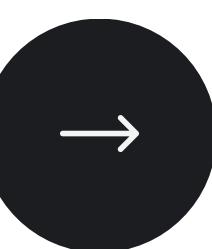


# Introduzione ai DB e SQL

---

Docente: Tommaso Muraca



# **SQL (Structured Query Language)**

---



**Structured Query Language (SQL) è un linguaggio di interrogazione (query) utilizzato per creare, modificare e gestire i dati in un database relazionale.**

**Si tratta di un linguaggio specifico di dominio (DSL) usato per comunicare con i sistemi di gestione di database relazionali (RDBMS).**

---

# SQL: COSA CI SERVE

Useremo un DBMS(database management system) ovvero un sistema software progettato per consentire la creazione, la manipolazione e l'interrogazione efficiente di database, per questo detto anche "gestore o motore del database".

Nello specifico useremo MySQL che si basa sul modello relazionale quindi un RDBMS(relational database management system).

Cosa faremo :

- Scaricheremo e installeremo MySQL (tramite MAMP)
- Andremo a vedere alcune connotazioni basilari di MySQL

# COS'E' MAMP?

---

**MAMP è un ambiente server locale gratuito che si installa in pochi clic su MacOS e Windows.**

**MAMP offre tutti gli strumenti necessari per eseguire MySQL o qualsiasi altro software sul proprio PC desktop a scopo di test o sviluppo .**

**Con l'aiuto del server DNS locale NAMO , si possono testare facilmente i progetti anche su dispositivi mobili.**

**MAMP è compatibile con server web Apache o Nginx oltre e linguaggi di programmazione come PHP, Python, Perl o Ruby.**

---

# **INSTALLIAMO MAMP**

---

**Andiamo a scaricare MAMP al seguente link:**

**<https://www.mamp.info/en/downloads/>**

**Importiamo i 2 DB di TEST**

---

# MySQL

---

**MySQL o Oracle MySQL è un relational database management system (RDBMS) composto da un client a riga di comando e un server. Ambo i costituenti sono multipiattaforma e sono disponibili ufficialmente su tutte le distribuzioni conosciute, quali Debian, Ubuntu e CentOS, sebbene lo abbiano sostanzialmente sostituito con MariaDB a partire dal 2012.**



**È software libero rilasciato a doppia licenza, compresa la GNU General Public License, sviluppato per essere il più possibile conforme agli standard ANSI SQL e ODBC SQL.**

**I sistemi e i linguaggi di programmazione che lo supportano sono molto numerosi, fra cui ODBC, Java, Mono, .NET, PHP, Python.**

# SQL

**SQL è un linguaggio standard per l'archiviazione, la manipolazione e il recupero dei dati nei database. Nonostante sia uno standard ANSI/ISO ci sono molte versioni SQL ed estensioni proprietarie dei vari software di database.**

**Un database si può riassumere in una serie di tabelle identificate con un nome e possono essere relazionate tra loro, queste tabelle sono divise in colonne per ogni attributo e in righe (record) per le informazioni che vogliamo salvare.**

	Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy	GNP	GNPOld	LocalName
▶	ABW	Aruba	North America	Caribbean	193.00	NULL	103000	78.4	828.00	793.00	Aruba
	AFG	Afghanistan	Asia	Southern and Central Asia	652090.00	1919	22720000	45.9	5976.00	NULL	Afganistan/Afghanistan
	AGO	Angola	Africa	Central Africa	1246700.00	1975	12878000	38.3	6648.00	7984.00	Angola
	AIA	Anguilla	North America	Caribbean	96.00	NULL	8000	76.1	63.20	NULL	Anguilla
	ALB	Albania	Europe	Southern Europe	28748.00	1912	3401200	71.6	3205.00	2500.00	Shqipëria
	AND	Andorra	Europe	Southern Europe	468.00	1278	78000	83.5	1630.00	NULL	Andorra
	ANT	Netherlands Antilles	North America	Caribbean	800.00	NULL	217000	74.7	1941.00	NULL	Nederlandse Antillen

# LE QUERY

**L'interrogazione di un database avviene mediante una query.**

**Ovvero una espressione o una serie di espressioni con delle richieste da fare al database per avere una risposta, sotto forma di tabella.**

1 • | `SELECT * FROM world.country;`

Questa query sta interrogando il database "world" andando a selezionare la tabella "country" dando quindi come risultato direttamente le righe della tabella country.

	Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy	GNP	GNPOld	LocalName
▶	ABW	Aruba	North America	Caribbean	193.00	NULL	103000	78.4	828.00	793.00	Aruba
	AFG	Afghanistan	Asia	Southern and Central Asia	652090.00	1919	22720000	45.9	5976.00	NULL	Afganistan/Afghanistan
	AGO	Angola	Africa	Central Africa	1246700.00	1975	12878000	38.3	6648.00	7984.00	Angola
	AIA	Anguilla	North America	Caribbean	96.00	NULL	8000	76.1	63.20	NULL	Anguilla
	ALB	Albania	Europe	Southern Europe	28748.00	1912	3401200	71.6	3205.00	2500.00	Shqipëria
	AND	Andorra	Europe	Southern Europe	468.00	1278	78000	83.5	1630.00	NULL	Andorra
	ANT	Netherlands Antilles	North America	Caribbean	800.00	NULL	217000	74.7	1941.00	NULL	Nederlandse Antillen

# ESEMPO DI QUERY

	Code	Name	Continent	Region
▶	ABW	Aruba	North America	Caribbean
	AFG	Afghanistan	Asia	Southern and Central Asia
	AGO	Angola	Africa	Central Africa
	AIA	Anguilla	North America	Caribbean
	ALB	Albania	Europe	Southern Europe
	AND	Andorra	Europe	Southern Europe

```
1 •   SELECT Continent  
2      FROM world.country;
```

In questo esempio andiamo a recuperare solo la colonna "Continent" della tabella country dando come risultato una tabella con una singola colonna con solo i valori di Continent di tutte le righe di country.

	Continent
▶	North America
	Asia
	Africa
	North America
	Europe
	Europe

# SQL: SINTASSI

---

**SQL è un linguaggio da una sintassi molto specifica che non è sensibile alle variazioni tra MAIUSCOLE e minuscole: "select" e "SELECT" sono uguali. Non è lo stesso per il nome delle tabelle e delle colonne.**

**Alcuni sistemi di database richiedono un punto e virgola alla fine di ogni istruzione SQL. Il punto e virgola è il modo standard per separare ogni istruzione SQL nei database che consentono l'esecuzione di più istruzioni SQL nella stessa chiamata al server (come MySQL).**

---

# SQL: SINTASSI

---

## Alcuni dei comandi SQL più importanti:

- **SELECT** - estrae i dati da un database
  - **UPDATE** - aggiorna i dati in un database
  - **DELETE** - elimina i dati da un database
  - **INSERT INTO** - inserisce nuovi dati in un database
  - **CREATE DATABASE** - crea un nuovo database
  - **ALTER DATABASE** - modifica un database
  - **CREATE TABLE** - crea una nuova tabella
  - **ALTER TABLE** - modifica una tabella
  - **DROP TABLE** - elimina una tabella
  - **CREATE INDEX** - crea un indice (chiave di ricerca)
  - **DROP INDEX** - elimina un indice
-

# **SQL: SCHEMA RELAZIONALE**

---

**Uno schema relazionale in SQL definisce la struttura e le relazioni tra le tavole all'interno di un database relazionale.**

**In SQL, gli schemi sono utilizzati per organizzare e raggruppare oggetti del database, come tavole, viste, procedure, e altro ancora.**

---

# SQL: Chiavi primarie

---

- La chiave primaria è un attributo (o un insieme di attributi) in una tabella che identifica univocamente ogni riga nella tabella.
- Ogni tabella in un database relazionale deve avere una chiave primaria.
- La chiave primaria garantisce l'unicità delle righe e facilita l'individuazione e la manipolazione dei dati in modo efficiente.
- Può essere composta da uno o più campi (attributi).

```
■ CREATE TABLE Students (
    ■   StudentID INT PRIMARY KEY,
    ■   FirstName VARCHAR(50),
    ■   LastName VARCHAR(50),
    ■   -- Altri attributi);
```

# SQL: Chiave Esterna

---

- La chiave esterna è un attributo in una tabella che stabilisce una relazione con la chiave primaria di un'altra tabella.
- La chiave esterna è utilizzata per collegare le righe di una tabella con le righe di un'altra tabella, creando così una relazione tra di esse.
- La tabella che contiene la chiave esterna è chiamata tabella figlia, e la tabella a cui fa riferimento la chiave esterna è chiamata tabella padre.
- La chiave esterna garantisce l'integrità referenziale tra le tabelle.

```
■ CREATE TABLE Enrollments (
    ■ EnrollmentID INT PRIMARY KEY,
    ■ StudentID INT,
    ■ CourseID INT,
    ■ Grade VARCHAR(2),
    ■ FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
    ■ FOREIGN KEY (CourseID) REFERENCES Courses(CourseID));
```

# SQL: COMMENTI

---

**Come in altri linguaggi è possibile inserire del testo che viene ignorato.**

**In SQL si può commentare una riga inserendo "--" prima del testo da commentare:**

- 1.-- selezioniamo tutte le colonne**
- 2.SELECT \* FROM table\_name -- dalla tabella**

**È possibile anche fare un commento multi riga inserendolo tra "/\*" e "\*/".**

- 1./\* selezioniamo tutte le colonne**
- 2.SELECT \* FROM table\_name \*/**
- 3. SELECT \* FROM world.country -- dalla tabella country del database world**

