LINGUAGGIO SQL

Il linguaggio SQL (Structured Query Language) si suddivide in base alla funzionalità delle proprie istruzioni in

- **DDL**: Data Definition Language. Istruzioni per la creazione della base di dati (creazione tabelle, chiavi, vincoli, tipi di dato…)
- **DML**: Data Manipulation Language. Istruzioni per manipolare i dati (inserimento, eliminazione, modifica).
- **QL:** Query Language. Istruzioni per le interrogazioni

Come detto, nonostante la definizione degli standard, ogni DBMS usa un proprio "dialetto" di SQL che comprende la maggior parte delle istruzioni standard più altre istruzioni realizzate appositamente per quello specifico DBMS. Noi utilizzeremo l'SQL del DBMS **MariaDB**.

Una delle problematiche principali di SQL è la non completa portabilità fra DBMS diversi. Fra le altre cose uno dei fattori che limita la portabilità è il diverso modo di gestire i valori **NULL** da parte dei diversi DBMS.

Storia del DBMS MariaDB

Negli anni '80 del secolo scorso la società svedese MySQL AB di Michael "Monty" Widenius realizza il DBMS MySQL, distribuito sia con licenza GNU GPL sia con licenza commerciale.

L'azienda viene acquisita nel 2008 da SUN Microsystems (quelli che hanno inventato Java). Nel 2010 la SUN Microsystem viene acquistata da Oracle.

Nel frattempo (2009) il padre di SQL Michael Widenius, in disaccordo con alcune scelte di SUN Microsystem, fonda la società Monty Program AB che sviluppa il DBMS MariaDB come fork di MySQL.

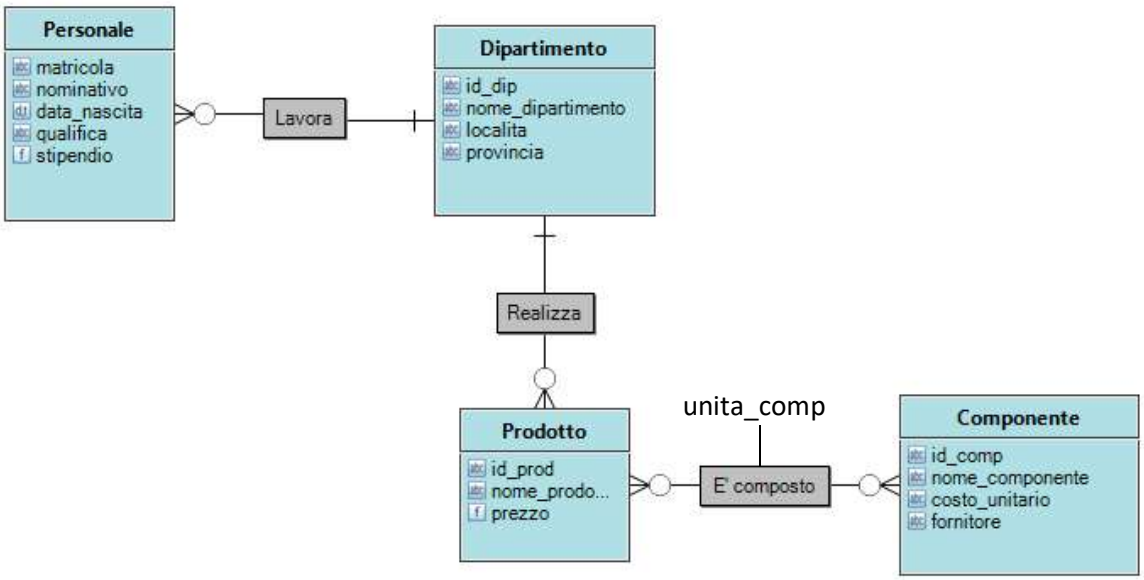
Esempio di riferimento

Per illustrare i comandi di SQL si utilizzerà il seguente DB relativo ad una azienda.

L'azienda è suddivisa in dipartimenti nei quali lavorano degli impiegati.

L'azienda produce dei prodotti, ogni prodotto è realizzato da uno dei dipartimenti dell'azienda, inoltre ogni prodotto è composto da un certo numero di componenti. Interessa memorizzare i componenti di cui è composto ogni prodotto.

Diagramma E/R



Schema del DB

Personale (matricola, nominativo, data_nascita, qualifica, stipendio, *id_dip*)

PK: matricola

FK: id_dip → dipartimento.id_dip

Dipartimenti (id_dip, nome_dipartimento, località, provincia)

PK: id_dip

Prodotti (id_prod, nome_prodotto, prezzo, *id_dip*)

PK:id_prod

FK: id_dip → dipartimento.id_dip

Componenti (id_comp, nome_componente, costo_unitario, fornitore)

PK: id_comp

Composizione (*id_prod*, *id_comp*, unità_comp)

PK : id_prod, id_comp

FK: id_prod → prodotti.id_prod

FK: id_cmp → componenti.id_comp

L’ultima tabella è ottenuta dalla associazione N a N fra Prodotti e Componenti e la chiave primaria è composta dalle due chiavi esterne. L’attributo unità_comp indica quanti elementi della entità componenti sono necessari per comporre ciascun prodotto.

Descrizione delle tabelle

Personale

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito
1	**matricola**	varchar(5)	latin1_swedish_ci		No	*Nessuno*
2	**nominativo**	varchar(30)	latin1_swedish_ci		No	*Nessuno*
3	**data_nascita**	date			No	*Nessuno*
4	**qualifica**	varchar(3)	latin1_swedish_ci		Sì	*NULL*
5	**stipendio**	float			No	*Nessuno*
6	**id_dip**	varchar(3)	latin1_swedish_ci		Sì	*NULL*

Dipartimenti

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito
1	**id_dip**	varchar(3)	latin1_swedish_ci		No	*Nessuno*
2	**nome**	varchar(30)	latin1_swedish_ci		No	*Nessuno*
3	**localita**	varchar(30)	latin1_swedish_ci		No	*Nessuno*
4	**provincia**	varchar(2)	latin1_swedish_ci		No	*Nessuno*

Prodotti

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito
1	**id_prod**	varchar(5)	latin1_swedish_ci		No	*Nessuno*
2	**id_dip**	varchar(5)	latin1_swedish_ci		No	*Nessuno*
3	**nome_prodotto**	varchar(30)	latin1_swedish_ci		No	*Nessuno*
4	**prezzo**	double			No	*Nessuno*

Composizioni

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito
1	**id_prod**	varchar(5)	latin1_swedish_ci		No	*Nessuno*
2	**id_comp**	varchar(5)	latin1_swedish_ci		No	*Nessuno*
3	**unita_comp**	int(11)			No	*Nessuno*

Componenti

	#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito
☐	1	id_comp	varchar(5)	latin1_swedish_ci		No	*Nessuno*
☐	2	nome_componente	varchar(30)	latin1_swedish_ci		No	*Nessuno*
☐	3	costo_unitario	double			No	*Nessuno*
☐	4	fornitore	varchar(30)	latin1_swedish_ci		No	*Nessuno*

Regole di scrittura utilizzate

Per ogni singolo campo di un database sarebbe possibile specificare se si vuole che i dati inseriti vengano interpretati come case sensitive oppure no. Di default il linguaggio non è case sensitive.

In maiuscolo scriviamo le parole chiave del linguaggio, chiamate anche **CLAUSOLE** (SELECT, FROM, WHERE…) e in minuscolo i nomi delle tabelle, le condizioni ecc. (my_table)

Inoltre per illustrare la sintassi utilizzeremo la seguente simbologia:

<……> : **parte variabile del comando che deve essere specificata al momento della formulazione effettiva dello stesso.**

[…..] : **indica una parte di comando opzionale (può esserci, può non esserci)**

{…..} : **indica parti di comando che contengono alternative**

| : **simbolo che separa le alternative**

Ad esempio, il generico comando di selezione è indicato nel seguente modo

SELECT <elenco_campi>	il nome reale dei campi va elencato al posto di <…..>
FROM <lista_tabelle>	il nome reale delle tabelle va elencato al posto di <…..>
[WHERE <condizione>];	la condizione, se presente, va indicata al posto di <…..>

Un esempio reale del comando di selezione è:

```
SELECT nominativo, stipendio
FROM   personale
WHERE personale.stipendio>2000
```

Una generica istruzione del tipo

{INSERT | DELETE | UPDATE}

significa che nell'istruzione reale si dovrà indicare uno solo fra i tre comandi senza le parentesi graffe

Il linguaggio SQL prevede l'utilizzo del **";"** a termine dell'istruzione (anche se molti DBMS accettano la sua omissione)

Le stringhe vengono indicate con apici (Esempio: 'Pierone'). In alcuni casi sono accettate anche le virgolette ("Pierone").

