MOD1

IL LINGUAGGIO SQL

Ing. Daniele Corti

Ver. 1.0

PREREQUISITI

- ✓ Conoscenze sullo sviluppo/progettazione di un DB: fase concettuale, logica e fisica.
- ✓ Conoscenze utilizzo software di gestione DB (mySQL).

OBIETTIVI

- ✓ Realizzare interrogazioni di un DB in SQL.

ARGOMENTI

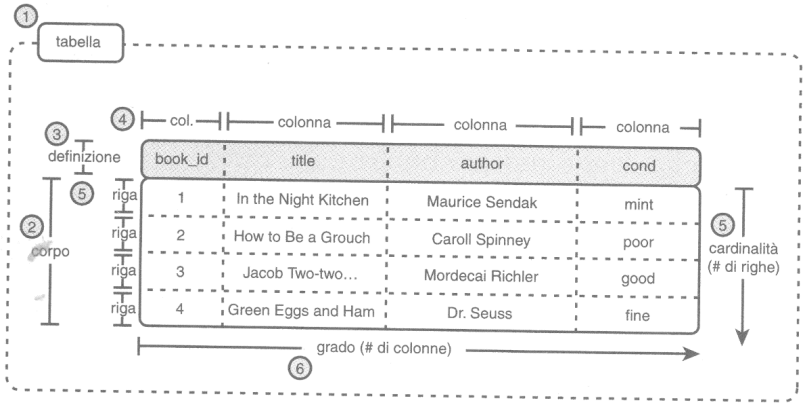
- ✓ Il software di gestione DB: mySQL.
- ✓ Introduzione – L'SQL.
- ✓ Componenti di SQL.
- ✓ Proposizioni.
- ✓ Operatori.
- ✓ Funzioni di aggregazione.
- ✓ Operazioni DDL.
- ✓ Interrogazioni con SQL.

IL LINGUAGGIO SQL

MYSQL

MySQL è un RDBMS (Relational Database Management System) ovvero un Sistema di gestione delle basi di dati basato sul modello relazionale. mySQL, composto da un Client e da un Server, è un software Open Source incorporato nella piattaforma easyPhp.

Terminologia delle tabelle mySQL



I componenti di una tabella (relazione nel modello relazionale) MySQL sono:

- Una tabella, collezione strutturata di dati, composta dai seguenti elementi:
- Un corpo, contenente righe e colonne di dati…
- La definizione della tabelle che contiene le colonne, le quali descrivono il tipo di dati presenti nelle righe.
- Ogni colonna ha un nome (per esempio book_id) ed un tipo (per esempio INT). Ciascun tipo rappresenta un insieme di valori (come i numeri interi o le lettere dell'alfabeto). Se un dato appartiene a un certo tipo, esso dovrà assumere uno dei valori previsti da quel tipo (per esempio, una colonna di tipo INT deve contenere un numero intero come 1 o 6504, ma non una stringa di caratteri, come "daniele corti", e neppure un numero decimale come 3.14).

- Ciascuna riga contiene un valore per ogni colonna dell'intestazione. I valori devono appartenere al tipo definito nella colonna alla quale corrispondono. Per esempio, la colonna book_id è del tipo SMALLINT e quindi solo valori di questo tipo possono essere memorizzati in quella posizione della riga.
- Il numero di colonne (**campi**) di una tabella (relazione) ne definisce il grado.
- Il numero di righe (record) di una tabella (relazione) ne definisce la **cardinalità**.

Tipi di tabelle in mySQL più utilizzate

- MyISAM: tipo di tabella di default, garantisce performance di velocità straordinarie e ha di fatto rimpiazzato la vecchia ISAM.
- INNODB: questo tipo di tabella supporta alcune features che MySQL prima non supportava (transazioni e foreign key).

INTRODUZIONE – L'SQL

L'SQL (Structured Query Language) è il linguaggio ormai assunto come standard per la trattazione di Data Base (DB) relazionali. La sua stesura è dovuta essenzialmente a Edgar Frank Ted Codd (1923-2003), informatico britannico, all'inizio degli anni '70. L'ANSI ha definito le specifiche (SQL ANSI-89), ma le case produttrici spesso propongono dialetti SQL non completamente conformi, sia perché non implementano alcuni costrutti, sia perché includono nuove caratteristiche.

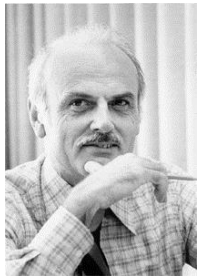


Figura 1 - E. Ted Codd

SQL è un **linguaggio non procedurale**. I linguaggi non procedurali, si distinguono dai linguaggi imperativi (come per esempio il linguaggio C, Visual Basic, etc.) perché permettono di descrivere gli obiettivi da raggiungere, astraendo dagli algoritmi che svolgono il compito effettivo.

L'SQL, il cui nome può essere tradotto in italiano come Linguaggio Strutturato di Interrogazione, è utilizzato per creare, manipolare e interrogare una base dati relazionale.
In generale, nei linguaggi per DB si distinguono le seguenti categorie semantiche:

- Primitive per la definizione dei dati: **DDL** (Data Definition Language). Serve per creare e definire nuovi DB, tabelle, campi, indici, vincoli.
- Primitive per la gestione dei dati: **DML** (Data Management Language). Serve per creare query che consentono di ordinare, filtrare, cancellare, modificare, estrarre dati dal DB.
- Primitive per il controllo del linguaggio: **DCL** (Data Control Language). Consente il controllo degli accessi per più utenti e i permessi per gli utenti autorizzati.

In questo corso si fa riferimento al linguaggio SQL per software mySQL e si approfondirà solo la parte relativa ai comandi per interrogazioni (SELECT) sui DB.

Componenti di SQL

Componenti di SQL sono: comandi, proposizioni, operatori, funzioni di aggregazione e predicati.
Qui di seguito si elencano le principali parole chiave. Successivamente si introducono altre parole chiave.
Anche in SQL si può mantenere la classificazione dei comandi in DDL e DML.

Comandi DDL

Comando	Descrizione
CREATE	Consente di creare nuove tabelle, campi e indici.
DROP	Consente di eliminare tabelle e indici dal DB.
ALTER	Consente di modificare tabelle aggiungendovi campi o modificandone la definizione.

Comandi DML

Comando	Descrizione
SELECT	Consente di richiedere i record del DB rispondenti a criteri specifici.

INSERT	Consente di caricare gruppi di dati nel DB con una sola operazione.
UPDATE	Consente di modificare i valori di particolari record e campi.
DELETE	Consente di rimuovere record da una tabella del DB.

Operazioni DDL

Il linguaggio di definizione dei dati permette di creare tabelle e indici, di modificare le tabelle aggiungendovi o rimuovendo colonne ed indici.
Nella pratica si preferisce di solito usare per questo compito altri metodi più diretti (come Access, in casa Microsoft, che grazie alla modalità visuale, semplifica grandemente il compito e riduce gli errori formali). Di seguito si indicano, per ciascuna delle funzioni, esempi di uso del SQL.

Creazione di tabelle

Creare una tabella di nome Persone, con due colonne di testo, Cognome, di 30 caratteri e Nome di 20 caratteri:

```
CREATE TABLE Persone([Cognome] TEXT (30), [Nome] TEXT (20));
```

Aggiunta e rimozione di colonne

Aggiungere alla tabella Persone una colonna Indirizzo di 50 caratteri:

```
ALTER TABLE Persone ADD COLUMN Indirizzo TEXT(50);
```

Eliminare dalla tabella Persone la colonna Nome:

```
ALTER TABLE Persone DROP COLUMN Nome;
```

Modificare la colonna Indirizzo portandola a 40 caratteri. Prima bisogna rimuoverla e successivamente aggiungere la colonna della dimensione scelta:

```
ALTER TABLE Persone DROP COLUMN Indirizzo;
ALTER TABLE Persone ADD Indirizzo TEXT(40);
```

Creazione ed eliminazione di indici

1° metodo: Creare l'indice all'atto della creazione della tabella

Si usa la parola chiave CONSTRAINT, che deve comparire all'inizio della definizione di un indice.

CONSTRAINT garantisce il mantenimento dell'integrità referenziale.

Creazione di una tabella con indice su una sola colonna (DataNascita)

```
CREATE TABLE Persone([Cognome] TEXT (30), [Nome] TEXT(20), _
[DataNascita] DATETIME CONSTRAINT IndicePersone PRIMARY);
```

Creazione di una tabella con indice su tre campi:

```
CREATE TABLE Persone([Cognome] TEXT (30), [Nome] TEXT(20), _
[DataNascita] DATETIME, CONSTRAINT IndicePersoneUNIQUE _
([Cognome], [Nome], [DataNascita]);
```

2° metodo: con il comando CREATE INDEX

Se si desidera creare un indice in un momento successivo alla creazione, si usa questa modalità.

Creazione di un indice sul campo DataNascita:

```
CREATE UNIQUE INDEX MioIndice ON Persone([DataNascita]);
```

Con la proposizione facoltativa WITH si possono aggiungere ulteriori condizioni:

Condizione	**Descrizione**
PRIMARY	Identifica la colonna primaria dell'indice
DISALLOW NULL	La colonna non può essere vuota.

DDL: CREARE E GESTIRE UNA TABELLA

Per creare una tabella si usa la seguente sintassi:

```
CREATE TABLE NomeTabella
(
Attributo1 Tipo1 Vincolo1,
Attributo2 Tipo2 Vincolo2,
……
AttributoN TipoN VincoloN,
AttributoChiaveEsterna Tipo Vincolo,
VincoloTabella
FOREIGN KEY(AttributoChiaveEsterna) REFERENCES
NomeTabellaAssociata(ChiavePrimaria)
)
```


Dove Vincolo1, Vincolo2, …, VincoloN sono vincoli per un singolo attributo, e VincoloTabella è un vincolo di ennupla (su un gruppo di attributi) o di integrità referenziale.

Esempio:

```
CREATE TABLE Dipendente
(
Matricola char(5) NOT NULL PRIMARY KEY,
Cognome char(30),
Nome char(30),
CodFisc char(16) NOT NULL,
Assunto date,
Filiale smallint,
Funzione char(15),
Livello smallint,
StipBase currency,
Via char(30),
Cap char(5),
Citta char(20),
Prov char(2),
Tel char(10),
DataNascita date,
PremioProduzione currency DEFAULT 0,
CodReparto char(2),
FOREIGN KEY(CodReparto) REFERENCES Reparto(CodReparto),
CHECK(StipBase>1000),
UNIQUE(CodFisc)
)
```

NB il vincolo CHECK non è implementato in Access.

