Linguaggio SQL (parte 2)

Mattia Cozzi
cozzimattia@gmail.com

a.s. 2024/2025

Contenuti

Struttura di un comando SQL

Un comando in SQL si suddivide in:

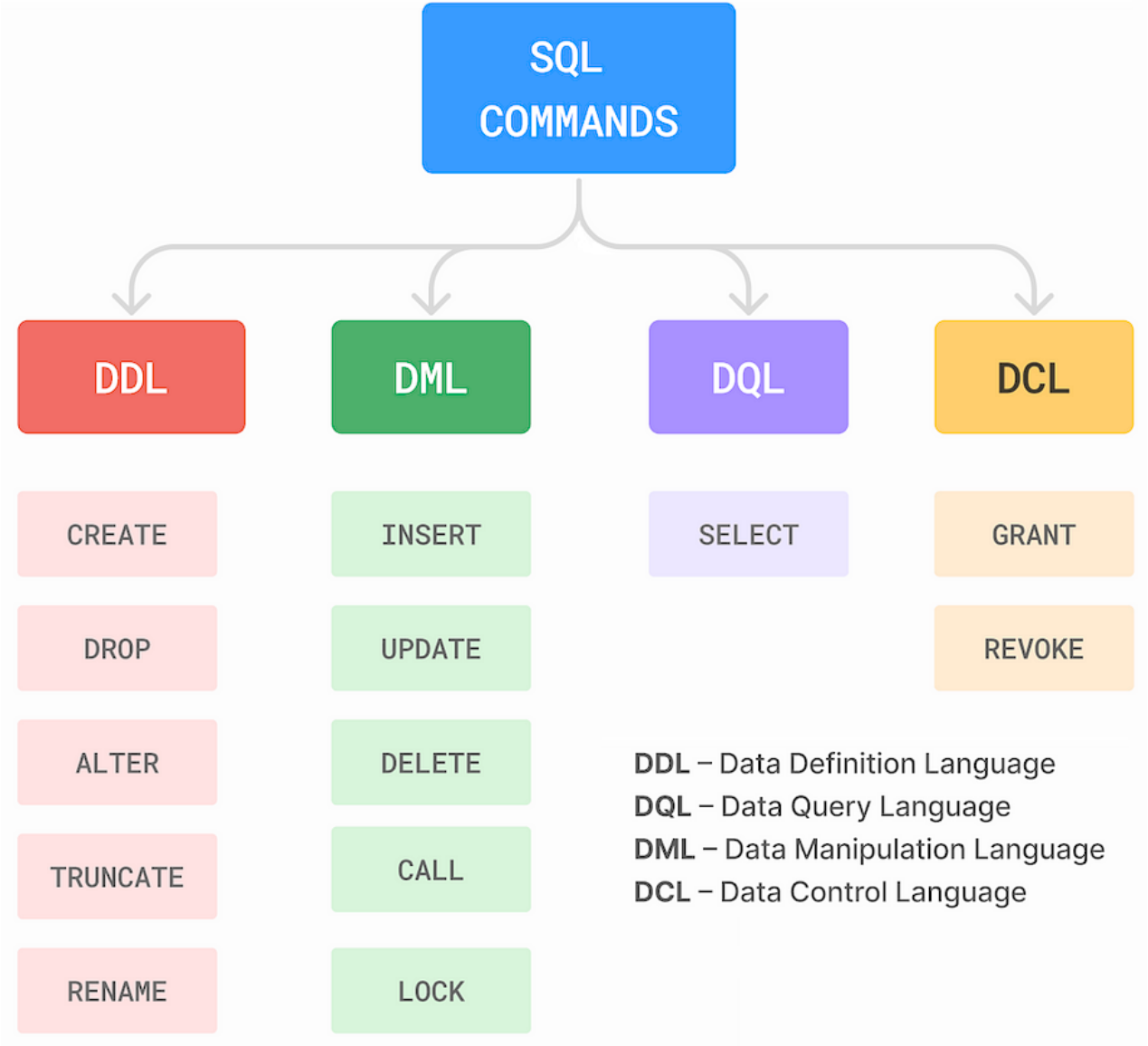
- costrutto, cioè il tipo di operazione effettuata, come `SELECT`, `INSERT` o `CREATE`;
- parametri, come `FROM`, `WHERE` o `IN`.