Le condizioni sono realizzate con gli operatori di confronto (=,<,>,>=,<=,< >) realizzando espressioni booleane con i connettivi logici scritti interamente (AND, OR, NOT)

Per dare la precedenza ad alcune operazioni rispetto ad altre si usano le parentesi tonde ( ).

Tutti gli esempi di codice SQL di seguito riportati possono essere testati utilizzando il Database Azienda (disponibile sulla piattaforma di supporto alla didattica Moodle) realizzato con DBMS MariaDB in ambiente PhpMyAdmin (o qualunque altro software di interfacciamento, front end, per MariaDB, ad esempio SQLyog, MySQLWorkbanch)

Per commentare in SQL si usa /*…commento */

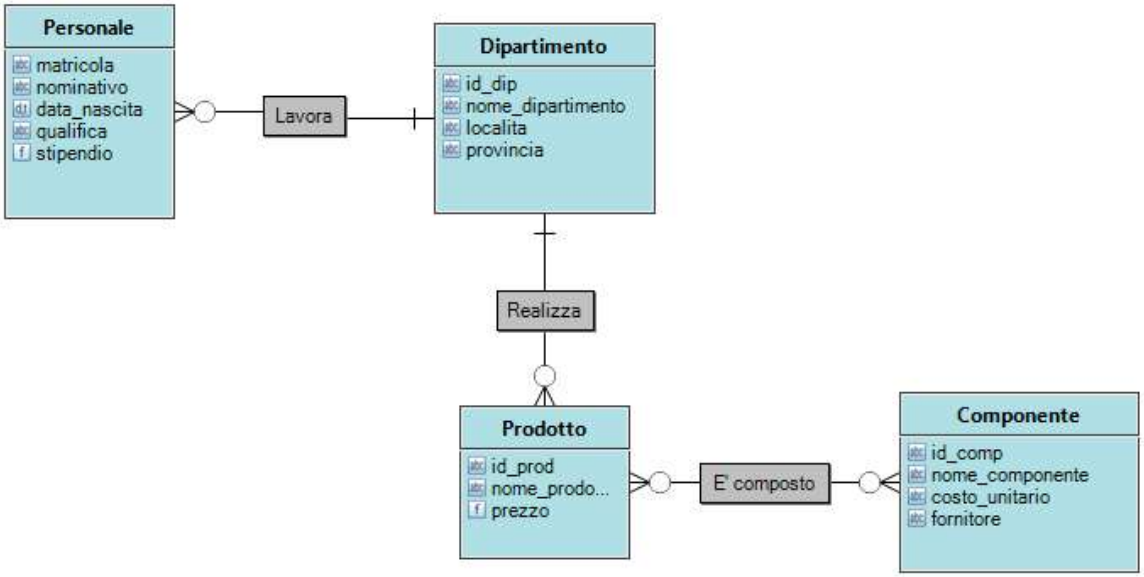
Cose da sapere con phpMyAdmin e MariaDB:

- Per esportare/importare tabelle e DB, utilizzare la scheda Importa e Esporta effettuando le operazioni **nel formato SQL**
- L'importazione agisce sulle singole tabelle. Per importare un DB precedentemente esportato bisogna quindi
    - prima creare un DB vuoto in phpMyAdmin e dargli un nome
    - poi importare il file sql del DB da importare dall'interno del database.
- Per rinominare e eliminare DB vai sul menu Operazioni.
- **Quando si fa la creazione con l'interfaccia grafica delle associazioni fra tabelle, le chiavi esterne devono o avere un indice di tipo index**
- **I file dei database creati si trovano nella cartella c:\xampp\mysql\data**

1. I COMANDI DDL DEL LINGUAGGIO SQL: CREATE, ALTER, DROP

I comandi DDL (Data Definition Language) di SQL sono i comandi che consentono di "creare" lo schema della base di dati: il database, le tabelle, i campi, vincoli di integrità, altri vincoli, gli indici. Gli indici sono assegnati a determinati campi per rendere più veloci le ricerche di dati che coinvolgono tali campi. I campi per i quali sono definiti degli indici vengono detti indicizzati. Metaforicamente l'indice è proprio come l'indice di un libro. Così come l'indice del libro permette di ricercare velocemente un argomento all'interno di un libro, così un indice permette al motore del DBMS di ricercare più velocemente un dato.

Per comprendere i comandi DDL utilizziamo il seguente esempio di un database di un'azienda:



IL COMANDO CREATE

Il comando CREATE consente di *creare* un nuovo database, una nuova tabella di un database, un indice, una vista di un database (vedremo successivamente cosa sono le viste).

Creazione di un nuovo database:

CREATE DATABASE <nome_database>

esempio:

CREATE DATABASE azienda

La creazione delle tabelle con CREATE TABLE

La sintassi è la seguente:

CREATE TABLE <nome_tabella> (

<nome campo1> <tipo1> [<vincolo1>],

…….,

<nome campo2> <tipo2> [<vincolo2>]);

I campi separati fra loro dalla virgola

Il tipo di dato relativo al campo

Il vincolo è facoltativo, specifica alcune caratteristiche del campo, può essere formato da più parti. Tali parti sono indicate una di seguito all'altra.

Esempio esplicativo:

creazione della tabella Personale

Valore assegnato di default al campo

Stringa con 5 caratteri (poi vedremo meglio i tipi di dato)

CREATE TABLE Personale (

matricola VARCHAR(5) NOT NULL DEFAULT ' ',

id_dip VARCHAR(5) DEFAULT NULL,

nominativo VARCHAR(30) DEFAULT NULL,

data_nascita DATE NOT NULL,

qualifica VARCHAR(5) DEFAULT NULL,

stipendio DOUBLE DEFAULT 1200);

Quando si aggiunge una riga il campo DEVE essere valorizzato (non può essere NULL)

La parola chiave **DEFAULT** consente di indicare un valore di default da assegnare al campo per ogni nuovo inserimento nel caso il campo non venisse avvalorato. Se DEFAULT non viene specificato in fase di creazione della tabella e durante l'inserimento dei dati il campo non viene avvalorato, il DBMS assegna a tale campo il valore **VUOTO**, **che NON E' NULL**, il valore VUOTO vale "" (stringa vuota) nel caso di VARCHAR, vale 0 nel caso di campo numerico, vale 0000-00-00 nel caso di una data.

Il vincolo **NOT NULL** obbliga a valorizzare un campo, ossia quel campo deve sempre avere un valore e non può essere mai NULL. L'inserimento di una riga con valore NULL non viene accettata.

UN CAMPO VUOTO NON E' NULL

ATTENZIONE: UN CAMPO VUOTO NON E' NULL! Se durante l'inserimento di dati non si specifica alcun valore per un campo, viene inserito un campo vuoto, ossia "" (stringa vuota) se è di tipo VARCHAR, oppure 0 se è di tipo numerico.

Se si vuole inserire un valore NULL (assenza di informazione) in un campo, **lo si deve specificare** in fase di inserimento (vedremo INSERT INTO…., oppure nell'interfaccia phpMyAdmin lo si specifica con la spunta sulla casella NULL).

Se in una tabella vi sono dei campi con restrizione NOT NULL (nell'esempio il campo data_nascita) e durante l'inserimento dei dati non viene specificato un valore per quel campo, il DBMS accetterà l'inserimento e assegnerà a quel campo il valore vuoto (nel caso della data 0000-00-00). Se invece si cercherà di assegnare un valore NULL a quel campo il DBMS rifiuterà l'inserimento!