Una volta creata una tabella, la si può successivamente modificare:

- con l'istruzione ADD si aggiunge una nuova colonna.

```
ALTER TABLE NomeTabella
ADD NomeColonna Tipo ;
```

Esempio:

```
ALTER TABLE Dipendente
ADD DataContratto date ;
```

- con l'istruzione DROP COLUMN si elimina una colonna.

```
ALTER TABLE NomeTabella
DROP COLUMN NomeColonna ;
```

- con l'istruzione MODIFY si modifica il tipo di una colonna e non il nome

```
ALTER TABLE NomeTabella
MODIFY NomeColonna NuovoTipo ;
```

- per cancellare una tabella si utilizza il comando DROP.

```
DROP TABLE NomeTabella ;
```

DML

- Per inserire i valori delle righe si usa il comando INSERT INTO.

```
INSERT INTO NomeTabella(Attr1, …, AttrN)
VALUES("val1",…, "valN") ;
```

Esempio

```
INSERT INTO Reparto(CodReparto, NomeReparto, Tel)
VALUES("A1","Produzione",”02543627) ;
```

- Per modificare una o più righe di una tabella si usa il comando UPDATE.

```
UPDATE NomeTabella
SET Attributo1 = valore1, Attributo2 = valore2
WHERE condizione ;
```

Esempio

```
UPDATE Reparto
SET Tel="02123456"
WHERE CodReparto="A1" ;
```

Esempio

```
UPDATE Dipendente
SET StipBase= StipBase*1.2
WHERE CodReparto="A1" ;
```

- Per cancellare una o più righe di una tabella si usa il comando DELETE.

```
DELETE FROM NomeTabella
WHERE condizione ;
```

Esempio

```
DELETE FROM Dipendente
WHERE Matricola="a12" ;
```

- Per estrarre dei dati dal DB si usa il comando SELECT con le seguenti clausole:

SELECT *lista attributi o espressioni*

FROM *lista tabelle*

WHERE *condizioni semplici*

GROUP BY *lista attributi di raggruppamento*

HAVING *condizioni aggregate*

ORDER BY *lista attributi di ordinamento*

INTERROGAZIONI SQL

Proposizioni

Le proposizioni sono condizioni che consentono di definire i dati che si desidera selezionare o gestire.

Proposizione	Descrizione
FROM	Consente di specificare il nome delle tabelle di cui si desidera selezionare i record.
WHERE	Consente di specificare i criteri a cui debbono corrispondere i record da selezionare.
GROUP BY	Consente di suddividere in gruppi i record selezionati.
HAVING	Consente di specificare la condizione a cui deve corrispondere ciascun gruppo.
ORDER BY	Consente di ordinare i record selezionati in base all'ordine specificato.

Funzioni di aggregazione

Le funzioni di aggregazione vengono utilizzate in una proposizione SELECT ed applicate a più gruppi di record per restituire un valore singolo riferito ad un gruppo di record.

Funz. di Aggregazione	Descrizione
AVG	Consente di ottenere la media dei valori di un particolare campo

COUNT	Restituisce il numero dei record selezionati
SUM	Restituisce la somma di tutti i valori di un particolare campo
MAX	Restituisce il valore massimo del campo specificato
MIN	Restituisce il valore minimo del campo specificato

Gli identificatori degli oggetti possono contenere spazi. In questo caso è obbligatorio racchiudere l'identificatore tra parentesi quadre[ ]; negli altri casi è facoltativo.

COMANDI PER INTERROGAZIONI CON SQL

I comandi d'interrogazione – QUERY – consentono di effettuare operazioni di ricerca sui dati contenuti nelle tabelle del database, impostando le condizioni che tali dati devono soddisfare.

COMANDO SELECT

SELECT è il comando per il reperimento/estrazione dei dati da un DB.

La SELECT opera sulle tabelle e restituisce come risultato una tabella.

Le interrogazioni, query, in SQL sono espresse in modo dichiarativo: si dice cosa si vuole come risultato, senza specificare la maniera per ottenerlo.

Un'interrogazione in SQL ha sempre una forma del tipo:

Struttura	Descrizione
SELECT <colonne>	seleziona <colonne>
FROM <tabelle>	da <tabelle>
WHERE <condizioni> ;	con <condizioni> ;

Esempio:

```
SELECT *
FROM utenti
WHERE idUtente = 1 ;
```

La precedente query consente di estrarre tutti campi (parola chiave *) dalla tabella utenti per i soli record il cui campo idUtente corrisponda a 1.

Una query può coinvolgere anche più di una tabella.

Il risultato di una query è una tabella con le seguenti caratteristiche:

- Verranno prese in considerazione solo le tabelle indicate in <tabelle>.
- Il risultato conterrà solo le colonne specificate nella lista <colonne>; tali colonne saranno tutte prese dalle tabelle indicate.
- Le tuple (righe, record) scelte saranno quelle che verificano le condizioni specificate in <condizioni>. A tal proposito, può succedere che nessuna delle righe verifichi le condizioni poste, e in tal caso la tabella sarà vuota.

Si noti che una query non modifica le tabelle su cui lavora: una query può quindi essere vista come una specie di filtro (filtraggio per riga e per colonna).

Il comando SELECT consente di estrarre le colonne, e la clausola WHERE di recuperare le righe.

SINTASSI DEL COMANDO SELECT

```
<Comando-select> ::=
SELECT [ALL | DISTINCT] <lista-di-selezione>
<espressione-di-tabella>
[<clausola-di-ordinamento>]

dove <espressione-di-tabella> è così definita:
<espressione-di-tabella> ::=
<clausola-FROM>
[<clausola-WHERE>]
[<clausola-GROUP-BY>]
[<clausola-HAVING>]
```

Come risulta dalla sintassi, nel comando è obbligatorio specificare, oltre alla <lista di selezione>, la sola clausola FROM. Le specifiche d'interrogazione – ALL e DISTINCT - la cui funzione sarà spiegata più avanti - e le altre clausole sono opzionali (pertanto nella sintassi sono riportate tra parentesi quadra). ALL e DISTINCT sono tra loro alternative (nella sintassi sono quindi separate da |); in mancanza di entrambe viene assunta valida per default la specifica ALL.

La clausola FROM indica la tabella o le tabelle su cui eseguire la selezione:

```
<clausola-FROM> ::= FROM <nome-tabella> [ { ,<nome-tabella> } … ]
```

Deve essere specificato il nome di almeno una tabella; successivamente, separati da virgole, possono essere indicati i nomi di altre tabelle.

LA LISTA DI SELEZIONE

La forma più semplice del comando di selezione è quella in cui la lista di selezione è costituita da uno o più nomi di colonne della stessa tabella, ovvero:

<lista_di_selezione> ::= <nome_colonna> [{ ,<nome_colonna>) } …]

Il risultato del comando è una tabella che contiene le colonne indicate della tabella specificata nella clausola FROM.

Esempio

```
modelli(codModello, nomeModello)

SELECT codModello, nomeModello
FROM modelli
```

RIDENOMINARE I NOMI DELLE COLONNE

```
SELECT cod_Modello AS Codice, nome_Modello AS Nome
FROM modelli
```

SELEZIONE DI TUTTE LE COLONNE

```
SELECT *
FROM Modelli
```

ELIMINARE I VALORI DUPLICATI

```
veicoli(codVeicolo, nome, targa, modello)
```

Supponiamo di voler visualizzare tutti i modelli che compaiono nella tabella veicoli:

```
SELECT modello
FROM veicoli
```

Il risultato è una tabella con tante righe quante sono le righe di veicoli, alcune delle quali contengono lo stesso modello.

Per eliminare i valori duplicati esiste la specifica d'interrogazione DISTINCT da anteporre al campo contenente i duplicati:

```
SELECT DISTINCT modello
FROM veicoli
```

Il risultato è una tabella in cui nessun valore di modello viene mai ripetuto.

NB La clausola DISTINCT può apparire, all'interno del comando SELECT, al più una volta.

CLAUSOLA WHERE

La clausola WHERE permette di specificare delle condizioni nella selezione. La sua sintassi è la seguente:

```
<clausola_WHERE> ::= WHERE <condizione>
```

Data una tabella e una condizione logica definita sui suoi attributi, la selezione restituisce una tabella con gli stessi attributi di quella di partenza, ma con le sole righe che soddisfano la condizione.

La clausola WHERE non è l'unico modo per operare delle restrizioni sulle righe di una tabella, poiché, per esempio, anche la clausola DISTINCT che abbiamo già esaminato opera in tal senso; tuttavia è certamente il modo più flessibile e più frequentemente utilizzato.

Le condizioni della clausola WHERE sono specificate mediante gli operatori di confronto, i connettori logici e gli operatori BETWEEN, IN, LIKE, IS NULL, che studieremo nei prossimi paragrafi.

Operatori

SQL prevede due tipi di operatori: logici e di confronto.

Gli operatori di confronto permettono di comparare due espressioni. Casi tipici sono quelli in cui i valori contenuti nelle colonne di una tabella vengono confrontati con delle costanti, con i valori di altre colonne della stessa riga della tabella o in generale con delle espressioni.

Operatore di confronto	Descrizione
<	Minore di
<=	Minore o uguale a

>	Maggiore
>=	Maggiore o uguale
=	Uguale
<>	Diverso
BETWEEN	Consente di specificare un intervallo di valori.
LIKE	Utilizzato per criteri di ricerca.
IN	Consente di specificare record in DB.

Tipicamente gli operatori si usano all'interno di una proposizione WHERE.

Esempi

```
SELECT *
FROM veicoli
WHERE cilindrata>1000
SELECT *
FROM Veicoli
WHERE Targa = 'AX23476'

SELECT *
FROM veicoli
WHERE cilindrata > cavalli_Fiscali * 20
```

In generale, i valori confrontati dagli operatori di confronto possono essere anche costanti ed espressioni calcolate sui valori di più colonne. Le espressioni aritmetiche possono contenere gli usuali operatori + , - , * , / per effettuare addizioni, sottrazioni, moltiplicazioni e divisioni. Anche i valori alfanumerici possono essere utilizzati nelle espressioni; in questo caso verranno impiegati operatori e funzioni che operano su sequenze di caratteri, quale per esempio l'operatore di concatenazione.