Nonostante SQL non sia *case-sensitive*, è bene scrivere il costrutto e i parametri in maiuscolo, il resto della stringa in minuscolo.

```
SELECT nome,cognome,telefono
  FROM Atleti
  WHERE altezza>170 AND peso<95;
```

Attenzione al punto e virgola a fine comando.

Schema comandi di SQL



Creazione di una tabella con SQL

La sintassi SQL per creare una nuova tabella è come la seguente:

```
CREATE TABLE Alunni (   --crea la tabella Alunni
  matricola INT AUTO_INCREMENT,    --impostazione dei campi
  cognome VARCHAR(20),
  nome VARCHAR(20),
  telefono VARCHAR(12),
  data_nascita DATE NOT NULL, --attenzione!
);
```

Tipi di campo

`VARCHAR(size)`	stringhe
`TEXT(size)`	stringhe lunghe
`SET(val1,val2,...)`	array di valori (max 64)
`BOOLEAN`	valore booleano (yes/no, 0/1)
`INT`	numeri interi
`FLOAT`	numeri con virgola
`DATE`	data in formato YYYY-MM-DD
`YEAR`	anno, YYYY

Chiavi e relazioni in SQL

Una relazione si crea grazie ad una coppia chiave primaria-chiave esterna (in inglese, **PRIMARY KEY**/**FOREIGN KEY**).

Una relazione pone dei vincoli (*constraints*) sui valori di un campo.

```
CREATE TABLE mansioni (
  ID_mansione INT AUTO_INCREMENT,
  nomeMansione VARCHAR(20) NOT NULL,
  tariffaOraria FLOAT,
  PRIMARY KEY (ID_mansione));

CREATE TABLE dipendenti (
  matricola INT AUTO_INCREMENT,
  nomeDip VARCHAR(20) NOT NULL,
  cognomeDip VARCHAR(20) NOT NULL,
  id_mansione INT,
  stipendio FLOAT,
  PRIMARY KEY (matricola),
  FOREIGN KEY (id_mansione) REFERENCES mansioni(ID_mansione));
```

Creazione tabella con Python

Per creare una tabella da Python, usiamo il seguente script:

```
1  import mysql.connector
2
3  mydb = mysql.connector.connect(
4      host = "localhost",user = "root",password = "",
5      database="mydatabase",
6  )
7
8  mycursor = mydb.cursor()
9  #le righe precedenti verranno omesse da qui in poi
10 mycursor.execute("CREATE TABLE Mansioni (ID_mansione INT
   ↪  AUTO_INCREMENT PRIMARY KEY, nomeMansione VARCHAR(20) NOT
   ↪  NULL, tariffaOraria INT)")
```

Controlliamo da phpMyAdmin (o con uno script Python) che tutto sia andato a buon fine.

Esercizi

1. Crea un database con un nome a tua scelta e inserisci in esso almeno due tabelle (ognuna con una chiave primaria) con una relazione che le lega. Mostra poi l'elenco delle tabelle che hai creato.
2. Aggiungi una nuova entità (collegata ad almeno una delle altre) al database dell'esercizio precedente. Mostra poi il nuovo elenco delle tabelle.

