ITIS-LS "Francesco Giordani" Caserta
prof. Ennio Ranucci
a.s. 2019-2020

Manuale SQL

Esercitazioni svolte in ambiente MySQL
(EasyPHP Devserver 17.0)

Accesso iniziale a mySql:

Prompt di DOS: "cmd";

- cd \; //change directory

Percorso

C:\Program Files (x86)\EasyPHP-Devserver-17\eds-binaries\dbserver\mysql5717x86\bin

Per accedere: "mysql -h -u -p "; //con username e password

Per accedere in localhost senza password: "mysql -u root".

Tipi di variabili:

Tipo	Descrizione	Esempi e range di variabilità
BIT	Singolo bit: Definisce il tipo booleano.	**0** (corrisponde a false o No) **1** (corrisponde a true o Si)
BIT(N)	Stringa di bit di lunghezza fissa pari a N.	**BIT(5)** Esempio di costante: "01101"
BIT VARYNG(N)	Stringa di bit di lunghezza variabile avente dimensione massima pari a N.	**BITVARYNG(5)** Esempio di costante:" 0011011"
CHARACTER o CHAR	Singolo carattere.	"C", "5", "%", "c"
CHARACTER(N) o CHAR(N)	Stringa di caratteri di lunghezza fissa pari a N caratteri. CHAR(1) corrisponde a CHARACTER o CHAR.	N varia da 1 a 15000. Esempio di costante: "rty56"
CHARACTER VARYNG(N) o VARCHAR(N)	Stringa di caratteri di lunghezza variabile con dimensione massima pari a N.	**VARCHAR(5)** Esempio di costante: "rty56"
DATE	Data nel formato "AAAA/MM/GG" oppure "AAAA-MM-GG".	"2006/12/25" "2006-12-25"

DECIMAL(P,D) o DEC(P,D)	Numero reale in	**DECIMAL(6,2)**
	fixedpoint con un numero massimo di cifre prima del punto decimale pari a P e un numero massimo di cifre dopo il punto decimale pari a D. Le dimensioni massime di P e D sono definite dall'implementazione	Esempio di costanti sono: 100.3, +0.7, -.3
DOUBLE PRECISIONE	Numero reale in floatingpoint con precisione per la mantissa doppia rispetto alla precisione di un FLOAT.	L'esponente generalmente varia da -127 a +128
FLOAT	Numero reale in floatingpoint con precisione definita dall'implementazione. Generalmente vale 15 per la mantissa.	Da 1E-28 a 1E+38. Stessi esempi del **REAL** per le costanti
FLOAT(P)	Numero reale in floatingpoint con precisione P per la mantissa.	La dimensione di P è definita dall'implemetazione. Generalmente P varia da 1 a 45
INTEGER o INT	Numero intero con precisione superiore a **SMALLINT**(generalmente occupano 4 byte, ma dipende dall'implementazione).	Generalmente da -2147483648 a +2147483647
INTEGER(N) o INT(N)	Numero intero con precisione N (numero massimo di cifre che il numero può contenere). Anche senon standard, è supportato dalla maggior parte delle implementazioni SQL.	**INT(5)**
SMALLINT	Numero intero con precisione inferiore a **INTEGER** (generalmente occupano 2 byte a memoria).	Generalmente da -32768 a +32767

REAL	Numero reale in floatingpoint con precisione definita dall'implementazione. Generalmente vale 7 per la mantissa.	Da 1E-38 a 1E+38. Esempi di costanti sono: 10E4 +24.7E-3 – 7.897E+12
TIME	Ora nel formato ora, minuti, secondi e millisecondi.	"10:34:25:42"
TIMESTAMP	Data e orario nel formato anno, mese, giorno, ora, minuti, secondi e millisecondi.	"2006/12/25 10:34:25:42"

DECIMAL (3,1) SIGNIFICA CHE IL TOTALE DELLE CIFRE E' 3 DI CUI 2 PARTE INTERA E 1 PARTE REALE

Comandi SQL DDL - Data Definition Language

(Agisce sulla struttura del Database) :

- **SHOW DATABASES;**

- **CREATE DATABASE <NomeDatabase> ;**

- **USE <NomeDatabase> ;**

- **SHOW TABLES;**

- **CREATE TABLE (...)**
Esempio: CREATE TABLE studenti (matStudente varchar (5) NOT NULL, cogStudente varchar (50) NOT NULL, nomStudente varchar (50) NOT NULL, cittaStudente varchar (50), PRIMARY KEY (matStudente));

- **DESCRIBE <nome tabella>;** //visualizza struttura tabella

~~- **ALTER TABLE ADD ( ) [BEFORE** <nomeAttributo>**];** //per aggiungere un campo~~

Esempio: ALTER TABLE studenti ADD (indirizzo varchar (50));

- **ALTER TABLE** studenti **ADD COLUMN** indirizzo char(50) **AFTER** CITTA**;** //per aggiungere un campo

- ALTER TABLE CHANGE; //modifica del nome di un campo

Esempio: ALTER TABLE studenti CHANGE indirizzo indStudenti varchar (50);

- **ALTER TABLE DROP;** //per eliminare un campo

Esempio: ALTER TABLE studenti DROP indStudenti;

- DROP TABLE <nome tabella> ;//cancella una tabella
- DROP DATABASE <nome database>; //cancella un database

- CREATE UNIQUE INDEX ON (); //crea un indice su attributi chiave, se non specificato, su attributi non chiave

CREATE UNIQUE INDEX <nomeIndice> ON <nomeTabella>(<nomeAttibuto1>,<nomeAttributo2>,…<nomeAttributoN>); //crea indice

-DROP INDEX <nomeindice> **ON** <nomeTabella>; //elimina indice

- **ALTER TABLE** studenti **ENGINE**=InnoDB; //supporta le foreign key

- ALTER TABLE studenti ADD (siglaClasse varchar (5), FOREIGN KEY (siglaClasse) REFERENCES classi (siglaClasse)); // aggiunge la chiave esterna "siglaClassi" alla tabella "studenti";

- **SET STORAGE_ENGINE**=InnoDB; //Impostare engine per tutta la durata della sessione per le tabelle create successivamente a questo comando;
Per impostare il motore anche per le successive sessioni, modificare la riga default-storage-engine=InnoDB del file my.ini che si trova nella cartella MySql.

Comandi SQL DML - Data Manipulation Language

(inserire, modificare e gestire dati memorizzati):

- INSERT INTO VALUES(…); //inserimento dei valori in una tabella

Esempio: INSERT INTO studenti (matStudente, cogStudente, nomStudente, cittaStudente) VALUES ("00001", "Ardivelo", "Nazareno", "Frignano");

- **UPDATE SET;** //aggiorna i dati di una tabella

Esempio: UPDATE<Tabella>

SET <nome campo> = <nuovo valore>

[WHERE<condizione>]

Esempio: UPDATE studenti SET nomstud=’caio’ WHERE matstud=’mat1’;

Esempio: UPDATE impiegati SET stipendio=stipendio+(stipendio*10/100) where stipendio >1200;

DEFAULT

-Create table voti (voto(2) not null, materia char(20) default “inf”);

insert into voti(voto) values(10);

- **DELETE FROM WHERE;** //viene utilizzata per eliminare le righe di una tabella

Esempio:

DELETE FROM <nome tabella> WHERE <condizione>;
DELETE FROM studenti WHERE voto=8;

Comandi SQL DCL - Data Control Language

(impostare politiche relative al controllo ed alla sicurezza dei dati):

Quando viene installato **MySQL**, è governato da un unico utente con privilegi illimitati: l'utente *root*, infatti, è l'amministratore del nostro server MySQL e come tale può fare qualsiasi cosa al suo interno. Normalmente, tuttavia, quando si amministra un server MySQL è buona norma NON utilizzare sempre l'utente *root*: se si crea un'applicazione che utilizza uno specifico database, ad esempio, è corretto creare ed utilizzare per questa un utente specifico con privilegi limitati al singolo database utilizzato e circoscritti alle specifiche esigenze di produzione.
Il comando **GRANT** permette allo stesso tempo di creare un utente e di assegnargli dei permessi specifici.
Se l'utente non esiste, GRANT consente di crearlo. In caso contrario il comando si limiterà ad assegnargli nuovi permessi e/o privilegi.