# **SQL: CREATE/DROP DATABASE**

---

**Sono delle operazioni che richiedono i privilegi di un amministratore.**

**CREATE DATABASE ci permette di creare un nuovo database:**

**1. CREATE DATABASE databasename;**

**DROP DATABASE ci permette di eliminare un database eliminando tutto il suo contenuto:**

**1. DROP DATABASE databasename;**

# SQL: CREATE TABLE

---

**CREATE TABLE** ci permette di creare una nuova tabella nel database database specificando il nome delle colonne con le loro caratteristiche:

- 1. CREATE TABLE table\_name (**
- 2. column1 datatype,**
- 3. column2 datatype,**
- 4. column3 datatype,**
- 5. ....**
- 6. );**

**Ci sono numerosi datatype che possiamo dare alle colonne, i più usati sono il VARCHAR(nMax di char) e l'INT.**

# SQL: CREATE TABLE

**Esempio:**

```
1. CREATE TABLE Persons (
2.   PersonID int,
3.   LastNames varchar(255),
4.   FirstNames varchar(255),
5.   Address varchar(255),
6.   City varchar(255)
7. );
```

PersonID	LastNames	FirstNames	Address	City

# SQL: CREATE TABLE

**CREATE TABLE** ci permette di creare una nuova tabella anche da un'altra tabella, copiando le caratteristiche delle tabelle selezionate e i suoi record:

- 1. CREATE TABLE new\_table\_name**
- 2. SELECT column1, column2,...**
- 3. FROM existing\_table\_name**
- 4. WHERE ....;**

**Un esempio di sintassi:**

- 1. CREATE TABLE TestTable**
- 2. SELECT customername, contactname**
- 3. FROM customers;**

# SQL: DATATYPE

---

In MySQL ci sono tre tipi di dati principali: stringa, numerico, data e ora.  
Qui elencheremo i principali.

I principali tipi sono:

- **CHAR(size): una stringa di grandezza fissa(max 255);**
- **VARCHAR(max\_size): una stringa a grandezza variabile fino al massimo indicato(max 65535);**
- **MEDIUMTEXT: una stringa lunga fino a 16,777,215 caratteri;**
- **LONGTEXT: una stringa lungafino a 4,294,967,295 caratteri;**
- **ENUM(val1, val2, val3, ...): una lista di valori possibili;**
- **BOOL: un valore booleano, 0 per false e 1 per true;**
- **INT: un valore numerico intero;**
- **FLOAT: un valore numero con virgola mobile;**
- **DATE: data in formato YYYY-MM-DD;**

# SQL: DROP/TRUNCATE TABLE

---

**DROP TABLE** ci permette di eliminare una tabella con tutto il suo contenuto:

**1. `DROP TABLE table_name`**

**TRUNCATE TABLE** ci permette di eliminare tutti i record di una tabella senza eliminare la tabella:

**1. `TRUNCATE TABLE table_name`**

# SQL: ALTER TABLE

---

**ALTER TABLE** ci permette di aggiungere, eliminare o modificare colonne in una tabella esistente con:

- **ADD:** per aggiungere una colonna;
- **DROP COLUMN:** per eliminare una colonna
- **MODIFY COLUMN:** per modificare una colonna;

**1. `ALTER TABLE table_name`**  
**2. `ADD column_name datatype;`**

**NOTA:** è possibile modificare anche i vincoli di una tabella.

---

# SQL: CONSTRAINT

**Per ogni colonna si possono specificare dei vincoli sul tipo di dato.**

```
1. CREATE TABLE table_name (  
2.   column1 datatype constraint,  
3.   .... );
```

**I vincoli più utilizzati sono:**

- **NOT NULL** - non ci può essere un valore NULL;
- **UNIQUE** - tutti i valori in una colonna sono differenti;
- **PRIMARY KEY** - identifica in modo univoco ogni riga di una tabella;
- **FOREIGN KEY** - identifica una chiave esterna;
- **CHECK** - i valori in una colonna devono soddisfare una condizione;
- **DEFAULT** - imposta un valore di default;
- **CREATE INDEX** - crea un index per recuperare dati più velocemente.

# SQL: CONSTRAINT

**NOT NULL esempio:**

```
1. CREATE TABLE Persons (
2.   ID int NOT NULL,
3.   LastName varchar(255) NOT NULL,
4.   FirstName varchar(255) NOT NULL,
5.   Age int
6.);
```

**Per poi modificare in successiva Age:**

```
1. ALTER TABLE Persons
2. MODIFY Age int NOT NULL;
```

# SQL: CONSTRAINT

**UNIQUE esempio:**

```
1. CREATE TABLE Persons (
2.   ID int NOT NULL,
3.   LastName varchar(255) NOT NULL,
4.   FirstName varchar(255),
5.   Age int,
6.   UNIQUE (ID));
```

**È possibile indicare più colonne UNIQUE modificando la sintassi indicando il nome del vincolo:**

```
1. CONSTRAINT ctnt_name UNIQUE (ID,LastName)
```