- **Aggiunta di chiavi e vincoli alle tabelle con CREATE TABLE**

Il comando CREATE TABLE consente anche di indicare, in una tabella, quali sono le chiavi primarie (questo aggiunge implicitamente il vincolo e di **unicità ma NON il vincolo NOT NULL, questo va indicato esplicitamente!**), quali sono le chiavi esterne (quindi aggiungere il vincolo di **integrità referenziale**) e aggiungere altri vincoli (ad esempio vincolo di unicità su altri campi non chiave, vincoli NOT NULL)

Esempio esplicativo:

creazione della tabella Composizione:

Va indicato esplicitamente!

ChiavePrimaria è il nome che è stato dato al vincolo (scelto a piacere)

PRIMARY KEY (elenco campi) permette di definire i campi della CHIAVE PRIMARIA e quindi il vincolo di UNICITA' e NOT NULL. Nell'elenco i campi sono separati da virgole

CONSTRAINT (vincolare, costringere): è la parola chiave che indica l'aggiunta di un vincolo

```
CREATE TABLE Composizione(
id_prod VARCHAR(5) NOT NULL,
id_comp VARCHAR(5) NOT NULL,
unita_comp INTEGER,
CONSTRAINT ChiavePrimaria PRIMARY KEY (id_prod, id_comp),
CONSTRAINT ProdottiComposizione FOREIGN KEY (id_prod)
REFERENCES Prodotti(id_prod),
CONSTRAINT ComponentiComposizione FOREIGN KEY (id_comp)
REFERENCES Composizione(id_comp));
```

REFERENCES *nomeTabella(elenco campi)* individua la chiave primaria a cui fa riferimento la chiave esterna. Questo definisce il vincolo di INTEGRITA' REFERENZIALE.

ProdottiComposizione è il nome che è stato dato al vincolo (scelto a piacere)

FOREIGN KEY (elenco campi) permette di definire i campi di una CHIAVE ESTERNA .

I vincoli introdotti sono separati fra loro da una virgola

L'integrità referenziale fra due tabelle legate da associazione uno a molti, come detto, non consente di aggiungere una riga alla tabella *"a molti"* se il valore della chiave esterna non è già presente come chiave primaria nella tabella che partecipa alla associazione con ruolo *"a uno"*. Ad esempio non è possibile aggiungere un nuovo dipendente alla tabella *Personale* che abbia un valore di id_dip (dipartimento) che non sia già presente nella tabella *Dipartimenti*.

Quando si aggiunge il vincolo di integrità referenziale ad una tabella (ossia la chiave esterna) è possibile specificare cosa deve accadere ai valori delle tabelle del database quando nella tabella "a uno" viene modificata una chiave primaria che è chiave esterna nella tabella "a molti".

Ad esempio: cosa deve accadere alla tabella *Personale* se l'amministratore del DB **elimina** o **modifica** il valore di *id_dip* nella tabella *Dipartimenti*? Ci sono due possibili soluzioni alternative:

1. La modifica o eliminazione della chiave primaria (id_dip) in Dipartimenti non viene consentita finchè vi sono valori collegati nella tabella Personale.
2. La modifica o eliminazione della chiave primaria (id_dip) in Dipartimenti viene consentita e viene applicata "a cascata" sulle righe della tabella Personale.

La soluzione 1 garantisce una maggiore protezione dell'integrità dei dati.

La soluzione 2 è più comoda ma rischiosa. Con questa soluzione ad esempio se elimino il dipartimento D2 dalla tabella Dipartimenti, elimino automaticamente i dati di tutti i dipendenti del dipartimento D2 dalla tabella Personale (magari invece volevo semplicemente "spostarli" ad un altro dipartimento).

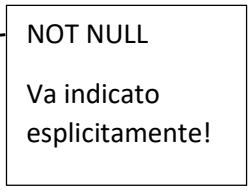
Di default SQL implementa la soluzione 1 (più prudente). Per implementare la soluzione 2 è necessario aggiungere una o entrambe le seguenti clausole **quando si definisce la chiave esterna**:

- **ON DELETE CASCADE** (se elimino un valore di chiave primaria, elimina tutte le righe che hanno tale valore come chiave esterna)
- **ON UPDATE CASCADE** (se modifico un valore di chiave primaria, modifica tutte le righe che hanno tale valore come chiave esterna)

Esempio: Creare due tabelle (Dipartimenti2 e Personale2) poi aggiungere dall'interfaccia due dipartimenti e due dipendenti (uno per dipartimento) e verificare il funzionamento in cascata modificando e poi eliminando il valore della chiave primaria.

```
CREATE TABLE dipartimenti2
(id_dip VARCHAR(5) NOT NULL PRIMARY KEY,
nome_dipartimento VARCHAR(30) NOT NULL,
localita VARCHAR(30) NOT NULL,
provincia VARCHAR(4) NOT NULL
);

CREATE TABLE Personale2
(matricola VARCHAR(5) NOT NULL PRIMARY KEY,
 nominativo VARCHAR(30) NOT NULL,
```



```
data_nascita DATE NOT NULL,
qualifica VARCHAR(5),
stipendio DOUBLE,
id_dip VARCHAR(5) NOT NULL,
CONSTRAINT ChiaveEsterna FOREIGN KEY (id_dip) REFERENCES
dipartimenti2(id_dip) ON DELETE CASCADE ON UPDATE CASCADE
);
```

Il comportamento ON DELETE e ON UPDATE, oltre a CASCADE, che aggiorna in cascata i valori secondo la gerarchia della integrità referenziale, può essere dato da una delle seguenti altre clausole.

ON DELETE e ON UPDATE:

CASCADE: la modifica è consentita e si ripercuote a cascata

RESTRICT (o **NO ACTION**): la modifica non viene consentita sulla tabella che contiene la chiave primaria, l'istruzione termina con un errore 1452 (è l'impostazione di default)

SET NULL: La modifica è consentita e il valore modificato nella tabella con chiave primaria viene sostituito, nella tabella con chiave esterna, con NULL

Vincolo UNIQUE

Per ogni singolo campo può essere definito il vincolo di unicità UNIQUE specificandolo come attributo contestualmente alla definizione del campo. Questo vincolo fa sì che il contenuto del campo NON possa ripetersi.

```
CREATE TABLE personale
(matricola varchar(5) PRIMARY KEY,
 nominativo VARCHAR(30) NOT NULL UNIQUE,
 data_nascita DATE NOT NULL,
 qualifica VARCHAR(5) NOT NULL,
 stipendio FLOAT NOT NULL,
 id_dip VARCHAR(5) NOT NULL,
 CONSTRAINT personale_dipartimenti FOREIGN KEY (id_dip)  REFERENCES dipartimenti(id_dip));
```

Il vincolo unique può essere aggiunto ad un **insieme** di attributi, in questo caso il valore di ogni singolo attributo può ripetersi, ciò che non si può ripetere è il valore di un **insieme** di attributi.

Ad esempio si potrebbe imporre che in un dipartimento non vi possa essere più di un dipendente con una determinata qualifica, quindi si impone il vincolo sulla coppia (qualifica,id_dip)

```
CREATE TABLE personale
(matricola varchar(5) PRIMARY KEY,
 nominativo VARCHAR(30) NOT NULL,
 data_nascita DATE NOT NULL,
 qualifica VARCHAR(5) NOT NULL,
```

stipendio FLOAT NOT NULL,
id_dip VARCHAR(5) NOT NULL,
CONSTRAINT personale_dipartimenti FOREIGN KEY (id_dip) REFERENCES dipartimenti(id_dip),
CONSTRAINT unica_qualifica UNIQUE(id_dip, qualifica));

Vincolo CHECK

La sintassi dell'SQL2003 prevede la possibilità di imporre dei vincoli di attributo. Tale vincolo consente di inserire una tupla solamente se è verificata l'espressione di controllo che deve essere un predicato dell'algebra di Boole. Tale predicato si può esprimere su uno o più attributi. La sintassi è la seguente

CONSTRAINT controllo_stipendio CHECK (stipendio>=0)

"Lo stipendio non può essere negativo"

Oppure, nel caso di predicato su più attributi

CONSTRAINT controllo_stipendio CHECK (stipendio>2000 AND qualifica="Q5")

"Se qualifica =Q5 lo stipendio deve essere >2000 e viceversa"

Dalla versione 10.2.1 MariaDB accetta il vincolo CHECK, nelle versioni precedenti tale vincolo non era accettato per motivi di compatibilità con altri DBMS. Dettagli sull'utilizzo al seguente link: (https://mariadb.com/kb/en/constraint/)