Inserimento di dati in una tabella con SQL

Sintassi SQL:

```
--Inserimento di una riga completa
INSERT INTO tabella
  VALUES (val1,val2,val3,...);

--Inserimento solo di alcuni valori
INSERT INTO tabella(campo1,campo3,campo8)
  VALUES (val1,val3,val8);
```

Ad esempio:

```
INSERT INTO dipendenti
  VALUES ("17A8J","Marco","Rossi",4, 1650.54);

INSERT INTO dipendenti(matricola, nomeDip, cognomeDip, mansione)
  VALUES ("78L2D","Andrea","Bianchi",2);
-- ATTENZIONE: se un campo ha l'opzione AUTO_INCREMENT, non
-- va indicato il valore di quel campo durante l'inserimento
```

Inserimento di dati in una tabella con Python

Con Python, creiamo prima una stringa con dei "segnaposto" (lavorando con stringhe, useremo il segnaposto %s):

```
10  #Inserimento di una riga completa
11  comando = "INSERT INTO dipendenti VALUES (%s, %s, %s, %s, %s)"
12  valori = ("17A8J","Marco","Rossi",4, 1650.54)
13  mycursor.execute(comando, valori)
14
15  #Inserimento solo di alcuni valori
16  comando2 = "INSERT INTO dipendenti(matricola, nomeDip,
    ↪  cognomeDip, mansione) VALUES (%s, %s, %s, %s)"
17  valori2 = ("78L2D","Andrea","Bianchi",2)
18  mycursor.execute(comando2, valori2)
19
20  #il comando seguente fa eseguire effettivamente
21  #le operazioni di modifica richieste sul database
22  mydatabase.commit()
```

Inserimento multiplo con Python

Possiamo inserire diversi record (le righe di una tabella) con:

```
10 comando = "INSERT INTO dipendenti VALUES (%s, %s, %s, %s, %s)"
11
12 #creo una LISTA DI TUPLE
13 valori = [
14   ("17A8J","Marco","Rossi",4, 1650.54),
15   ("78L2D","Andrea","Bianchi",2, 1800.00),
16   ("17A8J","Carlo","Verdi",1, 2013.65),
17   ("17A8J","Anna","Neri",2, 1800.00),
18 ]
19 #attenzione a come cambia questa riga
20 mycursor.executemany(comando, valori)
21
22 mydatabase.commit()
```

Id dell'ultimo valore inserito

Possiamo interrogare il cursore per conoscere l'id dell'ultimo record inserito:

```
10 comando = "INSERT INTO dipendenti VALUES (%s, %s, %s, %s, %s)"
11 valori = ("17A8J","Marco","Rossi",4, 1650.54)
12 mycursor.execute(comando, valori)
13 mydatabase.commit()
14
15 print("Aggiunto nuovo record con ID: ", mycursor.lastrowid)
```

Esercizi

3. Inserisci un nuovo record nella tabella creata nell'esercizio precedente.
4. Inserisci almeno quattro nuovi record nella stessa tabella dell'esercizio precedente, usando la sintassi per l'inserimento multiplo.
5. [db-profumi] Aggiungi te stesso/a come "naso" nella tabella corretta e crea un nuovo profumo per il brand "Chanel". Tu dovrai essere il naso per quel profumo.
6. [db-cosmesi] Crea un algoritmo che chieda all'utente tutti i dati di un nuovo prodotto. Aggiungi il nuovo prodotto alla tabella corretta e mostra il suo ID a schermo.

Eliminazione di dati con SQL

In SQL, il comando **DELETE** permette di eliminare uno più record da una tabella.

```
--Svuotamento di una tabella
DELETE FROM tabella;

--Eliminazione di alcuni valori
DELETE FROM tabella
  WHERE condizione;
```

Ad esempio:

```
--Svuotamento di una tabella
DELETE FROM mansioni;

--Eliminazione di alcuni valori
DELETE FROM dipendenti
  WHERE stipendio > 1800.00;
```

Eliminazione di dati con Python

Per eliminare record, usiamo:

```
10  #eliminiamo usando una condizione
11  comando = "DELETE FROM dipendenti WHERE stipendio > 1800.00"
12  mycursor.execute(comando)
13  mydatabase.commit()
14  print("Numero di righe cancellate: ", mycursor.rowcount)
15  #rowcount restituisce il numero di righe
16  #si cui il cursore ha operato
17
18  #usando un parametro e una condizione:
19  comando2 = "DELETE FROM dipendenti WHERE nome = %s"
20  valore2 = ("Marco",) #deve essere una tupla!
21  mycursor.execute(comando2, valore2)
22  mydatabase.commit()
23  print("Numero di righe cancellate: ", mycursor.rowcount)
```