Gli operatori logici (o connettori logici) consentono di collegare due espressioni logiche o di negarne una:

- AND
- OR
- NOT

Esempi:

```
SELECT *
FROM veicoli
WHERE codCombustibile = '01' AND cilindrata > 1000
```

Il comando seleziona le righe in cui il valore di codCombustibile è uguale a 01 e contemporaneamente il valore di cilindrata a è maggiore di 1000.

```
SELECT *
FROM veicoli
WHERE codModello = '001' OR codModello = '004'
```

Il comando seleziona le righe in cui il valore di codModello è uguale a 001 e le righe in cui il valore di codModello è uguale a 004.

```
SELECT *
FROM veicoli
WHERE NOT codCategoria = '01'
```

In questo caso lo stesso risultato poteva essere ottenuto scrivendo

```
WHERE codCategoria <> '01' ;
```

In altre circostanze le alternative non sono così immediate.
Tramite l'operatore NOT si realizza l'operazione di complemento di una tabella rispetto a una condizione, definita nel Capitolo 3, dedicato all'algebra relazionale. L'ordine di valutazione degli operatori prevede che vengano prima applicati gli operatori NOT, poi gli AND, infine gli OR. Questo significa per esempio che nella condizione multipla

```
NOT codCategoria = '01' AND Cilindrata > 1500
```

l'operatore NOT è riferito solo alla condizione `codCategoria = '01'`.

Nella condizione multipla

```
codCategoria = '03' OR codCategoria = '01' AND cilindrata > 1500
```

viene prima eseguito l'AND tra `codCategoria = '01'` e `cilindrata > 1500` e successivamente il risultato viene combinato in OR con `codCategoria = '03'`.

Per definire un ordine diverso di valutazione degli operatori nelle condizioni multiple possono essere usate le parentesi tonde:

```
NOT (codCategoria = '01' AND cilindrata > 1500)
```

In questo caso il NOT è riferito a tutta la condizione contenuta nella parentesi.

Nel seguente esempio:

```
(codCategoria = '01' OR codCategoria = '03') AND codCombustibile = '01'
```

viene eseguito prima l'OR tra le prime due condizioni, e successivamente il risultato viene combinato in AND con la terza condizione.

Le coppie di parentesi ( ) possono essere annidate una nell'altra, come nel seguente esempio, in cui vengono restituiti tutti veicoli esclusi quelli il cui codice combustibile è 01 o 02 con cilindrata minore di 1500:

```
NOT (
      (codCombustibile = '01' OR codCombustibile = '02')
      AND cilindrata < 1500
)
```

OPERATORE BETWEEN

L'operatore BETWEEN verifica se un argomento è compreso in un intervallo di valori; esso utilizza la seguente sintassi:

```
<operando> BETWEEN <operando> AND <operando>;
```

Normalmente il primo operando è un nome di colonna. Nel comando successivo si ottengono i veicoli la cui cilindrata è compresa tra 1500 e 1800:

```
SELECT *
FROM veicoli
WHERE cilindrata BETWEEN 1500 AND 1800
```

La precedente è una notazione equivalente ma più sintetica e intuitiva di:

```
SELECT *
FROM veicoli
WHERE cilindrata>=1500 AND cilindrata<=1800
```

I programmatori SQL, comunque, tendono a usare quest'ultima soluzione, sia per l'abitudine a lavorare con gli operatori di confronto, sia perché condizioni logiche complesse richiedono numerosi operatori e parentesi annidate, per cui può risultare più chiaro e immediato l'impiego di operatori

omogenei. È da notare, inoltre, che l'uso dell'operatore BETWEEN richiede che gli operandi siano espressi nell'ordine corretto, mentre questo non vale per le condizioni espresse tramite operatori di confronto, che producono lo stesso risultato a prescindere dall'ordine in cui vengono utilizzate.

OPERATORE IN

L'operatore IN verifica se un operando è contenuto in una sequenza di valori.

Nell'esempio selezioniamo le righe della tabella Veicoli in cui Targa è uguale ad A123456X, a D238765W o a XCH56GJK:

```
SELECT *
FROM veicoli
WHERE targa IN ('A123456Xt,'D238765W','XCH56GJK')
```

Naturalmente avremmo anche potuto scrivere:

```
SELECT *
FROM veicoli
WHERE targa='A123456X' OR targa='D238765W' OR targa='XCH56GJK'
```

Il primo comando è più sintetico e chiaro, in particolare qualora il numero di valori sia elevato; IN si ricollega esplicitamente agli operatori insiemistici.

Vedremo inoltre in seguito che l'utilizzo dell'operatore IN diviene essenziale qualora la lista di valori sui quali eseguire il confronto rappresenti il risultato di un'altra operazione di selezione.

L'OPERATORE LIKE

L'operatore LIKE verifica se una stringa di caratteri corrisponde a un determinato formato. Per definire il formato sono utilizzabili i seguenti caratteri jolly:

_ indica un singolo carattere qualsiasi;

% indica una sequenza di caratteri qualsiasi.

Nel comando

```
SELECT *
FROM veicoli
WHERE targa LIKE 'A__7___'
```

si richiedono le righe della tabella Veicoli dove la Targa è composta da otto caratteri di cui il primo è A, il secondo e il terzo possono essere qualsiasi, il quarto deve essere 7 e gli ultimi quattro di nuovo qualsiasi.
I caratteri maiuscoli vengono distinti dai corrispondenti minuscoli, a meno di non aver utilizzato delle opzioni particolari nella definizione dello schema logico; per-tanto il formato 'A__7___' risulta diverso da 'a__7___' .

Nell'esempio successivo la stringa in Targa deve iniziare con il carattere C seguito da una sequenza di caratteri qualsiasi:

```
SELECT *
FROM veicoli
WHERE targa LIKE 'C%'
```

Anche se il contenuto del database RegistroAutomobilistico non le prevede per omogeneità, se fossero state presenti Targhe con il primo carattere uguale a C di lunghezza maggiore o minore a otto caratteri, come C0333398V, le righe corrispondenti sarebbero state restituite.
Come abbiamo visto i caratteri _ e % vengono utilizzati come caratteri speciali. Si possono tuttavia presentare situazioni in cui gli stessi caratteri _ e % facciano parte del contenuto stesso delle colonne e debbano essere utilizzati nei confronti. Si consideri il caso in cui tali caratteri siano contenuti nella colonna NomeModello della tabella Modelli e vogliamo selezionare tutte le righe di tale tabella che corrispondono a un formato che li contiene. In tal caso è necessario operare nel modo seguente; supponiamo che sia il carattere _ da ricercare:

- individuare un particolare carattere, detto carattere di escape, che non sia ammissibile nella colonna su cui si sta impostando la condizione LIKE;
- nella specifica del formato da confrontare far precedere _, quando tale carattere debba essere utilizzato come carattere di confronto e non come carattere speciale, dal carattere di escape;
- specificare il carattere di escape individuato facendolo precedere dalla parola chiave ESCAPE, subito dopo il formato da confrontare.

Per esempio, per trovare tutti i modelli il cui nome inizia con C_:

```
SELECT *
FROM modelli
```



```
WHERE nomeModello LIKE 'C#_%' ESCAPE '#'
```

In questo caso abbiamo scelto # come carattere di escape; il carattere è utilizzato come carattere di confronto, essendo preceduto dal carattere di escape, mentre il carattere % mantiene il suo ruolo di carattere speciale. La procedura descritta può naturalmente essere applicata anche al carattere %.

OPERATORE IS NULL

L'operatore IS NULL verifica se il contenuto di un operando è NULL. Quest'ultimo è uno stato che indica che l'elemento della tabella nella specifica colonna non con-tiene alcun valore. (Si tenga presente che zero è un valore così come una stringa vuota.).
Per esempio, il comando

```
SELECT *
FROM veicoli
WHERE cilindrata IS NULL
```

restituisce le linee in cui il valore della colonna Cilindrata è NULL, cioè non contiene alcun valore

Gli ultimi operatori descritti sono presenti anche in forma negativa, ovvero nella forma:

```
NOT BETWEEN
NOT IN
NOT LIKE
NOT NULL
```

Nell'esempio

```
SELECT *
FROM veicoli
WHERE cilindrata NOT BETWEEN 1500 AND 1800
```

vengono perciò restituite le righe in cui la cilindrata non è compresa tra 1500 e 1800, o in altri termini è minore di 1500 o maggiore di 1800.
Analogamente, l'utilizzo dell'operatore NOT IN è utile per escludere un insieme di valori che non qualificano un'operazione di selezione.

CALCOLO DI ESPRESSIONI

La lista di selezione di un comando SELECT può contenere non solo nomi di colonne, come negli esempi sinora visti, ma anche espressioni calcolate sui valori di una o più colonne. Per esempio:

```
SELECT targa, velocità * 3.6 AS VEL-IN-MPS
FROM veicoli
```

Le colonne restituite riporteranno per ogni veicolo la targa e la velocità in metri all'ora, calcolata moltiplicando la velocità per 3.6. Questa colonna viene denominata `VEL-IN-MPS`. Il risultato sarà quello della prima tabella di pagina seguente.
Le espressioni possono essere utilizzate all'interno di un comando SELECT nella lista di selezione e nella clausola WHERE, come abbiamo visto, dove possa essere inserito il riferimento a una colonna.

FUNZIONI DI GRUPPO

Le espressioni finora considerate operano sulle singole righe di una tabella, restituendo sempre una tabella. Esistono però delle funzioni che consentono di calcolare espressioni su insiemi di righe e che, a differenza delle precedenti, restituiscono un caso molto particolare di tabella, ovvero un singolo valore scalare. Le funzioni disponibili sono le seguenti:

- MAX per la determinazione del valore massimo
- MIN per la determinazione del valore minimo
- SUM per il calcolo della somma di valori
- AVG per il calcolo della media di valori
- COUNT per il conteggio di valori

MAX e MIN

Le funzioni MAX e MIN calcolano rispettivamente il maggiore e il minore dei valori di una colonna.
Per esempio:

```
SELECT MAX(cilindrata)
FROM veicoli
```

La funzione MAX applicata alla colonna cilindrata prende in considerazione i valori di questa colonna e ne restituisce il più alto.

Le funzioni che operano su insiemi di righe evidenziano da una parte la potenza del linguaggio e dall'altra le differenze con i linguaggi procedurali. Per esempio, al fine di reperire il maggiore di una lista il programmatore C++ avrebbe dovuto combinare l'utilizzo di cicli e confronti, scrivendo codice più complesso e certamente meno chiaro di quello SQL. Ovviamente le funzioni di gruppo possono operare su un sottoinsieme delle righe di una tabella, come nel seguente esempio:

```
SELECT MIN(cilindrata)
FROM veicoli
WHERE codCombustibile='01'
```

In questo caso la funzione MIN è applicata ai soli valori della colonna Cilindrata relativi alle righe in cui codCombustibile='01'.