Esempio sintassi 1:

CREATE TABLE personale2

(matricola varchar(5) NOT NULL PRIMARY KEY,

nominativo varchar(30) NOT NULL,

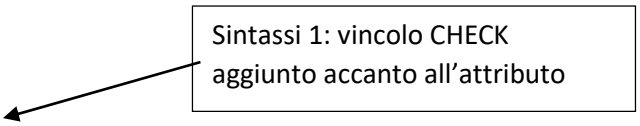
data_nascita date NOT NULL,

qualifica varchar(3),

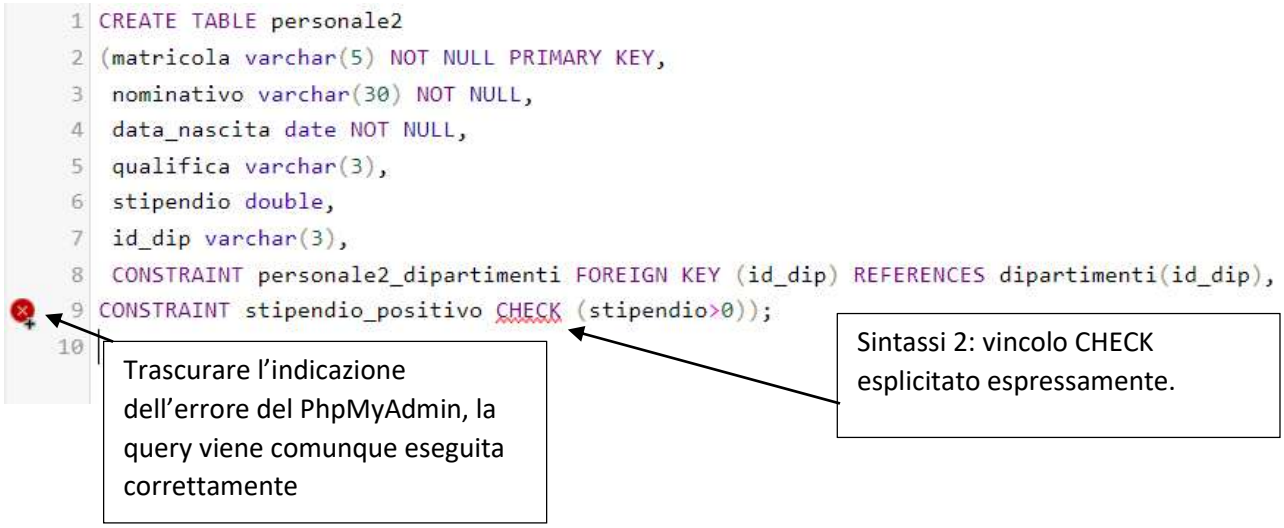
stipendio double NOT NULL CHECK (stipendio>0),

id_dip varchar(3),

CONSTRAINT personale2_dipartimenti FOREIGN KEY (id_dip) REFERENCES dipartimenti(id_dip));



Esempio sintassi 2



Esempio vincolo complesso

CREATE TABLE personale2
(matricola varchar(5) NOT NULL PRIMARY KEY,
nominativo varchar(30) NOT NULL,
data_nascita date NOT NULL,
qualifica varchar(3),
stipendio double NOT NULL,
id_dip varchar(3),
CONSTRAINT personale2_dipartimenti FOREIGN KEY (id_dip) REFERENCES dipartimenti(id_dip),
CONSTRAINT controllo_stipendio CHECK (stipendio>2000 AND qualifica='Q5'));

Creare un indice su un campo di una tabella esistente

Creare un indice su un campo di una tabella significa "segnalare" tale campo al motore del database (database engine), la parte che si occupa della memorizzazione e ricerca fisica dei dati, in modo da **velocizzare** le **ricerche** dei dati basate su quel campo. Il prezzo da pagare è che **rallentano** le operazioni di **inserimento** che coinvolgono quel campo, quindi gli indici vanno utilizzati con accortezza, e prevalentemente sui campi che hanno tante ricerche e pochi inserimenti.

CREATE [UNIQUE] INDEX <nome_indice>

ON <nome_tabella>(<nome_campo>)

La clausola **UNIQUE** è opzionale e consente di rendere il campo a cui viene assegnato l'indice *unico*, ossia aggiunge il vincolo UNIQUE a quel campo.

Esempio:

CREATE UNIQUE INDEX indice_nominativo
ON personale(nominativo);

Crea un indice sul campo nominativo della tabella personale

- **Visualizzare l'elenco degli indici presenti in una tabella**

  SHOW INDEX FROM <nomeTabella>

Esempio:

show index FROM personale;

- **Creare una tabella a partire da una query:**

  CREATE TABLE <nome_tabella>

  SELECT <nome_campo1, nom_campo2,..>

  FROM <lista_tabelle>

  [WHERE <condizione>];

  Esempio:

  CREATE TABLE Dirigenti

  SELECT matricola, nominativo

  FROM personale

  WHERE qualifica='Q5';

I COMANDI ALTER, RENAME E DROP

Attraverso questi comandi è possibile **apportare modifiche** a una **tabella** già realizzata.

1. **RENAME**

   RENAME TABLE <vecchio_nome> TO <nuovo_nome>

   Permette di rinominare una tabella

   Esempio:

   RENAME TABLE personale TO dipendenti; (poi rinominala di nuovo)

2. **ALTER**

   ALTER <nome_tabella> <operazione_di _modifica>

   Per mostrare il funzionamento creo una tabella CronologiaPersonale solamente definendo i campi e poi la modifico con ALTER TABLE

   CREATE TABLE Cronologia_personale

   (id_cronologia integer,

   matricola varchar(5),

   nota TEXT);

   ADD: Aggiunge un campo, un indice, un vincolo.

   Aggiunta chiave primaria:

   Alter table cronologia_personale ADD
   PRIMARY KEY (id_cronologia);

   Aggiunta di un campo data_inserimento_nota:

   ALTER TABLE cronologia_personale ADD
   data_inserimento_nota DATE;

   Aggiunta indice sulla data_inserimento_nota (sintassi diversa da CREATE INDEX)

   ALTER TABLE cronologia_personale ADD
   INDEX indice_data (data_inserimento_nota);

   Aggiunta chiave esterna a matricola

   ALTER TABLE cronologia_personale ADD

CONSTRAINT ChiaveEsterna FOREIGN KEY (matricola) REFERENCES personale(matricola);

CHANGE: per cambiare nome e il tipo di dato di un campo (il tipo di dato va sempre specificato)

Cambio il nome del campo note di tipo text in azioni di tipo varchar(250)

ALTER TABLE cronologia_personale CHANGE
nota azioni VARCHAR(250);

MODIFY: per modificare il tipo di dato e le caratteristiche di un campo (ma non il nome)

Cambio il campo id_cronologia in BIGINT e aggiungo la caratteristica **AUTO_INCREMENT (**questa caratteristica fa incrementare automaticamente un valore intero di 1 ogni volta che viene aggiunta una nuova riga alla tabella)

ALTER TABLE cronologia_personale MODIFY
id_cronologia BIGINT AUTO_INCREMENT;

DROP: rimuove un campo dalla tabella

Rimozione del campo Azioni dalla tabella

ALTER TABLE cronologia_personale DROP azioni;

Con **ALTER TABLE….DROP** si possono rimuovere anche indici e vincoli di chiavi esterne

Esempi:

ALTER TABLE cronologia_personale
DROP INDEX data_assunzione;

ALTER TABLE cronologia_personale
DROP FOREIGN KEY cronologia_personale_personale

3. **DROP**

Rimuove un elemento da un database (una tabella, un indice, il database stesso). Può agire sull'intero database, non solamente su una tabella

Esempio: eliminazione dell'indice *indice_data* dalla tabella
DROP INDEX indice_data ON cronologia_personale;

Esempio: eliminazione della tabella *cronologia_personale* dal database Azienda:
DROP TABLE cronologia_personale;

Esempio eliminazione del database Azienda2 creato precedentemente:
DROP DATABASE Azienda2;

2. I TIPI DI DATO IN SQL

Come visto, quando si crea un campo in una tabella è necessario indicare il tipo di dato relativo. Oltre ai tipi di dato definiti dallo standard SQL, ogni DBMS prevede dei propri tipi di dato specifici (impariamo quelli di MySQL/MariaDB poi, all'occorrenza ognuno può individuare i dati di cui ha bisogno nella documentazione dello specifico DBMS di interesse).