Esercizi

7. [db-profumi] Scrivi un algoritmo che chieda all'utente se vuole creare un nuovo profumo oppure eliminarne uno già esistente (usa l'ID del prodotto per l'eliminazione, non il nome). Esegui, se l'utente ha digitato tutto correttamente, l'operazione richiesta. Se l'utente ha commesso errori, mostra un messaggio di errore e chiedi di nuovo cosa si vuole fare.
8. [db-cosmesi] Scrivi un algoritmo che elimini tutti i prodotti il cui ID è multiplo di 19.
9. [db-cosmesi] Scrivi un algoritmo che elimini tutti i prodotti del brand "Dior".

Modifica e aggiornamento con SQL

In SQL il comando **UPDATE** permette di modificare i dati di una tabella.

```
--Modifica di tutte le righe
UPDATE tabella SET campo = valore;

--Modifica di alcune righe, con condizione
UPDATE tabella SET campo = valore WHERE condizione;
```

Ad esempio:

```
UPDATE dipendenti SET stipendio = 2300.00;

UPDATE dipendenti SET stipendio = 1800.00 WHERE nomeDip =
↪  "Marco";
```

Modifica di dati con Python

Per modificare record, usiamo:

```
10  #cambia tutti i valori
11  comando = "UPDATE dipendenti SET stipendio = 2300.00"
12  mycursor.execute(comando)
13  mydatabase.commit()
14  print("Numero di righe modificate: ",mycursor.rowcount)
15
16  #usando un parametro e una condizione:
17  comando2 = "UPDATE dipendenti SET stipendio = 1800.00 WHERE
    ↪  nomeDip = %s"
18  valore2 = ("Marco",) #deve essere una tupla!
19  mycursor.execute(comando2, valore2)
20  mydatabase.commit()
21  print("Numero di righe modificate: ",mycursor.rowcount)
```

Esercizi

10. [db-profumi] Scrivi un algoritmo che permetta all'utente di modificare il nome di un profumo a tua scelta.
11. [db-profumi] Scrivi un algoritmo che richieda all'utente l'ID di un profumo e chieda successivamente quale attributo vuole modificare. Fai inserire il nuovo valore ed esegui l'operazione richiesta.
12. [db-cosmesi] Scrivi un algoritmo che richieda all'utente l'ID di un prodotto e chieda successivamente se vuole modificarne il prezzo. Se l'utente accetta, fai inserire un valore e aggiorna il record del prodotto.

Modifica di una tabella con SQL

La struttura di una tabella può essere modificata con il comando **ALTER** in combinazione con **ADD**, **DROP** e **RENAME**. Esempi:

```
--Aggiungere un campo ad una tabella
ALTER TABLE mansioni ADD COLUMN data_assegnazione DATE;
--Eliminare un campo
ALTER TABLE mansioni DROP COLUMN data_assegnazione;
--Nuovo indice
ALTER TABLE dipendenti ADD INDEX nome_completo(cognome,nome);
--Rinominare una tabella
ALTER TABLE OldName RENAME NewName;
--Eliminare una tabella
DROP TABLE mansioni;  --Elimina se esiste: DROP TABLE IF EXISTS
```

Il codice Python è facilmente intuibile, dati gli esempi precedenti.

Esercizi

13. [db-profumi] Aggiungi l'attributo "Nazione" all'entità "naso" e compila almeno due record con due diverse nazionalità.
14. [db-profumi] Elimina la colonna creata nell'esercizio precedente.
15. [db-profumi] Aggiungi la colonna "inVita" all'entità "naso" e compilala con un valore booleano che indichi se il naso è ancora vivo.
16. [db-cosmesi] Aggiungi l'attributo "Nazione" all'entità "brand" e compila almeno due record con due diverse nazionalità.
17. [db-cosmesi] Elimina la colonna creata nell'esercizio precedente.

