Create la tabella Dipendenti:

create table Dipendenti(
- ➔ id int(3) not null auto_increment,
- ➔ livello int(1) not null,
- ➔ stipendio int(4) not null,
- ➔ primary key(id));

```
mysql> describe Dipendenti;
+-----------+--------+------+-----+---------+----------------+
| Field     | Type   | Null | Key | Default | Extra          |
+-----------+--------+------+-----+---------+----------------+
| id        | int(3) | NO   | PRI | NULL    | auto_increment |
| livello   | int(1) | NO   |     |         |                |
| stipendio | int(4) | NO   |     |         |                |
+-----------+--------+------+-----+---------+----------------+
```

Inserite dei valori come mostrato di seguito:

```
mysql> insert into Dipendenti
    -> values(NULL,2,2800);
Query OK, 1 row affected (0.03 sec)
```

Visualizzate il contenuto della tabella (* =**tutte le colonne**)
Non vi è la clausola **WHERE** pertanto vengono visualizzate **tutte le righe**

```
mysql> select * from Dipendenti;
+----+---------+-----------+
| id | livello | stipendio |
+----+---------+-----------+
|  1 |       3 |      1200 |
|  2 |       5 |       800 |
|  3 |       1 |     10000 |
|  4 |       2 |      5000 |
|  5 |       2 |      5000 |
|  6 |       2 |      5000 |
|  7 |       2 |      4000 |
|  8 |       2 |      4400 |
|  9 |       3 |      1400 |
| 10 |       3 |      1300 |
| 11 |       4 |      1000 |
| 12 |       4 |       950 |
| 13 |       2 |      2800 |
+----+---------+-----------+
13 rows in set (0.00 sec)
```

RESTRIZIONE
Se aggiungete la clausola WHERE come mostrato di seguito verranno visualizzate solo alcune RIGHE

```
mysql> select * from Dipendenti
    -> where livello=2;
+----+---------+-----------+
| id | livello | stipendio |
+----+---------+-----------+
|  4 |       2 |      5000 |
|  5 |       2 |      5000 |
|  6 |       2 |      5000 |
|  7 |       2 |      4000 |
|  8 |       2 |      4400 |
| 13 |       2 |      2800 |
+----+---------+-----------+
6 rows in set (0.02 sec)
```

PROIEZIONE
Potete di decidere di visualizzare **solo alcune colonne** sostituendo all' asterisco i nomi delle colonne

```
mysql> select livello,stipendio from Dipendenti
    -> where livello=2;
+---------+-----------+
| livello | stipendio |
+---------+-----------+
|       2 |      5000 |
|       2 |      5000 |
|       2 |      5000 |
|       2 |      4000 |
|       2 |      4400 |
|       2 |      2800 |
+---------+-----------+
6 rows in set (0.00 sec)
```

Questo è un esempio di **proiezione**(solo **alcune colonne**->livello e stipendio) e di **restrizione**(solo **alcune righe**->WHERE)

Creiamo un'altra tabella Dipendenti2:

```
mysql> create table Dipendenti2(
    -> id int(3) not null auto_increment,
    -> livello int(1) not null,
    -> stipendio int(4) not null,
    -> nome varchar(10) not null,
    -> cognome varchar(10) not null,
    -> primary key(id));
Query OK, 0 rows affected (0.09 sec)
```

Inseriamo dei valori anche in Dipendenti2:

```
mysql> insert into Dipendenti2
   -> values(NULL,2,2800,"Mario","Rossi");
Query OK, 1 row affected (0.05 sec)
```

Osserviamo che se proviamo ad assegnare NULL a qualcuno dei campi otteniamo un messaggio di errore, perché avevamo specificato come vincoli che tali campi non possono essere nulli:

```
mysql> insert into Dipendenti2
   -> values(NULL,2,3000,"Tizio",NULL);
ERROR 1048 (23000): Column 'cognome' cannot be null
```

Possiamo però benissimo inserire una stringa vuota come cognome

```
mysql> insert into Dipendenti2
   -> values(NULL,2,3000,"Tizio","");
Query OK, 1 row affected (0.03 sec)
```

Ovviamente ciò non è desiderabile, anche se ci siamo stufati di inserire cognomi è opportuno inserire un altro vincolo ovvero che nome e cognome non possano essere stringhe vuote.
Per introdurre questi ulteriori vincoli devo modificare la dichiarazione della tabella in maniera simile a quanto segue(CHAR_LENGTH(arg) ritorna la lunghezza in caratteri di arg)

```
mysql> create table Dipendenti3(
   -> id int(3) not null auto_increment,
   -> livello int(1) not null,
   -> stipendio int(4) not null,
   -> nome varchar(10) not null,
   -> cognome varchar(10) not null,
   -> primary key(id),
   -> check (char_length(nome)>0),
   -> check (char_length(cognome)>0));
```

Oppure

```
mysql> create table Dipendenti3(
   -> id int(3) not null auto_increment,
   -> livello int(1) not null,
   -> stipendio int(4) not null,
   -> nome varchar(10) not null,
   -> cognome varchar(10) not null,
   -> primary key(id),
   -> check (nome<> “”),
   -> check (cognomen<> “”));
```

In realtà però **in MySQL 5.0 il costrutto CHECK non è supportato**, per cui **l’effetto di tali istruzioni sarà nullo.**
Come si fa allora?
Se si stanno inserendo i dati attraverso una pagina web è necessario implementare il controllo nella pagina jsp(php,asp etc…) . Ovvero sarà il server ad effettuare il controllo e non MySQL!