SUM

La funzione SUM calcola la somma dei valori di una colonna. Tramite le specifiche ALL e DISTINCT è possibile indicare se devono essere sommati rispettivamente tutti i valori o solo i valori distinti. Il default è ALL.
Nell'esempio seguente vengono selezionate tutte le righe della tabella modelli in cui codFabbrica è uguale a 001, e su queste viene calcolata la somma dei valori della colonna numeroVersioni:

```
SELECT SUM(numeroVersioni)
FROM modelli
WHERE codFabbrica='001'
```

AVG

La funzione AVG calcola la media (average) dei valori non nulli di una colonna.
Tramite le specifiche ALL e DISTINCT è possibile indicare se devono essere considerati tutti i valori o solo i valori distinti. Il default è ALL. Con il comando

```
SELECT AVG(cilindrata)
FROM veicoli
```

viene calcolata la media della colonna Cilindrata delle righe di Veicoli. Si noti che nel calcolo non sono considerati i valori nulli (NULL), mentre sarebbero considerati eventuali valori posti a zero.

Indichiamo con n il numero dei valori non nulli di una colonna e con m il numero dei valori distinti.

Se nessun valore nella colonna si ripete allora n è uguale a m. La clausola DISTINCT fa sì che AVG effettui la somma degli m valori distinti e divida il risultato per m. Se n è uguale a m si ottiene lo stesso risultato specificando o no la clausola DISTINCT.

Nell'esempio successivo nel calcolo della media desideriamo prendere in considerazione solamente i valori differenti della colonna Cilindrata o, in altre parole, considerare una sola volta ciascun valore. A questo scopo facciamo precedere il nome della colonna su cui opera la funzione AVG dalla specifica DISTINCT.

```
SELECT AVG(DISTINCT cilindrata)
FROM veicoli
```

Nel caso specifico il risultato è uguale a quello ottenuto nell'esempio precedente, in quanto non vi sono valori uguali.

COUNT

La funzione COUNT conta le righe. Tramite le specifiche *, ALL e DISTINCT è possibile contare rispettivamente le righe selezionate, tutti i valori non nulli, i valori distinti. Il default è ALL. Il comando

```
SELECT COUNT(*) AS CONTA
FROM veicoli
```

calcola il numero di righe della tabella Veicoli.

Se desideriamo ottenere il numero delle righe in cui cavalliFiscali è diverso da NULL scriveremo

```
SELECT COUNT(ALL cavalliFiscali)
FROM veicoli
```

Eventuali veicoli con cavalli fiscali uguale a zero non verrebbero invece esclusi. La specifica ALL, come nella maggior parte dei comandi di selezione, viene utilizzata come default ed è pertanto possibile ometterla. Vediamo ora l'uso della specifica DISTINCT nella funzione di gruppo COUNT. Il comando

```
SELECT COUNT(DISTINCT cavalliFiscali)
FROM veicoli
```

calcola il numero di righe della tabella Veicoli che hanno cavalli fiscali differenti.

È importante notare che, per quanto visto finora, non è possibile utilizzare in una stessa lista di selezione funzioni di gruppo e funzioni su singole righe; non è possi-bile, per esempio, ricercare la

targa del veicolo con cilindrata più alta. In seguito verranno esaminate le tecniche per risolvere questo tipo di problema.

LA CLAUSOLA GROUP BY

La clausola GROUP BY aumenta ulteriormente le possibilità di interrogazione tramite le funzioni di gruppo. Si supponga di dover calcolare il numero totale di versioni di modelli prodotti da ciascuna fabbrica. Per farlo dobbiamo reperire tutte le differenti fabbriche presenti nella tabella Modelli; successivamente per ogni fabbrica dobbiamo calcolare la somma delle versioni. Utilizzando gli strumenti del linguaggio esaminati finora i passi da fare sono i seguenti.

La clausola GROUP BY consente di calcolare espressioni su insiemi di righe. Il suo formato è:

```
<clausola_GROUP_BY> ::= GROUP BY <nome_colonna>
```

Il problema precedente viene così risolto molto più semplicemente.

```
SELECT codFabbrica, SUM(numeroVersioni)
FROM veicoli
GROUP BY codFabbrica
```

LA CLAUSOLA HAVING

La clausola HAVING può essere usata correttamente solo in combinazione con la clausola GROUP BY; essa consente di porre una condizione su una funzione di gruppo, come la clausola WHERE lo consente su singole righe.

Si supponga per esempio di voler calcolare la somma delle versioni prodotte da ciascuna fabbrica, come nel caso precedente, ma solo per le fabbriche con almeno 2 modelli:

```
SELECT codFabbrica, COUNT(*), SUM(numeroVersioni)
FROM modelli
GROUP BY codFabbrica
HAVING COUNT(*) >= 2
```

La clausola GROUP BY indica che per ogni valore differente in CodFabbrica siano raggruppate tutte le righe con tale valore; la clausola HAVING indica che sia contato per ogni gruppo il numero di righe reperito e controlla che questo sia maggiore o uguale a 2. Dei gruppi che hanno verificato tale condizione viene visualizzato il codice fabbrica, il numero dei modelli e la somma del numero delle versioni per quella fabbrica.

Nella clausola HAVING possono essere utilizzati gli operatori di confronto, i connettori logici e gli operatori BETWEEN, IN, LIKE. Affinché una clausola HAVING sia corretta deve essere espressa su

una funzione di gruppo. Si noti che una stessa operazione di selezione può utilizzare sia la clausola WHERE sia la clausola GROUP BY. È necessario avere ben chiaro in questo caso che la WHERE opera una restrizione a priori, ossia sulle righe alle quali applicare la funzione di gruppo; la clausola HAVING esegue invece una selezione a posteriori, sulle righe risultanti dalla esecuzione della selezione. Il seguente esempio illustra meglio questo concetto.

Utilizzando la tabella Modelli, vogliamo calcolare la somma totale delle versioni prodotte da fabbriche che producono almeno un modello in 4 versioni. Una possibile soluzione è la seguente:

```
SELECT codFabbrica, SUM(numeroVersioni)
FROM modelli
GROUP BY cod_Fabbrica
HAVING MAX(numeroVersioni) >= 4
```

In questo caso la clausola HAVING è basata sul fatto che la condizione che almeno un modello abbia un numero di versioni maggiore o uguale a 4 corrisponde a controllare se tale condizione è vera per il modello con il numero di versioni più alto.

La query richiede quindi solo le fabbriche per cui esiste un numero di versione maggiore o uguale a 4, ma il calcolo della somma viene eseguito su tutti i modelli di tali fabbriche. Ben diverso sarebbe stato esprimere la seguente operazione di selezione, apparentemente simile, ma che calcola invece la somma per tutte le fabbriche escludendo però dal calcolo i modelli con numero di versioni inferiore a 4:

```
SELECT codFabbrica, SUM(numeroVersioni)
FROM modelli
WHERE numeroVersioni >= 4
GROUP BY codFabbrica
```

LA CLAUSOLA DI ORDINAMENTO

La clausola di ordinamento serve a dare un ordinamento al risultato di una selezione.

Se tale clausola non viene specificata l'ordine delle righe di un comando SELECT non è definito, e lo stesso comando può produrre ordinamenti diversi se eseguito in contesti diversi. Ciò rivela come la teoria relazionale derivi dalla teoria degli insiemi: un insieme non è, per definizione, ordinato.

L'ordinamento può essere effettuato in base a una o più colonne e per ciascuna colonna può essere crescente o decrescente. La clausola ha il formato:

```
<clausola-di-ordinamento> ::= ORDER BY <ordine> [ {,<ordine> } ...]
```

dove


```
<ordine>::= <nome_colonna l <numero_colonna> [ASC | DESC]
```

Ricordiamo che il simbolo | nella sintassi indica un'alternativa, pertanto la colonna su cui effettuare l'ordinamento può essere indicata come <nome colonna> o come <numero colonna>. In ogni tabella le colonne sono numerate da sinistra verso destra nell'ordine in cui sono state definite, per cui per esempio la colonna 3 veicoli è codCategoria. Questo modo di fare riferimento alle colonne è comunque sconsigliato, soprattutto per motivi di leggibilità.

La specifiche ASC e DESC, anch'esse alternative tra loro, indicano se l'ordinamento deve essere crescente, nel caso di ASC, o decrescente, nel caso di DESC; il valore di default è ASC.

Il comando seguente restituisce le righe della tabella Veicoli in cui la cilindrata risulta maggiore di 1500. La clausola ORDER BY fa sì che il risultato venga ordinato primariamente in modo decrescente in base ai cavalli fiscali, e secondariamente in modo crescente per targa:

```
SELECT *
FROM veicoli
WHERE cilindrata > 1500
ORDER BY cavalliFiscali DESC, targa
```

JOIN SU DUE TABELLE

Vediamo come effettuare la ricerca di informazioni mettendo in relazione i dati presenti in tabelle diverse; utilizzeremo a questo scopo la tecnica delle join (congiunzioni), che realizza ulteriori operazioni definite dall'algebra relazionale. Consideriamo due tabelle che abbiano colonne contenenti dati in comune. Se due tabelle soddisfano questa condizione possono essere correlate tramite una operazione di join. Spesso, come nei nostri esempi, i nomi stessi delle due colonne sono uguali, ma ciò non è strettamente necessario; le due colonne devono in ogni caso avere lo stesso significato, ovvero rappresentare lo stesso insieme di entità. In termini algebrici, diremo che i valori delle colonne delle due tabelle devono appartenere allo stesso dominio.

Questa caratteristica rappresenta una delle maggiori potenzialità dei sistemi relazionali: i collegamenti tra i dati vengono implementati dinamicamente al momento dell'esecuzione delle query basandosi sul contenuto di colonne omogenee senza dover utilizzare dei puntatori diretti.

Ciò rende sempre possibile collegare tra di loro informazioni contenute in tabelle diverse ma accomunate da un identico valore in una particolare colonna. Grazie a questa caratteristica i modelli dei dati di un sistema relazionale possono crescere al crescere delle esigenze del sistema che rappresentano, senza richiedere, la maggior parte delle volte, interventi di riprogettazione.