Data Query Language

Per interrogare un database in SQL (cioè per eseguire una *query*) usiamo il comando **SELECT**, che esegue operazioni di:

- selezione (lavora su tutta una riga) e proiezione (lavora su alcuni campi di una riga);

```
--selezione
SELECT * FROM amici WHERE telefono="11235813";
--proiezione
SELECT cognome,nome FROM amici WHERE telefono="11235813";
```

- congiunzione (lavora su più tabelle, detta anche **JOIN**).

```
--attenzione alla notazione con il punto (dot notation)!
SELECT * FROM amici,telefoni
  WHERE amici.ID_Amico = telefoni.id_amico;
```

Interrogare un database con Python

Per vedere i risultati della nostra *query* usiamo il metodo `fetchall()` ("prendi tutto").

```
10  comando = "SELECT cognome,nome FROM amici WHERE
    ↪  telefono="11235813""
11  mycursor.execute(comando)
12
13  #salva in result ciò che si ottiene dalla query
14  result = mycursor.fetchall()
15
16  #se vogliamo prendere solo la prima
17  #riga, usiamo invece il metodo fetchone()
18
19  #stampa tutti i risultati
20  for riga in result: #i risultati vengono restituiti
21    print(riga)       #come LISTA DI TUPLE
```

Alias e campi calcolati

Se nella nostra query vogliamo mostrare un dato che non esiste nelle tabelle di partenza, come ad esempio un dato calcolato a partire dai dati presenti, possiamo usare una sintassi come la seguente:

```
SELECT pagaOraria, oreLavorate, pagaOraria*oreLavorate AS
↪  "Stipendio" FROM dipendenti;
```

Operatori di confronto

Gli operatori di confronto di SQL sono:

Operatore	Descrizione
>, <	Maggiore, minore
>=, <=	Maggiore/minore o uguale
<>	Diverso
=	Uguale
`BETWEEN x AND y`	Compreso tra `x` e `y`

Ad esempio:

```
SELECT nome FROM tabella WHERE numero > 35;
```

Ricorda di usare le virgolette quando confronti stringhe!

Operatori aritmetici e logici

Gli operatori aritmetici e quelli logici sono i soliti:

Operatore	Descrizione
`x+y`	Somma
`x-y`	Sottrazione
`x*y`	Moltiplicazione
`x/y`	Divisione
`x%y`	Modulo (resto)
`x^y`	Potenza
`sqrt(x)`	Radice quadrata

Operatore	Descrizione
`x AND y`	Congiunzione
`x OR y`	Disgiunzione
`NOT x`	Negazione

Esercizi

18. [db-cosmesi] Stampa a schermo l'elenco dei prodotti cosmetici che hanno una valutazione uguale o maggiore di 4.
19. [db-profumi] Per i nasi il cui anno di nascita e morte è indicato, calcola e mostra a schermo quanti anni hanno vissuto.
20. [db-cosmesi] Calcola la media delle valutazioni ricevute dai prodotti cosmetici.

Esercizi

21. [db-cosmesi] Estrai dal database i prodotti che costano più di 8 € e che hanno una valutazione superiore a 4.
22. [db-cosmesi] Estrai dal database i prodotti con una valutazione superiore a 4.2 adatti a pelli sensibili.
23. [db-profumi] Estrai dal database i profumi maschili usciti dopo il 2007.