**Per poi eliminare un vincolo bisogna fare:**

```
1. ALTER TABLE table_name
2. DROP INDEX ctnt_name;
```

# SQL: CONSTRAINT

**PRIMARY KEY esempio:**

```
1. CREATE TABLE Persons (
2.   ID int NOT NULL,
3.   LastName varchar(255) NOT NULL,
4.   FirstName varchar(255),
5.   Age int,
6.   PRIMARY KEY (ID));
```

È possibile indicare una PRIMARY KEY su più colonne con un CONSTRAINT:

```
1. CONSTRAINT PK_name PRIMARY KEY (column1, column2)
```

Per poi eliminare una PRIMARY KEY bisogna fare:

```
1. ALTER TABLE table_name
2. DROP PRIMARY KEY;
```

# SQL: CONSTRAINT

**FOREIGN KEY esempio:**

```
1. CREATE TABLE Orders (
2.   OrderID int NOT NULL,
3.   OrderNumber int NOT NULL,
4.   PersonID int,
5.   PRIMARY KEY (OrderID),
6.   FOREIGN KEY (PersonID) REFERENCES Persons(PersonID) );
```

È possibile indicare una **FOREIGN KEY** con un **CONSTRAINT**:

```
1. CONSTRAINT FK_name FOREIGN KEY (column) REFERENCES ref_table(column)
```

Per poi eliminare una **FOREIGN KEY** bisogna fare:

```
1. ALTER TABLE table_name
2. DROP FOREIGN KEY FK_name;
```

# SQL: CONSTRAINT

**CHECK esempio:**

```
1. CREATE TABLE Persons (
2.   ID int NOT NULL,
3.   LastName varchar(255) NOT NULL,
4.   FirstName varchar(255),
5.   Age int,
6.   CHECK (Age>=18);
```

**È possibile indicare un CHECK con un CONSTRAINT:**

```
1. CONSTRAINT CHK_name CHECK (con1 AND con2)
```

**Per poi eliminare un CHECK bisogna fare:**

```
1. ALTER TABLE table_name
2. DROP CHECK CHK_name;
```

# SQL: CONSTRAINT

**DEFAULT esempio:**

```
1. CREATE TABLE Persons (
2.   ID int NOT NULL,
3.   LastName varchar(255) NOT NULL,
4.   FirstName varchar(255),
5.   Age int,
6.   City varchar(255) DEFAULT 'Sandnes' );
```

**È possibile indicare un valore con una function:**

```
1. OrderDate date DEFAULT CURRENT_DATE()
```

**Per poi eliminare o modificare un DEFAULT bisogna fare:**

```
1. ALTER TABLE table_name
2. ALTER column DROP DEFAULT; / ALTER column SET DEFAULT value;
```

# SQL: CONSTRAINT

**AUTO\_INCREMENT** esempio:

```
1. CREATE TABLE Persons (
2.   Personid int NOT NULL AUTO_INCREMENT,
3.   LastName varchar(255) NOT NULL,
4.   FirstName varchar(255),
5.   Age int,
6.   PRIMARY KEY (Personid) );
```

È possibile indicare un valore con cui far iniziare la sequenza:

```
1. ALTER TABLE Persons AUTO_INCREMENT=100;
```

**NOTE:** la colonna con il valore **AUTO\_INCREMENT** non permette di inserirgli valori.

# SQL: DATE

---

**MySQL fornisce diversi tipi di DATE per la memorizzazione di una data o di un valore di data / ora nel database:**

- **DATE - formato AAAA-MM-DD ('2038-01-19')**
- **DATETIME - formato: AAAA-MM-GG HH:MI:SS ('2038-01-19 03:14:07.999999')**
- **TIMESTAMP - formato: AAAA-MM-GG HH:MI:SS ('2038-01-19 03:14:07.999999')**
- **YEAR - formato AAAA o YY ('2038', '00' - '69' per il 2000, '70' - '99' per il 1900)**

**NOTE: è possibile confrontare con operatori di uguaglianza solo DATE senza valori temporali (data, ora e minuti).**

---

# SQL: SELECT

---

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

**1. SELECT \* FROM table\_name;**

**I dati restituiti vengono archiviati in una tabella temporanea, denominata result-set.**

**1. SELECT column1, column2, ...**

**2. FROM table\_name;**

**Con SELECT andiamo a selezionare le colonne di una tabella andando a specificare il nome, invece il "\*" serve ad indicare tutte le colonne.**

**Con la "," possiamo selezionare più colonne contemporaneamente.**

# SQL: SELECT DISTINCT

L'istruzione **SELECT DISTINCT** funziona esattamente come **SELECT** solo che restituisce valori distinti tra loro, quindi senza ripetizioni.

- 1. SELECT DISTINCT column1, column2, ...**
- 2. FROM table\_name;**

Con **SELECT DISTINCT** possiamo ad esempio sapere il numero di valori unici di una colonna usando il comando **COUNT** che conta i valori.