- **istruzioni_consentite**: E' una lista di istruzioni di SQL che si vogliono permettere all'utente (CREATE, SELECT, UPDATE, DELETE, ALTER, DROP, ecc..). Se si vuole dare all'utente permessi completi si può utilizzare la parola chiave "ALL".
- **database**: E' il nome del database sul quale l'utente potrà eseguire le istruzioni consentite. Se si vuole fare riferimento a tutti i database del sistema si può utilizzare il carattere asterisco (*).
- **tabella**: Specificando il nome di una tabella, si fa riferimento solo ad essa: i permessi dell'utente, quindi, riguarderanno solo questa tabella e non le altre presenti nel database. Se si vuole fare riferimento a tutte le tabella si può utilizzare il carattere asterisco (*).
- **utente**: Specifica il nome dell'utente che vogliamo creare o al quale vogliamo assegnare nuovi permessi.
- **host**: Specifica il/gli host da cui è ammessa la connessione.
- **password**: Specifica la password associata all'utente che stiamo creando. La password va scritta "in chiaro". Se si desidera inserire la password in forma criptata tramite la funzione PASSWORD() di MySQL, si deve far precedere la stringa criptata dalla parola PASSWORD.

- GRANT: Per concedere a gruppi di persone i diritti di accesso necessari ad interagire su un determinato insieme di dati.

Sintassi:

- GRANT <ElencoPrivilegi> ON [<NomeDatabase>.]<NomeTabella>

TO (<ElencoUtentiAutorizzati>) [PUBLIC]

[IDENTIFIED BY]

[WITH GRANT OPTION];

ESEMPI

-GRANT SELECT ON cinema.* TO 'visitatore'@'%' IDENTIFIED BY 'password_visitatore';

-GRANT SELECT, INSERT, UPDATE, DELETE ON cinema.* TO 'editore'@'localhost' IDENTIFIED BY 'password_editore';

-GRANT ALL ON cinema.* TO 'admin'@'123.123.123.123' IDENTIFIED BY 'password_admin';

-SHOW GRANTS FOR 'editore'@'localhost';

-SHOW GRANTS FOR CURRENT_USER;

-CREATE USER alberto@localhost;

-CREATE USER fabio@localhost IDENTIFIED BY ‘password’;

-DROP USER alberto@localhost;

-SET PASSWORD = PASSWORD(‘pw’);

-SET PASSWORD FOR paolo@localhost = PASSWORD(‘pw’);

REVOKE: Per revocare determinati permessi di accesso.

Sintassi:

- REVOKE <ElencoPrivilegi> ON <NomeTabella>

FROM <ElencoUtentiAutorizzati> [PUBLIC];

Dove i componenti di <ElencoPrivilegi> possono essere:

ALL che consente l'accesso globale alla tabella specificata;

SELECT che consente l'accesso in lettura a tutti i campi della tabella specificata;

INSERT - UPDATE [(<Attributo>)] che consente l'inserimento di dati eventualmente solo sulla colonna specificata da <Attributo>;

DELETE che consente la cancellazione di righe della tabella specificata;

REFERENCES [(<Attributo>)] che consente il riferimento a tutti i campi della tabella o eventualmente solo a quello specificato da <Attributo> nei vincoli di integrità;

ALL PRIVILEGES rappresenta simultaneamente tutti i privilegi possibili riferiti a un oggetto;

Ulteriori informazioni circa le autorizzazioni MySql
Nei DBMS SQL ogni operazione deve essere autorizzata, ovvero l'utente che esegue l'operazione deve avere i privilegi necessari. Abbiamo visto che i privilegi vengono concessi e revocati per mezzo delle istruzioni **GRANT** e **REVOKE**
Un utente che ha ricevuto un certo privilegio può a sua volta accordarlo ad altri utenti solo se è stato esplicitamente autorizzato a farlo: **WITH GRANT OPTION**

GRANT ALL ON DATABASE TO Pippo WITH GRANT OPTION;

in cui la clausola **WITH GRANT OPTION** autorizza l'utente Pippo a passare l'autorità ad altri utenti.

ALTER: attribuisce il diritto di modificare la definizione di una tabella
DELETE:attribuisce il diritto di cancellare righe di una tabella
INDEX: attribuisce il diritto di creare un indice sulla tabella
INSERT:attribuisce il diritto di inserire righe nella tabella
REFERENCES: attribuisce il diritto di definire foreign keys in altre tabelle che referenziano la tabella
SELECT: attribuisce il diritto di eseguire query sulla tabella/vista e di definire VIEW
UPDATE:attribuisce il diritto di modificare righe della tabella/vista

Le viste

In SQL è possibile definire un'altra classe di tabelle, chiamate viste, che non sono fisicamente memorizzate nella base di dati (sono infatti costruite nella memoria RAM), ma che possono essere definite solo logicamente.

Sintassi:
- CREATE VIEW <NomeVista> AS <Query>;

dove:

<NomeVista> è il nome assegnato alla tabella fittizia della vista;

<Query> è una normale query formulata con il comando SELECT;

Per cancellare una vista:

- DROP VIEW <NomeVista>;

Query:

- Unione

L'unione di due tabelle è una nuova tabella che ha per schema lo stesso schema e per istanza l'unione delle tuple delle due tabelle (N.B. è necessario che le due tabelle abbiano gli attributi dello stesso tipo).

Esempio:

- (SELECT Cognome,Nome FROM registi) UNION (SELECT Cognome,Nome FROM attori);

- Intersezione

L'intersezione tra due tabelle è una nuova tabella che ha per schema lo stesso schema e per istanza l'intersezione delle tuple delle due tabelle.

Esempio:
- (SELECT Cognome,Nome FROM registi) INTERSECT (SELECT Cognome,Nome FROM attori);

MySQL non supporta INTERSECT per cui lo stesso obiettivo si può ottenere con la seguente:

- SELECT registi.cognome,registi.nome FROM registi INNER JOIN attori ON CodAttore = CodRegista;

Salvare una query in una tabella
-Create table clienti2010 as select nome,cognome,clienti.codFiscale from clienti inner join fatture on clienti.codFiscale=fatture.codFiscale where year(data)=2010

- Differenza

La differenza tra due tabelle è una nuova tabella che ha per schema lo stesso schema e per istanza tutte le tuple della prima tabella che non sono presenti nella seconda tabella.

Esempio:
- (SELECT Cognome,Nome FROM registi) MINUS (SELECT Cognome,Nome FROM attori);

MySQL non supporta MINUS per cui lo stesso obiettivo si può ottenere con la seguente:

- SELECT * FROM clienti LEFT JOIN clienti09 ON clienti.CodCliente = clienti09.CodCliente WHERE clienti09.CodCliente IS NULL;

- Proiezione:

Data una tabella applicare l'operatore di proiezione significa creare una nuova tabella che ha per schema gli attributi indicati nella proiezione e per istanza la stessa istanza della tabella di partenza (senza ripetizioni).

Sintassi:

- SELECT * FROM <nome tabella> ; //mostra i valori contenuti in una tabella

- SELECT <elenco di colonne> FROM <nome tabella> //proiezione con ripetizione ;

- SELECT DISTINCT <elenco di colonne>FROM <nome tabella> ; //proiezione senza ripetizioni

- Selezione:

Data una tabella applicare l'operatore di selezione significa creare una nuova tabella che ha per schema lo stesso schema della tabella di partenza e per tuple (righe) quelle che verificano la condizione indicata.