La cosa migliore in realtà è effettuare tale controllo prima sul browser dell'utente e successivamente per motivi di sicurezza sul server.
Questo genere di controlli viene fatto abitualmente quando si inseriscono dei dati nei form di iscrizione ad un sito…

Riprendiamo la tabella Dipendenti2:

```
mysql> select * from Dipendenti2;
+----+---------+-----------+----------+---------+
| id | livello | stipendio | nome     | cognome |
+----+---------+-----------+----------+---------+
|  1 |       2 |      2800 | Mario    | Rossi   |
|  2 |       2 |      3000 | Raffaele | Zitano  |
|  3 |       2 |      3000 | Tizio    |         |
|  4 |       2 |      1000 | Mario    | Rossi   |
+----+---------+-----------+----------+---------+
4 rows in set (0.00 sec)
```

Osserviamo che vi sono due dipendenti che hanno lo stesso nome.
Se voglio visualizzare nome e cognome di ogni dipendente dell'azienda otterrei:

```
mysql> select nome,cognome from Dipendenti2;
+----------+---------+
| nome     | cognome |
+----------+---------+
| Mario    | Rossi   |
| Raffaele | Zitano  |
| Tizio    |         |
| Mario    | Rossi   |
+----------+---------+
4 rows in set (0.00 sec)
```

Se voglio che non vengano visualizzate le righe uguali utilizzo l'opzione DISTINCT dopo la select

```
mysql> select distinct nome,cognome from Dipendenti2;
+----------+---------+
| nome     | cognome |
+----------+---------+
| Mario    | Rossi   |
| Raffaele | Zitano  |
| Tizio    |         |
+----------+---------+
3 rows in set (0.00 sec)
```

Creiamo una nuova tabella Hobbies:

```
mysql> create table Hobbies(
    -> nome_hobby varchar(10) not null,
    -> unique(nome_hobby));
Query OK, 0 rows affected (0.09 sec)
```

Si noti che abbiamo specificato un vincolo con UNIQUE, tale vincolo comporta che i valori assegnati all'attributo nome_hobby devono essere distinti, non possiamo inserire due volte lo stesso valore.

Inseriamo nella nuova tabella qualche elemento:

```
mysql> insert into Hobbies
    -> values("pesca");
Query OK, 1 row affected (0.03 sec)

mysql> insert into Hobbies
    -> values("sci");
Query OK, 1 row affected (0.03 sec)
```

Proviamo a reinserire sci:

```
mysql> insert into Hobbies
    -> values("sci");
ERROR 1062 (23000): Duplicate entry 'sci' for key 1
```

Abbiamo un errore com'era prevedibile MySQL ci segnala che è stato violato il vincolo UNIQUE, pertanto l'inserimento non può essere effettuato!

```
mysql> select * from Hobbies;
+------------+
| nome_hobby |
+------------+
| pesca      |
| sci        |
+------------+
2 rows in set (0.00 sec)
```

PRODOTTO CARTESIANO

Adesso effettuiamo una select sulle **due tabelle** Dipendenti2 e Hobbies:

```
mysql> select nome,cognome,nome_hobby from Dipendenti2,Hobbies;
+----------+---------+------------+
| nome     | cognome | nome_hobby |
+----------+---------+------------+
| Mario    | Rossi   | pesca      |
| Mario    | Rossi   | sci        |
| Raffaele | Zitano  | pesca      |
| Raffaele | Zitano  | sci        |
| Tizio    |         | pesca      |
| Tizio    |         | sci        |
+----------+---------+------------+
6 rows in set (0.00 sec)
```

Possiamo osservare che la query effettuata ha avuto l'effetto di effettuare il **prodotto cartesiano** tra gli elementi **della tabella Dipendenti2 e Hobbies**, **ovvero per ogni riga di Dipendenti2 ho preso tutte le righe di Hobbies.**

Alla fine **sono stati visualizzati solamente i campi nome, cognome e nome_hobby come specificato nella select.**

Ricordiamo che il prodotto cartesiano significa prendere tutte le possibili combinazioni degli elementi di un insieme con quelli di un altro.

Es. Sia A={1,2,3} e B={5,6,7} A x B={(1,5),(1,6),(1,7),
(2,5),(2,6),(2,7),
(3,5),(3,6),(3,7)}

Nel caso delle tabelle, combino ogni riga della prima tabella, con ogni riga della seconda tabella.
Se la prima tabella ha 5 righe e la seconda 6, la tabella risultante avrà 5*6=30 righe!!
C'è un altro modo per effettuare il prodotto cartesiano in SQL è quello di utilizzare l'operatore **JOIN**
mysql> select nome,cognome,nome_hobby from Dipendenti2 **join** Hobbies;

Il risultato è del tutto identico a quello già visto.

UNIONE, INTERSEZIONE E DIFFERENZA

Ricordiamo che:
L'unione di due insiemi è un'insieme formato dagli **elementi di entrambi gli insiemi** di partenza
L'intersezione di due insiemi è un'insieme formato dagli elementi che **appartengono ad entrambi** gli insiemi di partenza, **ma non ad uno solo dei due**
La differenza di due insiemi A e B è un'insieme formato dagli **elementi di A che non appartengono a B.**

Es. A={1,2,3} e B={1,6,7}

Unione: A∪ B ={1,2,3,6,7} Intersezione: A∩ B ={1} Differenza: A-B= ={2,3}

INTERSEZIONE

Vogliamo sapere se qualcuno dei dipendenti inserito nella tabella Dipendenti prende lo **stesso stipendio** di un dipendente inserito nella tabella Dipendenti2 , ovvero cerchiamo se esiste il medesimo valore nelle due colonne stipendio delle due tabelle.
La query sarebbe qualcosa di simile
mysql> (select stipendio from Dipendenti) intersect (select stipendio from Dipendenti2);
Ma la query genera un errore…
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near 'intersect (select stipendio from Dipendenti2)' at line 1

Purtroppo **MySQL non supporta INTERSECT !!**

Come faccio ad effettuare un simile controllo?
La soluzione è la seguente mysql> select **Dipendenti.stipendio** from Dipendenti,Dipendenti2 where **Dipendenti.stipendio=Dipendenti2.stipendio**;

```
+-----------+
| stipendio |
+-----------+
|      1000 |
|      2800 |
+-----------+
2 rows in set (0.01 sec)
```

La query **ritorna gli elementi** della colonna stipendio **della tabella Dipendenti che hanno lo stesso valore di elementi della colonna omonima della tabella Dipendenti2**.

Quando si ha a che fare con tabelle che hanno campi omonimi si deve usare la notazione **NOMETABELLA . NomeAttributo** per specificare le colonne in maniera non ambigua!

In entrambe le tabelle dunque vi sono almeno due persone che percepiscono uno stipendio identico.

