

# Un database

Un **database** è una raccolta di dati organizzata in modo da facilitare l'accesso, nonché una gestione e un aggiornamento efficienti. Un database è costituito da **tabelle** che memorizzano le informazioni rilevanti. Ad esempio, useresti un database, se dovessi creare un sito web come YouTube, che contiene molte informazioni come video, nomi utente, password, commenti.



In questo corso impareremo come creare e interrogare database usando SQL!

# Tabelle database

Una tabella memorizza e visualizza i dati in un formato strutturato costituito da **colonne** e un database è costituito da **tabelle** che memorizzano le informazioni rilevanti. Ad esempio, useresti un database, se dovessi creare un sito web come YouTube, che contiene molte informazioni come video, nomi utente, password, commenti. **righe** simili a quelle visualizzate nei fogli di calcolo Excel. I database spesso contengono più tabelle, ciascuna progettata per uno scopo specifico. Ad esempio, immagina di creare una tabella di database di **nomi e numeri di telefono** . Innanzitutto, imposteremo colonne con i titoli *FirstName* , *LastName* e *TelephoneNumber* . Ogni tabella include il proprio insieme di campi, in base ai dati che memorizzerà.

FirstName	LastName	TelephoneNumber
John	Smith	715-555-1230
David	Williams	569-999-1719
Chloe	Anderson	715-777-2010
Emily	Adams	566-333-1223
James	Roberts	763-777-2956

Una tabella ha un numero specificato di colonne ma può avere un numero qualsiasi di righe.

## Chiavi primarie

Una chiave primaria (primary key) è un campo nella tabella che identifica in modo univoco i record della tabella. Caratteristiche principali della chiave primaria (primary key):

- Deve contenere una per ogni riga.
- Non può contenere valori.

Ad esempio, la nostra tabella contiene un record per ogni nome in una rubrica. Il numero univoco è una buona scelta per una chiave primaria nella tabella, poiché c'è sempre la possibilità che più di una persona abbia lo stesso nome.

ID	FirstName	LastName	TelephoneNumber
1	John	Smith	715-555-1230
2	David	Williams	569-999-1719
3	Chloe	Anderson	715-777-2010
4	Emily	Adams	566-333-1223
5	James	Roberts	763-777-2956

- Le tabelle sono limitate alla chiave primaria ciascuna.
  - Il valore della chiave primaria deve essere diverso per ogni riga.
- UNO**

# Cos'è SQL?


Una volta capito cos'è un database, capire SQL è facile. **SQL** è l'acronimo di **S**tructured **Q**uery **L**anguage . **SQL** viene utilizzato per accedere e manipolare un **database** . **MySQL** è un **programma** che comprende **SQL** . SQL può: - inserire, aggiornare o eliminare record in un database. - creare nuovi database, tabelle, stored procedure, viste. - recuperare i dati da un database, ecc.

SQL è uno standard ANSI (American National Standards Institute), ma esistono diverse versioni del linguaggio SQL.

La maggior parte dei programmi di database SQL ha le proprie estensioni proprietarie oltre allo standard SQL, ma tutti supportano i comandi principali.

## Comandi SQL di base

Il comando **SHOW TABLES** viene utilizzato per visualizzare tutte le tabelle nel database MySQL attualmente selezionato.



The screenshot shows a MySQL web interface with a top navigation bar containing icons and labels for 'Browse', 'Structure', 'SQL', 'Search', 'Insert', and 'Export'. Below the navigation bar is a 'Show query box' label. A yellow message box states 'Your SQL query has been executed successfully.' Below this, the command 'SHOW TABLES' is displayed in a purple font. At the bottom, there is a '+ Options' section with a dropdown menu showing 'Tables\_in\_my\_database' and a list of tables including 'customers'.

Show query box

Your SQL query has been executed successfully.

SHOW TABLES

+ Options

Tables\_in\_my\_database

customers

Per il nostro esempio, abbiamo creato un database, **my\_database** , con una tabella chiamata **clienti** .

## Comandi SQL di base

**SHOW COLUMNS** visualizza le informazioni sulle colonne in una determinata tabella. Il seguente esempio mostra le colonne nella nostra tabella dei **clienti** :

**SHOW COLUMNS FROM** customers

**Risultato:**

+ Options

Field	Type	Null	Key	Default	Extra
ID	int(11)	NO	PRI	NULL	
FirstName	varchar(60)	NO		NULL	
LastName	varchar(60)	NO		NULL	
City	varchar(30)	NO		NULL	
ZipCode	int(10)	NO		NULL	

SHOW COLUMNS visualizza i seguenti valori per ciascuna colonna della tabella:

**Field** : nome colonna

**Type** : tipo di dati colonna

**Null**: permette valore Null?

**Key** : indica se la colonna è indicizzata

**Default** : valore predefinito assegnato alla colonna

**Extra** : può contenere qualsiasi informazione aggiuntiva disponibile su un data colonna

Anche le colonne per la tabella dei **clienti** sono state create utilizzando lo strumento PHPMyAdmin.

## Istruzione SELECT

L'istruzione **SELECT** viene utilizzata per selezionare i dati da un database.

Il risultato viene memorizzato in una tabella dei risultati, denominata **set di risultati**.

Una **query** può recuperare le informazioni dalle colonne selezionate o da tutte le colonne nella tabella. Per creare una semplice istruzione SELECT, specificare il nome della colonna o delle colonne necessarie dalla tabella. **Sintassi dell'istruzione SQL SELECT:**

**SELECT** column\_list **FROM** table\_name

- **column\_list** include una o più colonne da cui vengono recuperati i dati (si può usare \* per tutte)

- **table-name** è il nome della tabella da cui vengono recuperate le informazioni

**Di seguito sono riportati i dati dalla nostra tabella dei customers:**

La seguente istruzione SQL seleziona il **FirstName** dalla tabella dei **customers** :

SELECT \* FROM customers

SELECT FirstName FROM customers:

FirstName
John
David
Chloe
Emily
James

Un'istruzione SELECT recupera zero o più righe da una o più tabelle di database.

## Query multiple

SQL consente di eseguire più query o comandi contemporaneamente. La seguente istruzione SQL seleziona le colonne **FirstName** e **City** dalla tabella dei customers:

```
SELECT FirstName FROM customers;  
SELECT City FROM customers;
```

City
New York
Los Angeles
Chicago
Houston
Philadelphia

Ricordarsi di terminare ogni istruzione SQL con un punto e virgola per indicare che l'istruzione è completa e pronta per essere interpretata.

In questo tutorial, useremo il punto e virgola alla fine di ogni istruzione SQL



## **Maiuscole / minuscole**

SQL non fa distinzione tra maiuscole e **minuscole** .

Le seguenti istruzioni sono equivalenti e produrranno lo stesso risultato:

**select** City from customers;

**SELECT** City FROM customers;

**sElEcT** City From customers;

È pratica comune scrivere tutti i comandi SQL in **maiuscolo** .