- 1. SELECT COUNT(DISTINCT Name) FROM world.country;**

	COUNT(DISTINCT Name)
▶	239

# SQL: WHERE Clause

---

**L'istruzione WHERE è definita una "clausola", serve per mettere una condizione in modo da filtrare i record che andranno nel result-set.**

- 1. SELECT column1, column2, ...**
- 2. FROM table\_name**
- 3. WHERE condition;**

**Le condizioni possono essere di vario tipo e vanno scritte insieme all'operatore WHERE, ad esempio con valori stringa che vanno scritti tra singoli apici come ('testo').**

- 1. SELECT \* FROM world.country**
- 2. WHERE Region ='Antarctica';**

# SQL: WHERE Clause

Un altro esempio può essere con un valore numerico:

1. **SELECT \* FROM world.country**
2. **WHERE Population=0;**

Le operazioni che si possono fare come condizione sono:

- = - uguale;
- > - maggiore di;
- < - minore di;
- >= - maggiore uguale di;
- <= - minore uguale di;
- != - non uguale (in alcune versioni di SQL si scrive "!="; in altre ancora accetta entrambe le scritture come in MySQL);
- **BETWEEN** - tra un certo intervallo ("BETWEEN 0 and 10", valori compresi);
- **LIKE** - cerca un pattern, ovvero cerca una egualianza parziale o totale specificando il pattern fisso da trovare e le parti del dato variabili con un *metacarattere* come il "%" (esempio: "City LIKE 's%' " per trovare i valori che iniziano con "s" o "S");
- **IN** - per specificare più valori possibili (esempio "IN (0, 1000)");

# SQL: WHERE (IL VALORE **NULL**)

---

**Un campo con valore **NULL**** è un campo senza valore che è diverso da un valore zero o da un campo che contiene spazi. Un campo con un valore **NULL** è un campo che è stato lasciato vuoto durante la creazione di record.

**Non è possibile verificare i valori **NULL** con operatori di confronto (come =, < o >). Bisogna usare gli operatori **IS NULL** & **IS NOT NULL**.**

- 1. SELECT column\_names**
- 2. FROM table\_name**
- 3. WHERE column\_name IS NULL;**

# SQL: AND - OR - NOT

---

Questi operatori permettono di combinare più condizioni, stanno ad indicare:

**AND** - visualizza il record se tutte le condizioni da sono soddisfatte;

**OR** - visualizza il record se almeno una condizione è soddisfatta;

**NOT** - visualizza il record se la condizione **NON** sono soddisfatte;

1. **SELECT column1, column2, ...**

2. **FROM table\_name**

3. **WHERE condition1 **AND** condition2 **AND** condition3 ...;**

Questi operatori possono essere affiancati da parentesi per indicare un gruppo di condizioni

1. **SELECT \* FROM table\_name**

2. **WHERE condition1 **AND** (condition2 **OR NOT** condition3);**

# SQL: ORDER BY

---

**L'istruzione ORDER BY serve ad ordinare il result-set su una o più colonne con due possibili criteri:**

- **ASC - crescente**
- **DESC - decrescente**

**Non è obbligatorio specificare il criterio di ordinamento, di default prende il criterio crescente (ASC).**

- 1. SELECT column1, column2, ...**
- 2. FROM table\_name**
- 3. ORDER BY column1, column2, ... ASC | DESC;**

# SQL: ORDER BY

In questo esempio ordiniamo in base alla colonna Region (string) e con ulteriore ordinamento decrescente in base alla colonna SurfaceArea (int). Così facendo avremo un result-set ordinato in base alla regione, e le regioni uguali ordinate in base alla superficie più ampia.

1. **SELECT \* FROM world.country**
2. **ORDER BY Region, SurfaceArea DESC;**

Region	SurfaceArea
Antarctica	13120000.00
Antarctica	7780.00
Antarctica	3903.00
Antarctica	359.00
Antarctica	59.00
Australia and New Zealand	7741220.00
Australia and New Zealand	270534.00
Australia and New Zealand	135.00
Australia and New Zealand	36.00
Australia and New Zealand	14.00
Baltic Countries	65301.00
Baltic Countries	64589.00

# SQL: GROUP BY

---

**L'istruzione GROUP BY serve ad raggruppare i record con stessi valori del result-set su una o più colonne. Viene spesso utilizzato con funzioni per raggruppare il result-set.**

- 1. SELECT column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE condition**
- 4. GROUP BY column\_name(s)**
- 5. ORDER BY column\_name(s);**

# SQL: GROUP BY

**Un esempio:**

- 1. SELECT Country, COUNT(CustomerID)**
- 2. FROM Customers**
- 3. GROUP BY Country;**

Country	COUNT(CustomerID)
Argentina	3
Austria	2
Belgium	2
Brazil	9
Canada	3

# SQL: GROUP BY

**Un esempio con l'ORDER BY:**

- 1. SELECT Country, COUNT(CustomerID)**
- 2. FROM Customers**
- 3. GROUP BY Country**
- 4. ORDER BY COUNT(CustomerID) DESC;**