UNIONE

Ora diciamo di voler conoscere quanti elementi sono contenuti nelle due tabelle complessivamente.
Come facciamo?
Proviamo con la seguente query
mysql> (select stipendio from Dipendenti) **union** (select stipendio from Dipendenti2);

Stavolta MySQL supporta il commando, e ciò che ci viene restituito è:
mysql> (select stipendio from Dipendenti) union (select stipendio from Dipendenti2);

```
+-----------+
| stipendio |
+-----------+
|      1200 |
|       800 |
|     10000 |
|      5000 |
|      4000 |
|      4400 |
|      1400 |
|      1300 |
|      1000 |
|       950 |
|      2800 |
|      3000 |
```

```
+-----------+
12 rows in set (0.00 sec)
```

Notiamo che i **duplicati sono stati eliminati**, il DBMS infatti automaticamente esegue una DISTINCT sui risultati.
Per ottenere quello che vogliamo dobbiamo utilizzare UNION ALL al posto di UNION

```
mysql> (select stipendio from Dipendenti) union all (select stipendio from Dipendenti2);
+-----------+
| stipendio |
+-----------+
|      1200 |
|       800 |
|     10000 |
|      5000 |
|      5000 |
|      5000 |
|      4000 |
|      4400 |
|      1400 |
|      1300 |
|      1000 |
|       950 |
|      2800 |
|      2800 |
|      3000 |
|      3000 |
|      1000 |
+-----------+
17 rows in set (0.19 sec)
```

Un modo alternativo per ottenere tale informazione è quello di utilizzare la seguente query(l'utilizzo della funzione COUNT sarà spiegato più avanti, mentre per altri esempi di operazioni si rimanda alle fotocopie del prof. Perfetto):

```
mysql> select ((select count(*) from Dipendenti) + (select count(*) from Dipendenti2 ));
+-------------------------------------------------------------------------------------+
| ((select count(*) from Dipendenti) + (select count(*) from Dipendenti2 ))   |
+-------------------------------------------------------------------------------------+
|                                                                                  17 |
+-------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

Com'è possibile vedere, la SELECT può essere utilizzata per fare operazioni, inoltre il risultato di una select può comparire all'interno di un'altra clausola select.
Di ciò parleremo diffusamente più avanti.

DIFFERENZA

Si vuole ottenere l'elenco degli stipendi dei dipendenti della tabella Dipendenti che non coincidono con quelli dei dipendenti della tabella Dipendenti2

La query sarebbe :
(select stipendio from Dipendenti) except (select stipendio from Dipendenti2);
ma anche in questo caso **MySQL non supporta EXCEPT**

Dobbiamo cercare pertanto una strada alternativa per effettuare una query equivalente.

mysql> select stipendio from Dipendenti **where stipendio not in (select stipendio from Dipendenti2);**

```
+-----------+
| stipendio |
+-----------+
|      1200 |
|       800 |
|     10000 |
|      5000 |
|      5000 |
|      5000 |
|      4000 |
|      4400 |
|      1400 |
|      1300 |
|       950 |
+-----------+
11 rows in set (0.44 sec)
```

Otteniamo infatti i valori di Dipendenti.stipendio meno i valori 2800 e 1000 che appartengono all'intersenzione tra Dipendenti.stipendio e Dipendenti2.stipendio
Questa query ci permette di utilizzare **l'operatore IN** che richiede che **il valore cercato appartenga ad un certo insieme di valori** specificato in questo caso dai valori ritornati dalla select che figura nella clausola where.(Questo operatore è stato introdotto per specificare i vincoli di dominio)
(**Rivedere gli operatori BETWEEN <Min> AND <Max>, LIKE <Espressione> sul libro!)**

FUNZIONI DI AGGREGAZIONE

Riprendiamo la tabella Dipendenti (attenzione non Dipendenti2!!):

```
mysql> select * from Dipendenti;
+----+---------+-----------+
| id | livello | stipendio |
+----+---------+-----------+
|  1 |       3 |      1200 |
|  2 |       5 |       800 |
|  3 |       1 |     10000 |
|  4 |       2 |      5000 |
|  5 |       2 |      5000 |
|  6 |       2 |      5000 |
|  7 |       2 |      4000 |
|  8 |       2 |      4400 |
|  9 |       3 |      1400 |
| 10 |       3 |      1300 |
| 11 |       4 |      1000 |
| 12 |       4 |       950 |
| 13 |       2 |      2800 |
+----+---------+-----------+
13 rows in set (0.00 sec)
```

E forniamo esempi di query in cui compaiono "funzioni di aggregazione", tali funzioni si chiamano così perché servono ad elaborare i dati di un'intera colonna(o di una parte degli elementi della colonna se utilizziamo WHERE per selezionare solo alcune righe).

Tali funzioni sono:

1. COUNT: conta il numero di elementi della colonna specificata
2. MIN: restituisce l'elemento minore, tra quelli della colonna
3. MAX: restituisce l'elemento maggiore, tra quelli della colonna
4. SUM: restituisce la somma degli elementi della colonna
5. AVG(average=media): restituisce la media degli elementi della colonna

La seguente query ritorna il **numero dei Dipendenti**, ovvero degli elementi della colonna id

```
mysql> select count(id) from Dipendenti;
+-----------+
| count(id) |
+-----------+
|        13 |
+-----------+
1 row in set (0.00 sec)
```

Avrei potuto utilizzare anche la seguente query

mysql> select **count(*)** from Dipendenti;

```
+----------+
| count(*) |
+----------+
|       13 |
+----------+
1 row in set (0.00 sec)
```

infatti **count(*) ritorna il numero di righe della tabella**

Come faccio a ricavare qual'è lo **stipendio minore**? (La query è simile per lo stipendio maggiore con **max** al posto di min)

```
mysql> select min(stipendio) from Dipendenti;
+----------------+
| min(stipendio) |
+----------------+
|            800 |
+----------------+
1 row in set (0.00 sec)
```

E se voglio effettuare la **somma degli stipendi** di tutti i dipendenti?
Tutte le volte che si incontra calcolare il totale di …. Si deve ricorrere a questa funzione

```
mysql> select sum(stipendio) from Dipendenti;
+----------------+
| sum(stipendio) |
+----------------+
|          42850
+----------------+
1 row in set (0.00 sec)
```

E se voglio calcolare lo **stipendio medio dei dipendenti di livello 3**?