## Regole di sintassi

Una singola istruzione SQL può essere inserita su una o più righe di testo. Inoltre, più istruzioni SQL possono essere combinate su una singola riga di testo.

Gli spazi bianchi e più righe vengono ignorati in SQL.

Ad esempio, la seguente query è assolutamente corretta.

ID	FirstName	LastName	City	ZipCode
1	John	Smith	New York	10199
2	David	Williams	Los Angeles	90052
3	Chloe	Anderson	Chicago	60607
4	Emily	Adams	Houston	77201
5	James	Roberts	Philadelphia	19104

```
SELECT City
```

```
FROM customers;
```

Tuttavia, si consiglia di evitare spazi e linee bianche non necessarie.

In combinazione con la corretta spaziatura e rientro, la suddivisione dei comandi in righe logiche renderà le istruzioni SQL molto più facili da leggere e mantenere.

## Selezione di più colonne

Come accennato in precedenza, l'istruzione SQL SELECT recupera i record dalle tabelle nel database SQL. Puoi selezionare più colonne della tabella contemporaneamente.

Elenca semplicemente i nomi delle colonne, separati da **virgole**

SELECT **FirstName, LastName, City**

FirstName	LastName	City
John	Smith	New York
David	Williams	Los Angeles
Chloe	Anderson	Chicago
Emily	Adams	Houston
James	Roberts	Philadelphia

Non inserire una virgola dopo il nome **dell'ultima** colonna.

## La parola chiave **DISTINCT**

In situazioni in cui sono presenti più

FirstName
John
David
Chloe
Emily
James

record duplicati in una tabella, potrebbe essere più sensato restituire solo record univoci, invece di recuperare i duplicati. La parola chiave SQL **DISTINCT** viene utilizzata insieme a **SELECT** per eliminare tutti i record duplicati e restituire solo quelli univoci.

La sintassi di base di **DISTINCT** è la seguente:

```
SELECT DISTINCT nome_colonna1, nome_colonna2 FROM nome_tabella;
```

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia
6	Andrew	Thomas	New York
7	Daniel	Harris	New York
8	Charlotte	Walker	Chicago
9	Samuel	Clark	San Diego
10	Anthony	Young	Los Angeles

← Notare che ci sono nomi di **città** duplicati . La seguente istruzione SQL seleziona solo valori distinti dalla colonna Città:

SELECT **DISTINCT** City FROM clienti;

Ciò produrrebbe il seguente risultato. Le voci duplicate sono state rimosse.

La parola chiave DISTINCT recupera solo i valori univoci.

City
New York
Los Angeles
Chicago
Houston
Philadelphia
San Diego

## La parola chiave LIMIT

Per impostazione predefinita, vengono restituiti tutti i risultati che soddisfano le condizioni specificate nell'istruzione SQL. Tuttavia, a volte è necessario recuperare solo un sottoinsieme di record. In MySQL, ciò si ottiene utilizzando la parola chiave **LIMIT**. La sintassi per **LIMIT** è la seguente:

SELEZIONA l'elenco delle colonne DA nome\_tabella **LIMIT** [numero di record];

Ad esempio, possiamo recuperare i primi **cinque** record dalla tabella dei **clienti**.

SELECT ID, FirstName, LastName, City FROM customers **LIMIT 5**;

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia

Per impostazione predefinita, vengono restituiti tutti i risultati che soddisfano le condizioni specificate nell'istruzione SQL.

Puoi anche prelevare una serie di record da un particolare **offset** .

Nell'esempio seguente, raccogliamo **quattro** record, a partire dalla **terza** posizione:

```
SELECT ID, FirstName, LastName, City FROM customers LIMIT 3,4;
```

ID	FirstName	LastName	City
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia
6	Andrew	Thomas	New York
7	Daniel	Harris	New York

Il motivo per cui produce risultati a partire dall'ID numero quattro, e non tre, è che MySQL inizia a contare da **zero** , il che significa che l'offset della prima riga è 0, non 1.



## Fully Qualified Names (Nome Completo)

In SQL, è possibile fornire il nome della tabella prima del nome della colonna, separandoli con un **punto** .

**Le seguenti dichiarazioni sono equivalenti:**

```
SELECT City FROM customers;
```

```
SELECT customers.City FROM customers;
```

**Il termine per la sintassi sopra menzionata è chiamato "Fully Qualified Names (Nome Completo)" della colonna.**

Questa forma di scrittura è particolarmente utile quando si lavora con più tabelle che possono condividere gli stessi nomi di colonna.

## Order By

**ORDER BY** viene utilizzato con **SELECT** per **ordinare** i dati restituiti. L'esempio seguente ordina la nostra tabella dei **clienti in** base alla colonna *FirstName* .

```
SELECT * FROM customers ORDER BY FirstName;
```

ID	FirstName	LastName	City
6	Andrew	Thomas	New York
10	Anthony	Young	Los Angeles
8	Charlotte	Walker	Chicago
3	Chloe	Anderson	Chicago
7	Daniel	Harris	New York
2	David	Williams	Los Angeles
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia
1	John	Smith	New York
9	Samuel	Clark	San Diego

**Risultato:** come puoi vedere, le righe sono ordinate **alfabeticamente in** base alla colonna **FirstName** .

Per impostazione predefinita, la parola chiave **ORDER BY** ordina i risultati in ordine **crescente** .

## Ordinamento di più colonne

ORDER BY può ordinare i dati recuperati da più colonne. Quando si utilizza ORDER BY con più di una colonna, basta separare l'elenco di colonne con **virgole** .

Ecco la tabella dei **clienti** , che mostra i seguenti record:

ID	FirstName	LastName	Age
1	John	Smith	35
2	David	Smith	23
3	Chloe	Anderson	27
4	Emily	Adams	34
5	James	Roberts	31
6	Andrew	Thomas	45
7	Daniel	Harris	30

Per ordinare per **lastName** ed **Age** :

```
SELECT * FROM customers ORDER BY lastName , Age;
```

ID	FirstName	LastName	Age
4	Emily	Adams	34
3	Chloe	Anderson	27
7	Daniel	Harris	30
5	James	Roberts	31
2	David	Smith	23
1	John	Smith	35
6	Andrew	Thomas	45

Questa istruzione ORDER BY restituisce il seguente risultato:poiché abbiamo due **Smith** , verranno ordinati in ordine crescente dalla colonna **Età** .

Il comando ORDER BY inizia a ordinare nella stessa sequenza delle colonne. Ordinerà in base alla prima colonna elencata, quindi alla seconda e così via.

La dichiarazione WHERE

La clausola **WHERE** viene utilizzata per estrarre solo i record che soddisfano un criterio specificato.