Sintassi:

- SELECT * FROM <nome tabella> WHERE<criterio di selezione> ;

- Proieizione e selezione:

- SELECT <elenco di campi> FROM <nome tabella> WHERE <criterio di selezione> ;

- Congiunzione (JOIN):

Data due tabelle con almeno un attributo in comune, applicare l'operatore di congiunzione significa creare una nuova tabella che ha per schema l'unione dei due schemi riportando una sola volta gli attributi in comune e per istanza la concatenazione delle tuple che hanno lo stesso valore negli attributi in comune.

Sintassi:
SELECT * FROM <tabella 1> INNER JOIN <tabella 2> ON tabella1.attributo = tabella2.attributo;

Esempio:
SELECT * FROM studenti INNER JOIN classi ON studenti.siglaClasse = classi.siglaClasse;

Left Join:

Il LEFT OUTER JOIN oltre alle tuple del join restituisce anche tutte le righe della prima tabella, anche se non ci sono corrispondenze nella seconda tabella.

Elenco dei clienti del 2010 che erano già clienti nel 2009 e dei clienti del 2009 che non sono rimasti clienti nel 2010:

- SELECT * FROM clienti09 LEFT JOIN clienti ON clienti09.CodCliente = clienti.CodCliente; Elenco dei clienti del 2009 che non sono rimasti clienti nel 2010 (corrisponde alla differenza):

- SELECT * FROM clienti09 LEFT JOIN clienti ON clienti09.CodCliente = clienti.CodCliente WHERE clienti09.CodCliente IS NULL;

Right Join:

Il RIGHT OUTER JOIN oltre alle tuple del join restituisce anche tutte le righe della seconda tabella, anche se non ci sono corrispondenze nella prima tabella.

- SELECT * FROM clienti09 RIGHT JOIN clienti ON clienti09.CodCliente = clienti.CodCliente;

- SELECT * FROM clienti09 RIGHT JOIN clienti ON clienti09.CodCliente = clienti.CodCliente WHERE clienti09.CodCliente IS NULL;

Full Join:

Elenco dei clienti del 2009 e del 2010:
- SELECT * FROM clienti09 FULL JOIN clienti;

Il comando FULL JOIN non è supportato da MySQL quindi si ricorre all'unione dei due join esterni.

- (SELECT * FROM clienti09 LEFT JOIN clienti ON clienti09.CodCliente = clienti.CodCliente)

UNION

(SELECT * FROM clienti09 RIGHT JOIN clienti ON clienti09.CodCliente = clienti.CodCliente);

Cross Join (prodotto cartesiano):

Il CROSS JOIN è l'insieme di tutte le possibili concatenazioni.

- SELECT * FROM clienti09 CROSS JOIN clienti;

PERSONALE		
NOME	**COGNOME**	**RUOLO**
Lucio	Capelli	Ragioniere
Marco	Tardi	Ragioniere
Maurizio	Maugeri	Usciere
Pasquale	Catozzo	Manager
Milena	Piconi	Responsabile Acq.

DIPENDENTI		
NOME	**COGNOME**	**RUOLO**
Lucio	Capelli	Ragioniere
Marco	Tardi	Ragioniere
Silvana	Perugia	Consulente
Maurizio	Maugeri	Usciere

INNER JOIN

```
SELECT p.nome, p.cognome, p.ruolo, d.nome, d.cognome, d.ruolo
FROM personale p INNER JOIN dipendenti d
ON p.nome=d.nome AND p.cognome=d.cognome AND p.ruolo=d.ruolo
```

P_NOME	P_COGNOME	P_RUOLO	D_NOME	D_COGNOME	D_RUOLO
Lucio	Capelli	Ragioniere	Lucio	Capelli	Ragioniere
Marco	Tardi	Ragioniere	Marco	Tardi	Ragioniere
Maurizio	Maugeri	Usciere	Maurizio	Maugeri	Usciere

LEFT OUTER JOIN

```
SELECT p.nome, p.cognome, p.ruolo, d.nome, d.cognome, d.ruolo
FROM personale p LEFT JOIN dipendenti d
ON p.nome=d.nome AND p.cognome=d.cognome AND p.ruolo=d.ruolo
```

P_NOME	P_COGNOME	P_RUOLO	D_NOME	D_COGNOME	D_RUOLO
Lucio	Capelli	Ragioniere	Lucio	Capelli	Ragioniere
Marco	Tardi	Ragioniere	Marco	Tardi	Ragioniere
Maurizio	Maugeri	Usciere	Maurizio	Maugeri	Usciere
Pasquale	Catozzo	Manager			
Milena	Piconi	Responsabile Acq.			

RIGHT OUTER JOIN

```
SELECT p.nome, p.cognome, p.ruolo, d.nome, d.cognome, d.ruolo
FROM personale p RIGHT JOIN dipendenti d
ON p.nome=d.nome AND p.cognome=d.cognome AND p.ruolo=d.ruolo
```

P_NOME	P_COGNOME	P_RUOLO	D_NOME	D_COGNOME	D_RUOLO
Lucio	Capelli	Ragioniere	Lucio	Capelli	Ragioniere
Marco	Tardi	Ragioniere	Marco	Tardi	Ragioniere
			Silvana	Perugia	Consulente
Maurizio	Maugeri	Usciere	Maurizio	Maugeri	Usciere

FULL JOIN

P_NOME	P_COGNOME	P_RUOLO	D_NOME	D_COGNOME	D_RUOLO
Lucio	Capelli	Ragioniere	Lucio	Capelli	Ragioniere
Marco	Tardi	Ragioniere	Marco	Tardi	Ragioniere
Maurizio	Maugeri	Usciere	Maurizio	Maugeri	Usciere
Pasquale	Catozzo	Manager			
Milena	Piconi	Responsabile Acq.			
			Silvana	Perugia	Consulente

CROSS JOIN

P_NOME	P_COGNOME	P_RUOLO	D_NOME	D_COGNOME	D_RUOLO
Lucio	Capelli	Ragioniere	Lucio	Capelli	Ragioniere
Lucio	Capelli	Ragioniere	Marco	Tardi	Ragioniere
Lucio	Capelli	Ragioniere	Maurizio	Maugeri	Usciere
Lucio	Capelli	Ragioniere	Silvana	Perugia	Consulente
Marco	Tardi	Ragioniere	Lucio	Capelli	Ragioniere
Marco	Tardi	Ragioniere	Marco	Tardi	Ragioniere
Marco	Tardi	Ragioniere	Maurizio	Maugeri	Usciere
Marco	Tardi	Ragioniere	Silvana	Perugia	Consulente
Maurizio	Maugeri	Usciere	Lucio	Capelli	Ragioniere
Maurizio	Maugeri	Usciere	Marco	Tardi	Ragioniere
Maurizio	Maugeri	Usciere	Maurizio	Maugeri	Usciere
Maurizio	Maugeri	Usciere	Silvana	Perugia	Consulente
Pasquale	Catozzo	Manager	Lucio	Capelli	Ragioniere
Pasquale	Catozzo	Manager	Marco	Tardi	Ragioniere
Pasquale	Catozzo	Manager	Maurizio	Maugeri	Usciere
Pasquale	Catozzo	Manager	Silvana	Perugia	Consulente
Milena	Piconi	Responsabile Acq.	Lucio	Capelli	Ragioniere
Milena	Piconi	Responsabile Acq.	Marco	Tardi	Ragioniere
Milena	Piconi	Responsabile Acq.	Maurizio	Maugeri	Usciere
Milena	Piconi	Responsabile Acq.	Silvana	Perugia	Consulente