```
mysql> select avg(stipendio) from Dipendenti where livello=3;
+----------------+
| avg(stipendio) |
+----------------+
|      1300.0000 |
+----------------+
```

ORDINAMENTO

E se voglio ordinare le righe della tabella in base al valore del campo stipendio?

Uso la clausola **ORDER BY [ASC|DESC]** dove ASC specifica che voglio che le righe siano ordinate in ordine crescente di stipendio(dal più piccolo al più grande), DESC invece specifica che voglio che le righe siano ordinate in ordine decrescente di stipendio(dal più grande al più piccolo). Se non specifico DESC, gli elementi vengono visualizzati in ordine CRESCENTE di DEFAULT

```
mysql> select * from Dipendenti order by(stipendio) desc;
+----+---------+-----------+
```

| id | livello | stipendio |
+----+---------+-----------+
3	1	10000
4	2	5000
5	2	5000
6	2	5000
8	2	4400
7	2	4000
13	2	2800
9	3	1400
10	3	1300
1	3	1200
11	4	1000
12	4	950
2	5	800
+----+---------+-----------+
13 rows in set (0.00 sec)

Diciamo che voglia conoscere qual è lo **stipendio medio di un dipendente di livello 2**...come faccio?
No, non serve la calcolatrice…☺ basta utilizzare la clausola GROUP BY e la funzione AVG già vista.

GROUP BY <Attributo1>,…<Attributo N> [HAVING <CondizioneGruppo>]

serve ad applicare le funzioni di aggregazione appena viste a gruppi di elementi invece che a tutti gli elementi di una colonna. I gruppi sono costituiti dagli elementi che hanno lo stesso valore per l'attributo specificato nel nostro caso tale attributo sarà livello.

mysql> select livello,avg(stipendio) from Dipendenti **group by livello**;
+---------+----------------+
| livello | avg(stipendio) |
+---------+----------------+
1	10000.0000
2	4366.6667
3	1300.0000
4	975.0000
5	800.0000
+---------+----------------+
5 rows in set (0.00 sec)

Se vogliamo sapere **quanti dipendenti vi sono per ogni livello** utilizziamo la seguente query:

mysql> select livello,count(livello) from Dipendenti **group by livello**

```
+---------+-----------------+
| livello | count(livello) |
+---------+-----------------+
|       1 |               1 |
|       2 |               6 |
|       3 |               3 |
|       4 |               2 |
|       5 |               1 |
+---------+-----------------+
5 rows in set (0.00 sec)
```

Possiamo effettuare entrambe le queries contemporaneamente:

mysql> select livello, avg(stipendio), count(livello) from Dipendenti group by livello

Se vogliamo restringere la query ai soli livelli superiori al 3 aggiungiamo anche la clausola **HAVING**

mysql> select livello,avg(stipendio),count(livello) from Dipendenti group by livello **having livello>3;**

Ecco il risultato:

```
+---------+---------------+----------------+
| livello | avg(stipendio) | count(livello) |
+---------+---------------+----------------+
|       4 |      975.0000 |              2 |
|       5 |      800.0000 |              1 |
+---------+---------------+----------------+
2 rows in set (0.00 sec)
```

La clausola HAVING permette di effettuare controlli sul risultato di funzioni di aggregazion, cosa non permessa dalla clausola WHERE(non è possibile scrivere ad esempio where max(stipendio)>10 è necessario utilizzare having).

CHIAVI ESTERNE E GIUNZIONE DI DUE TABELLE

Creiamo le seguenti tabelle:

Tabella Agenti:

```
mysql> create table Agenti(
    -> codag int(2) not null auto_increment,
    -> nomeag varchar(10) not null,
    -> tel int(10),
    -> primary key(codag));
Query OK, 0 rows affected (0.09 sec)
```

Tabella Clienti:

```
mysql> create table Clienti(
    -> codcli int(3) not null auto_increment,
    -> nomecli varchar(10) not null,
    -> via varchar(20),
    -> citta varchar(10),
    -> numcivico int(2),
    -> primary key(codcli),
    -> cod_agente int(2) not null,
    -> foreign key(cod_agente) references Agenti(codag));
Query OK, 0 rows affected (0.14 sec)
```

Dopo qualche inserimento:

```
Es. mysql> insert into Agenti
     ➔ values(NULL,"Francesca",0234333);
Query OK, 1 row affected (0.02 sec)
```

avremo la seguente tabella:

```
mysql> select * from Agenti;
+-------+-----------+-------------+
| codag | nomeag    | tel         |
+-------+-----------+-------------+
|     1 | Mario     |  86523443   |
|     2 | Paolo     | 865234333   |
|     3 | Vincenzo  |   6234333   |
|     4 | Francesca |    234333   |
+-------+-----------+-------------+
4 rows in set (0.00 sec)
```

Ora procediamo a fare la stessa cosa per la tabella Clienti, facendo attenzione ad inserire il valore corretto nel campo chiave cod_agente che è la chiave esterna che lega la tabella Clienti con quella degli Agenti(relazione 1Agente → N Clienti).

```
mysql> select * from Clienti;
+--------+----------+-----------------------+---------+-----------+---------------+
| codcli | nomecli  | via                   | citta   | numcivico | cod_agente |
+--------+----------+-----------------------+---------+-----------+---------------+
|      1 | Bianchi  | Corso Risorgimento | Isernia |        23 |             2 |
|      2 | Rossi    | Corso Garibaldi    | Isernia |        43 |             2 |
|      3 | Esposito | Corso Garibaldi    | Isernia |        11 |             2 |
|      4 | Esposito | Giovanni XXIII     | Isernia |        11 |             1 |
|      5 | Zullo    | Contrada le Piane  | Isernia |        11 |             3 |
|      6 | Iannone  | Cavour             | Roma    |        11 |             3 |
|      7 | Maglioli | da qualche parte   | Isernia |        16 |             4 |
+--------+----------+-----------------------+---------+-----------+---------------+
7 rows in set (0.00 sec)
```

Il motivo per cui esistono le chiavi esterne è quello di mettere in relazione dati di tabelle diverse.
Vogliamo visualizzare la relazione tra clienti e agenti espressa attraverso le chiavi esterne, **visualizzando i nomi dei clienti e degli agenti associati.**

Ricordiamo che ogni cliente è associato ad un agente attraverso la chiave esterna cod_agente.

Ciò viene realizzato utilizzato un operatore detto di "**congiunzione**" (nel senso che si congiungono i dati delle due tabelle attraverso le chiavi esterne).
Vediamo però direttamente in SQL quali sono le istruzioni da utilizzare.
Abbiamo già visto in precedenza l'istruzione JOIN per effettuare il prodotto cartesiano, tuttavia questa operazione non è molto utile.
Noi vogliamo che vengano associati i clienti con i propri agenti, non con tutti gli agenti!
Siamo interessati ad associare solo le righe per cui il valore di cod_agente e codag risultino coincidenti!.

Questa operazione viene effettuata attraverso l' **INNER JOIN**

La sintassi è la seguente <TABELLA1> **INNER JOIN** <TABELLA2> **ON** <Condizione>

La condizione specifica quali sono gli attributi che devono coincidere.

La query diventa dunque:

```
mysql> select nomecli,nomeag from clienti inner join agenti on clienti.cod_agente=agenti.codag;
+----------+-----------+
| nomecli  | nomeag    |
+----------+-----------+
| Esposito | Mario     |
| Bianchi  | Paolo     |
| Rossi    | Paolo     |
| Esposito | Paolo     |
| Zullo    | Vincenzo  |
| Iannone  | Vincenzo  |
| Maglioli | Francesca |
+----------+-----------+
7 rows in set (0.06 sec)
```

A questo punto possiamo notare che per effettuare un'operazione simile è possibile utilizzare solo il comando select e la clausola where, la seguente query:

```
mysql> select nomecli,nomeag from clienti,agenti where clienti.cod_agente=agenti.codag;
+----------+-----------+
| nomecli  | nomeag    |
+----------+-----------+
| Esposito | Mario     |
| Bianchi  | Paolo     |
| Rossi    | Paolo     |
| Esposito | Paolo     |
| Zullo    | Vincenzo  |
| Iannone  | Vincenzo  |
| Maglioli | Francesca |
+----------+-----------+
7 rows in set (0.00 sec)
```

produce infatti gli stessi identici risultati!
Abbiamo già visto d'altra parte che se nella select specifichiamo dopo il from due tabelle, difatto effettuiamo il prodotto cartesiano delle due tabelle e sulla tabella che ci viene restituita selezioniamo alcune combinazioni in base alla clausola where!

L'operazione di giunzione consiste pertanto nella sequenza di operazioni:

1. **PRODOTTO CARTESIANO(CLIENTI,AGENTI)**
2. **RESTRIZIONE(=WHERE)**

Tuttavia l'esempio appena fatto non è del tutto completo in quanto non abbiamo visto cosa accade quando nelle tabelle vi sono elementi non correlati. Siccome abbiamo specificato il vincolo di obbligatorietà per la chiave esterna, questa non potrà assumere il valore NULL.

Creiamo pertanto una nuova tabella:

```
mysql> create table Clienti2(
    -> codcli int(3) not null auto_increment,
    -> nomecli varchar(10) not null,
    -> via varchar(20),
    -> citta varchar(10),
    -> numcivico int(2),
    -> cod_agente int(2),
    -> primary key(codcli),
    -> foreign key(cod_agente) references Agenti(codag));
Query OK, 0 rows affected (0.24 sec)
```

In cui **non c'è più il vincolo sulla chiave esterna** e **copiamo al suo interno il contenuto della tabella Clienti** in questo modo:

```
mysql> insert into Clienti2(select * from Clienti);
Query OK, 7 rows affected (0.48 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

```
mysql> select * from Clienti2;
```

```
+--------+----------+-----------------------+---------+-----------+---------------+
| codcli | nomecli  | via                   | citta   | numcivico | cod_agente  |
+-------+----------+-----------------------+---------+-----------+---------------+
|      1 | Bianchi     | Corso Risorgimento | Isernia |          23 |               2 |
|      2 | Rossi       | Corso Garibaldi       | Isernia |          43 |               2 |
|      3 | Esposito    | Corso Garibaldi       | Isernia |          11 |               2 |
|      4 | Esposito    | Giovanni XXIII        | Isernia |          11 |               1 |
|      5 | Zullo       | Contrada le Piane     | Isernia |          11 |               3 |
|      6 | Iannone     | Cavour                | Roma    |          11 |               3 |
|      7 | Maglioli    | da qualche parte      | Isernia |          16 |               4 |
+--------+----------+-----------------------+---------+-----------+---------------+
7 rows in set (0.00 sec)
```

Adesso inseriamo una nuova riga, ovvero un nuovo cliente che non è associato a nessun agente

```
mysql> insert into Clienti2
    -> values(NULL,"Di Gennaro","vattelapesca","Somewhere",12,NULL);
Query OK, 1 row affected (0.44 sec)

mysql> select * from Clienti2;
+--------+-------------+-------------------------+--------------+-------------+-------------+
| codcli   | nomecli     | via                     | citta        | numcivico  | cod_agente |
+--------+-------------+-------------------------+--------------+-------------+-------------+
|      1 | Bianchi     | Corso Risorgimento      | Isernia      |          23 |               2 |
|      2 | Rossi       | Corso Garibaldi         | Isernia      |          43 |               2 |
|      3 | Esposito    | Corso Garibaldi         | Isernia      |          11 |               2 |
|      4 | Esposito    | Giovanni XXIII          | Isernia      |          11 |               1 |
|      5 | Zullo       | Contrada le Piane       | Isernia      |          11 |               3 |
|      6 | Iannone     | Cavour                  | Roma         |          11 |               3 |
|      7 | Maglioli    | da qualche parte        | Isernia      |          16 |               4 |
|      8 | Di Gennaro | vattelapesca             | Somewhere    |          12 |        NULL |
+--------+-------------+-------------------------+--------------+-------------+-------------+
8 rows in set (0.00 sec)
```

Ed eseguiamo la medesima query di prima:

```
mysql> select nomecli,nomeag from clienti2 inner join agenti on clienti2.cod_agente=agenti.codag;
+----------+-----------+
| nomecli  | nomeag    |
+----------+-----------+
| Esposito | Mario     |
| Bianchi  | Paolo     |
| Rossi    | Paolo     |
| Esposito | Paolo     |
| Zullo    | Vincenzo  |
| Iannone  | Vincenzo  |
| Maglioli | Francesca |
+----------+-----------+
7 rows in set (0.48 sec)
```

Ottenendo il medesimo risultato, come si può vedere la riga appena aggiunta non associata ad alcun elemento della tabella agenti non viene visualizzata!