I tipi di dato dell' SQL di MariaDB si suddividono in 3 categorie principali:

- dati numerici
- date e orari
- stringhe di caratteri

Dati numerici

Gli attributi fra parentesi quadre indicati nella tabella seguente sono opzionali. In generale si tenga presente che, se non diversamente specificato

> **UNSIGNED** indica che il numero è senza segno quindi non sono ammessi valori negativi e, per i valori interi, fa raddoppiare il massimo numero rappresentabile
>
> **ZEROFILL** indica di memorizzare nel server il numero specificato aggiungendo degli zeri davanti ad esso nel caso non siano occupati tutti i bit. (Il suo utilizzo consente di risparmiare spazio facilitando la compressione dei dati e poi ha un altro vantaggio che vedremo dopo)
>
> **M** indica il numero di cifre da visualizzare complessive del numero. (Es. se M=3 il numero valore 12 verrà visualizzato come 012, se M=5 verrà visualizzato come 00012)
>
> **D** indica il numero di cifre decimali da visualizzare del numero. Quindi un numero FLOAT (2,1), anche se il range è circa $-3.4*10^{38}$ e $3.4*10^{-38}$, avendo indicato M=2 e D=1 ha come valore minimo -9,9 e massimo 9,9 perché può avere al massimo due cifre di cui una decimale.

Le seguenti tabelle si possono trovare sul manuale dalla pag. 1122

BIT [(M)]	È un dato per memorizzare valori binari. Contiene il numero di bit specificato con M (1 per default), che può andare da 1 a 64. Un valore di questo tipo può essere indicato ad es. con *b'111'*, che rappresenta in questo caso tre bit a 1 (corrispondenti al valore decimale 7)	non consente le opzioni UNSIGNED e ZEROFILL
TINYINT [(M)] [UNSIGNED] [ZEROFILL] *Sinonimi* di TINYINT(1): BOOL BOOLEAN	intero a 1 byte	range -128 ..+127 range unsigned 0..255

SMALLINT [(M)] [UNSIGNED] [ZEROFILL]	intero a 2 byte	range -32768 ..+32767 range unsigned 0..655535
MEDIUMNT [(M)] [UNSIGNED] [ZEROFILL]	intero a 3 byte	range -8388608 ..+8388607 range unsigned 0..16777216
INT [(M)] [UNSIGNED] [ZEROFILL] *Sinonimi*: INTEGER	intero a 4 byte	range circa da -2 miliardi a +2 miliardi range unsigned circa da 0 a 4 miliardi
BIGINT [(M)] [UNSIGNED] [ZEROFILL]	intero a 8 byte	range circa da -9 miliardi di miliardi a +9 miliardi di miliardi range unsigned circa da 0 a 18 miliardi di miliardi
FLOAT [(M,D)] [UNSIGNED] [ZEROFILL]	numero reale a virgola mobile a precisione singola (4 byte)	range numeri negativi circa da $-3.4*10^{38}$ a $-1.17*10^{-38}$ range numeri positivi circa da $1.17*10^{-38}$ a $3.4*10^{38}$ oltre allo 0. *Per inserire valori esponenziali in PhpMyAdmin si usa e, ad esempio 3.3e38*
DOUBLE [(M,D)] [UNSIGNED] [ZEROFILL] *Sinonimi*: DOUBLE PRECISION REAL	numero reale a virgola mobile a precisione singola (8 byte)	range numeri negativi circa da $-1.79*10^{308}$ a $-2.22*10^{-308}$ range numeri positivi circa da $2.22*10^{-308}$ a $1.79*10^{308}$ oltre allo 0.
DECIMAL [(M,D)] [UNSIGNED] [ZEROFILL] *Sinonimi*: DEC NUMERIC FIXED	numero reale con *esattamente* M cifre di cui D decimali. (diversamente dai FLOAT e DOUBLE per i quali M e D sono una riduzione dei valori effettivi che si possono memorizzare). Di default M=10 e D=0. Valori massimi M=65 e D=30	Range limite uguali a quelli di DOUBLE

In generale per tali tipi di dati la precisione è affidabile fino alla **settima** cifra decimale per i FLOAT e alla **quindicesima** cifra decimale per i DOUBLE. L'uso di UNSIGNED disabilita la possibilità di inserire valori negativi ma non aumenta il range dei valori positivi.

Caso particolare nell'utilizzo di ZEROFILL. Ad esempio per memorizzare un numero di Partita IVA che è formato da 11 cifre si potrebbe utilizzare un BIGINT con 11 cifre ma è necessario considerare anche gli zero nella parte più significativa del dato. Per considerare tali zeri è possibile utilizzare ZEROFILL:

P IVA errata: 23223443

P IVA corretta: 00023223443

Date e orari

DATE	Date nel formato AAAA-MM-GG. E' possibile inserire date, oltre che con il formato indicato, anche sottoforma di stringhe o numeri	Range da 1000-1-1 (1 gennaio 1000) a 9999-12-31 (31 dicembre 9999) **In realtà anche valori oltre questi limiti possono funzionare ma MariaDB non ne garantisce il corretto funzionamento poiché non sono stati testati.**
DATETIME	Date e ore nel formato AAAA-MM-GG HH:MM:SS Anche in questo caso si possono usare formati diversi per inserire i dati	Range uguale a DATE
TIME	Orari in formato HH:MM:SS Anche in questo caso si possono usare formati diversi per inserire i dati	Range da -838:59:59 a 838:59:59 (per rappresentare intervalli di tempo anche diversi dalle ore del giorno)
TIMESTAMP [(M)]	Numero di secondi trascorsi dal 1 gennaio 1970 (Timestamp UNIX)	
YEAR [(2 | 4)]	Rappresenta un anno su 2 cifre o su 4 cifre	Range 2 cifre: da 70 (1970) a 69 (2069) Range 4 cifre: da 1901 a 2155

- Inserimento di date come stringhe: è possibile inserire i dati come stringhe utilizzando separatori diversi (slash, backslash, due punti, punto..) a patto di mantenere la sequenza anno – mese – giorno. **Gli zero non significativi si possono omettere.**

  Esempio: '1980/2/3' '1980\2\3' '1980.2.3'

- Inserimento date come numeri: è possibile inserire le date come numeri di 8 cifre che rappresentano AnnoMeseGiorno. **Gli zero non significativi non si possono omettere.** Esempio: 19800203

Formattazione delle date

Come detto il formato in cui vengono memorizzate le date e gli orari sono del tipo AAA-MM-GG HH:MM:SS. Spesso si vuole presentare tali dati in un formato diverso. Per fare questo si utilizza la seguente **funzione** messa a disposizione di SQL che si chiama **DATE_FORMAT().** Essa accetta due **argomenti**: la data da formattare (presa generalmente da una tabella) e una stringa nella quale viene specificato il nuovo formato che dovrà assumere la data

DATE_FORMAT(stringa_data da formattare, stringa_nuovo_formato)

La stinga per che specifica il nuovo formato è costituita da alcuni caratteri speciali preceduti da % che hanno il significato mostrato nella seguente tabella:

%d	giorno del mese numerico da 1 a 31
%m	mese dell'anno numerico da 1 a 12
%M	nome del mese da January a December
%H	ore da 00 a 23
%i	minuti da 00 a 59
%s	secondi da 00 a 59
%Y	anno a 4 cifre
%y	anno a 2 cifre

Se la data inserita in un campo di tipo data ha un formato non corretto il valore che viene inserito è il valore di default *zerovalue* costituito da tutti zero.

Si può testare con dei SELECT

Esempi:

SELECT DATE_FORMAT('1980-2-22', '%d %M %Y') AS data_formattata

Restituisce: 22 February 1980

SELECT DATE_FORMAT('19800222', '%d %M Anno: %Y') AS data_formattata

Restituisce: 22 February Anno:1980

Esercizio (fare): visualizzare il nome e la data di nascita di ciascun dipendente in formato Anno/mese/giorno ordinati dal più anziano al più giovane con il mese espresso come mese dell'anno.