Ordinare e limitare i risultati

Possiamo decidere l'ordinamento dei nostri risultati:

```
-- ordine alfanumerico
SELECT * FROM dipendenti ORDER BY cognomeDip;
-- ordine alfanumerico inverso
SELECT * FROM dipendenti ORDER BY nomeDip DESC;
```

Possiamo limitare ad un certo numero di risultati:

```
-- mostra solo i primi tre risultati
SELECT * FROM dipendenti LIMIT 3;

--possiamo anche usare le condizioni contemporaneamente
```

Esercizi

24. [db-profumi] Estrai dal database i profumi femminili usciti tra il 1991 e il 2000 e ordinali per data di uscita.
25. [db-cosmesi] Estrai dal database le creme solari e mostrale ordinate per valutazione.
26. [db-cosmesi] Estrai dal database le creme solari e mostrale ordinate per prezzo.

Esercizi

27. [db-cosmesi] Estrai dal database i tre prodotti più costosi.
28. [db-cosmesi] Estrai dal database i tre prodotti più economici.
29. [db-cosmesi] Estrai dal database i tre prodotti con la valutazione maggiore.
30. [db-cosmesi] Estrai dal database i tre prodotti con la valutazione minore.

Tipi di `JOIN`

Abbiamo già citato l'operazione di `JOIN` parlando di `SELECT`.

L'istruzione `JOIN`, con i suoi vari parametri, ci permette di eseguire operazioni insiemistiche sulle nostre tabelle:

Vedremo alcuni tipi particolari di `JOIN`:

- `INNER JOIN`;
- `LEFT JOIN` e `RIGHT JOIN`;
- `FULL JOIN`.

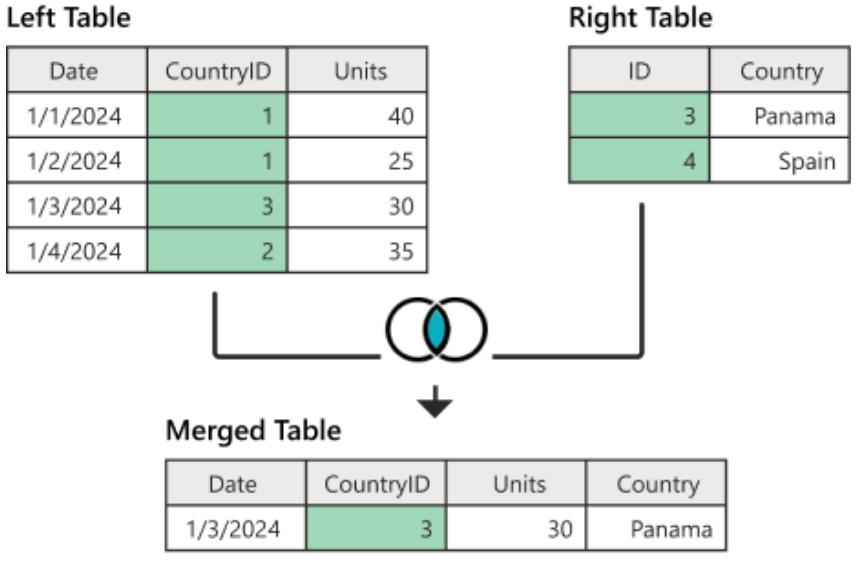
Codice Python per le JOIN

In tutta la spiegazione che segue, verranno portati come esempi solo i comandi SQL da inserire come stringhe nella variabile comando. Il resto del codice Python resta invariato:

```
1  import mysql.connector
2  mydb = mysql.connector.connect(host = "localhost",user =
   ↪  "root",password = "",database="mydatabase")
3  mycursor = mydb.cursor()
4  #il carattere \ mi permette di scrivere comandi su più righe
5  comando = "COMANDO IN SQL \
6             CHE VOGLIAMO   \
7             ESEGUIRE"
8  mycursor.execute(comando)
9  result = mycursor.fetchall()
10 for riga in result:
11   print(riga)
```