Adesso esaminiamo il caso contrario, ovvero quello in cui è qualche agente a non essere associato a nessun cliente. E' sufficiente aggiungere un nuovo agente nella tabella Agenti.

```
mysql> insert into Agenti
    -> values(NULL,"Giacomo",89909);
Query OK, 1 row affected (0.45 sec)

mysql> select * from Agenti;
+-------+-----------+--------------+
| codag | nomeag    | tel          |
+-------+-----------+--------------+
|     1 | Mario     |  86523443    |
|     2 | Paolo     | 865234333    |
|     3 | Vincenzo  |   6234333    |
|     4 | Francesca |    234333    |
|     5 | Giacomo   |     89909    |
+-------+-----------+--------------+
5 rows in set (0.00 sec)
```

In questo caso Giacomo non è associate a nessun cliente, in quanto il suo identificativo(codag) 5 non figura nella colonna cod_agente della tabella clienti.

Ripetiamo la query di prima:

```
mysql> select nomecli,nomeag from clienti2 inner join agenti on clienti2.cod_agente=agenti.codag;
+----------+-----------+
| nomecli  | nomeag    |
+----------+-----------+
| Esposito | Mario     |
| Bianchi  | Paolo     |
| Rossi    | Paolo     |
| Esposito | Paolo     |
| Zullo    | Vincenzo  |
| Iannone  | Vincenzo  |
| Maglioli | Francesca |
+----------+-----------+
7 rows in set (0.06 sec)
```

Ancora una volta, non notiamo alcuna differenza rispetto ai risultati precedenti.

In conclusione possiamo osservare che gli elementi(clienti o agenti) delle due tabelle non associati tra loro non vengono mostrati
A tutti gli effetti l'inner join equivale alla seguente query

mysql> **select nomecli,nomeag from clienti2,agenti where clienti2.cod_agente=agenti.codag;**

dove dopo aver effettuato il prodotto cartesiano delle righe di Clienti2 e di Agenti(FROM clienti2,agenti) vengono selezionate(con la clausola WHERE) solo le righe per cui i valori degli attributi cod_agente e codag coincidono!

Affinché ciò sia possibile è necessario che il cliente sia associato ad un agente e viceversa.

Ora diciamo invece di voler **visualizzare tutti i clienti(associati e non)** , mostrando nel contempo gli agenti associati per i clienti che ne hanno uno. Come facciamo?

Risposta : utilizziamo il **LEFT JOIN**

mysql> select nomecli,nomeag from clienti2 **left join** agenti **on** clienti2.cod_agente=agenti.codag;

nomecli	nomeag
Bianchi	Paolo
Rossi	Paolo
Esposito	Paolo
Esposito	Mario
Zullo	Vincenzo
Iannone	Vincenzo
Maglioli	Francesca
Di Gennaro	**NULL**

8 rows in set (0.00 sec)

Come per magia adesso è apparso anche il cliente Di Gennaro non associato a nessun agente

Viceversa se desiderassi **visualizzare tutti gli agenti(associati e non) ,** mostrando nel contempo i clienti associati per i clienti che ne hanno uno…
…utilizzerei il **RIGHT JOIN** come mostrato di seguito:

mysql> select nomecli,nomeag from clienti2 **right join** agenti **on** clienti2.cod_agente=agenti.codag;

nomecli	nomeag
Esposito	Mario
Bianchi	Paolo
Rossi	Paolo
Esposito	Paolo
Zullo	Vincenzo
Iannone	Vincenzo
Maglioli	Francesca
NULL	**Giacomo**

8 rows in set (0.00 sec)

Ecco comparire l'agente Giacomo non associato a nessun cliente!

SOTTOINTERROGAZIONI O SUBQUERIES

Una **subquery** non è altro che una **select che compare all'interno della clausola where di un'altra select.**, ovvero si selezionano i risultati di un'altra query!
Un esempio dissiperà(si spera) ogni dubbio…

Diciamo **che vogliamo sapere quale agente ha come cliente un dipendente tra quelli elencati nella tabella Dipendenti2**

Per prima cosa visualizziamo il contenuto delle tabelle Dipendenti2, Agenti e Clienti così sappiamo cosa aspettarci come risultato…

```
mysql> select * from Dipendenti2;
+----+---------+-----------+----------+---------+
| id | livello | stipendio | nome     | cognome |
+----+---------+-----------+----------+---------+
|  1 |       2 |      2800 | Mario    | Rossi   |
|  2 |       2 |      3000 | Raffaele | Zitano  |
|  3 |       2 |      3000 | Tizio    |         |
|  4 |       2 |      1000 | Mario    | Rossi   |
+----+---------+-----------+----------+---------+
4 rows in set (0.00 sec)
```

```
mysql> select * from Clienti;
+--------+----------+------------------------+---------+-----------+----------------+
| codcli | nomecli  | via                    | citta   | numcivico | cod_agente |
+--------+----------+------------------------+---------+-----------+----------------+
|      1 | Bianchi  | Corso Risorgimento | Isernia |        23 |              2 |
|      2 | Rossi    | Corso Garibaldi    | Isernia |        43 |              2 |
|      3 | Esposito | Corso Garibaldi    | Isernia |        11 |              2 |
|      4 | Esposito | Giovanni XXIII     | Isernia |        11 |              1 |
|      5 | Zullo    | Contrada le Piane  | Isernia |        11 |              3 |
|      6 | Iannone  | Cavour             | Roma    |        11 |              3 |
|      7 | Maglioli | da qualche parte   | Isernia |        16 |              4 |
+--------+----------+------------------------+---------+---------- --+--------------+
7 rows in set (0.03 sec)
```

```
mysql> select * from Agenti;
+-------+-----------+-----------+
| codag | nomeag    | tel       |
+-------+-----------+-----------+
|     1 | Mario     |  86523443 |
|     2 | Paolo     | 865234333 |
|     3 | Vincenzo  |   6234333 |
|     4 | Francesca |    234333 |
|     5 | Giacomo   |     89909 |
+-------+-----------+-----------+
5 rows in set (0.01 sec)
```

E’ possibile fare tutto con una la seguente query:

mysql> **select nomeag from agenti,dipendenti2, clienti2**
-> where (clienti2.cod_agente=agenti.codag) AND (nomecli=Dipendenti2.cognome);
+--------+
| nomeag |
+--------+
| Paolo |
| Paolo |
+--------+
2 rows in set (0.00 sec)

In Clienti c’è un Rossi, mentre nella tabella Dipendenti2 ve ne sono 2, per cui ci aspettiamo due risultati, poichè il Rossi che compare nella tabella Clienti è associate all’agente Paolo, la query che è effettuata solo sui nomi restituirà due volte il nome dell’agente Paolo.