Country	COUNT(CustomerID)
USA	13
Germany	11
France	11
Brazil	9
UK	7

# SQL: INSERT INTO

---

**L'istruzione INSERT INTO serve per inserire nuovi record in una tabella.**

- 1. `INSERT INTO table_name (column1, column2, column3, ...)`**
- 2. `VALUES (value1, value2, value3, ...);`**

**È possibile non specificare le colonne se aggiungiamo valori a tutte le colonne ma bisogna assicurarsi che i valori abbiano l'ordine corretto delle colonne in cui devono essere inseriti.**

- 1. `INSERT INTO table_name`**
- 2. `VALUES (value1, value2, value3, ...);`**

# SQL: UPDATE

**L'istruzione UPDATE serve per modificare i record esistenti in una tabella. Viene tendenzialmente usato insieme a WHERE per filtrare i record da modificare altrimenti modificherebbe tutti i record della tabella.**

- 1. UPDATE table\_name**
- 2. SET column1 = value1, column2 = value2, ...**
- 3. WHERE condition;**

**Con SET andiamo a specificare i valori da cambiare indicando la colonna con l'assegnazione del valore.**

- 1. UPDATE Customers**
- 2. SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'**
- 3. WHERE CustomerID = 1;**

# SQL: DELETE

L'istruzione **DELETE** serve per eliminare i record esistenti in una tabella. Viene tendenzialmente usato insieme a WHERE per filtrare i record da eliminare altrimenti eliminerebbe tutti i record della tabella.

1. **DELETE FROM table\_name**
2. **WHERE condition;**

Questa istruzione va ad eliminare solo i record, la struttura della tabella rimane invariata pure se eliminano tutti i record senza specificare una condizione.

1. **DELETE FROM Customers**
2. **WHERE CustomerName='Mario Rossi';**

# SQL: SELECT TOP (**LIMIT**)

L'istruzione **SELECT TOP** serve per specificare il numero di record da restituire.

Non tutti i sistemi di database supportano questa istruzione. MySQL supporta l'operazione **LIMIT** per selezionare un numero limitato di record.

1. **SELECT column\_name**
2. **FROM table\_name**
3. **WHERE condition**
4. **LIMIT number;**

**LIMIT** quindi ci permette di indicare il numero massimo di record da recuperare da un database, questa operazione tendenzialmente si esegue per tavole con un grande numero di record ottimizzando le prestazioni.

1. **SELECT \* FROM Customers**
2. **LIMIT 50;**

# SQL: ALIAS

---

**Gli ALIAS SQL servono per assegnare un nome temporaneo ad un elemento come una tabella, una colonna, il risultato di una funzione ecc ecc...; In questo modo possiamo richiamare direttamente il nome temporaneo assegnato all'elemento.**

**Gli ALIAS possono essere utili quando:**

- sono coinvolte più tabelle in una QUERY;
- vengono usate le funzioni in una QUERY;
- i nomi delle colonne sono grandi o poco leggibili;
- due o più colonne vengono combinate insieme;

**Per assegnare un ALIAS viene usata la parola chiave AS quando andiamo a selezione o richiamare in SELECT o in FROM l'elemento a cui assegnare il nome:**

- 1. SELECT column\_name AS alias\_columName**
- 2. FROM table\_name AS alias\_tableName;**

# SQL: MIN() - MAX()

---

**Sono delle funzioni che restituiscono rispettivamente il valore minimo e il valore massimo di una colonna, hanno lo stesso tipo di sintassi:**

- 1. SELECT MIN(column\_name)**
- 2. FROM table\_name**
- 3. WHERE condition;**

**Nel prossimo esempio prenderemo il prezzo più alto da una tabella di prodotti e associeremo il risultato ad un nome temporaneo tramite la parola chiave **AS** (ovvero un **ALIAS**):**

- 1. SELECT MAX(Price) **AS** LargestPrice**
- 2. FROM Products;**

# SQL: COUNT() - AVG() - SUM()

**Sono delle funzioni che eseguono un calcolo restituendo un valore specifico:**

- **COUNT()** - restituisce il numero di record selezionati;
- **AVG()** - restituisce il valore medio di una colonna con valori numerici;
- **SUM()** - restituisce la somma totale di una colonna con valori numerici;

**Tutte e tre le funzioni hanno lo stesso tipo di sintassi e possono essere associate a delle condizioni con WHERE:**

- 1. SELECT SUM(column\_name)**
- 2. FROM table\_name**
- 3. WHERE condition;**

# SQL: WILDCARD CHARACTERS

---

**In italiano metacaratteri, sono dei caratteri che servono per sostituire uno o più caratteri in una stringa. Vengono utilizzati nelle condizioni come WHERE e sono:**

- "%" - zero o più caratteri ("bl%" = bl, black, blue e blob);
- "\_" - un singolo carattere ("h\_t" = hot, hat e hit);

