
```

```
mysql> SELECT * FROM utenti;
+-----------+---------+---------+-------------+-------+
| Id_Utente | Nome    | Cognome | DataNascita | Sesso |
+-----------+---------+---------+-------------+-------+
|         1 | Mario   | Rossi   | 1990-03-29  | M     |
|         6 | Valerio | Bianchi | 1984-03-05  | M     |
|         7 | Carla   | Verdi   | 1995-05-21  | F     |
+-----------+---------+---------+-------------+-------+
3 rows in set (0.00 sec)
```

E con proiezione:

```
SELECT Nome, Cognome FROM utenti;
```

```
mysql> SELECT Nome, Cognome FROM utenti;
+---------+---------+
| Nome    | Cognome |
+---------+---------+
| Mario   | Rossi   |
| Valerio | Bianchi |
| Carla   | Verdi   |
+---------+---------+
3 rows in set (0.00 sec)
```

Oppure selezione:

```
SELECT * FROM utenti WHERE Id_utente > 2;
```

```
mysql> SELECT * FROM utenti WHERE Id_Utente > 2;
+-----------+---------+---------+-------------+-------+
| Id_Utente | Nome    | Cognome | DataNascita | Sesso |
+-----------+---------+---------+-------------+-------+
|         6 | Valerio | Bianchi | 1984-03-05  | M     |
|         7 | Carla   | Verdi   | 1995-05-21  | F     |
+-----------+---------+---------+-------------+-------+
2 rows in set (0.00 sec)
```

Oppure è anche possible la combinazione delle due operazioni assieme con:

```
SELECT Nome, Cognome FROM utenti WHERE Id_utente > 2;
```

```
mysql> SELECT Nome, Cognome FROM utenti WHERE Id_utente > 2;
+---------+---------+
| Nome    | Cognome |
+---------+---------+
| Valerio | Bianchi |
| Carla   | Verdi   |
+---------+---------+
2 rows in set (0.00 sec)
```

A seconda dei casi condizioni combinate in modo multiplo possono essere abbinate con gli operatori AND e OR. Ad esempio si cerca l'utente di Cognome 'Rossi' o quelli di sesso femminile. La query diverrà:

```
mysql> SELECT utenti.*
    -> FROM utenti
    -> WHERE Cognome = 'Rossi' OR Sesso = 'F';
+-----------+-------+---------+-------------+-------+
| Id_Utente | Nome  | Cognome | DataNascita | Sesso |
+-----------+-------+---------+-------------+-------+
|         1 | Mario | Rossi   | 1990-03-29  | M     |
|         7 | Carla | Verdi   | 1995-05-21  | F     |
+-----------+-------+---------+-------------+-------+
2 rows in set (0.00 sec)
```

Invece una query con combinazione di due condizioni che devono essere entrambe verificate è:

```
mysql> SELECT utenti.*
    -> FROM utenti
    -> WHERE Cognome = 'Rossi' AND Sesso = 'M';
+-----------+-------+---------+-------------+-------+
| Id_Utente | Nome  | Cognome | DataNascita | Sesso |
+-----------+-------+---------+-------------+-------+
|         1 | Mario | Rossi   | 1990-03-29  | M     |
+-----------+-------+---------+-------------+-------+
1 row in set (0.00 sec)
```

Prodotto cartesiano

Consideriamo un DB esteso rispetto al precedente costituito solo da una tabella:

```
CREATE TABLE utenti (
Id_Utente INT(10) UNSIGNED NOT NULL,
Nome VARCHAR(30) NOT NULL,
Cognome VARCHAR(30) NOT NULL,
DataNascita DATE NOT NULL,
Sesso ENUM('M','F') NOT NULL,
PRIMARY KEY (`Id_Utente`)  );

CREATE TABLE accessi (
Id_Accesso INT(10) UNSIGNED NOT NULL,
Momento DATETIME NOT NULL,
Id_Utente INT(10) UNSIGNED NOT NULL,
PRIMARY KEY (Id_Accesso)  );
```

L'operazione che produce i record risultato del prodotto cartesiano[3] tra due tabelle diverse è una delle più importanti e ricorrenti nel linguaggio SQL e nelle basi di dati relazionali.
Essa è data dalla seguente sintassi:

[3] Ossia i record risultanti da tutte le possibili combinazioni di tutti i record della prima e della seconda tabella.