Il risultato è quello atteso. Se assumiamo che i nomi degli agenti siano tutti diversi(ma non abbiamo imposto il vincolo UNIQUE) possiamo aggiungere un DISTINCT nella select per eliminare le duplicazioni, ma come già detto non abbiamo in questo caso nessuna garanzia che non si tratti di persone omonime!

Ma avremmo potuto risolvere il problema anche nel seguente modo:

1. Il primo passo consiste nel ricercare una corrispondenza tra i nomi che figurano nella tabella Dipendenti2 e quelli che figurano nella tabella Clienti.

2. Una volta ottenuti tali nomi, devo cercare i nomi degli agenti associati nella tabella Agenti

Il primo passo si risolve con la seguente query:

mysql> select **clienti2.cod_agente** from clienti2,dipendenti2
-> where nomecli=Dipendenti2.cognome;

che ritorna

+-------------+
| cod_agente |
+-------------+
| 2 |
| 2 |
+-------------+
2 rows in set (0.01 sec)

Quindi utilizziamo tali risultati all’interno della seconda query, ritornando i nomi degli agenti il cui codice è compreso tra quelli ritornati da questa query.

mysql> Select **nomeag** from agenti **where agenti.codag IN** *(select clienti2.cod_agente from clienti2,dipendenti2 where nomecli=Dipendenti2.cognome)*;
+--------+
| nomeag |
+--------+
| Paolo |
+--------+
1 row in set (0.00 sec)

In questo secondo caso il risultato è più preciso poichè la subquery ritorna i codici degli agenti e la query esterna semplicemente restituisce i nomi associati ai codici degli agenti che sono compresi tra l'insieme dei risultati della subquery.
In tal caso possiamo essere certi che vi è un solo agente Paolo associato al cliente Rossi, che potrebbe essere uno dei due omonimi della tabella Dipendenti2.

Sebbene in entrambi casi la query non ci fornisca la sicurezza che l'agente Paolo è associato ad uno dei due Mario Rossi di tabella Dipendenti2 per via del fatto che utilizzando solo il cognome invece di un identificativo numerico univoco non è possibile fare di meglio , **nel secondo caso almeno ci dice che l'agente è uno solo e non due!**

Quindi possiamo vedere che oltre ad essere più comprensibile della prima, la seconda query ci permette di ottenere una risposta migliore.

In realtà si dovrebbe fare un discorso anche sull'efficienza dell'operazione di ricerca nel database, ovvero su quale query mi permette di ottenere il risultato desiderato con meno operazioni. Qui però tralasciamo almeno per il momento questo discorso. Basti sapere che **query formulate diversamente possono avere performance molto diverse.**

APPROFONDIMENTI SULLE SUBQUERIES

IN – Not IN

Nella precedente interrogazione abbiamo utilizzato il "predicato" IN , perché la subquery restituiva non un singolo valore, ma più valori(nel caso generale).

IN ci dice che il valore di un attributo appartiene ad un certo insieme(in questo caso i valori ritornati dalla subquery)

NOT IN è il contrario di IN, ovvero controlla se il valore dell'attributo non figura tra i risultati della subquery

Se rieseguiamo la precedente query sostituendo ad IN, NOT IN otterremo i nomi di tutti gli agenti tranne Paolo

mysql> Select nomeag from agenti where agenti.codag **NOT IN** (select clienti2.cod_agente from clienti2,dipendenti2 where nomecli=Dipendenti2.cognome);
+-----------+
| nomeag |
+-----------+
| Mario |
| Vincenzo |
| Francesca |
| Giacomo |
+-----------+
4 rows in set (0.00 sec)

ANY

Any significa “almeno”, in questo caso ci dice che la condizione (nella clausola WHERE) in cui compare deve essere valida almeno per un elemento tra quelli ritornati dalla subquery.

In genere si utilizza in query simili alla seguente(s1 è un attributo della tabella t1 e t2):

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

Si noti che la seguente query:

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

è equivalente alla query:

```
SELECT s1 FROM t1 WHERE s1 IN (SELECT s1 FROM t2);
```

Un esempio varrà a chiarire tutto:

Diciamo che voglia conoscere **l’id e lo stipendio** dei dipendenti di tabella Dipendenti che hanno stipendio **superiore ad almeno uno degli stipendi dei dipendenti di classe 2**.

Si può osservare che la query non effettua il confronto con un valore costante, non conosco a priori qual è lo stipendio minimo di un dipendente di livello 2, devo trovare tale valore attraverso una subquery e poi confrontarlo con tutti gli stipendi della tabella.

Esaminiamo prima gli elementi della tabella Dipendenti, lo stipendio minimo di un dipendente di livello 2 è pari a 2800, dunque la query dovrà restituire l’id di tutti i dipendenti con stipendio > 2800

mysql> select * from Dipendenti;

```
+----+---------+-----------+
| id | livello | stipendio |
+----+---------+-----------+
|  1 |       3 |      1200 |
|  2 |       5 |       800 |
|  3 |       1 |     10000 |
|  4 |       2 |      5000 |
|  5 |       2 |      5000 |
|  6 |       2 |      5000 |
|  7 |       2 |      4000 |
|  8 |       2 |      4400 |
|  9 |       3 |      1400 |
| 10 |       3 |      1300 |
| 11 |       4 |      1000 |
| 12 |       4 |       950 |
| 13 |       2 |      2800 |
+----+---------+-----------+
13 rows in set (0.39 sec)
```

La query è la seguente:

```
mysql> select id,stipendio from dipendenti where stipendio > ANY (select stipendio from
Dipendenti where livello=2);
+----+-----------+
| id | stipendio |
+----+-----------+
|  3 |     10000 |
|  4 |      5000 |
|  5 |      5000 |
|  6 |      5000 |
|  7 |      4000 |
|  8 |      4400 |
+----+-----------+
6 rows in set (0.00 sec)
```

Come possiamo constatare il risultato è proprio quello atteso.