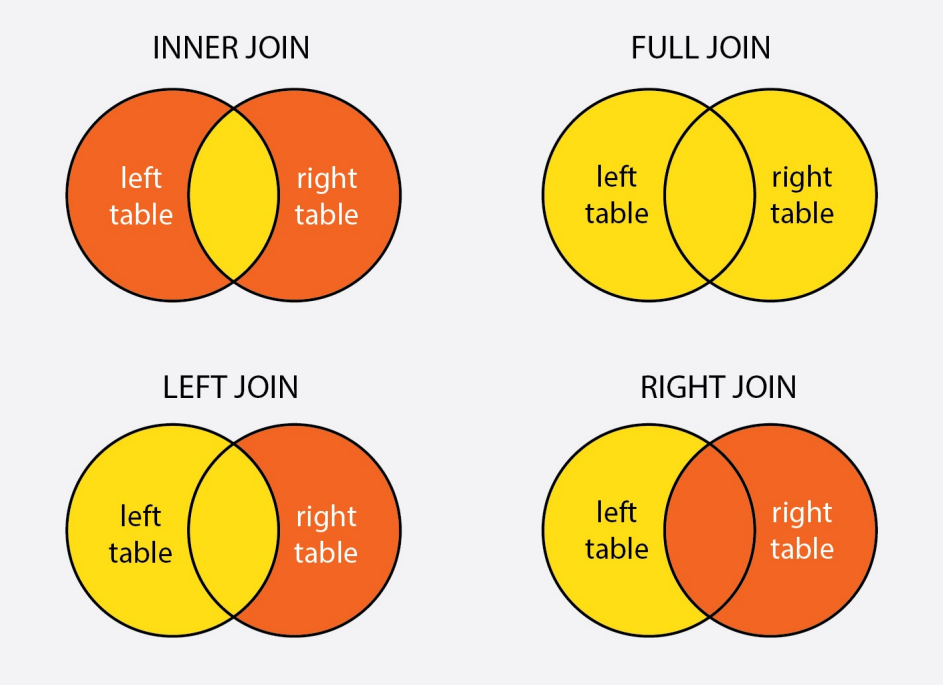


Tabella STUDENTE

Matricola	Nome	Cognome	NomeClasse
1111	Paolo	Rossi	5CA
2222	Gino	Verdi	4CA
3333	Luigi	Neri	NULL

Tabella CLASSE

NomeClasse	Piano
5CA	2
4CA	2
3CA	1

```
INNER JOIN CLASSE
ON STUDENTE.NomeClasse = CLASSE.NomeClasse;
```

INNER (o NATURAL) JOIN

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA

```
SELECT Nome, Cognome, STUDENTE.NomeClasse, CLASSE.NomeClasse
FROM STUDENTE
LEFT JOIN CLASSE
ON STUDENTE.NomeClasse = CLASSE.NomeClasse;
```

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA
Luigi	Neri	NULL	NULL

```
SELECT Nome, Cognome, STUDENTE.NomeClasse, CLASSE.NomeClasse
FROM STUDENTE
RIGHT JOIN CLASSE
ON STUDENTE.NomeClasse = CLASSE.NomeClasse;
```

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	4CA
NULL	NULL	NULL	3CA

(CROSS) JOIN

Nome	Cognome	NomeClasse	NomeClasse
Paolo	Rossi	5CA	5CA
Gino	Verdi	4CA	5CA
Luigi	Neri	NULL	5CA
Paolo	Rossi	5CA	4CA
Gino	Verdi	4CA	4CA
Luigi	Neri	NULL	4CA
Paolo	Rossi	5CA	3CA
Gino	Verdi	4CA	3CA
Luigi	Neri	NULL	3CA

Funzioni di aggregazione:

- COUNT

- SELECT COUNT() AS "Conteggio" FROM ;

- SELECT COUNT() FROM ; Esempio:

SELECT COUNT(*) AS "Conteggio" FROM classi;

SELECT COUNT(SiglaClasse) AS "Conteggio" FROM classi; SELECT COUNT(*) FROM classi;

- ORDER BY

- SELECT * FROM ORDER BY DESC; //decrescente

- SELECT * FROM ORDER BY ASC; //crescente

Esempio: SELECT * FROM classi ORDER BY numAula ASC;

- MIN E MAX

- SELECT MIN() FROM ;

- SELECT MAX() FROM ;

Esempio:

SELECT MIN(NumAula) FROM classi;

SELECT MAX(NumAula) FROM classi;

- GROUP BY

- SELECT COUNT() FROM GROUP BY ;

Esempi:
SELECT COUNT(*) FROM studenti GROUP BY citta;
SELECT citta,COUNT(*) FROM studenti GROUP BY citta;

- AVG

- SELECT AVG() FROM ;

Esempio: SELECT AVG(eta) FROM studenti;

- HAVING E BETWEEN

- SELECT COUNT() FROM GROUP BY HAVING ;
- SELECT COUNT() FROM GROUP BY HAVING BETWEEN;

Esempi:

SELECT eta,citta,COUNT(*) AS "conteggio" FROM studenti GROUP BY eta HAVING eta>18;

SELECT eta,citta,COUNT(*) AS "conteggio" FROM studenti GROUP BY eta HAVING eta BETWEEN 10 AND 30;

Operatori di confronto:

- IN:

Richiede che il valore di <Attributo> sia tra quelli specificati da: <Valore1>, .. , <ValoreN>

<Attributo> IN (<Valore1>, .. , <ValoreN>);

Esempio: SELECT * FROM clienti WHERE clienti.CodCliente IN(1,2);

- Between:

Richiede che il valore di <Attributo> sia compreso tra i valori <Min> e <Max>

<Attributo> BETWEEN <Min> AND <Max>;

- Not Between:

Richiede che il valore di <Attributo> non sia compreso tra i valori <Min> e <Max>

<Attributo> NOT BETWEEN <Min> AND <Max>;

- Like:

Richiede che il valore di <Attributo> assuma il formato specificato da <Espressione1>

<Attributo> LIKE <Espressione1>;

Esempio:

SELECT * FROM clienti WHERE clienti.Cognome LIKE("r%"); //tutti i cognomi dei clienti che iniziano per la lettera "r"

- Not Like:

Richiede che il valore di <Attributo> non assuma il formato specificato da <Espressione1>

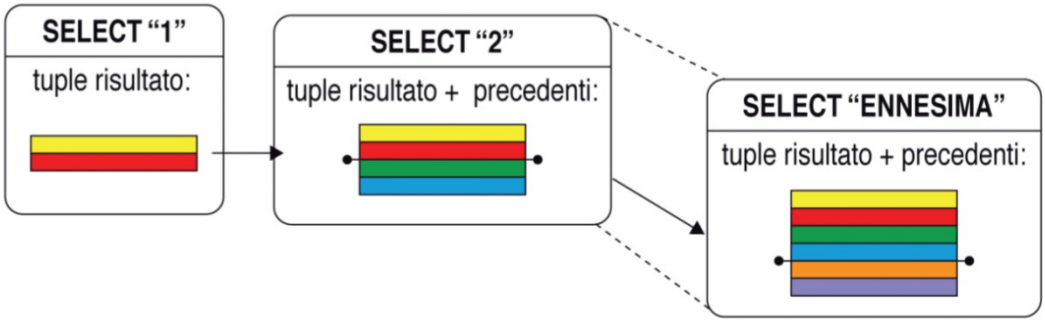
<Attributo> NOT LIKE <Espressione1>;

SubQuery e Set queries (query insiemistica)

Per eseguire query complesse è possibile strutturare opportunamente più comandi select. Le interrogazioni possono contenere all'interno altre interrogazioni, dette sottointerrogazioni o subquery. Le query binarie o "set queries", invece, permettono di combinare insieme più SELECT

Set Queries

Il risultato di una query può essere formato dall'UNIONE dei risultati di due o piu' query.
E' necessario, però, che il risultato della query composita sia omogeneo, ovvero è necessario che tutte le query da unire restituiscano lo stesso tipo di risultato, cioe' una tabella con **stesso nome, stesso numero di attributi, attributi con lo stesso nome e dello stesso tipo**



Nella "Select Unione" trovo le tuple senza ripetizioni delle tuple di ciascuna "Select" , quindi giallo,rosso (una volta sola)- verde, celeste- marrone,viola (tuple delle select 3,4, ecc.)