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia
6	Andrew	Thomas	New York
7	Daniel	Harris	New York
8	Charlotte	Walker	Chicago
9	Samuel	Clark	San Diego
10	Anthony	Young	Los Angeles

## La sintassi per la clausola WHERE:

```
SELECT elenco_colonne  
FROM nome_tabella  
WHERE condizione;
```

Considera la seguente tabella: Nella tabella sopra, per SELEZIONARE un record specifico:

```
SELECT * FROM clienti  
WHERE ID = 7;
```

Prova tu stesso

ID	FirstName	LastName	City
7	Daniel	Harris	New York

La clausola WHERE viene utilizzata per estrarre solo i record che soddisfano un criterio specificato.

## Operatori logici

Gli operatori logici possono essere usati per combinare due valori booleani e restituire un risultato di vero, falso o nullo.

Possono essere utilizzati i seguenti operatori:

Operator	Description
AND	TRUE if <b>both</b> expressions are TRUE
OR	TRUE if <b>either</b> expression is TRUE
IN	TRUE if the operand is equal to one of a list of expressions
NOT	Returns TRUE if expression is not TRUE

Quando si recuperano dati usando un'istruzione SELECT, usare gli operatori logici nella clausola WHERE per combinare più condizioni.

Se vuoi selezionare le righe che soddisfano tutte le condizioni date, usa l'operatore logico AND.

ID	FirstName	LastName	Age
1	John	Smith	35
2	David	Williams	23
3	Chloe	Anderson	27
4	Emily	Adams	34
5	James	Roberts	31
6	Andrew	Thomas	45
7	Daniel	Harris	30

Per trovare i nomi dei clienti tra i 30 e i 40 anni di età, impostare la query come visto qui:

```
SELECT ID, FirstName, LastName, Age FROM customers WHERE Age >= 30 AND Age <= 40;
```

Il risultato è il seguente output:

ID	FirstName	LastName	Age
1	John	Smith	35
4	Emily	Adams	34
5	James	Roberts	31
7	Daniel	Harris	30

È possibile combinare tutte le condizioni necessarie per ottenere i risultati desiderati.

## OR

Se vuoi selezionare le righe che soddisfano almeno una delle condizioni date, puoi usare l'operatore OR logico.

La seguente tabella descrive come funziona l'operatore OR logico:

Condition1	Condition2	Result
True	True	True
True	False	True
False	True	True
False	False	False

Per esempio, se volete trovare i clienti che vivono a New York o a Chicago, la query sarà simile a questa:

```
SELECT * FROM customers  
WHERE City = 'New York' OR City = 'Chicago';
```

Risultato:

ID	FirstName	LastName	City
1	John	Smith	New York
3	Chloe	Anderson	Chicago
6	Andrew	Thomas	New York
7	Daniel	Harris	New York
8	Charlotte	Walker	Chicago

You can OR two or more conditions.

## Combinazione di AND e OR

Le condizioni SQL AND e OR possono essere combinate per verificare più condizioni in una query. Questi due operatori sono chiamati operatori congiuntivi.

Quando si combinano queste condizioni, è importante usare le parentesi, in modo da conoscere l'ordine di valutazione di ogni condizione.

Si consideri la seguente tabella:

ID	FirstName	LastName	City	Age
1	John	Smith	New York	35
2	David	Williams	Los Angeles	23
3	Chloe	Anderson	Chicago	27
4	Emily	Adams	Houston	34
5	James	Roberts	Philadelphia	31
6	Andrew	Thomas	New York	45
7	Daniel	Harris	New York	30
8	Charlotte	Walker	Chicago	35
9	Samuel	Clark	San Diego	20
10	Anthony	Young	Los Angeles	33

La dichiarazione seguente seleziona tutti i clienti della città "New York" E con l'età uguale a "30" O "35":

```
SELECT * FROM customers  
WHERE City = 'New York'  
AND (Age=30 OR Age=35);
```

Risultato:

ID	FirstName	LastName	City	Age
1	John	Smith	New York	35
7	Daniel	Harris	New York	30

Potete annidare tutte le condizioni di cui avete bisogno.



## L'operatore IN

L'operatore IN si usa quando si vuole confrontare una colonna con più di un valore.

Per esempio, potresti aver bisogno di selezionare tutti i clienti di New York, Los Angeles e Chicago. Con la condizione OR, il tuo SQL sarebbe come questo:

```
SELECT * FROM customers
```

```
WHERE City = 'New York'
```

```
OR City = 'Los Angeles'
```

```
OR City = 'Chicago';
```

Risultato:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago
6	Andrew	Thomas	New York
7	Daniel	Harris	New York
8	Charlotte	Walker	Chicago
10	Anthony	Young	Los Angeles

L'operatore IN si usa quando si vuole confrontare una colonna con più di un valore.

Potete ottenere lo stesso risultato con una sola condizione IN, invece delle condizioni OR multiple:

```
SELECT * FROM customers
```

```
WHERE City IN ('New York', 'Los Angeles', 'Chicago');
```

Anche questo produrrebbe lo stesso risultato:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago
6	Andrew	Thomas	New York
7	Daniel	Harris	New York
8	Charlotte	Walker	Chicago
10	Anthony	Young	Los Angeles

Notate l'uso delle parentesi nella sintassi.

## L'operatore NOT IN

L'operatore NOT IN permette di escludere una lista di valori specifici dall'insieme dei risultati.

Se aggiungiamo la parola chiave NOT prima di IN nella nostra query precedente, i clienti che vivono in quelle città saranno esclusi:

```
SELECT * FROM customers  
WHERE City NOT IN ('New York', 'Los Angeles', 'Chicago');
```

Risultato:

ID	FirstName	LastName	City
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia
9	Samuel	Clark	San Diego

L'operatore NOT IN permette di escludere una lista di valori specifici dall'insieme dei risultati.

## La funzione CONCAT

La funzione CONCAT è usata per concatenare due o più valori di testo e restituisce la stringa concatenata.

Concateniamo il FirstName con la City, separandoli con una virgola:  
`SELECT CONCAT(FirstName, ', ', City) FROM customers;`

Il risultato di uscita è:

<b>CONCAT(FirstName, ', ', City)</b>
John, New York
David, Los Angeles
Chloe, Chicago
Emily, Houston
James, Philadelphia
Andrew, New York
Daniel, New York
Charlotte, Chicago
Samuel, San Diego
Anthony, Los Angeles

La funzione CONCAT() prende due o più parametri.

## La parola chiave AS

Una concatenazione dà come risultato una nuova colonna. Il nome predefinito della colonna sarà la funzione CONCAT.

Puoi assegnare un nome personalizzato alla colonna risultante usando la parola chiave AS:

```
SELECT CONCAT(FirstName,' ', City) AS new_column  
FROM customers;
```

E quando si esegue la query, il nome della colonna sembra essere cambiato.

new_column
John, New York
David, Los Angeles
Chloe, Chicago
Emily, Houston
James, Philadelphia
Andrew, New York
Daniel, New York
Charlotte, Chicago
Samuel, San Diego
Anthony, Los Angeles

Una concatenazione dà come risultato una nuova colonna.