Un caso molto comune e significativo di colonne comuni a due tabelle è quello in cui l'appartenenza allo stesso dominio deriva dalla presenza di relazioni tra le due entità rappresentate dalle due tabelle, e quindi dalla presenza di chiavi esterne. Facendo riferimento al database registroAutomobilistico,

categorie(codCategoria, ….)

veicoli(targa, …., codCategoria[categorie])

…….

rispondono a questa situazione le seguenti coppie di colonne:

- codCategoria della tabella categorie e codCategoria della tabella veicoli;
- codModello della tabella modelli e codModello della tabella veicoli;
- codCombustibile della tabella combustibili e codCombustibili e della tabella veicoli
- codFabbrica della tabella fabbriche e codFabbrica della tabella modelli;
- targa della tabella veicoli e targa della tabella proprietà;
- codProprietario della tabella proprietari e codProprietario della tabella proprietà.

EQUI-JOIN

La forma più utile e immediata della operazione di join è la equi-join tra due tabelle. Questa operazione restituisce una terza tabella le cui righe sono tutte e sole quelle ottenute dalle righe delle due tabelle di partenza in cui i valori delle colonne in comune sono uguali. L'operazione di equi-join è implementata in SQL come una forma particolare del comando di selezione:

- nella clausola FROM vanno indicate le due tabelle correlate su cui va effettuata la join;
- nella clausola WHERE va espresso il collegamento tra le due tabelle, mediante un'apposita condizione detta condizione di join;

Nella clausola WHERE e nella <lista di selezione> è possibile indicare i nomi delle colonne qualificandoli mediante il nome della tabella cui appartengono, utilizzato come prefisso nella forma.

```
nome_tabella.nome_colonna
```

Ciò spesso si rivela necessario per evitare ambiguità poiché, come abbiamo detto, in molti casi i nomi delle colonne contenenti dati comuni coincidono nelle due tabelle.

Utilizziamo ora la equi-join per visualizzare per ciascun veicolo la descrizione della relativa categoria; in questo caso l'operazione di ricerca coinvolge le due tabelle Veicoli e Categorie, poiché l'informazione relativa al nome della categoria, Nome Categoria, non è presente nella tabella

Veicoli. Le due tabelle sono correlate tra loro: ricordiamo infatti che nello schema concettuale esiste tra esse una relazione 1:N.

Il comando SELECT che realizza l'equi-join è il seguente:

```
SELECT targa, categorie.codCategoria, nomeCategoria
FROM veicoli, categorie
WHERE veicoli.codCategoria = categorie.codCategoria
```

Abbiamo realizzato l'operazione procedendo come segue:

- nella clausola FROM abbiamo indicato la tabella Veicoli e la tabella Categorie che dobbiamo correlare;
- nella clausola WHERE abbiamo espresso il collegamento con la condizione di join veicoli.codCategoria=categorie.codCategoria;
- nella clausola WHERE e nella <lista di selezione> abbiamo fatto precedere i nomi delle colonne codCategoria di veicoli e codCategoria di categorie dal nome della tabella di appartenenza seguito da un punto, eliminando in questo modo qualsiasi ambiguità. Si noti che le colonne relative alla targa del veicolo e al nome della categoria non sono state prefissate con il nome della tabella di appartenenza poiché, comparendo, rispettivamente solo, nella tabella Veicoli e Categorie e non in entrambe, non generano alcuna ambiguità.

Nella clausola WHERE possono essere aggiunte altre condizioni combinandole in AND con la condizione di join per restringere l'insieme delle righe restituite, o combinandole in OR con la condizione di join per estendere l'insieme delle righe restituite.

Possono inoltre essere applicate tutte le clausole (GROUP BY, HAVING, ORDER BY) viste in precedenza.

Restringiamo l'esempio appena visto ai soli veicoli con cilindrata maggiore di 1600 e ordiniamo il risultato per valori crescenti della targa:

```
SELECT targa, categorie.codCategoria, nomeCategoria
FROM veicoli, categorie
WHERE veicoli.codCategoria = categorie.codCategoria
AND cilindrata > 1600
ORDER BY targa
```

Ricerchiamo infine tutti i veicoli della categoria la cui descrizione è Autovettura, visualizzando tutte le colonne presenti nella tabella Categorie e nella tabella Veicoli.

Per evitare di specificare i nomi di tutte le colonne di una determinata tabella è definita la seguente notazione abbreviata:

<nome_tabella>.*

che utilizza il carattere speciale *, che avevamo già utilizzato a proposito della selezione su una sola tabella, estendendone l'uso ai casi in cui le tabelle utilizzate siano più di una. Il comando, dunque, sarà:

```
SELECT categorie.*, veicoli.*
FROM categorie, veicoli
WHERE veicoli.codCategoria = categorie.codCategoria
AND nomeCategoria= 'Autovettura'
```

La disponibilità di operazioni per l'esecuzione di join tra tabelle è uno degli aspetti qualificanti del linguaggio SQL. Basti pensare alla complessità che la scrittura di algoritmi di questo tipo, risolti in SQL in poche righe di codice, presenterebbe in un linguaggio di programmazione tradizionale.

INNER-JOIN

Esaminiamo la inner-join, di cui la equi-join è un caso particolare. Per la inner-join la condizione di join non deve necessariamente essere una condizione di uguaglianza.

Cerchiamo per esempio tutti i veicoli la cui categoria ha descrizione Autovettura o Furgone:

```
SELECT veicoli.*
FROM veicoli, categorie
WHERE veicoli.codCategoria = categorie.codCategoria
AND nomeCategoria IN ('Autovettura', 'Furgone')
```

Nella condizione di join è possibile utilizzare, oltre all'operatore IN, anche gli operatori di confronto e gli operatori BETWEEN e LIKE esaminati nel capitolo precedente.

L'esempio successivo mette in evidenza quanto detto relativamente alla possibilità di effettuare join tra colonne espresse sullo stesso dominio, seppure utilizzate per rappresentare informazioni non esplicitamente correlate.

La cilindrata media nella tabella Modelli e la cilindrata nella tabella Veicoli non sono legate da relazioni rappresentate da chiavi esterne. Tali dati sono utilizzati nella seguente query, che ricerca tutti i modelli dalla cilindrata media superiore alla cilindrata del veicolo con targa C76589AG.

```
SELECT modelli.*
FROM modelli, veicoli
WHERE cilindrataMedia > cilindrata
AND targa = 'C76589AG'
```


Negli esempi visti finora abbiamo utilizzato la forma tradizionale di scrittura di una join mediante l'uso del comando SELECT. Nelle ultime revisioni allo standard sono stati introdotti, per specificare il tipo di join, dei costrutti ad hoc che però spesso non sono supportati da DBMS commerciali.

Per realizzare la inner-join, e quindi anche, come caso particolare, per la equi-join, è stato introdotto il comando INNER JOIN, il cui formato è:

```
<comando_INNER_JOIN> ::=
<riferimento_a_tabella> [NATURAL] INNER JOIN
<riferimento_a_tabella>
[ON <condizione_di_join>
l USING <lista_colonne_di_join>]
```

Le clausole ON e USING sono alternative e servono entrambe a specificare la condizione di join. Utilizzando la prima clausola la condizione deve essere espressa esplicitamente e può contenere operatori diversi da quello di uguaglianza; la seconda è una forma abbreviata che esprime una equi-join effettuata in base alla lista di colonne che seguono la parola chiave USING. L'opzione NATURAL è un'ulteriore abbreviazione che indica una equi-join effettuata in base a tutte le colonne che nelle due tabelle sono identificate dallo stesso nome.

Riprendiamo l'esempio in cui volevamo ottenere per ciascun veicolo la descrizione della relativa categoria:

```
SELECT *
FROM veicoli, categorie
WHERE veicoli. codCategoria = categorie.codCategoria
```

Utilizzando il comando INNER JOIN anziché il comando SELECT, possiamo scrivere:

```
categorie NATURAL INNER JOIN veicoli
```

oppure

```
categorie INNER JOIN veicoli
USING codCategoria
```

oppure

```
categorie INNER JOIN veicoli
ON categorie.codCategoria = veicoli.codCategoria
```

La categoria sintattica <riferimento_a_tabella> include sia nomi di tabelle, come abbiamo visto negli esempi, sia comandi SELECT e ulteriori comandi di join.

Come per il comando INNER JOIN, i comandi OUTERJOIN e CROSS JOIN che verranno descritti nei paragrafi successivi sono definiti dallo standard, ma non sempre sono presenti nelle attuali versioni dei database commerciali.

OUTER-JOIN

Quando vengono correlate mediante una join due tabelle con colonne contenenti dati in comune, è possibile che un valore presente in una delle due colonne non compaia nell'altra.

Considerando la tabella categoria, effettuando la equi-join del primo esempio, in cui per ciascun veicolo si ricercava la corrispondente categoria, nessuna riga della tabella risultante ha codCategoria uguale a 0 2, poiché per tale categoria non è presente alcun veicolo. In alcuni casi è rilevante non perdere nella join la nozione di mancata corrispondenza.

Una join che preservi questa informazione è detta outer-join. La outer-join può mantenere i valori per cui non esiste corrispondenza per una, per l'altra o per entrambe le tabelle; i tre casi vengono rispettivamente definiti outer-join sinistra, outer-join destra, outer-join completa.

Prima delle ultime revisioni del linguaggio non esisteva un metodo standard per esprimere l'outerjoin, sebbene tutti i principali DBMS avessero implementato uno specifico costrutto. Attualmente lo standard prevede di esprimere la outer-join nel modo seguente:

```
<riferimento_a_tabella> [NATURAL] {LEFT l RIGHT I FULL} [OUTER] JOIN
<riferimento_a_tabella>
[ON <condizione_di_join>
l USING <lista_colonne_di_join>]
```

Nella outer-join sinistra vengono preservati tutti i valori della tabella che viene specificata come primo operando; nella outer-join destra vengono preservati tutti i valori della tabella che viene specificata come secondo operando. L'opzione NATURAL e le clausole ON e USING hanno lo stesso significato descritto nel comando INNER JOIN.

Per ottenere tutti i codici delle categorie, compresi quelli per cui non esiste alcun veicolo, sarà necessario scrivere

```
categorie LEFT OUTER JOIN veicoli
ON categorie.codCategoria = veicoli.codCategoria
```

oppure

```
categorie LEFT JOIN veicoli
ON categorie.codCategoria = veicoli.codVeicolo
```

oppure

```
veicoli RIGHT JOIN categorie
ON categorie.codCategoria = veicoli.codCategoria
```

Se nella tabella Veicoli fossero presenti veicoli con categoria non definita, in maniera del tutto analoga sarebbe possibile includerli nel risultato della join.