INNER JOIN

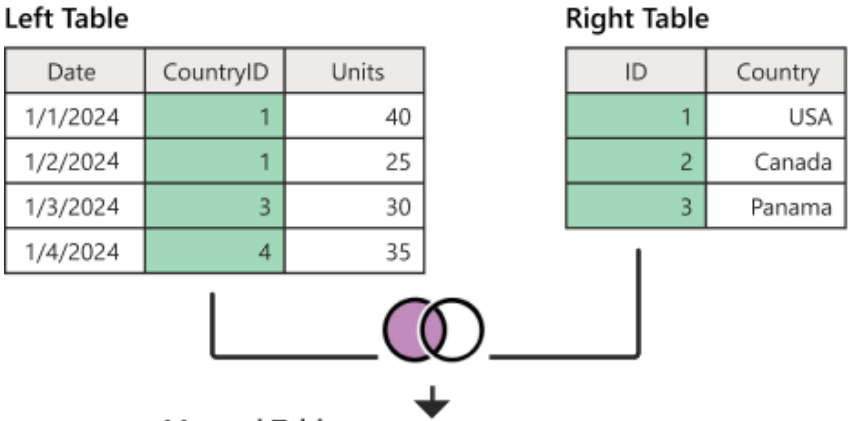
`INNER JOIN` corrisponde all'operazione insiemistica di intersezione.




```
SELECT * FROM leftTable
INNER JOIN rightTable
ON leftTable.key = rightTable.key --condizione
```

LEFT JOIN

`LEFT JOIN` prende tutta la tabella di sinistra e mostra, della tabella di destra, la parte per cui la condizione risulta vera.

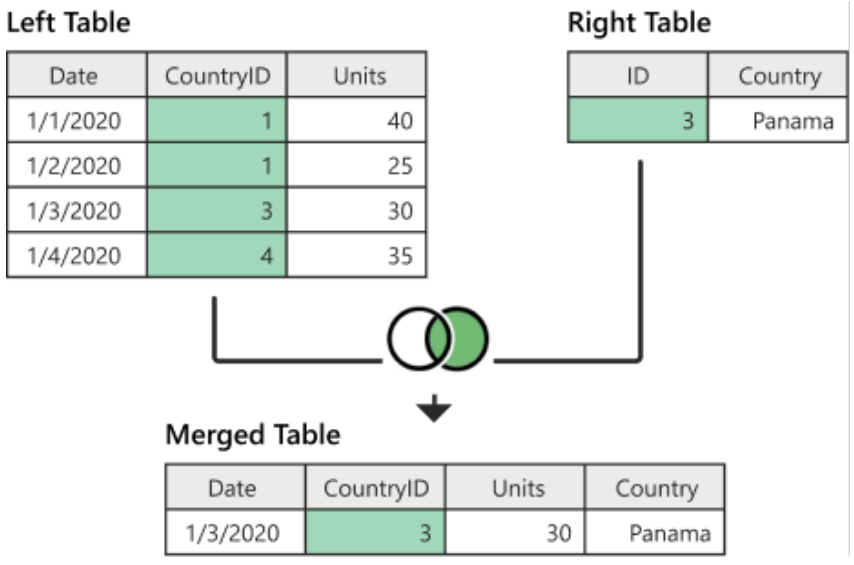



```sql
SELECT * FROM leftTable
LEFT JOIN rightTable ON leftTable.key = rightTable.key
```

Potremmo anche ottenere celle vuote (valore `NULL`).

RIGHT JOIN

`RIGHT JOIN` prende tutta la tabella di destra e mostra, della tabella di sinistra, la parte per cui la condizione risulta vera.




```
SELECT * FROM leftTable
RIGHT JOIN rightTable ON leftTable.key = rightTable.key
```

FULL JOIN

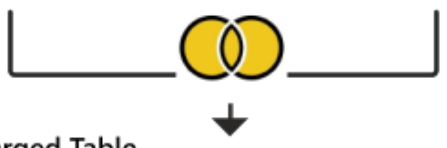
`FULL JOIN` corrisponde all'operazione insiemistica di unione.