## Operatori aritmetici

Gli operatori aritmetici eseguono operazioni aritmetiche su operandi numerici. Gli operatori aritmetici includono aggiunta (+), sottrazione (-), moltiplicazione (\*) e divisione (/).

La seguente tabella dei dipendenti mostra i nomi e gli stipendi dei dipendenti:

ID	FirstName	LastName	Salary
1	John	Smith	2000
2	David	Williams	1500
3	Chloe	Anderson	3000
4	Emily	Adams	4500
5	James	Roberts	2000
6	Andrew	Thomas	2500
7	Daniel	Harris	3000
8	Charlotte	Walker	3500
9	Samuel	Clark	4000
10	Anthony	Young	5000

L'esempio seguente aggiunge 500 allo stipendio di ogni dipendente e seleziona il risultato:

```
SELECT ID, FirstName, LastName, Salary+500 AS Salary  
FROM employees;
```

Risultato:

ID	FirstName	LastName	Salary
1	John	Smith	2500
2	David	Williams	2000
3	Chloe	Anderson	3500
4	Emily	Adams	5000
5	James	Roberts	2500
6	Andrew	Thomas	3000
7	Daniel	Harris	3500
8	Charlotte	Walker	4000
9	Samuel	Clark	4500
10	Anthony	Young	5500

Le parentesi possono essere usate per forzare un'operazione a prendere la priorità su qualsiasi altro operatore. Sono anche usate per migliorare la leggibilità del codice.

# La funzione UPPER

La funzione UPPER converte tutte le lettere della stringa specificata in maiuscolo.  
La funzione LOWER converte la stringa in minuscolo.

La seguente query SQL seleziona tutti i cognomi come maiuscoli:  
SELECT FirstName, UPPER(LastName) AS LastName  
FROM employees;

Risultato:

FirstName	LastName
John	SMITH
David	WILLIAMS
Chloe	ANDERSON
Emily	ADAMS
James	ROBERTS
Andrew	THOMAS
Daniel	HARRIS
Charlotte	WALKER
Samuel	CLARK
Anthony	YOUNG

Se ci sono caratteri nella stringa che non sono lettere, questa funzione non avrà alcun effetto su di essi.

## SQRT e AVG

La funzione SQRT restituisce la radice quadrata del valore dato nell'argomento.

Calcoliamo la radice quadrata di ogni salario:

```
SELECT Salary, SQRT(Salary)
```

```
FROM employees;
```

SQL

Risultato:

Salary	SQRT(Salary)
2000	44.721359549995796
1500	38.72983346207417
3000	54.772255750516614
4500	67.08203932499369
2000	44.721359549995796
2500	50
3000	54.772255750516614
3500	59.16079783099616
4000	63.245553203367585
5000	70.71067811865476

Allo stesso modo, la funzione AVG restituisce il valore medio di una colonna numerica:

```
SELECT AVG(Salary) FROM employees;
```

Risultato:

AVG(Salary)
3100.0000

Un altro modo per fare il SQRT è usare POWER con l'esponente 1/2. Tuttavia, SQRT sembra funzionare più velocemente di POWER in questo caso.

## La funzione SUM

La funzione SUM viene utilizzata per calcolare la somma dei valori di una colonna.

Per esempio, per ottenere la somma di tutti gli stipendi nella tabella degli impiegati, la nostra query SQL sarebbe come questa:

```
SELECT SUM(Salary) FROM employees;
```

Risultato:

SUM(Salary)
31000

La somma di tutti gli stipendi dei dipendenti è 31000.



## Sottoquery

Una subquery è una query dentro un'altra query.

Consideriamo un esempio. Potremmo aver bisogno della lista di tutti gli impiegati i cui stipendi sono superiori alla media.

Per prima cosa, calcoliamo la media:

```
SELECT AVG(Salary) FROM employees;
```

Poiché conosciamo già la media, possiamo usare un semplice WHERE per elencare gli stipendi che sono superiori a quel numero.

```
SELECT FirstName, Salary FROM employees
```

```
WHERE Salary > 3100
```

```
ORDER BY Salary DESC;
```

Risultato:

FirstName	Salary
Anthony	5000
Emily	4500
Samuel	4000
Charlotte	3500

La parola chiave DESC ordina i risultati in ordine decrescente.

Allo stesso modo, ASC ordina i risultati in ordine crescente.

Una singola sottoquery restituirà più facilmente lo stesso risultato.

```
SELECT FirstName, Salary FROM employees
```

```
WHERE Salary > (SELECT AVG(Salary) FROM employees)
```

```
ORDER BY Salary DESC;
```

SQL

Si otterrà lo stesso risultato.

FirstName	Salary
Anthony	5000
Emily	4500
Samuel	4000
Charlotte	3500

Racchiudere la sottoquery tra parentesi.

Inoltre, notate che non c'è nessun punto e virgola alla fine della subquery, poiché fa parte della nostra singola query.

## L'operatore Like

La parola chiave LIKE è utile quando si specifica una condizione di ricerca nella clausola WHERE.

```
SELECT column_name(s)
```

```
FROM table_name
```

```
WHERE column_name LIKE pattern;
```

Il pattern matching SQL vi permette di usare "\_" per abbinare qualsiasi singolo carattere e "%" per abbinare un numero arbitrario di caratteri (inclusi i caratteri zero).

Per esempio, per selezionare i dipendenti il cui nome inizia con la lettera A, si può usare la seguente query:

```
SELECT * FROM employees
```

```
WHERE FirstName LIKE 'A%';
```

SQL

Risultato:

ID	FirstName	LastName	Salary
6	Andrew	Thomas	2500
10	Anthony	Young	5000

Come altro esempio, la seguente query SQL seleziona tutti gli impiegati con un LastName che finisce con la lettera "s":

```
SELECT * FROM employees
```

```
WHERE LastName LIKE '%s';
```

SQL

Risultato:

ID	FirstName	LastName	Salary
2	David	Williams	1500
4	Emily	Adams	4500
5	James	Roberts	2000
6	Andrew	Thomas	2500
7	Daniel	Harris	3000

Il carattere jolly % può essere usato più volte all'interno dello stesso schema.

## La funzione MIN

La funzione MIN è usata per restituire il valore minimo di un'espressione in un'istruzione SELECT.

Per esempio, potreste voler conoscere lo stipendio minimo tra gli impiegati.

```
SELECT MIN(Salary) AS Salary FROM employees;
```

SQL

Salary
1500

Tutte le funzioni SQL possono essere combinate insieme per creare una singola espressione.

## Unire le tabelle

Tutte le query mostrate fino ad ora hanno selezionato da una sola tabella alla volta.

Una delle caratteristiche più vantaggiose di SQL è la capacità di combinare i dati di due o più tabelle.