**Il resto dei metacaratteri hanno bisogno della funzione "REGEXP\_LIKE(colonna, pattern)"**

- "[]" - ogni singolo carattere all'interno delle parentesi ("h[oa]t" = hot e hat);
- "^" - qualsiasi carattere non compreso tra parentesi ("h[^oa]t" = hit);
- "-" - qualsiasi carattere all'interno del range specificato ("c[a-b]t" = cat e cbt);

**I metacaratteri possono essere usati in combinazione come ad esempio '\_r%' che rappresenta valori con almeno una r al secondo carattere.**

---

# SQL: LIKE

---

**È un operatore che cerca un pattern preciso usando anche i metacaratteri:**

- 1. SELECT column1, column2, ...**
- 2. FROM table\_name**
- 3. WHERE columnN LIKE pattern;**

**In questo esempio seleziona tutti i clienti con il nome che inizia per "a":**

- 1. SELECT \* FROM Customers**
- 2. WHERE CustomerName LIKE 'a%';**

# SQL: IN

---

**È un operatore che consente di specificare più valori in una clausola, una scorciatoia per più condizioni OR.**

- 1. SELECT column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE column\_name IN (value1, value2, ...);**

**È possibile inserire anche una table all'interno dei valori possibili:**

- 1. SELECT column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE column\_name IN (SELECT STATEMENT);**

# SQL: IN

---

**Esempi di utilizzo:**

- 1. SELECT \* FROM Customers**
- 2. WHERE Country IN ('Germany', 'France', 'UK');**

**con NOT:**

- 1. SELECT \* FROM Customers**
- 2. WHERE Country NOT IN ('Germany', 'France', 'UK');**

**Confrontando valori di una table con un'altra table:**

- 1. SELECT \* FROM Customers**
- 2. WHERE Country IN (SELECT Country FROM Suppliers);**

# SQL: BETWEEN

**È un operatore che seleziona i valori (come: numeri, testo o date) all'interno di un determinato intervallo. L'operatore è inclusivo ovvero sono inclusi i valori di inizio e fine.**

- 1. SELECT column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE column\_name BETWEEN value1 AND value2;**

**Un esempio di utilizzo con più condizioni:**

- 1. SELECT \* FROM Products**
- 2. WHERE Price BETWEEN 10 AND 20**
- 3. AND CategoryID NOT IN (1,2,3);**

# SQL: BETWEEN

Un esempio utilizzando le date:

```
1. SELECT * FROM Orders  
2. WHERE OrderDate BETWEEN #07/01/1996# AND #07/31/1996#;
```

Un'altra sintassi per le date è:

```
1. SELECT * FROM Orders  
2. WHERE OrderDate BETWEEN '1996-07-01' AND '1996-07-31';
```

Un esempio utilizzando un ORDER BY:

```
1. SELECT * FROM Products  
2. WHERE ProductName NOT BETWEEN 'Carnarvon Tigers' AND 'Mozzarella di Giovanni'  
3. ORDER BY ProductName;
```

# Esercizi

---

**Esercizi:**

- 1) Scrivete una query SQL che restituisca solo i record dalla tabella "products" con un prezzo superiore a 50.**
  
  - 2) Scrivete una query SQL che restituisca tutti i record dalla tabella "orders" ordinati per data in ordine decrescente.**
  
  - 3) Scrivete una query SQL che aggiorni il prezzo di tutti i prodotti nella tabella "products" aumentandolo del 10%.**
-

# Esercizi1

---

## Esercizi:

- 1) Scrivete una query SQL che inserisca un nuovo utente nella tabella "customers".**
  
  - 2) Scrivete una query SQL che elimini tutti gli ordini nella tabella "orders" con uno stato di "Cancelled".**
  
  - 3) Scrivete una query SQL che restituisca tutti gli utenti dalla tabella "customers" il cui nome inizia con la S e vivono in California.**
-

# SQL: JOIN

**È una clausola che viene utilizzata per combinare righe da due o più tabelle, in base a una colonna correlata tra di loro.**

```
1. SELECT Orders.OrderID, Customers.CustomerName, Orders.OrderDate  
2. FROM Orders  
3. INNER JOIN Customers ON Orders.CustomerID=Customers.CustomerID;
```

**In questo esempio stiamo andando a prendere i record che hanno lo stesso valore CustomerID unendo i valori selezionati generando un result-set con 3 colonne:**