Set Queries

```
(SELECT nome, stelle, citta, nazione FROM hotel_italia)
UNION
(SELECT name AS nome, stars AS stelle, city AS citta, country AS nazione FROM hotel_europa)
ORDER BY stelle DESC, nome ASC;
```

Set Queries con filtro

Le due select sono racchiuse dentro le parentesi tonde e definiscono una nuova tabella a cui viene applicata la clausola ORDER BY come se si trattasse di una sola SELECT .

UNION opera come se fosse una SELECT DISTINCT. Qualora si desideri mostrare tutti i risultati estratti, con ripetizioni, sarà necessario utilizzare **UNION ALL**.

Non è possibile applicare una WHERE ad un unione di query; per poter raggiungere un tale risultato è necessario "importare" la UNION all'interno di un'istruzione di tipo SELECT ... FROM*:*

```
SELECT * FROM
 (
  SELECT nome, stelle, citta, nazione FROM hotel_italia
  UNION
  SELECT name AS nome, stars AS stelle, city AS citta, country AS nazione FROM hotel_europa
 )AS temp
  WHERE temp.stelle = 5;
```

Tabella Maternità

Maternità

cognome_fi	nome_figlic	cognome_m	nome_madi
Butti	Gino	Bari	Dina
Colombo	Patrizia	Bottari	Simona
Conti	Sara	Corelli	Lina
Conti	Gianna	Corelli	Lina
Corti	Loretta	Dini	Anna
Porta	Piera	Monti	Bianca
Valli	Mario	Tanzi	Mina
Valli	Paolo	Porta	Piera
Colombo	Piero	Fari	Sara

Tabella Paternità

Paternità

cognome_p	nome_padri	cognome_fi	nome_figlic
Butti	Gennaro	Butti	Gino
Colombo	Piero	Colombo	Patrizia
Conti	Andrea	Conti	Sara
Conti	Andrea	Conti	Gianna
Corti	Silvano	Corti	Loretta
Valli	Paolo	Valli	Mario
Porta	Pietro	Porta	Piera

La query insiemistica può essere vista come un'unione basata sugli attributi esposti nella target list delle due `SELECT`. In pratica, vengono unite tutte le tuple delle due tabelle che hanno gli attributi uguali, sia come tipo che come ordine sequenziale. Tale ordine viene chiamato notazione posizionale. Per esempio, si vogliono conoscere tutti i nomi dei genitori e dei relativi figli mediante la seguente query:

```
SELECT nome_figlio, cognome_figlio, nome_madre, cognome_madre
FROM Maternità
UNION
   SELECT nome_figlio, cognome_figlio, nome_padre, cognome_padre
   FROM Paternità
```

1 Primo passaggio intermedio:

nome_figlic	cognome_fi	nome_madı	cognome_m
Gino	Butti	Dina	Bari
Patrizia	Colombo	Simona	Bottari
Sara	Conti	Lina	Corelli
Gianna	Conti	Lina	Corelli
Loretta	Corti	Anna	Dini
Piera	Porta	Bianca	Monti
Mario	Valli	Mina	Tanzi
Paolo	Valli	Piera	Porta
Piero	Colombo	Sara	Fari

2 Secondo passaggio intermedio:

nome_figlic	cognome_fi	nome_padı	cognome_p
Gino	Butti	Gennaro	Butti
Patrizia	Colombo	Piero	Colombo
Sara	Conti	Andrea	Conti
Gianna	Conti	Andrea	Conti
Loretta	Corti	Silvano	Corti
Mario	Valli	Paolo	Valli
Piera	Porta	Pietro	Porta

Per poter realizzare un'unione è necessario che le due tabelle da unire possiedano lo stesso numero di colonne, e che i domini degli attributi siano uguali. L'unione delle due tabelle produce una nuova tabella che contiene tutte le righe delle tabelle di partenza, escludendo le righe doppie. In questo caso non vi sono righe doppie, per cui si ottiene:

nome_figlic	cognome_fi	nome_madı	cognome_m
Gianna	Conti	Andrea	Conti
Gianna	Conti	Lina	Corelli
Gino	Butti	Dina	Bari
Gino	Butti	Gennaro	Butti
Loretta	Corti	Anna	Dini
Loretta	Corti	Silvano	Corti
Mario	Valli	Mina	Tanzi
Mario	Valli	Paolo	Valli
Paolo	Valli	Piera	Porta
Patrizia	Colombo	Piero	Colombo
Patrizia	Colombo	Simona	Bottari
Piera	Porta	Bianca	Monti
Piera	Porta	Pietro	Porta
Piero	Colombo	Sara	Fari
Sara	Conti	Andrea	Conti
Sara	Conti	Lina	Corelli

E' buona regola assegnare un nome con alias ai campi risultato

```
SELECT nome_figlio, cognome_figlio, nome_madre AS "nome genitore",
cognome_madre AS "cognome genitore"
FROM Maternità
UNION
   SELECT nome_figlio, cognome_figlio, nome_padre, cognome_padre
   FROM Paternità
```

Se vogliamo conoscere il numero di figli complessivo per ciascun genitore:

```
SELECT P.cognome_padre AS cognome, P.nome_padre AS nome, COUNT(*)
AS "totale figli"
FROM Paternità P
GROUP BY P.cognome_padre, P.nome_padre
UNION
   SELECT M.cognome_madre, M.nome_madre, COUNT(*) AS "totale figli"
   FROM Maternità M
   GROUP BY M.cognome_madre, M.nome_madre
```

cognome	nome	"totale figli'
Bari	Dina	1
Bottari	Simona	1
Butti	Gennaro	1
Colombo	Piero	1
Conti	Andrea	2
Corelli	Lina	2
Corti	Silvano	1
Dini	Anna	1
Fari	Sara	1
Monti	Bianca	1
Porta	Piera	1
Porta	Pietro	1
Tanzi	Mina	1
Valli	Paolo	1

Altro esempio - Set Queries

```
SELECT Nonna.Madre AS Nonn, Mamma.Figlio AS Nipote
        FROM Maternita Nonna, Maternita Mamma WHERE Nonna.Figlio=Mamma.Madre
UNION
SELECT Nonno.Padre AS Nonn, Babbo.Figlio AS Nipote
        FROM Paternita Nonno, Paternita Babbo WHERE Nonno.Figlio=Babbo.Padre
UNION
SELECT Nonno.Padre AS Nonn, Mamma.Figlio AS Nipote
        FROM Paternita Nonno, Maternita Mamma WHERE Nonno.Figlio=Mamma.Madre
UNION
SELECT Nonna.Madre AS Nonn, Babbo.Figlio AS Nipote
FROM Maternita Nonna, Paternita Babbo WHERE Nonna.Figlio=Babbo.Padre
```

(*corretta, restituisce piu' attributi e piu' righe*)

```
 SELECT Nonna.Madre Mamma.Figlio
        FROM Maternita Nonna, Maternita Mamma WHERE Nonna.Figlio=Mamma.Madre
UNION
SELECT Nonno.Padre, Babbo.Figlio
        FROM Paternita Nonno, Paternita Babbo WHERE Nonno.Figlio=Babbo.Padre
```

(*Non corretta, restituisce attributi con nomi diversi*)

Set Queries – con filtro

Unisce i risultati di piu' query e poi li seleziona.

```
Select Tabella.Nonn FROM
      (
          SELECT Nonna.Madre AS Nonn, Mamma.Figlio AS Nipote
                  FROM Maternita Nonna, Maternita Mamma WHERE Nonna.Figlio=Mamma.Madre
          UNION
          SELECT Nonno.Padre AS Nonn, Babbo.Figlio AS Nipote
                  FROM Paternita Nonno, Paternita Babbo WHERE Nonno.Figlio=Babbo.Padre
          UNION
          SELECT Nonno.Padre AS Nonn, Mamma.Figlio AS Nipote
                  FROM Paternita Nonno, Maternita Mamma WHERE Nonno.Figlio=Mamma.Madre
          UNION
          SELECT Nonna.Madre AS Nonn, Babbo.Figlio AS Nipote
                  FROM Maternita Nonna, Paternita Babbo WHERE Nonna.Figlio=Babbo.Padre
      ) Tabella
WHERE Tabella.Nipote= qualcosa… ;
```