Nelle due tabelle che seguono, la tabella chiamata clienti memorizza informazioni sui clienti:

ID	Name	City	Age
1	John	New York	35
2	David	Los Angeles	23
3	Chloe	Chicago	27
4	Emily	Houston	34
5	James	Philadelphia	31

La tabella degli ordini memorizza le informazioni sui singoli ordini con il loro importo corrispondente:

ID	Name	Customer_ID	Amount
1	Book	3	5000
2	Box	5	3000
3	Toy	2	4500
4	Flowers	4	1800
5	Cake	1	6700

In SQL, "unire le tabelle" significa combinare i dati di due o più tabelle. Un'unione di tabelle crea una tabella temporanea che mostra i dati delle tabelle unite.

## Unire le tabelle

Invece di memorizzare il nome del cliente in entrambe le tabelle, la tabella ordini contiene un riferimento all'ID del cliente che appare nella tabella clienti. Questo approccio è più efficiente, invece di memorizzare gli stessi valori di testo in entrambe le tabelle.

Per essere in grado di selezionare i dati corrispondenti da entrambe le tabelle, abbiamo bisogno di unirle su questa condizione.

Per unire le due tabelle, specificarle come elenco separato da virgole nella clausola FROM:

```
SELECT customers.ID, customers.Name, orders.Name, orders.Amount
```

```
FROM customers, orders
```

```
WHERE customers.ID=orders.Customer_ID
```

```
ORDER BY customers.ID;
```

Ogni tabella contiene colonne "ID" e "Nome", quindi per selezionare l'ID e il Nome corretti, vengono usati nomi completamente qualificati.

Si noti che la clausola WHERE "unisce" le tabelle alla condizione che l'ID della tabella clienti sia uguale al customer\_ID della tabella ordini.

Risultato:

ID	Name	Name	Amount
1	John	Cake	6700
2	David	Toy	4500
3	Chloe	Book	5000
4	Emily	Flowers	1800
5	James	Box	3000

I dati restituiti mostrano gli ordini dei clienti e il loro importo corrispondente.

Specifica più nomi di tabelle nel FROM separandoli con una virgola.

## Nomi personalizzati

Anche i nomi personalizzati possono essere usati per le tabelle. Puoi abbreviare le dichiarazioni di unione dando alle tabelle dei "nicknames":

```
SELECT ct.ID, ct.Name, ord.Name, ord.Amount  
FROM customers AS ct, orders AS ord  
WHERE ct.ID=ord.Customer_ID  
ORDER BY ct.ID;
```

Come potete vedere, abbiamo abbreviato i nomi delle tabelle man mano che li abbiamo usati nella nostra query.

Tipi di join

I seguenti sono i tipi di JOIN che possono essere usati in MySQL:

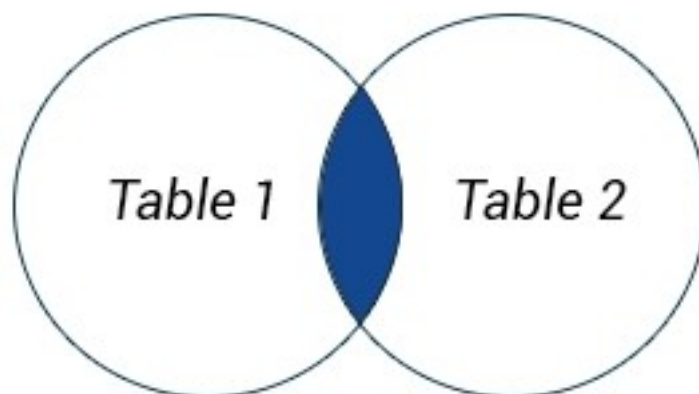
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN

INNER JOIN è equivalente a JOIN. Restituisce le righe quando c'è una corrispondenza tra le tabelle.

Sintassi:

```
SELECT column_name(s)  
FROM table1 INNER JOIN table2  
ON table1.column_name=table2.column_name;
```

Notate la parola chiave ON per specificare la condizione di inner join.  
L'immagine qui sotto dimostra come funziona la INNER JOIN:



Vengono restituiti solo i record che corrispondono alla condizione di unione.

# LEFT JOIN

La LEFT JOIN restituisce tutte le righe della tabella di sinistra, anche se non ci sono corrispondenze nella tabella di destra.

Questo significa che se non ci sono corrispondenze per la clausola ON nella tabella di destra, la join restituirà comunque le righe della prima tabella nel risultato.

La sintassi di base della LEFT JOIN è la seguente:

```
SELECT table1.column1, table2.column2...
```

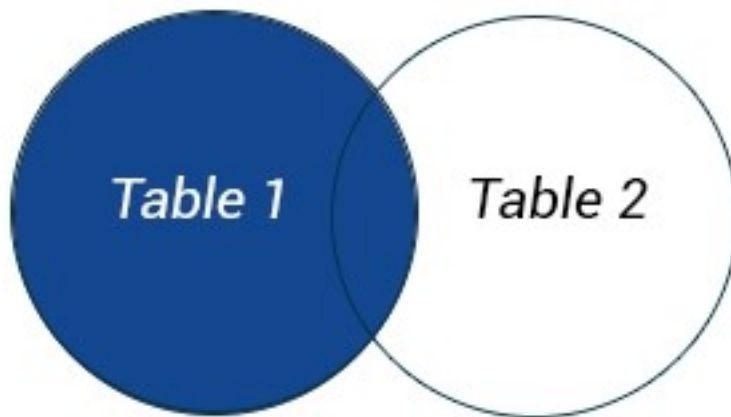
```
FROM table1 LEFT OUTER JOIN table2
```

```
ON table1.column_name = table2.column_name;
```

SQL

The OUTER keyword is optional, and can be omitted.

L'immagine qui sotto dimostra come funziona la LEFT JOIN:



Considerate le seguenti tabelle.

clienti:

ID	Name	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago
4	Emily	Adams	Houston
5	James	Roberts	Philadelphia
6	Andrew	Thomas	New York
7	Daniel	Harris	New York

articoli:

ID	Name	Cost	Seller_id
39	Book	5.9	1
24	Box	2.99	1
72	Toy	23.7	2
36	Flowers	50.75	2
18	T-Shirt	22.5	3
16	Notebook	150.22	4
74	Perfume	90.9	6

La seguente istruzione SQL restituirà tutti i clienti e gli articoli che potrebbero avere:

```
SELECT customers.Name, items.Name  
FROM customers LEFT OUTER JOIN items  
ON customers.ID=items.Seller_id;  
SQL
```

Risultato:

Name	Name
John	Book
John	Box
David	Toy
David	Flowers
Chloe	T-Shirt
Emily	Notebook
Andrew	Perfume
James	NULL
Daniel	NULL