Left Table

Date	CountryID	Units
1/1/2024	1	40
1/2/2024	1	25
1/3/2024	3	30
1/4/2024	2	35

Right Table

ID	Country
1	USA
2	Canada
3	Panama
4	Spain



Merged Table

Date	CountryID	Units	Country
1/1/2024	1	40	USA
1/2/2024	1	25	USA
1/4/2024	2	35	Canada
1/3/2024	3	30	Panama
null	*null*	*null*	Spain

```
SELECT * FROM leftTable
FULL JOIN rightTable ON leftTable.key = rightTable.key
```

Esercizi

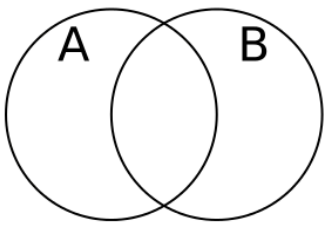
31. [db-profumi] Estrai dal database una tabella che mostri il nome di ogni profumo e il nome del brand che lo ha prodotto.
32. [db-cosmesi] Estrai dal database una tabella che mostri la categoria di appartenenza di tutti i prodotti con valutazione maggiore di 4.
33. [db-profumi] Estrai dal database a quanti anni di età ognuno dei nasi di cui conosci l'anno di nascita hanno creato i profumi a loro associati.

Schema riassuntivo

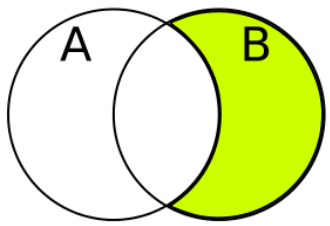
SQL JOINS

Arranged in a Karnaugh Map by Jason Charney (jrcharney@gmail.com)

No join (∅)

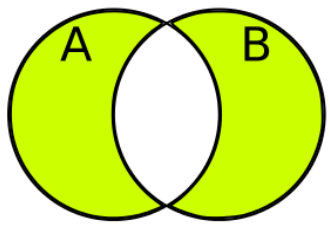


[Exclusive] Right Join (¬A)



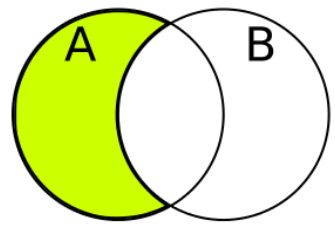

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

[Exclusive] Full Join (A⊕B)



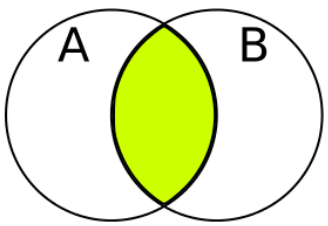

```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE B.key IS NULL
OR A.key IS NULL
```

[Exclusive] Left Join (¬B)



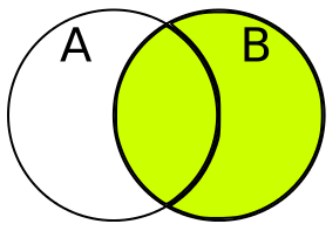

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

Inner Join (A∧B)



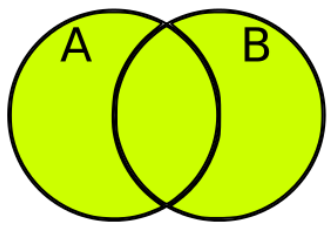

```
SELECT *
FROM A
INNER JOIN B ON A.key = B.key
```

[Inclusive] Right Join (B)



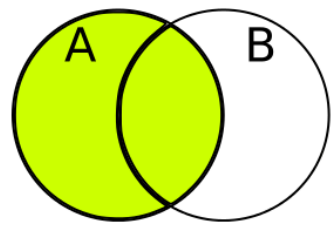

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
```

[Inclusive] Full Join (A∨B)




```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
```

[Inclusive] Left Join (A)




```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

Nella spiegazione precedente abbiamo utilizzato le versioni inclusive (riga inferiore dello schema).

Guide a SQL

Corsi video (italiano):

- Corso breve di CodeGrind
- Corso lungo di CodeGrind
- Corso completo di CodeGrind

Corsi web (inglese, con esercizi):

- Guida di W3Schools