Naturalmente in questo caso c'era un modo più semplice di effettuare la query:

```
mysql> select id,stipendio from dipendenti where stipendio > (select MIN(stipendio) from
Dipendenti where livello=2);
+----+-----------+
| id | stipendio |
+----+-----------+
|  3 |     10000 |
|  4 |      5000 |
|  5 |      5000 |
|  6 |      5000 |
|  7 |      4000 |
|  8 |      4400 |
+----+-----------+
```

6 rows in set (0.00 sec)

Ovvero mi calcolo il minimo stipendio di un dipendente di livello 2 e utilizzo questo valore per effettuare il confronto.

Come è possibile vedere vi sono diversi modi per ottenere lo stesso risultato!

ALL

Analogamente il predicato ALL indica che la condizione sarà soddisfatta per tutti i valori della subquery, se nell'esempio precedente sostituiamo ad any, ALL avremo

mysql> select id,stipendio from dipendenti where stipendio > **ALL** (select stipendio from Dipendenti where livello=2);
+----+-----------+
| id | stipendio |
+----+-----------+
| 3 | 10000 |
+----+-----------+
1 row in set (0.02 sec)

Che è equivalente in questo caso alla query:

mysql> select id,stipendio from dipendenti where stipendio > (select **MAX(stipendio)** from Dipendenti where livello=2);
+----+-----------+
| id | stipendio |
+----+-----------+
| 3 | 10000 |
+----+-----------+

Ovvero lo stipendio dovrà essere superiore a quello di tutti i dipendenti di livello 2 o in altri termini superiore allo stipendio massimo di un dipendente di livello 2.

Verificare che **NOT IN** è equivalente a **<> ALL.** (<> significa "diverso da")

EXISTS - NOT EXISTS

EXISTS indica che la subquery restituisce una tabella non vuota, NOT EXIST che al contrario la subquery non produce risultati

Esempio:

Voglio visualizzare il messaggio "Paolo lavora!" se l'agente Paolo ha qualche cliente
Per prima cosa scriviamo una query per visualizzare i clienti di Paolo:

```
mysql> select * from clienti  where cod_agente=(select codag from agenti where nomeag="Paolo");

+--------+----------+------------------------+---------+------------+-------------+
| codcli  | nomecli  | via                    | citta     | numcivico | cod_agente |
+--------+----------+------------------------+---------+------------+-------------+
|       1 | Bianchi   | Corso Risorgimento | Isernia  |            23|            2 |
|       2 | Rossi      | Corso Garibaldi       | Isernia  |            43|            2 |
|       3 | Esposito  | Corso Garibaldi       | Isernia  |            11|            2 |
+--------+----------+------------------------+---------+------------+-------------+
3 rows in set (0.01 sec)
```

Ora vediamo che cosa ritorna il predicato EXISTS, come si può vedere di seguito la select permette anche di visualizzare dei valori(che siano stringhe,numeri o effettuare operazioni). Inoltre attraverso abbiano nominato la colonna che contiene il risultato con il nome di Occupato. Parleremo dell'AS più avanti…

mysql> select **EXISTS**(select * from clienti where cod_agente=(select codag from agenti where nomeag="**Paolo**")) **AS Occupato**;

Il risultato è il seguente:

```
+-----------+
| Occupato |
+-----------+
|           1 |
+-----------+
1 row in set (0.02 sec)
```

Se ripetiamo la query per Giacomo avremo invece:

mysql> select EXISTS(select * from clienti where cod_agente=(select codag from agenti where nomeag="**Giacomo**")) as Occupato;

```
+-----------+
| Occupato |
+-----------+
|           0 |
+-----------+
1 row in set (0.00 sec)
```

Dunque l'**1 o lo 0 corrispondenti al true\false booleani** segnalano se un agente ha dei clienti oppure no, naturalmente avremmo potuto utilizzare la condizione COUNT(….) >0 o altre formulazioni della query per ottenere lo stesso risultato!.

CONSERVAZIONE DI RISULTATI PARZIALI

Concludiamo questa dispensa, con qualche accenno ad un'operazione che può risultare estremamente utile quando si effettuano query particolarmente complesse ovvero sulla conservazione dei risultati parziali o sull'utilizzo di "alias".

Se vogliamo memorizzare i risultati della query sui clienti di Paolo in una nuova tabella ClientiPaolo dobbiamo eseguire una *create* di questo tipo:

```
mysql> create table ClientiPaolo select * from clienti  where cod_agente=(select codag from agenti where nomeag="Paolo");
Query OK, 3 rows affected (0.59 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

Visualizziamo le tabelle presenti nel nostro DB:

```
mysql> show tables;
+--------------------+
| Tables_in_scuola |
+--------------------+
| agenti             |
| clienti            |
| clienti2           |
| clientipaolo       |
| dipendenti         |
| dipendenti2        |
| film               |
| hobbies            |
+--------------------+
8 rows in set (0.00 sec)
```

Come si può vedere abbiamo una nuova tabella **clientipaolo !**

Se visualizziamo il contenuto di questa tabella avremo:

```
mysql> select * from clientipaolo;
+--------+----------+------------------------+---------+-------------+-------------+
| codcli  | nomecli  | via                    | citta      | numcivico | cod_agente |
+--------+----------+------------------------+---------+-------------+-------------+
|      1     | Bianchi  | Corso Risorgimento | Isernia |              23 |              2 |
```

```
|     2    | Rossi     | Corso Garibaldi       | Isernia |              43 |              2 |
|     3    | Esposito | Corso Garibaldi       | Isernia |              11 |              2 |
+--------+----------+-----------------------+---------+-----------------+----------------+
3 rows in set (0.00 sec)
```

Per copiare il contenuto di una tabella in un'altra invece è molto comoda la seguente istruzione, già vista nella prima parte della presente dispensa:

INSERT INTO <NuovaTabella> SELECT * FROM <VecchiaTabella>;