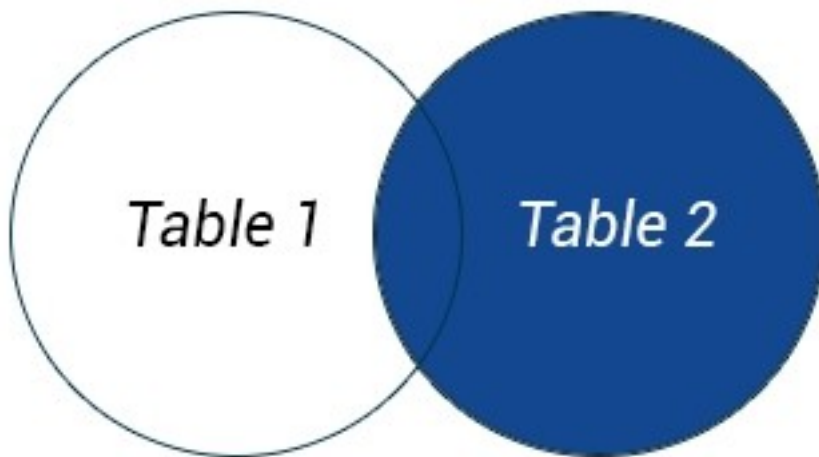
L'insieme dei risultati contiene tutte le righe della tabella di sinistra e i dati corrispondenti della tabella di destra.

Se non viene trovata alcuna corrispondenza per una particolare riga, viene restituito NULL.



## RIGHT JOIN

La RIGHT JOIN restituisce tutte le righe della tabella di destra, anche se non ci sono corrispondenze nella tabella di sinistra.



La sintassi di base della RIGHT JOIN è la seguente:

```
SELECT table1.column1, table2.column2...  
FROM table1 RIGHT OUTER JOIN table2  
ON table1.column_name = table2.column_name;
```

Di nuovo, la parola chiave OUTER è opzionale e può essere omessa.

Consideriamo lo stesso esempio della nostra lezione precedente, ma questa volta con una RIGHT JOIN:

```
SELECT customers.Name, items.Name FROM customers  
RIGHT JOIN items ON customers.ID=items.Seller_id;
```

Risultato:

Name	Name
John	Book
John	Box
David	Toy
David	Flowers
Chloe	T-Shirt
Emily	Notebook
Andrew	Perfume

La RIGHT JOIN restituisce tutte le righe della tabella di destra (articoli), anche se non ci sono corrispondenze nella tabella di sinistra (clienti).

Ci sono altri tipi di join nel linguaggio SQL, ma non sono supportati da MySQL.

# Operazione set

Occasionalmente, potresti aver bisogno di combinare i dati da più tabelle in un unico set di dati completo. Questo può essere per tabelle con dati simili all'interno dello stesso database o forse c'è la necessità di combinare dati simili tra database o anche tra server.

Per fare questo, usa gli operatori UNION e UNION ALL.

UNION combina più set di dati in un unico set di dati e rimuove qualsiasi duplicato esistente. UNION ALL combina più set di dati in un unico set di dati, ma non rimuove le righe duplicate. UNION ALL è più veloce di UNION, poiché non esegue l'operazione di rimozione dei duplicati sull'insieme di dati.

## UNION

L'operatore UNION è usato per combinare i set di risultati di due o più istruzioni SELECT.

Tutte le istruzioni SELECT all'interno dell'UNIONE devono avere lo stesso numero di colonne. Le colonne devono anche avere gli stessi tipi di dati. Inoltre, le colonne in ogni istruzione SELECT devono essere nello stesso ordine.

La sintassi di UNION è la seguente:

```
SELECT column_name(s) FROM table1
```

```
UNION
```

```
SELECT column_name(s) FROM table2;
```

```
SQL
```

Ecco la prima di due tabelle:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles

Ed ecco il Secondo:

ID	FirstName	LastName	City
1	James	Roberts	Philadelphia
2	David	Williams	Los Angeles

```
SELECT ID, FirstName, LastName, City FROM First
```

```
UNION
```

```
SELECT ID, FirstName, LastName, City FROM Second;
```

La tabella risultante sarà simile a questa:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
1	James	Roberts	Philadelphia

Come puoi vedere, i duplicati sono stati rimossi.

CONSIGLIO:

Se le tue colonne non corrispondono esattamente in tutte le query, puoi usare un valore NULL (o qualsiasi altro) come:

```
SELECT FirstName, LastName, Company FROM businessContacts  
UNION  
SELECT FirstName, LastName, NULL FROM otherContacts;
```

L'operatore UNION si usa per combinare gli insiemi di risultati di due o più istruzioni SELECT.

## UNION ALL

UNION ALL seleziona tutte le righe da ogni tabella e le combina in un'unica tabella.

Il seguente statement SQL usa UNION ALL per selezionare i dati dalla prima e dalla seconda tabella:

```
SELECT ID, FirstName, LastName, City FROM First  
UNION ALL  
SELECT ID, FirstName, LastName, City FROM Second;
```

La tabella risultante:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
1	James	Roberts	Philadelphia
2	David	Williams	Los Angeles

Come potete vedere, l'insieme dei risultati include anche le righe duplicate.

## Inserire i dati

Le tabelle SQL memorizzano i dati in righe, una riga dopo l'altra. L'istruzione INSERT INTO è usata per aggiungere nuove righe di dati ad una tabella nel database.

La sintassi SQL INSERT INTO è la seguente:

```
INSERT INTO table_name
```

```
VALUES (value1, value2, value3,...);
```

Assicurati che l'ordine dei valori sia lo stesso delle colonne della tabella.

Considera la seguente tabella Employees:

ID	FirstName	LastName	Age
1	Emily	Adams	34
2	Chloe	Anderson	27
3	Daniel	Harris	30
4	James	Roberts	31
5	John	Smith	35
6	Andrew	Thomas	45
7	David	Williams	23

Usa la seguente istruzione SQL per inserire una nuova riga:

```
INSERT INTO Employees
```

```
VALUES (8, 'Anthony', 'Young', 35);
```

I valori sono separati da virgole e il loro ordine corrisponde alle colonne della tabella.

Risultato:

ID	FirstName	LastName	Age
1	Emily	Adams	34
2	Chloe	Anderson	27
3	Daniel	Harris	30
4	James	Roberts	31
5	John	Smith	35
6	Andrew	Thomas	45
7	David	Williams	23
8	Anthony	Young	35

Quando si inseriscono record in una tabella usando l'istruzione SQL INSERT, è necessario fornire un valore per ogni colonna che non ha un valore predefinito o non supporta NULL.

## Inserire i dati

In alternativa, puoi specificare i nomi delle colonne della tabella nell'istruzione INSERT INTO:  
INSERT INTO table\_name (column1, column2, column3, ...,columnN)

VALUES (value1, value2, value3,...valueN);

SQL

column1, column2, ..., columnN sono i nomi delle colonne in cui volete inserire i dati.