Soluzione:

```
SELECT nominativo, DATE_FORMAT (data_nascita, "%Y %M %d") as nato
FROM personale
ORDER BY nato ASC
```

SQL mette a disposizione opportune **funzioni** per operare sulle date. Nel seguente elenco ne riportiamo alcune spesso utilizzate:

FUNZIONE	COSA FA	ESEMPIO (fare)
DATE_FORMAT('data da formattare', 'stringa_nuovo_formato')	Consente di mostrare una data e ora in un formato specificato	**SELECT nominativo, DATE_FORMAT(data_nascita, '%d/%m/%Y') AS data_formattata FROM personale ORDER BY data_nascita ASC;**
TIME_FORMAT('orario da formattare', 'stringa_nuovo_formato')	Consente di mostrare un orario in un formato specificato	**SELECT TIME_FORMAT('22:29:13','ore: %H minuti: %i secondi: %s') AS Orario;**
STR_TO_DATE()	Operazione inversa di DATE_FORMAT(), ossia restituisce un dato di tipo DATE O DATETIME O TIME. Riceve come argomenti due stringhe che sono la data e la stringa che, con gli specificatori, indica dove individuare i giorni, i mesi ecc…	**SELECT STR_TO_DATE('10,4,2018','%d,%m,%Y');**
NOW() CURRENT_TIMESTAMP() SYSDATE()	Restituiscono data e ora corrente	**SELECT NOW()** restituisce data e ora attuale nel formato predefinito AAAA-MM-GG **SELECT NOW()+0** restituisce data e ora in formato numerico 20190311

CURRENT_DATE()	Restituisce data corrente	**SELECT CURRENT_DATE();**
CURRENT_TIME()	Restituiscono ora corrente	**SELECT CURRENT_TIME();**
UNIX_TIMESTAMP()	Restituisce lo UNIX timestamp ossia il numero di secondi trascorsi dal 1 gennaio 1970	**SELECT UNIX_TIMESTAMP();**
YEAR() MONTH() DAY() WEEK()	Estraggono da un tipo di dato DATE l' anno, il mese, il giorno, il numero della settimana	**SELECT YEAR(data_nascita) AS anno, MONTH(data_nascita) AS mese, DAY(data_nascita) AS giorno, WEEK(data_nascita) AS settimana FROM personale;**
HOUR() MINUTE() SECOND()	Estrae le informazioni da un tipo di dato TIME	
ADDDATE(data, N)	Aggiunge N GIORNI a una data	**SELECT ADDDATE(NOW(),1);**
SUBDATE(data, N)	sottrae N GIORNI a una data	**SELECT SUBDATE(NOW(),365);**
DATEDIFF(data1,data2)	restituisce la differenza **IN GIORNI** fra due date	**SELECT DATEDIFF(NOW(),'2018-8-1');**
INTERVAL	per sommare/sotttrarre altri intervalli di tempo	**SELECT NOW()+INTERVAL 30 MINUTE SELECT NOW()+INTERVAL 30 HOUR SELECT NOW()+INTERVAL 30 SECOND**
TIMESTAMPDIFF (unità di data, data1, data2)	Restituisce la differenza fra le due date espressa in unità di data. unità di data può essere YEAR, MONTH, DAY, WEEK, HOUR, MINUTE, SECOND…….MICROSECOND	**SELECT nominativo,TIMESTAMPDIFF(YEAR,data_nascita,NOW()) AS eta FROM personale**

Stringhe di caratteri

Nella seguente tabella vengono riportati i tipi di dato che consentono di memorizzare stringhe e caratteri o stringhe di byte (stringhe che rappresentano byte espresse in esadecimale quindi, ad esempio per memorizzare il byte 11110011→f3);

CHAR(M)	Stringa di caratteri di lunghezza fissa M. Se M=10, anche se si memorizza la parola 'ciao' essa occuperà 10 caratteri, i caratteri mancanti verranno riempiti con degli zero e verranno eliminati in fase di lettura (il numero di byte dipende dal charset utilizzato: UTF8, UTF16 ecc..). Il dato è allocato staticamente nella memoria quindi l'accesso è più veloce. Si usa per memorizzare stringhe di lunghezza fissa, ad esempio per il sesso: M o F.	MAX 11255 caratteri
VARCHAR(M)	Stringa di caratteri di lunghezza variabile (massimo M caratteri). Il dato è allocato in memoria dinamicamente quindi l'accesso è più lento. Si usa per stringhe la cui lunghezza non è fissa, ad esempio il nome.	MAX 65535 caratteri (M=65535)
BINARY(M)	Corrisponde a CHAR ma consente di memorizzare in formato **esadecimale** delle stringhe che rappresentano dati binari.	MAX 255 BYTE esempio di valore memorizzato: f3 = '11110011'
VARBINARY(M)	Corrisponde a VARCHAR ma consente di memorizzare in formato esadecimale delle stringhe che rappresentano dati binari.	
TEXT [(M)] TINYTEXT MEDIUMTEXT LONGTEXT	Per memorizzare STRINGHE DI TESTO lunghe (quando VARCHAR non è sufficientemente grande)	Numero di caratteri massimo: TINYTEXT: 255 TEXT: 65535 MEDIUMTEXT :. 16 milioni LONGTEXT: 4 miliardi
BLOB TINYBLOB MEDIUMBLOB LONGBLOB	BLOB è l'acronimo di *Bynary Long OBjects*. Per memorizzare DATI BINARI lunghi (quando VARBINARY non è sufficientemente grande). Viene utilizzato per memorizzare file di qualunque tipo, ad esempio per le immagini.	Numero di byte massimo: TINYBLOB: 255 BLOB: 65535

	(Vantaggi: le immagini possono essere esportate direttamente dal database facendo un “esporta”. Svantaggi: il DB può diventare molto pesante)	MEDIUMBLOB:. 16 MByte LONGBLOB: 4 GByte phpMyadmin consente file solo fino a 2.048 Mb Per aggiungere un file con una query si usa l’istruzione **LOAD_FILE(pathname).** Ad esempio: **update prodotti set immagine=load_file("C:\Users\User\Desktop\img.jpg") WHERE id_prod="p001"**
ENUM(‘valore1’,’valore2’,…)	Può contenere uno dei valori fra quelli dell’elenco, oppure NULL oppure una stringa vuota, che viene assegnata nel caso di tentativo di assegnamento di un valore non valido	MAX 65535 valori

6 COMANDI DML: INSERT, DELETE, UPDATE

DML è l'acronimo di Data Manipulation Language, infatti questi comandi consentono di manipolare i dati nelle tabelle, ossia inserire, eliminare e modificare dati.

INSERT

Comando per l'aggiunta di righe alle tabelle. Ci sono due possibili sintassi:

Sintassi 1:

INSERT INTO <nome_tabella> [(lista_campi)] VALUES (<lista_valori>);

Sintassi 2:

INSERT INTO <nome_tabella> SET <campo1>=<valore1>, …, <campoN>=<valoreN>;

Nella sintassi 1 l'ordine dei valori deve rispettare l'ordine indicato per i nomi dei campi, nella sintassi 2 l'ordine è a piacere.

Cosa accade se alcuni campi non vengono valorizzati?

- Se il campo è PK, il comando viene rifiutato
- Se il campo può assumere valore NULL, assume il valore NULL
- Se il campo è NOT NULL assume il valore vuoto

Esempio: inserimento di un novo dipendente:

Si può omettere questo elenco se vengono inseriti dati per ogni campo, non c'è ambiguità.

sintassi 1:

```
INSERT INTO personale (matricola, nominativo, data_nascita, qualifica, stipendio, id_dip)
VALUES (00111, 'ANTONIO LICCIA','1970-2-3', 'Q3', '2200', 'D3');
```

sintassi 2:

```
INSERT INTO Personale SET
matricola=00222,id_dip='D2',nominativo='PINNA ALESSIO',qualifica='Q2',data_nascita='1988-3-1',stipendio=2100;
```

E' possibile inserire dati estrapolandoli da una query. In questo modo è possibile aggiungere più elementi alla tabella con una sola istruzione.