Query annidate (query annidata dopo il where)

query annidata in clausola WHERE

SELECT	elementi da selezionare	FROM	tabelle da selezionare	WHERE	(SELECT .....) e/o altre condizioni

Una query SELECT può essere annidata in un'altra query SELECT come parte di una espressione (all'interno delle clausole SELECT, WHERE, HAVING)

•la SELECT annidata (quella interna) deve restituire un unico valore affinché questo possa essere valutato nell'espressione:

SELECT Sum(Reddito) FROM Persone WHERE Eta > 30 AND Reddito > (SELECT Avg(Reddito) FROM Persone) ; (*corretta*)

SELECT Sum(Reddito) FROM Persone WHERE
Eta > 30 AND Reddito > (SELECT **Reddito** FROM Persone WHERE Nome LIKE 'A%') ;
(*non corretta, puo' restituire piu' righe*)

SELECT Sum(Reddito) FROM Persone WHERE
Eta > 30 AND Reddito > (SELECT **Reddito, Eta** FROM Persone Persone WHERE Nome LIKE 'A%');
(non corretta restituisce piu' attributi) (anche più righe)

Query annidate (query annidata dopo il from)

query annidata in clausola FROM

SELECT	elementi da selezionare	FROM	(SELECT .....) e/o altre tabelle	WHERE	condizioni

Una query SELECT può essere annidata in un'altra query SELECT all'interno della clausola FROM
la SELECT annidata (quella interna) puo' restituire una tabella con piu' righe e colonne.

```
SELECT MAX(tbl.quanti)
FROM (
  SELECT COUNT(id) AS quanti, cognome
  FROM amici
  GROUP BY cognome
) AS tbl;
```

altro esempio con un natural join:

NATURAL JOIN

È un tipo di operazione che ci permette di correlare due o più tabelle sulla base di valori uguali in attributi contenenti lo stesso tipo di dati.

INNER JOIN

È un tipo di join in cui le righe delle tabelle vengono combinate solo se i campi collegati con join soddisfano una determinata condizione.

```
SELECT * FROM
(SELECT Nonna.Madre AS Gen, Mamma.Figlio AS Fig FROM Maternita Nonna, Maternita
Mamma WHERE Nonna.Figlio=Mamma.Madre)
TabellaFem,
(SELECT Nonno.Padre AS Gen, Babbo.Figlio AS Fig FROM Paternita Nonno, Paternita Babbo
WHERE Nonno.Figlio=Babbo.Padre)
TabellaMas
WHERE TabellaFem.Fig LIKE 'A%'
```

Tipi di Subquery

Le query annidate di tipo scalare (restituisce un singolo valore)
Le subquery scalari annidate producono un result-set costituito da una sola tupla con un solo attributo.

query esterna	Operatore	query interna: *restituisce solo un attributo*
SELECT ... FROM ... WHERE espressione	= != > < !> !< >= <=	(SELECT ... FROM ... WHERE espressione)

La query che si trova tra le parentesi tonde, chiamata anche select interna o query annidata, viene eseguita per prima; quindi, sulla base del risultato ottenuto, viene eseguita l'altra query, chiamata anche select esterna o query principale.

In questo esempio si vede come è possibile usare una query annidata per verificare quali padri hanno avuto meno figli di "Corelli Lina":

```
SELECT cognome_padre, nome_padre, COUNT(*) AS figli
FROM paternità
GROUP BY cognome_padre, nome_padre
HAVING COUNT(*)<
   ( SELECT COUNT(*)
   FROM Maternità
   WHERE cognome_madre="Corelli" and nome_madre="Lina");
```

Tabella Dipendenti

Dipendenti

matricola	nome	cognome	mansione	stipendio
1	verdi	gianni	1	900
2	bianchi	gino	2	2900
3	corti	sandro	3	4800
4	bini	dino	4	1200
5	brutti	cino	5	1300
6	porto	guido	10	1650
7	dritto	guido	10	1450
8	storto	guido	10	1220
9	butti	sonia	9	6500

Tabella Mansioni

Mansioni

ID_mansion	nome	tariffa_orari
1	giardiniere	5
2	manager	125
3	top manager	145
4	impiegato 1 liv	12
5	impiegato 2 liv	15
6	impiegato 3 liv	18
7	dirigente	27
8	cassiere	24
9	segretaria	85
10	autista	22
11	uscere	7

```
SELECT cognome, nome, stipendio
FROM Dipendenti
WHERE stipendio <
   (SELECT AVG(stipendio) FROM Dipendenti
   WHERE mansione=9)
   and stipendio >
   (SELECT AVG(stipendio) FROM Dipendenti
   WHERE mansione=10)
```

cognome	nome	stipendio
gino	bianchi	2900
sandro	corti	4800
guido	porto	1650
guido	dritto	1450

Le query annidate di tipo non scalare

Subquery che restituiscono un unica riga (con più campi)

```
SELECT *
FROM amici
WHERE ROW(nome, cognome) = (
  SELECT nome, cognome FROM parenti WHERE id = 1
);
```

Subquery che restituiscono una sola colonna (molteplici righe)

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Predicati ANY e ALL

- **ANY:** la condizione della clausola WHERE è vera se il valore dell'attributo <attributo> compare in almeno uno dei valori forniti dalla subquery<sottoquery>;
- **ALL:** la condizione della clausola WHERE è vera se il valore dell'attributo <attributo> compare in tutti quelli restituiti dalla subquery<sottoquery>;

Sintassi:

```
SELECT <ListaAttributi>
FROM <ListaTabelle>
WHERE <Attributo><OperatoreRelazionale> ANY | ALL (<SottoQuery>);
```

Esempi:

Seleziona dalla tabella "amici" tutti i record in cui il numero di telefono è uguale (=) ad almeno uno (*ANY*) dei valori estratti dalla tabella "parenti".

Any e Some sono sinonimi

```
SELECT *
FROM amici
WHERE telefono = ANY(
  SELECT telefono FROM parenti
);
```

"Nome e cognome dei dipendenti con stipendio netto maggiore rispetto a tutti gli stipendi medi d'Europa"

```
 SELECT Nome,Cognome
FROM Dipendenti
WHERE StipendioNetto> ALL (SELECT DISTINCT StipendioMedio FROM Stipendi WHERE Continente = "Europa")
```

Supponiamo di voler conoscere i nomi dei dipendenti che guadagnano meno di 1500 euro con la stessa mansione di quelli che ne guadagnano di più.

```
SELECT cognome, nome, stipendio
FROM Dipendenti
WHERE stipendio <=1500
AND mansione =
   (SELECT mansione
      FROM Dipendenti
      WHERE stipendio>1500);
```

Errata (la query annidata non restituisce un solo valore

ANY

La query deve essere letta in questo modo: “elencare tutti i dipendenti che hanno stipendio minore o uguale a 1500 con mansione uguale a uno qualsiasi tra i dipendenti con la stessa mansione ma con stipendio maggiore di 1500”.

```
SELECT cognome, nome, stipendio
FROM Dipendenti
WHERE stipendio <=1500
AND mansione = ANY
   (SELECT mansione FROM Dipendenti
   WHERE stipendio>1500);
```

ALL

Il nome e la mansione dei dipendenti con uno stipendio più alto di quello di tutti i dipendenti con mansione “autista”.

```
SELECT M.nome AS mansione, D.cognome, D.nome, D.stipendio
FROM Dipendenti D INNER JOIN Mansioni M ON M.ID_mansione=D.mansione
WHERE D.stipendio > ALL
   (SELECT D.stipendio
   FROM Dipendenti D, Mansioni M
   WHERE M.nome=”autista”
   and M.ID_mansione=D.mansione)
```

Subquery che restituiscono righe

SELECT colonna1,colonna2

FROM t1

WHERE (colonna1,colonna2) IN (SELECT colonna1,colonna2 FROM t2);

Predicati IN e NOT IN

- **IN:** la condizione della clausola WHERE è vera se il valore dell'attributo <attributo> appartiene all'insieme dei valori fornito dalla subquery<sottoquery>;

- **NOT IN:** la condizione della clausola WHERE è vera se il valore dell'attributo <attributo> non appartiene all'insieme dei valori fornito dalla subquery<sottoquery>;

Sintassi:

SELECT <ListaAttributi>

FROM <ListaTabelle>

WHERE <Attributo><OperatoreRelazionale> IN | NOT IN (<SottoQuery>);

Esempio:

"Quanti sono gli animali dello zoo originari dell'Africa ?"