INSERT INTO Employees (ID, FirstName, LastName, Age)

VALUES (8, 'Anthony', 'Young', 35);

Questo inserirà i dati nelle colonne corrispondenti:

ID	FirstName	LastName	Age
1	Emily	Adams	34
2	Chloe	Anderson	27
3	Daniel	Harris	30
4	James	Roberts	31
5	John	Smith	35
6	Andrew	Thomas	45
7	David	Williams	23
8	Anthony	Young	35

Potete specificare il vostro ordine di colonne, purché i valori siano specificati nello stesso ordine delle colonne.

È anche possibile inserire dati solo in colonne specifiche.

INSERT INTO Employees (ID, FirstName, LastName)

VALUES (9, 'Samuel', 'Clark');

Risultato:

ID	FirstName	LastName	Age
1	Emily	Adams	34
2	Chloe	Anderson	27
3	Daniel	Harris	30
4	James	Roberts	31
5	John	Smith	35
6	Andrew	Thomas	45
7	David	Williams	23
8	Anthony	Young	35
9	Samuel	Clark	0

La colonna Età per quella riga è diventata automaticamente 0, poiché questo è il suo valore predefinito.

## Aggiornare i dati

L'istruzione UPDATE ci permette di modificare i dati nella tabella.

La sintassi di base di una query UPDATE con una clausola WHERE è la seguente:

```
UPDATE table_name
```

```
SET column1=value1, column2=value2, ...
```

```
WHERE condition;
```

Si specifica la colonna e il suo nuovo valore in una lista separata da virgole dopo la parola chiave SET.

Se ometti la clausola WHERE, tutti i record della tabella saranno aggiornati!

Aggiornare i dati

Consideriamo la seguente tabella chiamata "Employees":

ID	FirstName	LastName	Salary
1	John	Smith	2000
2	David	Williams	1500
3	Chloe	Anderson	3000
4	Emily	Adams	4500

Per aggiornare lo stipendio di John, possiamo usare la seguente query:

```
UPDATE Employees
```

```
SET Salary=5000
```

```
WHERE ID=1;
```

Risultato:

ID	FirstName	LastName	Salary
1	John	Smith	<b>5000</b>
2	David	Williams	1500
3	Chloe	Anderson	3000
4	Emily	Adams	4500

## Aggiornare più colonne

È anche possibile AGGIORNARE più colonne allo stesso tempo separandole con la virgola:

```
UPDATE Employees
```

```
SET Salary=5000, FirstName='Robert'
```

```
WHERE ID=1;
```

Risultato:

ID	FirstName	LastName	Salary
1	Robert	Smith	5000
2	David	Williams	1500
3	Chloe	Anderson	3000
4	Emily	Adams	4500

Potete specificare l'ordine delle colonne in qualsiasi modo vogliate nella clausola SET.

## Cancellare i dati

L'istruzione DELETE è usata per rimuovere i dati dalla tua tabella. Le query DELETE funzionano come le query UPDATE.

```
DELETE FROM table_name  
WHERE condition;
```

Per esempio, puoi eliminare un dipendente specifico dalla tabella:

```
DELETE FROM Employees  
WHERE ID=1;
```

Risultato:

ID	FirstName	LastName	Salary
2	David	Williams	1500
3	Chloe	Anderson	3000
4	Emily	Adams	4500

Se ometti la clausola WHERE, tutti i record della tabella saranno cancellati!

L'istruzione DELETE rimuove i dati dalla tabella in modo permanente.



# Tabelle SQL

Un singolo database può ospitare centinaia di tabelle, ognuna delle quali gioca un ruolo unico nello schema del database.

Le tabelle SQL sono composte da righe e colonne. Le colonne delle tabelle sono responsabili della memorizzazione di molti tipi diversi di dati, inclusi numeri, testi, date e persino file.

L'istruzione `CREATE TABLE` è usata per creare una nuova tabella.

La creazione di una tabella di base comporta la denominazione della tabella e la definizione delle sue colonne e del tipo di dati di ciascuna colonna.

## Creare una tabella

La sintassi di base per l'istruzione CREATE TABLE è la seguente:

```
CREATE TABLE table_name  
(  
column_name1 data_type(size),  
column_name2 data_type(size),  
column_name3 data_type(size),  
....  
columnN data_type(size)  
);
```

- Il parametro column\_names specifica i nomi delle colonne che vogliamo creare.
  - Il parametro data\_type specifica che tipo di dati può contenere la colonna. Per esempio, usa int per i numeri interi.
  - Il parametro size specifica la lunghezza massima della colonna della tabella.
- Nota le parentesi nella sintassi.

Supponiamo che tu voglia creare una tabella chiamata "Users" che consiste di quattro colonne: UserID, LastName, FirstName e City.

Usare la seguente istruzione CREATE TABLE:

```
CREATE TABLE Users  
(  
UserID int,  
FirstName varchar(100),  
LastName varchar(100),  
City varchar(100)  
);
```

varchar è il tipo di dato che memorizza i caratteri. Si specifica il numero di caratteri nelle parentesi dopo il tipo. Così nell'esempio sopra, i nostri campi possono contenere al massimo 100 caratteri di testo.

# Tipi di dati

I tipi di dati specificano il tipo di dati per una particolare colonna.

Se una colonna chiamata "LastName" deve contenere dei nomi, allora quella particolare colonna dovrebbe avere un tipo di dati "varchar" (carattere di lunghezza variabile).

I tipi di dati più comuni:

Numerico

INT - Un intero di dimensioni normali che può essere con segno o senza segno.

FLOAT(M,D) - Un numero in virgola mobile che non può essere senza segno. Potete opzionalmente definire la lunghezza di visualizzazione (M) e il numero di decimali (D).

DOUBLE(M,D) - Un numero in virgola mobile a doppia precisione che non può essere senza segno. Potete opzionalmente definire la lunghezza di visualizzazione (M) e il numero di decimali (D).

Data e ora

DATE - Una data nel formato YYYY-MM-DD.

DATETIME - Una combinazione di data e ora nel formato YYYY-MM-DD HH:MM:SS.

TIMESTAMP - Un timestamp, calcolato dalla mezzanotte del 1 gennaio 1970

TIME - Memorizza l'ora nel formato HH:MM:SS.

Tipo di stringa

CHAR(M) - Stringa di caratteri di lunghezza fissa. La dimensione è specificata tra parentesi. Max 255 byte.

VARCHAR(M) - Stringa di caratteri di lunghezza variabile. La dimensione massima è specificata tra parentesi.

BLOB - "Binary Large Objects" e sono usati per memorizzare grandi quantità di dati binari, come immagini o altri tipi di file.

TEXT - Grande quantità di dati di testo.

Scegliere il tipo di dati corretto per le tue colonne è la chiave per una buona progettazione del database.