Per ottenere invece tutti i casi di mancata corrispondenza, sia della tabella categorie sia della tabella veicoli, scriveremo

```
categorie FULL OUTER JOIN veicoli
ON categorie.codCategoria = veicoli.codCategoria
```

oppure

```
categorie FULL JOIN veicoli
ON categorie.codCategoria = veicoli.codCategoria
```

oppure

```
Veicoli FULL JOIN categorie
ON categorie.codCategoria = veicoli.codCategoria
```

CROSS-JOIN

L'operazione di prodotto tra due tabelle si ottiene mediante una operazione di join espressa in maniera tradizionale in cui non venga specificata la condizione di join. Se le due tabelle sono composte rispettivamente da n e da m righe, la nuova tabella conterrà n x m righe, ottenute accodando ciascuna riga della prima tabella con ciascuna riga della seconda, come in

```
SELECT categorie.*, fabbriche.*
FROM categorie, fabbriche.
```

Lo stesso risultato si ottiene utilizzando il comando INNER JOIN senza specificare né l'opzione NATURAL, né le clausole ON o USING.

Questo caso è previsto esplicitamente dalle nuove revisioni allo standard; è stato infatti definito il comando CROSS JOIN con la seguente sintassi:

```
<riferimento_a_tabella> CROSS JOIN
<riferimento_a_tabella>
```

Il seguente comando genera lo stesso risultato dell'esempio precedente:

```
categorie CROSS JOIN fabbriche
```

JOIN SU PIÙ DI DUE TABELLE

Talvolta un'interrogazione può dover coinvolgere più di due tabelle. Supponiamo per esempio di voler reperire il nome della fabbrica del modello con targa A123456X. Come possiamo dedurre dall'esame dello schema logico riportato in Figura 6.6, nell'operazione sono coinvolte le tre tabelle

```
veicoli, modelli e fabbriche
```

Il linguaggio permette di effettuare la join tra più di due tabelle. Le regole da rispettare sono le seguenti:

- nella clausola FROM vanno elencate tutte le tabelle coinvolte;
- la condizione di join deve contenere le informazioni necessarie a correlare tutte le tabelle coinvolte; se le tabelle sono n, sono necessarie almeno n-1 condizioni. Nella lista di selezione non devono necessariamente essere incluse colonne di tutte le tabelle coinvolte; in altri termini alcune delle tabelle della clausola FROM possono avere l'unico ruolo di essere utilizzate nella condizione di join e in altre condizioni della clausola WHERE. Con questa tecnica l'esempio precedente può essere tradotto in un unico comando SELECT:

```
SELECT nomeFabbrica
FROM veicoli, modelli, fabbriche
WHERE veicoli.codModello = modelli.CodModello
AND modelli.codFabbrica = fabbriche.codFabbrica
AND targa = 'A123456X'
```

La clausola WHERE esprime il legame, vincolando inoltre targa a A123456X.

La join di tre tabelle è rilevante perché viene spesso utilizzata nel caso di schemi che traducono relazioni N:N; per esempio, in

```
SELECT cognome, nome
FROM veicoli, proprietà, proprietari
WHERE veicoli.targa = proprietà.targa
AND Proprietà.codProprietario = proprietari.cod_Proprietario
AND targa 'A123456X'
```

due tabelle, veicoli e proprietari, rappresentano gli insiemi di entità e la terza, Proprietà, la relazione N:N tra esse. Il linguaggio SQL permette join tra un numero arbitrario di tabelle, anche se ragioni dovute alla necessità di ottenere dei tempi di risposta accettabili inducono normalmente a porsi delle limitazioni.

SELF-JOIN

Un caso molto particolare di join è quello che utilizza una singola tabella, correlandola con se stessa; si ha in questo caso una self-join. Una possibile situazione di questo genere è quella in cui si vogliono elencare tutte le coppie di veicoli che hanno lo stesso modello. La tecnica utilizzata in tale caso è la seguente: si suppone, anziché di avere una sola tabella dei veicoli, di averne due copie assolutamente identiche, diciamo V1 e V2; in questo caso la join può essere facilmente ricondotta a una equi-join che fa corrispondere a tutti i veicoli della prima tabella quelli della seconda con lo stesso modello.

Per mettere in pratica quanto detto è necessario utilizzare gli alias, ovvero la ridenominazione di tabelle; questa consente di definire, nella clausola FROM, dei nomi alternativi per le tabelle, che possono essere utilizzati sia nella lista di selezione sia nelle clausole successive (WHERE, ORDER BY ecc.). La sintassi è la seguente:

```
<nome_tabella> <nome_alias>
```

Per ridenominare in un comando SELECT la tabella Veicoli con l'alias V1 scriveremo nella clausola FROM:

```
FROM veicoli V1
```

In prima approssimazione la soluzione al problema proposto è la seguente:

```
SELECT V1.targa, V1.codModello, V2.targa, V2.codModello
FROM veicoli AS V1, veicoli AS V2
WHERE V1.codModello = V2.codModello
```

Il risultato ottenuto può apparire in prima analisi scorretto, producendo una riga anche per quei veicoli che compaiono con un unico modello. Ricordando però di aver lavorato su due copie identiche della tabella dei veicoli, il risultato è corretto.

Infatti per esempio il modello 004 presente in un veicolo della prima tabella è presente nello stesso veicolo della seconda tabella. Inoltre le coppie di veicoli che hanno realmente lo stesso modello compaiono due volte, poiché se è vero che il modello del veicolo X è uguale a quello del veicolo Y, è anche vero il viceversa. Occorre quindi apportare delle correzioni al fine di eliminare le coppie in cui entrambi gli elementi sono uguali e quelle duplicate, ovvero quelle in cui gli elementi sono uguali a quelli di un'altra coppia ma in ordine diverso. Questo si ottiene ponendo la condizione V1.Targa > V2.Targa, poiché in questo modo i due codici saranno certamente diversi e il primo sarà maggiore del secondo, escludendo così necessariamente la seconda coppia con gli stessi elementi. Scriveremo pertanto:

```
SELECT V1.Targa, V1.Cod_Modello, V2.Targa,
V1.Cod_Modello
FROM Veicoli V1, Veicoli V2
WHERE V1.Cod_Modello = V2.Cod_Modello AND
V1.Targa > V2.Targa
```

Sebbene la ridenominazione di tabelle sia indispensabile solo nelle self-join. È possibile utilizzare tale tecnica anche nelle operazioni di join viste in precedenza. In questo modo la scrittura risulta essere abbreviata.

PROGETTO / ESERCIZIO 1

Lo scopo di questo esercizio (reale) è monitorare alcune zone d'Italia al fine di verificare se gli agenti inquinanti superano le soglie che garantiscono la salute dell'essere umano.

http://www2.arpalombardia.it/sites/qaria/_layouts/15/qaria/idati.aspx

DESCRIZIONE

PM (Particulate Matter) è il termine generico con il quale si definisce un mix di particelle solide e liquide (particolato) che si trovano in sospensione nell'aria. Il PM può avere origine sia da fenomeni naturali (processi di erosione del suolo, incendi boschivi, dispersione di pollini ecc.) sia da attività antropiche, in particolar modo dai processi di combustione e dal traffico veicolare (particolato primario). Esiste, inoltre, un particolato di origine secondaria che si genera in atmosfera per reazione di altri inquinanti come gli ossidi di azoto (NOx), il biossido di zolfo (SO2), l'ammoniaca (NH3) ed i Composti Organici Volatili (COV), per formare solfati, nitrati e sali di ammonio.

Gli studi epidemiologici hanno mostrato una correlazione tra le concentrazioni di polveri in aria e la manifestazione di malattie croniche alle vie respiratorie, in particolare asma, bronchiti, enfisemi. A livello di effetti indiretti inoltre il particolato agisce da veicolo per sostanze ad elevata tossicità, quali ad esempio gli idrocarburi policiclici aromatici.

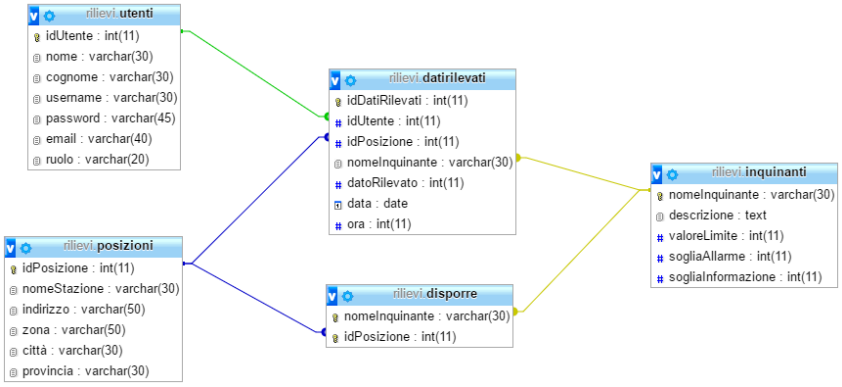
Le particelle di dimensioni inferiori costituiscono un pericolo maggiore per la salute umana, in quanto possono penetrare in profondità nell'apparato respiratorio; è per questo motivo che viene attuato il monitoraggio ambientale di PM10 e PM2.5 che rappresentano, rispettivamente, le frazioni di particolato aerodisperso aventi diametro aerodinamico inferiore a 10 µm e a 2.5 µm.

OBIETTIVO

La soglia di concentrazione in aria delle polveri fini PM2.5 è stabilita dal D.Lgs. 155/2010 e calcolata su base temporale annuale. La caratterizzazione dei livelli di concentrazione in aria di PM2.5 nel Veneto al 2015 si è basata sul superamento, registrato presso le stazioni della rete regionale ARPAV della qualità dell'aria che misurano questo inquinante, del Valore Limite (VL) annuale per la protezione della salute umana pari a 25 µg/m3. Da osservare che, fino al 2014, la stessa concentrazione di 25µg/m3, era considerata un Valore Obiettivo. Dal 2015 invece questa soglia rappresenta a tutti gli effetti un Valore Limite (VL).

REALIZZAZIONE

Realizzare in mySQL il Data Base *Inquinamento* costituito dalla seguenti tabelle:



NB Si ricorda che occorre avviare il Server Virtuale easyPHP e dalla barra degli indirizzi del vostro browser digitare:

http://localhost/modules/phpmyadmin/

ESEMPIO

Vediamo come sia possibile scrivere un'interrogazione in SQL

```
SELECT nomeInquinante, datoRilevato, data, ora
FROM datirilevati
WHERE idPosizione = 6 ;
```

La query realizzata consente la visualizzazione dei dati rilevati (nomeInquinante, datoRilevato, data e ora) dalla stazione numero 6.