```
SELECT * FROM utenti, accessi
```

Che potrebbe anche essere scritto in maniera alternativa come:

```
SELECT * FROM utenti inner join accessi;
```

Preferiremo durante l'anno utilizzare la prima forma, per non essere dispersivi.

Nel nostro caso se le tabelle utenti, accessi, contenevano 3 record ciascuno il risultato era costituitoda 9 record, ossia 3 x 3.

Di solito una query così fatta non è utile per effettuare ricerche in quanto combina in modo disaggregato tutti i dati di una tabella con dati di un altra. Più utile è prendere un preciso sottinsieme di questo insieme di record che verifica precise condizioni. Iniziamo da una selezione su questo insieme riguardante il quesito più ovvio considerato il DB stesso, ossia dato un utente indicare i suoi accessi al sistema.

Eseguendo il prodotto cartesiano abbiamo il risultato:

```
mysql> SELECT * FROM utenti, accessi;
+-----------+---------+---------+-------------+-------+------------+---------------------+-----------+
| Id_Utente | Nome    | Cognome | DataNascita | Sesso | Id_Accesso | Momento             | Id_Utente |
+-----------+---------+---------+-------------+-------+------------+---------------------+-----------+
|         1 | Mario   | Rossi   | 1990-03-29  | M     |          1 | 2018-02-02 10:10:00 |         1 |
|         6 | Valerio | Bianchi | 1984-03-05  | M     |          1 | 2018-02-02 10:10:00 |         1 |
|         7 | Carla   | Verdi   | 1995-05-21  | F     |          1 | 2018-02-02 10:10:00 |         1 |
|         1 | Mario   | Rossi   | 1990-03-29  | M     |          2 | 2018-02-10 16:53:58 |         6 |
|         6 | Valerio | Bianchi | 1984-03-05  | M     |          2 | 2018-02-10 16:53:58 |         6 |
|         7 | Carla   | Verdi   | 1995-05-21  | F     |          2 | 2018-02-10 16:53:58 |         6 |
|         1 | Mario   | Rossi   | 1990-03-29  | M     |          3 | 2018-01-23 16:54:15 |         6 |
|         6 | Valerio | Bianchi | 1984-03-05  | M     |          3 | 2018-01-23 16:54:15 |         6 |
|         7 | Carla   | Verdi   | 1995-05-21  | F     |          3 | 2018-01-23 16:54:15 |         6 |
+-----------+---------+---------+-------------+-------+------------+---------------------+-----------+
9 rows in set (0.00 sec)
```

Dove si vede che ad ogni utente sono associati sia i suoi accessi, sia accessi non suoi, ossia di altri utenti.

Come fare a produrre un risultato che associ correttamente i due dati (utente ed accessi) ?

La risposta è che bisogna inserire delle condizioni aggiuntive alla espressione cartesiana sopra indicata.

Queste condizioni, nella quasi totalità di casi sono condizioni di ugualianza e nel nostro caso condizioni di uguaglianza tra l'Id_Utente, presente nella tabella Utenti e l'Id_Utente dalla tabella Accessi.

Quindi di tutte le combinazioni prodotte dal prodotto cartesiano delle due tabelle, si prendono quelle che hanno questa corrispondenza, ossia associano elementi che hanno una affinità logica (l'accesso di quell'utente indicato, non di altri).

In pratica quindi l'istruzione di selezione diviene:

```
SELECT * FROM utenti, accessi WHERE utenti.Id_Utente = accessi.Id_Utente;
```

```
mysql> SELECT * FROM utenti, accessi WHERE utenti.Id_Utente = accessi.Id_Utente;
+-----------+---------+---------+-------------+-------+------------+---------------------+-----------+
| Id_Utente | Nome    | Cognome | DataNascita | Sesso | Id_Accesso | Momento             | Id_Utente |
+-----------+---------+---------+-------------+-------+------------+---------------------+-----------+
|         1 | Mario   | Rossi   | 1990-03-29  | M     |          1 | 2018-02-02 10:10:00 |         1 |
|         6 | Valerio | Bianchi | 1984-03-05  | M     |          2 | 2018-02-10 16:53:58 |         6 |
|         6 | Valerio | Bianchi | 1984-03-05  | M     |          3 | 2018-01-23 16:54:15 |         6 |
+-----------+---------+---------+-------------+-------+------------+---------------------+-----------+
3 rows in set (0.00 sec)
```

questa parte aggiuntiva che permette la selezione di alcune tuple in base alla condizione di uguaglianza è quella indicata in fondo all'istruzione SQL, essa indica che di tutte le combinazioni tra utenti ed accessi, devono essere prese quelle in cui l'identificatore dell'utente corrisponde a quello dell'utente nell'accesso (ossia prendendo solo gli accessi effettuati da quell'utente).

Se volessimo sapere quindi gli accessi dell'utente 'Rossi' dovremmo mettere poi una condizone aggiuntiva sull'utente. Tale condizione và accostata (aggiunta) con un AND, dato che essa aumenta la restrittitività della selezione:

```
mysql> SELECT * FROM utenti, accessi
    -> WHERE utenti.Id_Utente = accessi.Id_Utente AND utenti.Cognome = 'Rossi';
+-----------+-------+---------+-------------+-------+------------+---------------------+-----------+
| Id_Utente | Nome  | Cognome | DataNascita | Sesso | Id_Accesso | Momento             | Id_Utente |
+-----------+-------+---------+-------------+-------+------------+---------------------+-----------+
|         1 | Mario | Rossi   | 1990-03-29  | M     |          1 | 2018-02-02 10:10:00 |         1 |
+-----------+-------+---------+-------------+-------+------------+---------------------+-----------+
1 row in set (0.00 sec)
```

L'operazione appena effettuata è detta di inner join (detto anche join naturale), in questo caso sulle due tabelle utenti ed accessi.

Volendo evidenziare solo i dati salienti del problema ed eliminare quelli superflui potremo decidere diaggiungere l'operazione di proiezione:

```
mysql> SELECT Nome, Cognome, Momento FROM utenti, accessi WHERE utenti.Id_Utente = accessi.Id_Utente
+---------+---------+---------------------+
| Nome    | Cognome | Momento             |
+---------+---------+---------------------+
| Mario   | Rossi   | 2018-02-02 10:10:00 |
| Valerio | Bianchi | 2018-02-10 16:53:58 |
| Valerio | Bianchi | 2018-01-23 16:54:15 |
+---------+---------+---------------------+
3 rows in set (0.00 sec)
```

La stessa istruzione potrebbe essere scritta nella forma (che di norma non utilizzeremo, perchè leggermente più complessa da scrivere):

```
mysql> SELECT Nome, Cognome, Momento
    -> FROM utenti INNER JOIN accessi
    -> ON utenti.Id_Utente = accessi.Id_Utente;
+---------+---------+---------------------+
| Nome    | Cognome | Momento             |
+---------+---------+---------------------+
| Mario   | Rossi   | 2018-02-02 10:10:00 |
| Valerio | Bianchi | 2018-02-10 16:53:58 |
| Valerio | Bianchi | 2018-01-23 16:54:15 |
+---------+---------+---------------------+
3 rows in set (0.00 sec)
```