SELECT COUNT(CodAnimale)

FROM zoo

WHERE CodRazza IN (SELECT CodRazzaFROM razza WHERE Continente="Africa");

Predicati EXISTS e NOT EXISTS

EXISTS: la condizione della clausola WHERE è vera se la subquery<sottoquery> produce una tabella non vuota;

NOT EXISTS: la condizione della clausola WHERE è vera se il risultato della subquery<sottoquery> è una tabella vuota, senza alcuna riga;

Sintassi:

- SELECT <ListaAttributi>

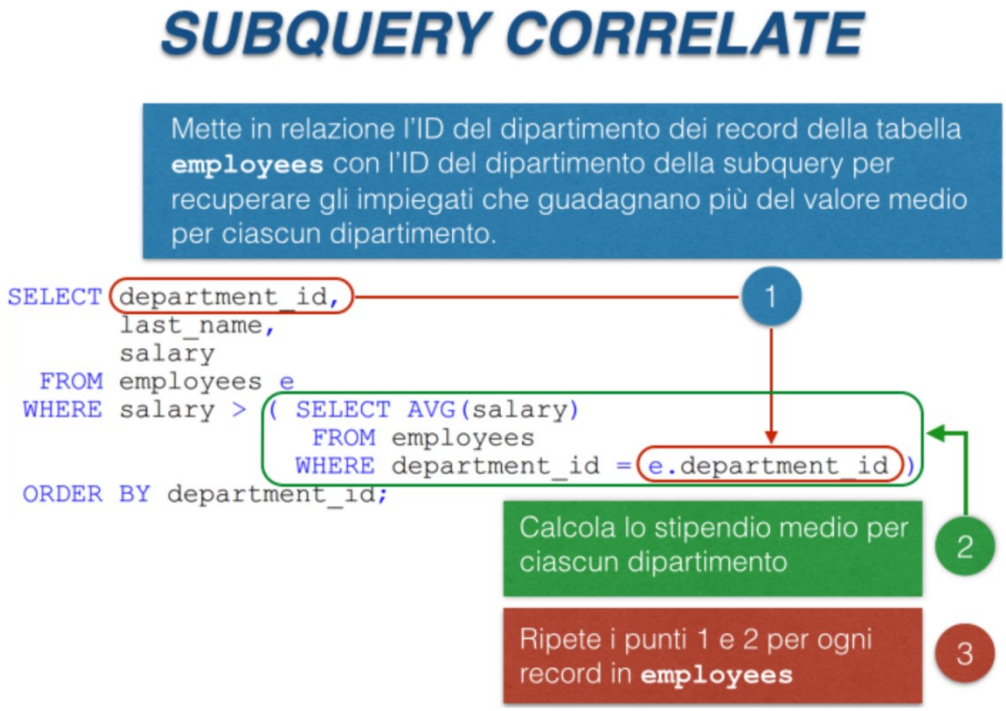
FROM <ListaTabelle>

WHERE EXISTS | NOT EXISTS (<SottoQuery>);

Esempio: "Quali sono i clienti che hanno richiesto di viaggiare ?"

- SELECT * FROM CLIENTE C

WHERE EXISTS (SELECT * FROM Richiede H WHERE C.CodCli = H.CodCli);



Si dicono **correlate** le *subquery* che contengono un riferimento ad una delle tabelle che fanno parte della query esterna:

```
SELECT *
FROM amici
WHERE nome = ANY(  SELECT nome
                   FROM parenti
                   WHERE parenti.citta = amici.citta );
```

Con questa query selezioniamo dalla tabella "amici" tutti i record in cui il nome è uguale ad almeno uno di quelli restituiti dalla tabella "parenti" a loro volta selezionati in base ad un'uguaglianza di città tra le due tabelle. E', appunto, quest'ultimo filtro a definire la *subquery* come correlata essendo stato utilizzato nella clausola *WHERE* di quest'ultima un campo della tabella esterna.

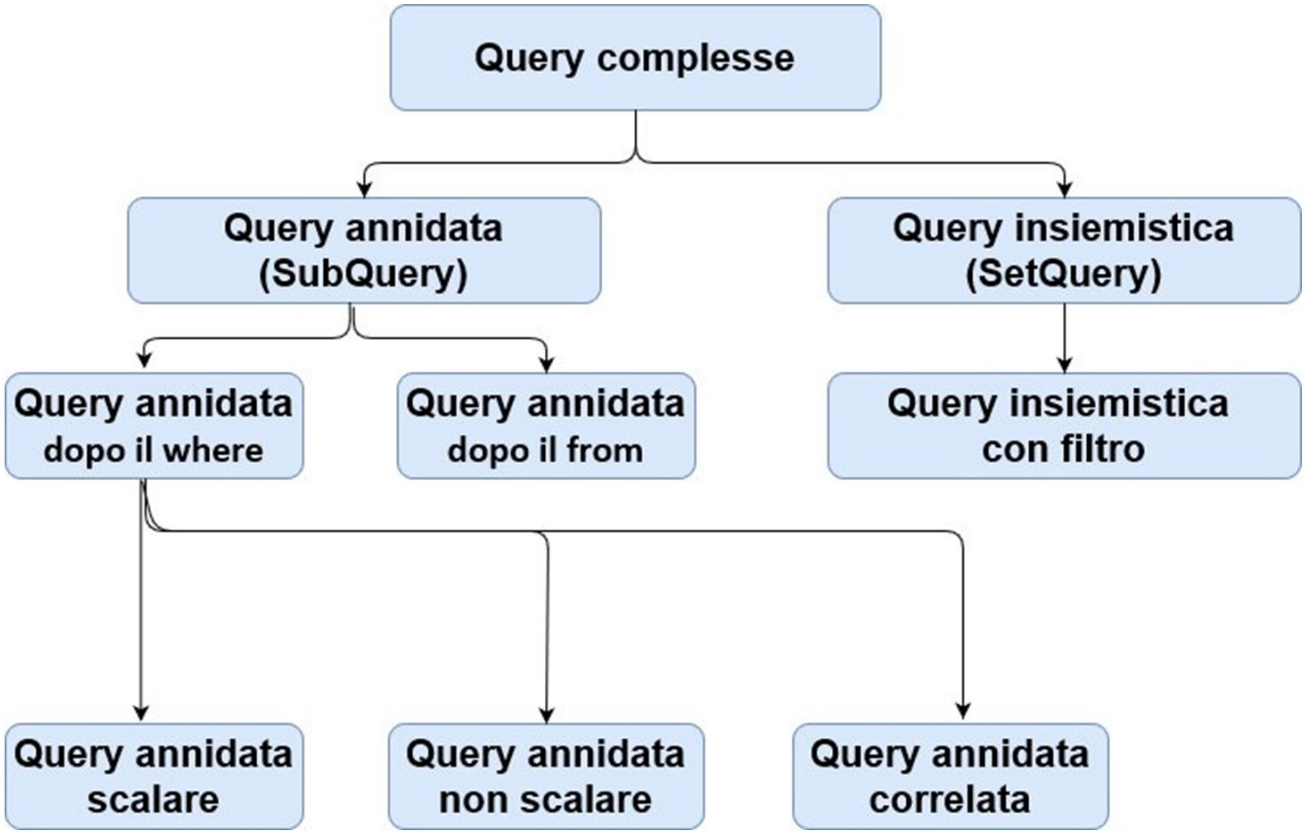
```
mysql> select * from amici;
+-------+---------+
| nome  | citta   |
+-------+---------+
| isa   | napoli  |
| luca  | caserta |
| mario | caserta |
+-------+---------+
3 rows in set (0.00 sec)

mysql> select * from parenti;
+--------+---------+
| nome   | citta   |
+--------+---------+
| franco | caserta |
| mario  | caserta |
| sara   | napoli  |
| mario  | napoli  |
+--------+---------+
4 rows in set (0.00 sec)

mysql> select * from amici where nome = all (select nome from parenti where parenti.citta = amici.citta);
Empty set (0.00 sec)

mysql> select * from amici where nome = any (select nome from parenti where parenti.citta = amici.citta);
+-------+---------+
| nome  | citta   |
+-------+---------+
| mario | caserta |
+-------+---------+
1 row in set (0.00 sec)

mysql>
```