NB **Il risultato dell'esecuzione di una query è una tabella.**

RICHIESTA DELL'ESERCIZIO

Dopo aver valorizzato la tabella con almeno 20 record si realizzino le seguenti interrogazioni:

1. Elencare i dati rilevati nell'anno 2016 dalla Centralina 1.

```
SELECT nomeInquinante, datoRilevato, YEAR(data), ora
FROM datirilevati
WHERE Year(data)=2016 AND idPosizione = 1 AND nomeInquinante=PM10;
```

In questa query abbiamo utilizzato la funzione YEAR(campo) che estrare dal campo data, passatogli come parametro, l'anno.

nomeInquinante	datoRilevato	YEAR(data)	ora
PM10	80	2016	8
PM10	130	2016	12
PM10	90	2016	8
PM10	100	2016	12
PM10	120	2016	8
PM10	120	2016	8
PM10	134	2016	8
PM10	130	2016	8
PM10	160	2016	8
PM10	140	2016	8
PM10	111	2016	8
PM10	100	2016	8
PM10	120	2016	8
PM10	90	2016	8

È possibile rinominare le intestazioni di colonna in questo modo:


```
SELECT nomeInquinante, datoRilevato, YEAR(data) AS ANNO, ora
FROM datirilevati
WHERE Year(data)=2016 AND idPosizione = 1
```

nomeInquinante	datoRilevato	ANNO	ora
PM10	80	2016	8
PM10	130	2016	12
PM10	90	2016	8
PM10	100	2016	12
PM10	120	2016	8
PM10	120	2016	8
PM10	134	2016	8
PM10	130	2016	8
PM10	160	2016	8
PM10	140	2016	8
PM10	111	2016	8
PM10	100	2016	8
PM10	120	2016	8
PM10	90	2016	8

2. Visualizzare la media giornaliera dei dati rilevati dalla stazione 1. PURAMENTE DIMOSTRATIVA – DA NON STUDIARE ORA

```
SELECT data, AVG(datoRilevato)
FROM datirilevati
WHERE idPosizione=1 AND nomeInquinante="PM10"
GROUP BY data
```

data	AVG(datoRilevato)
2016-01-01	105.0000
2016-02-01	95.0000
2016-03-02	120.0000
2016-04-19	134.0000
2016-05-03	130.0000
2016-06-16	160.0000
2016-07-06	140.0000
2016-08-17	111.0000
2016-09-01	100.0000
2016-10-06	120.0000
2016-11-11	90.0000
2016-12-15	120.0000

```
SELECT data, ROUND(AVG(datoRilevato), 2)
FROM datirilevati
WHERE idPosizione=1 AND nomeInquinante="PM10"
GROUP BY data
```

data	ROUND(AVG(datoRilevato), 2)
2016-01-01	105.00
2016-02-01	95.00
2016-03-02	120.00
2016-04-19	134.00
2016-05-03	130.00
2016-06-16	160.00
2016-07-06	140.00
2016-08-17	111.00
2016-09-01	100.00
2016-10-06	120.00
2016-11-11	90.00
2016-12-15	120.00

ESERCIZIO 2

Data la tabella `Valutazioni(IdValutazione, IdAlunno, Classe, Voto)` visualizzare la media dei voti per il solo alunno con IdAlunno uguale a 1023.

```
SELECT AVG(Voto)
FROM Valutazioni
GROUP BY IdAlunno
HAVING IdAlunno=1023;
```

ESERCIZIO 3

Data la tabella `Rilevazioni(IdRilevazione, Valore, Giorno, Ora)` visualizzare il numero di rilevazioni effettuate alle ore 10.

```
SELECT Count(IdRilevazione) AS ConteggioDiIdRilevazioni
FROM Rilevazioni
GROUP BY Ora
HAVING Ora=10;
```


ESERCIZIO 4

Data la relazione `Persone(IdPersona, Nome, Cognome, DataNascita)`, visualizzare il nome e il cognome delle persone che hanno più di 17 anni:

```
SELECT Nome, Cognome, (YEAR(NOW)-YEAR(DataNascita)) AS ETA
FROM Persona
WHERE ETA > 17 ;
```

ESERCIZIO 5

Sia data la seguente tabella:

Query2 | Utenti

IdUtente	Cognome	Punti
1	Bianchi	10
2	Rossi	80
3	Verdi	80
4	Grigi	70
5	Neri	60

Utenti(IdUtente, Cognome, Punti)

1) Visualizzare i punti dei soli utenti con punteggio massimo:

```
SELECT Max(Punti) AS Max
FROM Utenti;
```

2) Visualizzare Cognome e Punti per i soli utenti con punteggio massimo.

```
SELECT Cognome, Punti
FROM Utenti
WHERE Punti IN (
                    SELECT Max(Punti) AS MaxDiPunti
                    FROM Utenti
                ) ;
```

ESERCIZIO 6 – completo

Sia dato il seguente schema relazionale:

Dipendenti(Matricola, Cognome, Nome, CodFisc, Assunto, Filiale, Mansione, Livello, StipBase, Via, Cap, Citta, Prov, Tel, DataNascita, PremioProduzione, CodReparto[Reparto])

Reparti(CodReparto, NomeReparto, Tel)

OPERAZIONI RELAZIONALI

q1. Visualizzare Nome, Cognome di tutti i dipendenti.

```
SELECT     Nome, Cognome
FROM       Dipendenti ;
```

q2. Visualizzare le funzioni che un dipendente può svolgere.

```
SELECT     DISTINCT Mansione
FROM       Dipendenti ;
```

q3. Visualizzare/Elencare tutti i dipendenti (tutte le loro caratteristiche).

```
SELECT     *
FROM       Dipendenti ;
```

NB L'asterisco (*) di SELECT * indica al DB di fornire tutte le colonne associate alla tabella specificata dalla clausola FROM.

q4. Visualizzare Nome, Cognome e Codice Fiscale degli impiegati con funzione "Impiegato".

```
SELECT     Nome, Cognome, CodFisc
FROM       Dipendenti
WHERE      Mansione = "Impiegato" ;
```

q5. Elencare i dipendenti che abitano a "Milano".

```
SELECT     *
FROM       Dipendenti
WHERE      Citta = " Milano" ;
```

q6. Elencare le provincie di abitazione degli impiegati, rinominando l'intestazione in Provincia.

```
SELECT     DISTINCT Prov AS Provincia
FROM       Dipendenti
WHERE      Citta = " Milano" ;
```

NB Query senza ripetizioni: possiamo visualizzare la colonna Prov facendo comparire i valori ripetuti una sola volta. Occorre anteporre alla colonna la clausola DISTINCT.

q7. Visualizzare Nome e Cognome dei dipendenti che guadagnano più di 2000 euro.

```
SELECT     Nome, Cognome
FROM       Dipendenti
WHERE      StipBase > 2000 ;
```


q8. Visualizzare Nome e Cognome dei dipendenti che guadagnano più di 2000 euro e che vivono a "Milano".

```
SELECT     Nome, Cognome
FROM       Dipendenti
WHERE      StipBase > 2000 AND Citta = " Milano" ;
```

q9. Visualizzare Nome e Cognome dei dipendenti, gli stipendi e gli stipendi con una variazione del 10%.

```
SELECT     Nome, Cognome, Stipendio, Stipendio*1.2 AS NuovoStipendio
FROM       Dipendenti
```

q10. Visualizzare Nome e Cognome dei dipendenti che svolgono una certa mansione (decisa dall'operatore).

```
SELECT     Nome, Cognome
FROM       Dipendenti
WHERE      Mansione = ["Inserisci mansione"] ;
```

FUNZIONI DI AGGREGAZIONE: COUNT, SUM, AVG, MIN, MAX

q11. Visualizzare il numero di dipendenti con stipendio maggiore di 2000 euro.

```
SELECT     COUNT(StipBase)
FROM       Dipendenti
WHERE      StipBase > 2000 ;
```

q12. Visualizzare l'esborso totale (somma degli stipendi) per gli stipendi dei dipendenti.

```
SELECT     SUM(StipBase)
FROM       Dipendenti ;
```

q13. Visualizzare l'esborso totale per gli stipendi dei dipendenti di livello 5.

```
SELECT     SUM(StipBase)
FROM       Dipendenti
WHERE      Livello = 5 ;
```

q14. Visualizzare il valore Massimo degli stipendi dei dipendenti.

```
SELECT     MAX(StipBase)
FROM       Dipendenti ;
```

q15. Visualizzare il valore Medio degli stipendi dei dipendenti.

```
SELECT     AVG(StipBase)
FROM       Dipendenti ;
```

q16. Visualizzare il valore Minimo e Massimo degli stipendi dei dipendenti.

```
SELECT     MAX(StipBase), MIN(StipBase)
FROM       Dipendenti ;
```

FUNZIONI ARITMETICHE: ABS, CEIL, FLOOR

CEIL(ARG). Questa funzione fornisce il più piccolo numero intero che è maggiore o uguale al suo argomento.

FLOOR(ARG). Questa funzione fornisce il più grande numero intero che è minore o uguale al suo argomento.

SIGN(ARG). La funzione SIGN restituisce –1 se il suo argomento è minore di zero e restituisce 1 se il suo argomento è maggiore o uguale a zero.