Esempio: creare una tabella *nuovi_componenti*. Inserire nella tabella *componenti* gli elementi di *nuovi_componenti*. Come esercizio si utilizzino i comandi DDL per la creazione della tabella nuovi_componenti (campi uguali a quelli della tabella componenti) e DML per l'inserimento di due righe.

```
CREATE TABLE nuovi_componenti
(id_comp VARCHAR(5) NOT NULL PRIMARY KEY,
 nome_componente VARCHAR(30) NOT NULL,
 costo_unitario DOUBLE,
 fornitore VARCHAR(30));

INSERT INTO nuovi_componenti (id_comp, nome_componente, costo_unitario, fornitore)
VALUES ('C021', 'Componente 21', 0.3,'Fornitore4');

INSERT INTO nuovi_componenti (id_comp, nome_componente, costo_unitario, fornitore)
VALUES ('C022', 'Componente 22', 0.5,'Fornitore4');
```

Inserisco in nuovi componenti nella tabella componenti attraverso una query:

```
INSERT INTO componenti (id_comp, nome_componente, fornitore)
SELECT id_comp, nome_componente,fornitore
FROM nuovi_componenti;
```

Si noti che l'inserimento con query **non utilizza la parola chiave VALUES** e, in questo caso, aggiunge solo 3 campi (non valorizza il prezzo nella tabella componenti).

Se i campi risultanti dalla query fossero stati TUTTI i campi della tabella di destinazione, non sarebbe stato necessario elencarli. Sarebbe bastato

```
INSERT INTO componenti
SELECT *
FROM nuovi_componenti;
```

DELETE

Sintassi per eliminare elementi:

DELETE FROM <nome_tabella> [WHERE <condizione>]

Esempio: eliminare dalla tabella componenti gli elementi C021 e C022 aggiunti nell'esempio precedente:

```
DELETE FROM componenti
WHERE id_comp='C021' OR id_comp='C022';
```

In assenza della clausola WHERE vengono eliminate tutte le righe della tabella.

La clausola WHERE determina quali righe che verranno eliminiate

UPDATE

Sintassi per aggiornare gli elementi di una tabella:

UPDATE <nome_tabella> SET <nome_campo>=<valore>,….

Esempio: aggiornare lo stipendio del dipendente 00013 del 10%:

```
UPDATE personale SET stipendio=personale.stipendio*1.1
WHERE matricola='00013';
```

Con UPDATE è possibile aggiornare più campi (nome, prezzo) di tutte le righe che soddisfano la clausola WHERE.

si possono usare AND OR NOT, IN (NOT e IN vogliono la condizione fra parentesi) nelle condizioni, >,<,=,>=,<=,!=,<> nei criteri di selezione

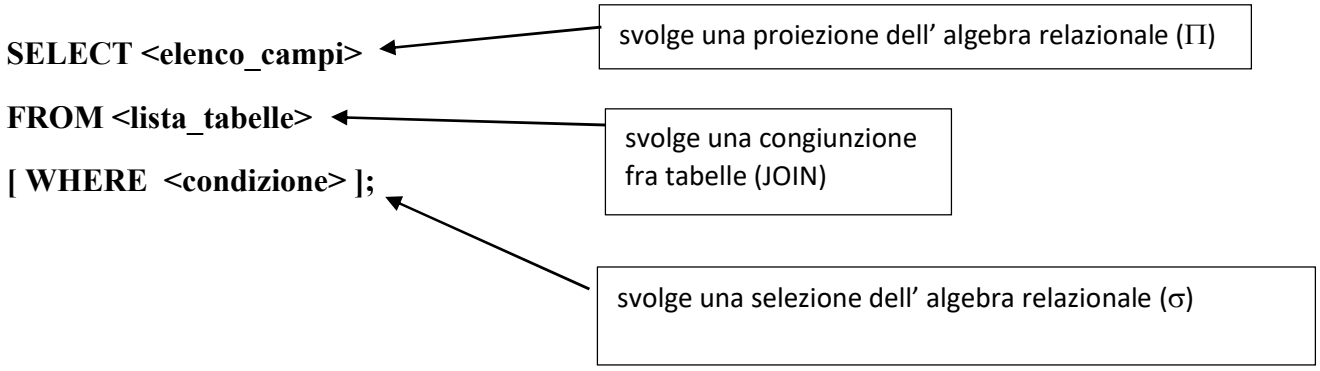
Esempio: aggiorno il nome e lo stipendio del dipendente Neri Giovanni

```
UPDATE personale
SET nominativo="Neri Giovanni Antonio", stipendio=2800
WHERE nominativo="Neri Giovanni"
```

La clausola WHERE determina quali righe verranno aggiornate

1. IL COMANDO SELECT E L'ALGEBRA RELAZIONALE

L'operazione fondamentale è quella basata sul seguente costrutto ed è chiamata **mapping**:



L'istruzione restituisce le colonne "elenco_campi" delle tabelle "elenco_tabelle" che soddisfano la "condizione"

Esempi

1. Per visualizzare il nome e il prezzo di tutti prodotti realizzati dal dipartimento con codice D2:

   In sql si traduce con:

   SELECT nome_prodotto, prezzo **(è l'elenco della proiezione)**

   FROM Prodotti

   WHERE id_dip='D2'; **(è il predicato della selezione)**

2. Per visualizzare il nome e il prezzo di tutti prodotti realizzati dal dipartimento con codice D2 il cui prezzo è superiore a 10€ si utilizza il connettivo logico AND nella condizione

   SELECT nome_prodotto, prezzo

   FROM Prodotti

   WHERE id_dip='D2' AND prezzo>10;

Nella clausola WHERE si possono utilizzare i connettivi logici **AND OR NOT** e gli operatori di confronto soliti:

>,<,>=,<=,=,!= (oppure <>)

OSSERVAZIONI ED ESEMPI

1. Per indicare tutti i campi di una tabella dopo la parola chiave SELECT si usa **l'asterico ***

   SELECT *

   FROM Prodotti

   WHERE id_dip='D2' AND prezzo>10;

2. Per indicare un insieme di valori che devono essere soddisfatti dalla clausola WHERE si usa la parola chiave **IN** seguita dall'elenco dei valori

   SELECT nominativo, qualifica

   FROM Personale

   WHERE id_dip IN ('D2', 'D4','D5');

   Mostra i nomi dei dipendenti che lavorano in uno dei dipartimenti indicati, la condizione equivale a:

   WHERE id_dip='D2' OR id_dip='D4' OR id_dip='D5' ;

   Utilizzando l'operatore booleano NOT è possibile ottenere l'elenco dei dipendenti che **NON** appartengono all'elenco specificato

   WHERE id_dip NOT IN ('D2', 'D4','D5');

ESERCIZIO fare loro. Scrivere le query per selezionare i seguenti elementi

- Matricola e nominativo degli dipendenti con stipendio >2500
- tutti i campi dei dipartimenti che hanno sede in provincia di Milano o di Brescia.

3. Utilizzo della costante NULL nelle interrogazioni

   La costante NULL può essere utilizzata nella clausola WHERE per selezionare tuple in cui alcuni valori non sono stati **avvalorati**. Per utilizzare NULL la sintassi è la seguente (**IS NULL** nella condizione).

   Attenzione: NULL è diverso dalla stringa vuota!

   SELECT *

   FROM Personale

   WHERE qualifica IS NULL;

   Elenco di tutti i campi dei Dipendenti che non hanno alcuna qualifica

   IS NULL (staccato)

Lo stesso risultato si può ottenere con la seguente sintassi

```
SELECT *
FROM  Personale
WHERE  ISNULL (qualifica);
```

Elenco Dipendenti che non hanno alcuna qualifica

ISNULL (attaccato)

4. Ordinare i risultati dell'interrogazione. Le tuple che costituiscono il risultato dell'interrogazione possono essere ordinate in base a uno o più determinati campi (parola chiave **ORDER BY** messa **di seguito** alla clausola WHERE) e secondo un ordine crescente (**ASC**) o decrescente (**DESC**)

```
SELECT matricola, nominativo, stipendio
FROM  Personale
WHERE  id_dip='D4'
ORDER BY stipendio DESC, nominativo ASC ;
```

Elenco dipendenti del dipartimento D4 in ordine di stipendio decrescente e, **a parità di stipendio** in ordine alfabetico

5. Non ripetere le righe duplicate. Alcune interrogazioni, come visto, possono generare tuple duplicate, per evitare questo è necessario indicare specificatamente al DBMS tale volontà con la parola chiave **DISTINCT** subito dopo la parola chiave SELECT.

```
SELECT DISTINCT prezzo
FROM  prodotti ;
```