Query complesse
Query annidata (SubQuery)
Query insiemistica (SetQuery)
Query annidata dopo il where
Query annidata dopo il from
Query insiemistica con filtro
Query annidata scalare
Query annidata non scalare
Query annidata correlata

Vincoli di dominio:

relativi al singolo attributo, generalmente utilizzati nella creazione della tabella (NOT NULL, DEFAULT, CHECK);

- **Check:** al suo interno è possibile usare gli operatori IN, BETWEEN, LIKE;

Il linguaggio SQL standard prevede la clausola CHECK nel comando CREATE TABLE proprio per controllare il valore dei dati immessi in una tabella.
CHECK Constraint: consente di assicurarsi che tutti i valori presenti in una colonna soddisfino determinati criteri.

```
CREATE TABLE t1
(
  CHECK (c1 <> c2),
  c1 INT CHECK (c1 > 10),
  c2 INT CONSTRAINT c2_positive CHECK (c2 > 0),
  c3 INT CHECK (c3 < 100),
  CONSTRAINT c1_nonzero CHECK (c1 <> 0),
  CHECK (c1 > c3)
);
```

Il primo vincolo è un **vincolo di tabella**: si verifica all'esterno di qualsiasi definizione di colonna, p Questo vincolo contiene riferimenti a colonne non ancora definite. Non viene specificato alcun nome di vincolo, pertanto MySQL genera un nome.
I tre vincoli successivi sono **vincoli di colonna**: ognuno verifica all'interno di una definizione di colonna e pertanto può fare riferimento solo alla colonna da definire. Uno dei vincoli è denominato in modo esplicito. MySQL genera un nome per ognuno degli altri due.
Gli ultimi due vincoli sono vincoli di tabella. Uno di essi è denominato in modo esplicito. MySQL genera un nome per l'altro.

MySQL ammette l'uso della cluasola clausola check, ma solo per compatibilità con lo standard SQL, e tale opzione non ha alcun effetto concreto nel controllare i dati immessi nella tabella. Solo a partire dalla versione 8.016 check effettua un controllo effettivo.

Trigger

Un trigger è un insieme di comandi che viene eseguito automaticamente a causa di eventi che modificano il database e può essere usato in sostituzione della clausola "Check".
Con la versione 5 di MySQL sono state introdotte nuove funzionalità che hanno potenziato notevolmente questo DBMS rendendolo concorrenziale con alternative commerciali ritenute più avanzate (si pensi per esempio ad Oracle); una delle novità più importanti attiene alla possibilità di creare trigger per la gestione delle procedure.
In pratica un trigger inserito all'interno di un database consente di gestire in modo automatico determinate procedure tramite regole definite, il tutto senza richiedere l'intervento dell'utente che dovrà digitare un numero inferiore di comandi ed istruzioni SQL; in questo modo la base di dati avrà la possibilità di dar vita a particolari comportamenti agendo in risposta al verificarsi di eventi esterni, parliamo quindi di basi di dati "attive".

Un trigger è un componente di un database abbinato a una tabella con la seguente sintassi:

- CREATE TRIGGER ON FOR EACH ROW

Esempio:

- CREATE TRIGGER InserimentoEditori BEFORE INSERT ON Editori

FOR EACH ROW BEGIN

IF NEW.nome IS NULL SET NEW.nome = concat('manca il nome!')

END IF;

END;

- **NOT NULL:** Il campo di una colonna non può avere valore "NULL";

- Default: consente di fornire un valore predefinito ad una colonna;

Esempio
CREATE TABLE personale (codice INT, paga DECIMAL(4,2));
CREATE TRIGGER test_sum BEFORE INSERT ON personale
FOR EACH ROW SET @sum = @sum + NEW.paga;

Nell'esempio abbiamo creato una semplice tabella denominata personale in cui inserire il codice identificativo di ogni impiegato e la relativa paga; per quanto riguarda invece il secondo gruppo di istruzioni, questo è stato necessario per la creazione di un INSERT trigger con il compito di sommare i valori nella colonna paga.

Vincoli di ennupla o tupla (riferiti a più attributi):

- **PRIMARY KEY;**

- **UNIQUE:** (crea un indice su attributi chiave, se non specificato, su attributi non chiave);

- **CHECK:** (riguarda più attributi);

Vincoli di integrità referenziale (riferiti alle associazioni, più tabelle): //FOREIGN KEY ... REFERENCES.

Esempio derivazione logica di un'associazione 1:N (Integrità Referenziale):

CREATE TABLE studenti (matricola varchar (5), siglaClassevarchar (5),nome varchar (50) NOT NULL, cognome varchar (50) NOT NULL, citta varchar(50), PRIMARY KEY (matricola), FOREIGN KEY (siglaClasse) REFERENCES classi);

Opzioni della FOREIGN KEY:

Le transazioni sono delle macro-operazioni che racchiudono, al loro interno, una serie di sub-operazioni le quali possono raggiungere il successo o il fallimento solo restando unite. Se una sola delle operazioni previste all'interno della transaction fallisce, allora fallisce l'intera transazione generando un rollback che ripristina la situazione al momento immediatamente antecedente all'inizio delle operazioni.

L'engine InnoDB fornisce, oltre alle Foreign keys, un sistema di transazioni con rollback e possibilità di recupero a seguito di un crash, un sistema di locking ben funzionante, prestazioni elevate con tabelle con una grande mole di dati, la possibilità di creare tabelle di ogni dimensione anche con filesystem con limiti nella dimensione dei file.
Dalla versione 4.0 di MySQL questo storage engine è attivo di default
Non è necessario che tutte le tabelle del database siano di tipo InnoDB, basta che lo siano le tabelle che dovranno fare uso delle funzioni garantite solo da questo storage engine.

Possiamo definire azioni diverse da far attivare in caso di cancellazione o modifica:

- **RESTRICT, NO ACTION** oppure nessuna opzione (restituisce errore e non consente di proseguire)

- **SET DEFAULT** (non utilizzabile in MySQL);

- **SET NULL** (imposta gli attributi a null);

- **CASCADE** (consente l'operazione e la estende ai collegati);

CASCADE
In questo caso la cancellazione o modifica di un record nella tabella padre genererà la cancellazione o la modifica dei record collegati nella tabella figlia.
Se cancelliamo un editore cancelleremo tutti i libri collegati, se modifichiamo un editore cambiando l'id ad esempio sostituiremo il valore del campo editore anche nella tabella libri.

SET NULL
Questa azione è attivabile solo se il campo interessato della tabella figlia non è impostato a NOT NULL. Come si può intuire in caso di eliminazione o modifica di un record nella tabella padre i record collegati della tabella figlia verranno modificati impostando il campo NULL.

NO ACTION o RESTRICT
In questo caso si potrebbe pensare che il record della tabella padre venga cancellato o modificato senza che cambi niente. Queste due azioni invece impediscono direttamente la modifica o la cancellazione dei record della tabella padre.