APPROFONDIMENTO

FUNZIONI TRIGONOMETRICHE: COS, SIN, TAN (operano su argomenti in rad)

ALTRE FUNZIONI MATEMATICHE: EXP, LN, LOG, POWER(BASE, ESPONENTE), SQRT

FUNZIONI CARATTERE

- **CONCAT**: ha la funzione di concatenare due o più stringhe;
- **UPPER**: restituisce una stringa formata da tutti i caratteri minuscoli convertiti in maiscoli;
- **SUBSTRING(str, aPartireDa, quantiChar)**: restituisce la sottostringa di una stringa a partire dalla prima occorrenza di un pattern;
- **LOWER**: restituisce una stringa formata da tutti i caratteri maiuscoli trasformati in minuscoli.
- **LENGTH()**: restituisce la lunghezza della stringa.

```
SELECT CONCAT(UPPER(SUBSTRING(nome_campo,1,1)),
LOWER(SUBSTRING(nome_campo, FROM 2)));
```

```
SELECT SUBSTRING('Quadratically',5);
        -> 'ratically'
SELECT SUBSTRING('foobarbar' FROM 4);
        -> 'barbar'
SELECT SUBSTRING('Quadratically',5,6);
        -> 'ratica'
SELECT SUBSTRING('Sakila', -3);
        -> 'ila'
SELECT SUBSTRING('Sakila', -5, 3);
        -> 'aki'
SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
        -> 'ki'
```

ORDINAMENTI

q17. Visualizzare i dipendenti in ordine alfabetico.

```
SELECT     *
FROM       Dipendenti
ORDER BY   Cognome, Nome ;
```

q18. Visualizzare Nome, Cognome e Stipendio dei dipendenti con stipendi maggiori di 2000 euro in ordine decrescente (cioè dallo stipendio più alto a quello più basso, ma sempre maggiore di 2000 euro).

```
SELECT     Cognome, Nome, StipBase
FROM       Dipendenti
WHERE      StipBase > 2000
ORDER BY   StipBase DESC ;
```

q19. q18. Visualizzare Nome, Cognome e Stipendio dei dipendenti con stipendi maggiori di 2000 euro in ordine decrescente per stipendio e Cognome.

```
SELECT     Cognome, Nome, StipBase
```

```
FROM       Dipendenti
WHERE      StipBase > 2000
ORDER BY   StipBase DESC, Cognome ;
```

NB **ASC,** anteposto ad un campo in ORDER BY applica un ordinamento ascendente (di default); **DESC** invece un ordinamento decrescente.

RAGGRUPPAMENTI

q20. Raggruppare i dipendenti in base al loro livello e visualizzare lo stipendio medio per livello.

```
SELECT     Livello, AVG(StipBase)
FROM       Dipendenti
GROUP BY   Livello ;
```

q21. Raggruppare i dipendenti in base al loro livello e visualizzare lo stipendio medio e il numero di dipendenti per livello.

```
SELECT     Livello, AVG(StipBase), COUNT(Livello)
FROM       Dipendenti
GROUP BY   Livello ;
```

q22. Raggruppare i dipendenti in base al loro livello, solo però per quelli maggiori del terzo, e visualizzare lo stipendio medio e il numero di dipedenti per livello.

```
SELECT     Livello, AVG(StipBase), COUNT(Livello)
FROM       Dipendenti
GROUP BY   Livello
HAVING     Livello > 3 ;
```

q23. Raggruppare i dipendenti in base al loro livello, e visualizzare lo stipendio medio e il numero di dipedenti per livello, solo però per i livello in cui ci siano almeno 5 dipendenti.

```
SELECT     Livello, AVG(StipBase), COUNT(*)
FROM       Dipendenti
GROUP BY   Livello
HAVING     COUNT(*) > 5 ;
```

ESPRESSIONI E OPERATORI CONDIZIONALI

Tutte le volte che si vuole trovare un particolare elemento o gruppo di elementi in un database, occorre specificare una o più condizioni. Le condizioni sono introdotte dalla clausola WHERE.

OPERATORE LOGICO		SIGNIFICATO
OR	**||**	**alternanza**
AND	**&&**	**congiunzione**
NOT	**!**	**negazione**

q24. Visualizzare Nome, Cognome dei dipendenti che si chiamano Daniele e Luca.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Nome = "Daniele" OR Nome = "Luca" ;
```

Oppure

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Nome = "Daniele" || Nome = "Luca" ;
```

q25. Visualizzare Nome, Cognome dei dipendenti con stipendio compreso fra 1500 e 2000 euro.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      StipBase >= 1500 AND StipBase <= 2000 ;
```

Oppure

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      StipBase >= 1500 && StipBase <= 2000 ;
```

q26. Visualizzare Nome, Cognome dei dipendenti che non si chiamano Daniele.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      NOT Nome = "Daniele" ;
```

Oppure

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Nome != "Daniele" ;
```

Oppure

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Nome <> "Daniele" ;
```

NB **Gli operatori: maggiore (>) , maggiore o uguale (>=), minore (<), minore o uguale (<=), diverso (<>),** si usano allo stesso modo di come si usa l'operatore di uguaglianza.

GESTIONE DATE

q27. Visualizzare Nome e Cognome degli impiegati assunti dopo il 2009.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      YEAR(Assunto) > 2009 ;
```

q28. Visualizzare Nome e Cognome degli impiegati assunti fra il 2008 e il 2010.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      YEAR(Assunto) > 2008 AND YEAR(Assunto) < 2010 ;
```

RICERCHE SEMPLIFICATE – UTILIZZO DELLE CLAUSOLE IN, LIKE, IS

CARATTERI JOLLY		
Significato	Access	MySQL
1 carattere	"#"	"_"
Un numero qualsiasi di caratteri	"*"	"%"

q29. Visualizzare gli impiegati che abitano nelle provincie di MI, VA e NO.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Prov IN ("MI", "VA", "NO") ;
```

q30. Visualizzare Cognome e Nome dei dipendenti con il cognome che inizia con la lettera R.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Cognome LIKE "R*" ;
```

q31. Visualizzare Cognome e Nome dei dipendenti che non hanno fornito il loro numero di telefono.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Tel IS NULL ;
```

q32. Visualizzare Cognome e Nome dei dipendenti che si chiamano Paolo o Paola.

```
SELECT     Cognome, Nome
FROM       Dipendenti
WHERE      Nome LIKE "Paol#"  ;
```

OPERAZIONI DI GIUNZIONE - JOIN

q33. Visualizzare il Nome del Reparto e il relative telefono in cui lavora il dipendente Rossi Mario.

```
SELECT     Reparto.NomeReparto, Reparto.Tel
FROM       Dipendente, Reparto
WHERE      Dipendente.Cognome = "Rossi" AND Dipendente.Nome = "Mario"
AND        Dipendente.CodReparto = Reparto.CodReparto ;
```


ESERCIZIO 4

Sia dato lo schema relazionale costituito dalle tabelle (le chiavi primarie sono sottolineate):

```
ATTORI (CodAttore, Nome, AnnoNascita, Nazionalità);
RECITA (CodAttore, CodFilm)
FILM (CodFilm, Titolo, AnnoProduzione, Nazionalità, Regista, Genere)
PROIEZIONI (CodProiezione, CodFilm, CodSala, Incasso, DataProiezione)
SALE (CodSala, Posti, Nome, Città)
```

Trovare, con l'opportuna query SQL:

1. Il numero di sale di Torino con più di 60 posti

```
SELECT count(*)
FROM SALE AS s
WHERE s.Città = "Torino" and s.Posti > 60
```

2. Il numero totale di posti nelle sale di Torino

```
SELECT sum(s.Posti)
FROM SALE AS s
WHERE s.Città = "Torino";
```

3. Per ogni città, il numero di sale con più di 60 posti

```
SELECT s.Città, count(*)
FROM SALE AS s
WHERE s.Posti > 60
GROUP BY s.Città;
```

4. Per ogni regista, l'incasso totale di tutte le proiezioni dei suoi film

```
SELECT f.Regista, sum(p.Incasso) AS IncassoTotale
FROM FILM AS f, PROIEZIONI AS p
WHERE f.CodFilm = p.CodFilm
GROUP BY f.Regista;
```

5.Per ogni film di S. Spielberg, il titolo del film, il numero totale di proiezioni a Torino e l'incasso totale (sempre a Torino)

```
SELECT f.Titolo, count(*)
AS NumeroProiezioni, sum(p.Incasso) AS IncassoTotale
FROM FILM AS f, PROIEZIONI AS p, SALE AS s
WHERE f.CodFilm = p.CodFilm and p.CodSala=s.CodSala
AND f.Regista = 'S.Spielberg' and s.Città = 'Torino'
GROUP BY f.CodFilm, f.Titolo;
```

6.Il regista ed il titolo dei film in cui recitano meno di 6 attori

```
SELECT f.Regista, f.Titolo
```

```
FROM FILM AS f, RECITA AS r
WHERE f.CodFilm = r.CodFilm
GROUP BY f.CodFilm, f.Titolo, f.Regista
HAVING count(*) < 6;
oppure
SELECT f.Regista, f.Titolo
FROM FILM AS f
WHERE 6 > (SELECT count (*)
FROM Recita r
WHERE f.CodFilm = r.CodFilm);
```

7.Il titolo dei film dello stesso regista di “Casablanca”

```
SELECT f.Titolo
FROM FILM AS f
WHERE f.Regista = (SELECT f.Regista
FROM Film f
WHERE f.Titolo = "Casablanca");
```

8.Il titolo dei film in cui recita M. Mastroianni oppure S.Loren

```
SELECT DISTINCT f.Titolo
FROM FILM AS f, RECITA AS r, ATTORE AS a
WHERE (a.Nome = "M.Mastroianni" OR a.Nome = "S.Loren")
AND f.CodFilm = r.CodFilm
AND r.CodAttore = a.CodAttore;
```

9.Il titolo dei film in cui recitano M. Mastroianni e S.Loren

```
SELECT f.Titolo
FROM FILM AS f,
WHERE "M.Mastroianni" IN
(SELECT a.Nome
FROM ATTORI AS a, RECITA AS r
WHERE f.CodFilm = r.CodFilm AND r.CodAttore = a.CodAttore)
AND "S.Loren" IN
(
SELECT a.Nome
FROM ATTORI AS a, RECITA AS r
WHERE f.CodFilm = r.CodFilm AND r.CodAttore = a.CodAttore);
```

10.I titoli dei film che non sono mai stati proiettati a Torino

```
SELECT f.Titolo
FROM FILM AS f
```



```
WHERE NOT EXISTS
(SELECT *
FROM PROIEZIONI
AS p, SALA AS s
WHERE s.Città="Torino" AND f.CodFilm=p.CodFilm AND p.CodSala =s.CodSala);
oppure
SELECT f.Titolo
FROM FILM AS f
WHERE "Torino" NOT IN
(SELECT s.Città
FROM PROIEZIONI AS p, SALA AS s
WHERE f.CodFilm = p.CodFilm AND p.CodSala =s.CodSala);
```

11.I titoli dei film che sono stati proiettati solo a Torino

```
SELECT f.Titolo
FROM FILM AS f
WHERE NOT EXISTS
(SELECT *
FROM PROIEZIONI AS p, SALA AS s
WHERE Città < > "Torino" AND f.CodFilm = p.CodFilm
AND p.CodSala = s.CodSala);
```

RIFERIMENTO AL LINGUAGGIO SQL

http://www.html.it/guide/guida-linguaggio-sql/