Elenco dei prezzi dei prodotti. Senza DISTINCT, ogni prezzo sarebbe stato ripetuto tante volte quanti sono i prodotti con quel prezzo

ESERCIZIO fare loro. Scrivere le query per selezionare i seguenti elementi

- Matricola, nominativo, data di nascita dei dipendenti ordinati dal più anziano al più giovane

- Dalla tabella personale, l'elenco degli id_dip senza ripetizioni

6. Ricerche *pattern – matching* nei campi di tipo stringa. Con l'operatore **LIKE** è possibile effettuare delle ricerche nella clausola WHERE indicando una sottostringa ed individuando, grazie al carattere jolly **%**, che "significa" "qualsiasi stringa", tutte le stringhe che contengono la sottostringa indicata.

```
SELECT *
FROM  Personale
WHERE  nominativo LIKE 'Ros%' ;
```

Elenco dipendenti il cui nominativo inizia con Ros. Ad esempio: Rossi, Rossini, Rosario ecc..

```
SELECT *
FROM  Personale
WHERE  nominativo LIKE '%ert%' ;
```

Elenco dipendenti il cui nominativo contiene ert. Ad esempio: Roberto, Bertolini, ecc..

Un altro carattere jolly è underscore (_) . Esso sostituisce qualsiasi carattere, ma UN SOLO carattere

Esempio: elenco dei dipendenti che si chiamano Paolo o Paola

```
SELECT *
FROM  Personale
WHERE  nominativo LIKE ' %Paol_' ;
```

Esempio: (fare loro) elenco dei dipendenti il cui cognome (prima parte del nominativo) è esattamente di 5 lettere

ESERCIZIO fare loro. Scrivere le query per selezionare i seguenti elementi

- Tutti i campi del personale delle personale che hanno due nomi o piu' (quindi hanno due spazi)

ESERCIZIO fare loro. Scrivere le query per selezionare i seguenti elementi

- Tutti i prodotti con nome che inizia con Prodotto, ordinati in ordine alfabetico

7. Ricerche in un intervallo di valori. Con l'operatore **BETWEEN** nella clausola WHERE è possibile indicare nella condizione un intervallo di valori all'interno dei quali effettuare la ricerca. La sintassi è la seguente

   WHERE <campo> BETWEEN <valore iniziale> AND <valore finale>

   che equivale a

   WHERE <campo> >= <valore iniziale> AND <campo> <= <valore finale>

```
SELECT *
FROM  Personale
WHERE  data_nascita BETWEEN '1960-1-1'AND '1969-12-31' ;
```

Elenco dipendenti la cui data di nascita è compreso fra il 1960 e 1969

Si noti che la costante utilizzata per indicare la data è una **stringa** nel formato **AAA-MM-GG**.

Un'altra possibilità è quella di utilizzare la **funzione YEAR** che estrae da un campo data l'anno di nascita. In tal caso le costanti utilizzate per indicare gli anni sono dei numeri interi.

```
SELECT *
FROM  Personale
WHERE YEAR(data_nascita) BETWEEN 1960 AND 1969 ;
```

Elenco dipendenti il cui anno di nascita è compreso fra il 1960 e 1969

Altre funzioni sulle date, le funzioni devono avere le parentesi alla fine:

MONTH (data) restituisce un intero corrispondente al mese (da 1 a 12)

DAY (data) restituisce un intero corrispondente al giorno (da 1 a 31)

TIMESTAMPDIFF (YEAR, data1, data1) fornisce la differenza in anni (oppure in mesi o giorni) fra data2 e data1.

CURRENT_DATE(), CURRENT_TIME(), CURRENT_TIMESTAMP(), NOW(): forniscono data e ora attuali.

8. Creazione e nominazione di **campi temporanei** (chiamati **alias**) nei risultati delle query. In seguito ad operazioni di elaborazione (**calcoli**) svolte sui valori dei campi può essere utile creare dei nuovi campi temporanei da visualizzare nei risultati delle query. Tali campi sono definiti **temporanei** perché non sono memorizzati nel DB e la loro vita si esaurisce una volta eseguita la query. Per assegnare dai nomi a tali campi si usa la parola chiave **AS** nel SELECT.

SELECT nominativo,data_nascita, TIMESTAMPDIFF (YEAR, data_nascita, CURRENT_DATE) AS eta

FROM personale

order BY eta DESC

Elenco dipendenti del dipartimento 'D2' ordinati per eta decrescente mostrando l'alias eta

Se il nome del campo *alias* contiene uno spazio è necessario che venga messo fra apici.

Ad esempio: *'eta dipendente'*;

La parola chiave **AS** può essere utilizzata anche per rinominare un campo di una tabella temporaneamente nella formulazione di una query:

SELECT matricola, nominativo AS cognome_nome, stipendio

FROM Personale ;

Elenco che riporta matricola, cognome_nome e stipendio di tutti i dipendenti

Quindi il DBMS mette a disposizione anche delle funzioni (ad esempio quelle per operare sulle date, ma non solo) che vedremo un po' alla volta.

Esempio:

Per visualizzare il numero di matricola, nominativo e stipendio e una eventuale gratifica del 10% dello stipendio da corrispondere ai dipendenti del dipartimento 'D2' con qualifica 'Q4' la query è la seguente:

SELECT matricola, nominativo, stipendio, 0.1*stipendio AS gratifica

FROM Personale

WHERE id_dip='D2' AND qualifica='Q4' ;

OSSERVAZIONE IMPORTANTE: **Non è consentito fare riferimento a un alias di colonna in una clausola WHERE, poiché prima viene svolta la clausola WHERE poi la clausola SELECT, quindi l'alias non esiste ancora quando si svolge la selezione delle tuple con WHERE.**

Quindi questo **NON FUNZIONA**:

SELECT matricola, nominativo, stipendio, 0.1*stipendio AS gratifica

FROM Personale

WHERE gratifica>100 ;

ESERCIZI:

Scrivere e testare le query per visualizzare i seguenti dati:

- Elenco dei componenti forniti dal fornitore "Fornitore3"
- Elenco dei prodotti ordinati dal più costoso al meno costoso
- Elenco degli id_prodotto (solo l'id_prod) che contengono almeno 2 unità del componente C001 ordinati da quello che ne contiene di più a quello che ne contiene di meno

```
SELECT id_prod FROM
composizione
WHERE id_comp="C001" and unita_comp>1
ORDER BY unita_comp DESC
```

- Si applichi uno sconto del 10% a tutti i prodotti. Si visualizzino il nome del prodotto, il prezzo, lo sconto, il prezzo scontato solo dei prodotti dei dipartimenti "D2" e "D3" il cui sconto è compreso fra 0,3 e 0,5 €, ordinati dal meno costoso al più costoso. Ricordarti che non puoi usare prezzo_scontato nella clausola WHERE.

```
SELECT nome_prodotto,prezzo,(prezzo-prezzo*0.1) AS sconto
FROM prodotti
WHERE id_dip IN ('D1', 'D2') and (prezzo*0.1) BETWEEN 0.3 AND 0.5
ORDER BY prezzo ASC
```

- Ipotizzando che si vada in pensione al compimento dei 68 anni: visualizzare matricola, nominativo, data di nascita, età, anni mancanti alla pensione di ogni dipendente

```
SELECT matricola, nominativo, data_nascita,timestampdiff(year,data_nascita, now())
as eta, 68-timestampdiff(year,data_nascita, now()) as anni_mancanti_pensione
FROM personale
```

- Visualizzare che data sarà fra 1000 giorni. Cercare nel manuale o su internet la funzione da utilizzare:

```
SELECT (ADDDATE(CURRENT_DATE(),1000));
```

ESERCIZIO RIASSUNTIVO fare loro: Svolgere le seguenti query sul database cinematografico della figura a p. A138:

- Titolo e anno di realizzazione dei film compresi negli anni 90
- Elenco degli attori (nominativo, età) ordinati dal più anziano al più giovane
- Elenco degli attori che iniziano con la lettera A
- Elenco dei titoli dei film americani degli anni ‘90
- Elenco dei film (titolo, anno) che non hanno alcuna valutazione
- Elenco dei film (titolo, anno) il cui titolo contiene l’articolo ‘il’