## Chiave primaria

UserID è la scelta migliore per la chiave primaria della nostra tabella Utenti. Definiscilo come chiave primaria durante la creazione della tabella, usando la parola chiave PRIMARY KEY.

```
CREATE TABLE Users
(
  UserID int,
  FirstName varchar(100),
  LastName varchar(100),
  City varchar(100),
  PRIMARY KEY(UserID)
);
```

Specificare il nome della colonna tra le parentesi della parola chiave PRIMARY KEY.

Creare una tabella

Ora, quando eseguiamo la query, la nostra tabella verrà creata nel database.

UserID	FirstName	LastName	City

Ora puoi usare le query INSERT INTO per inserire dati nella tabella.

# Vincoli SQL

I vincoli SQL sono usati per specificare le regole per i dati della tabella.

I seguenti sono vincoli SQL comunemente usati:

NOT NULL - Indica che una colonna non può contenere alcun valore NULL.

UNIQUE - Non permette di inserire un valore duplicato in una colonna. Il vincolo UNIQUE mantiene l'unicità di una colonna in una tabella. Più di una colonna UNIQUE può essere usata in una tabella.

PRIMARY KEY - Impone alla tabella di accettare dati unici per una colonna specifica e questo vincolo crea un indice unico per accedere più velocemente alla tabella.

CHECK - Determina se il valore è valido o meno da un'espressione logica.

DEFAULT - Durante l'inserimento dei dati in una tabella, se non viene fornito alcun valore a una colonna, allora la colonna ottiene il valore impostato come DEFAULT.

Per esempio, il seguente significa che la colonna name non ammette valori NULL.

nome varchar(100) NOT NULL

SQL

Durante la creazione della tabella, specificate i vincoli a livello di colonna dopo il tipo di dati di quella colonna.

# AUTOINCREMENTO

L'autoincremento permette di generare un numero unico quando un nuovo record viene inserito in una tabella.

Spesso, vorremmo che il valore del campo chiave primaria fosse creato automaticamente ogni volta che viene inserito un nuovo record.

Per default, il valore iniziale di AUTO\_INCREMENT è 1, e aumenterà di 1 per ogni nuovo record. Impostiamo il campo UserID per essere una chiave primaria che genera automaticamente un nuovo valore:

```
UserID int NOT NULL AUTO_INCREMENT,  
PRIMARY KEY (UserID)
```

L'autoincremento permette di generare un numero unico quando un nuovo record viene inserito in una tabella.

L'esempio seguente dimostra come creare una tabella usando i vincoli.

```
CREATE TABLE Users (  
id int NOT NULL AUTO_INCREMENT,  
username varchar(40) NOT NULL,  
password varchar(10) NOT NULL,  
PRIMARY KEY(id)  
);
```

Il seguente SQL fa sì che le colonne "id", "username" e "password" non accettino valori NULL. Definiamo anche che la colonna "id" sia un campo chiave primaria ad incremento automatico.

Ecco il risultato:

#	Column	Type	Null	Default	Extra
1	id	int(11)	No	None	AUTO_INCREMENT
2	username	varchar(40)	No	None	
3	password	varchar(10)	No	None	

Quando si inserisce un nuovo record nella tabella Users, non è necessario specificare un valore per la colonna id; un nuovo valore unico sarà aggiunto automaticamente.

## ALTER TABLE

Il comando ALTER TABLE è usato per aggiungere, cancellare o modificare colonne in una tabella esistente.

Puoi anche usare il comando ALTER TABLE per aggiungere e rimuovere vari vincoli da una tabella esistente.

Considera la seguente tabella chiamata People:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago

Il seguente codice SQL aggiunge una nuova colonna chiamata DateOfBirth  
ALTER TABLE People ADD DateOfBirth date;

Risultato:

ID	FirstName	LastName	City	DateOfBirth
1	John	Smith	New York	NULL
2	David	Williams	Los Angeles	NULL
3	Chloe	Anderson	Chicago	NULL

Tutte le righe avranno il valore predefinito nella colonna appena aggiunta, che, in questo caso, è NULL.

## Dropping

Il seguente codice SQL dimostra come cancellare la colonna chiamata DateOfBirth nella tabella People.

```
ALTER TABLE People  
DROP COLUMN DateOfBirth;
```

The People table will now look like this:

ID	FirstName	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago

La colonna, insieme a tutti i suoi dati, sarà completamente rimossa dalla tabella.

Per cancellare l'intera tabella, usa il comando DROP TABLE:

```
DROP TABLE People;
```

Fai attenzione quando elimini una tabella. L'eliminazione di una tabella comporta la perdita completa delle informazioni memorizzate nella tabella!



## Rinominare

Il comando ALTER TABLE è usato anche per rinominare le colonne:

```
ALTER TABLE table_name  
CHANGE column_1 column_2 ["Data Type"];
```

\*es VARCHAR(50)

Questa query rinominerà la colonna chiamata FirstName in name.

Risultato:

ID	name	LastName	City
1	John	Smith	New York
2	David	Williams	Los Angeles
3	Chloe	Anderson	Chicago

Rinominare le tabelle

Puoi rinominare l'intera tabella usando il comando RENAME:

```
RENAME TABLE People TO Users;
```

Questo rinominerà la tabella People in Users.

# Views

In SQL, una VISTA è una tabella virtuale che si basa sull'insieme dei risultati di un'istruzione SQL. Una vista contiene righe e colonne, proprio come una tabella reale. I campi in una vista sono campi di una o più tabelle reali nel database.

Le viste ci permettono di:

- Strutturare i dati in un modo che gli utenti o le classi di utenti trovano naturale o intuitivo.
- Limitare l'accesso ai dati in modo tale che un utente possa vedere e (a volte) modificare esattamente ciò di cui ha bisogno e non di più.
- Riassumere i dati da varie tabelle e usarli per generare rapporti.

Per creare una vista:

```
CREATE VIEW view_name AS  
SELECT column_name(s)  
FROM table_name  
WHERE condition;
```

La query SELECT può essere complessa quanto serve. Può contenere più JOINS e altri comandi. Consideriamo la tabella Employees, che contiene i seguenti record:

ID	FirstName	LastName	Age	Salary
1	Emily	Adams	34	5000
2	Chloe	Anderson	27	10000
3	Daniel	Harris	30	6500
4	James	Roberts	31	5500
5	John	Smith	35	4500
6	Andrew	Thomas	45	6000
7	David	Williams	23	3000

Creiamo una vista che visualizzi FirstName e Salary di ogni dipendente.

```
CREATE VIEW List AS  
SELECT FirstName, Salary  
FROM Employees;
```

Ora, potete interrogare la vista List come fareste con una tabella reale.

```
SELECT * FROM List;
```

Questo produrrebbe il seguente risultato:

FirstName	Salary
Emily	5000
Chloe	10000
Daniel	6500
James	5500
John	4500
Andrew	6000
David	3000

Una vista mostra sempre dati aggiornati! Il motore del database usa l'istruzione SQL della vista per ricreare i dati ogni volta che un utente interroga una vista.