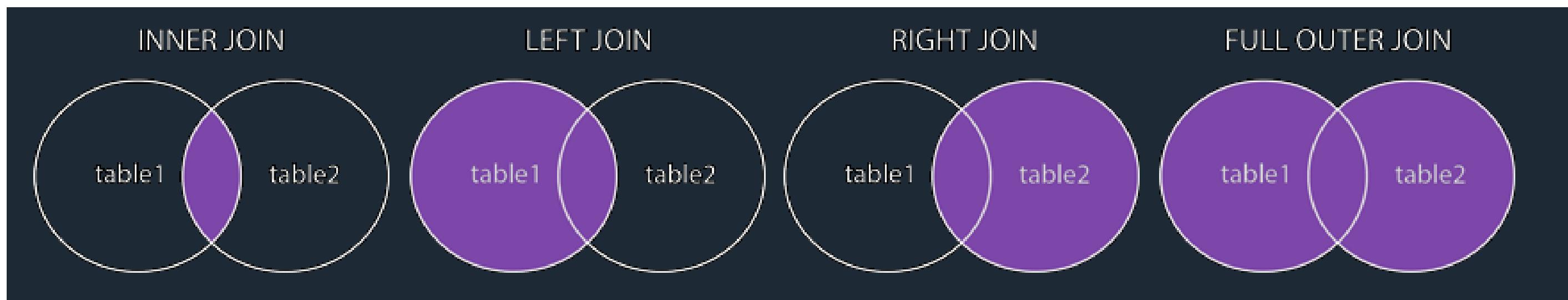
- 1. OrderID della tabella orders**
- 2. CustomerName della tabella customers**
- 3. OrderDate della tabella orders**

# SQL: JOIN

---

**Ci sono 4 tipi di join:**

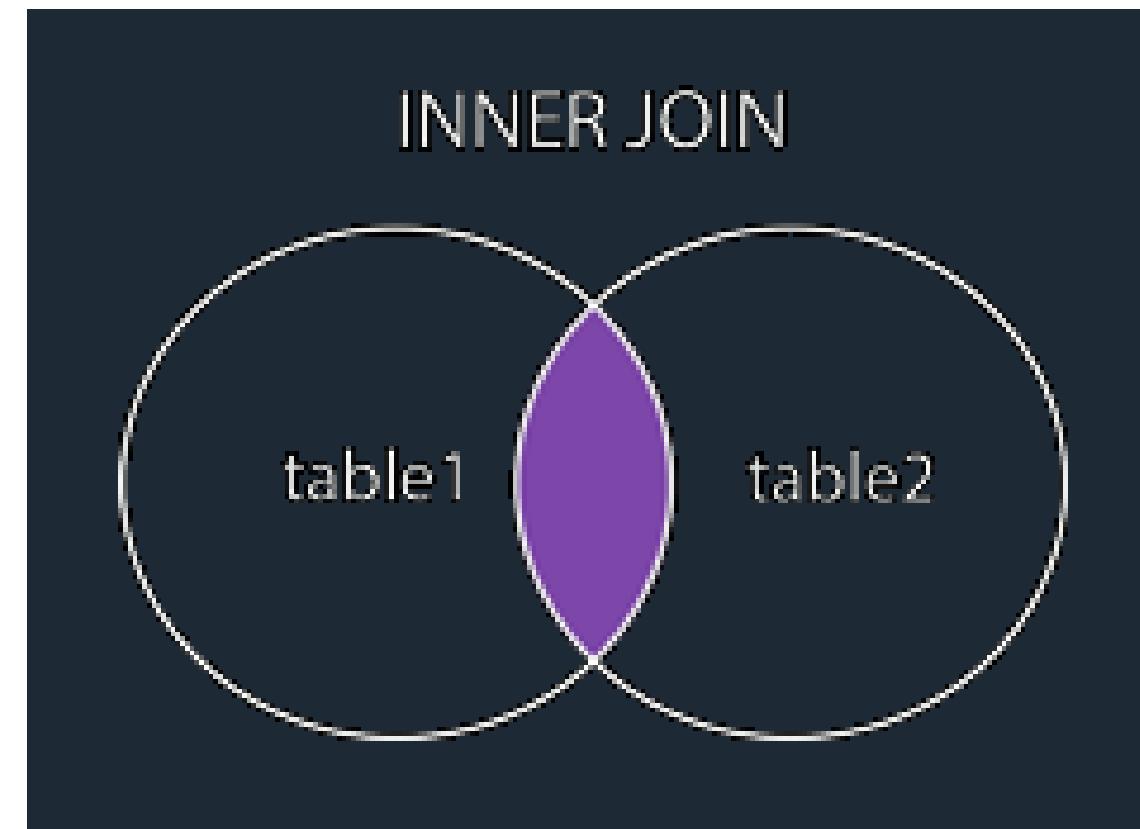
- 1.(INNER) JOIN: restituisce record con valori in comune**
- 2.LEFT (OUTER) JOIN: restituisce tutti i record della tabella di sinistra e i record con valori in comune**
- 3.RIGHT (OUTER) JOIN: restituisce tutti i record della tabella di destra e i record con valori in comune**
- 4.FULL (OUTER) JOIN: restituisce tutti i record quando è presente una corrispondenza in uno dei due a sinistra o tavolo a destra**



# SQL: INNER JOIN

**INNER JOIN** seleziona i record con valori corrispondenti in entrambe le tabelle.

- 1. SELECT column\_name(s)**
- 2. FROM table1**
- 3. INNER JOIN table2**
- 4. ON table1.column\_name = table2.column\_name;**



# SQL: INNER JOIN

**Esempio di utilizzo:**

```
1. SELECT Orders.OrderID, Customers.CustomerName  
2. FROM Orders  
3. INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

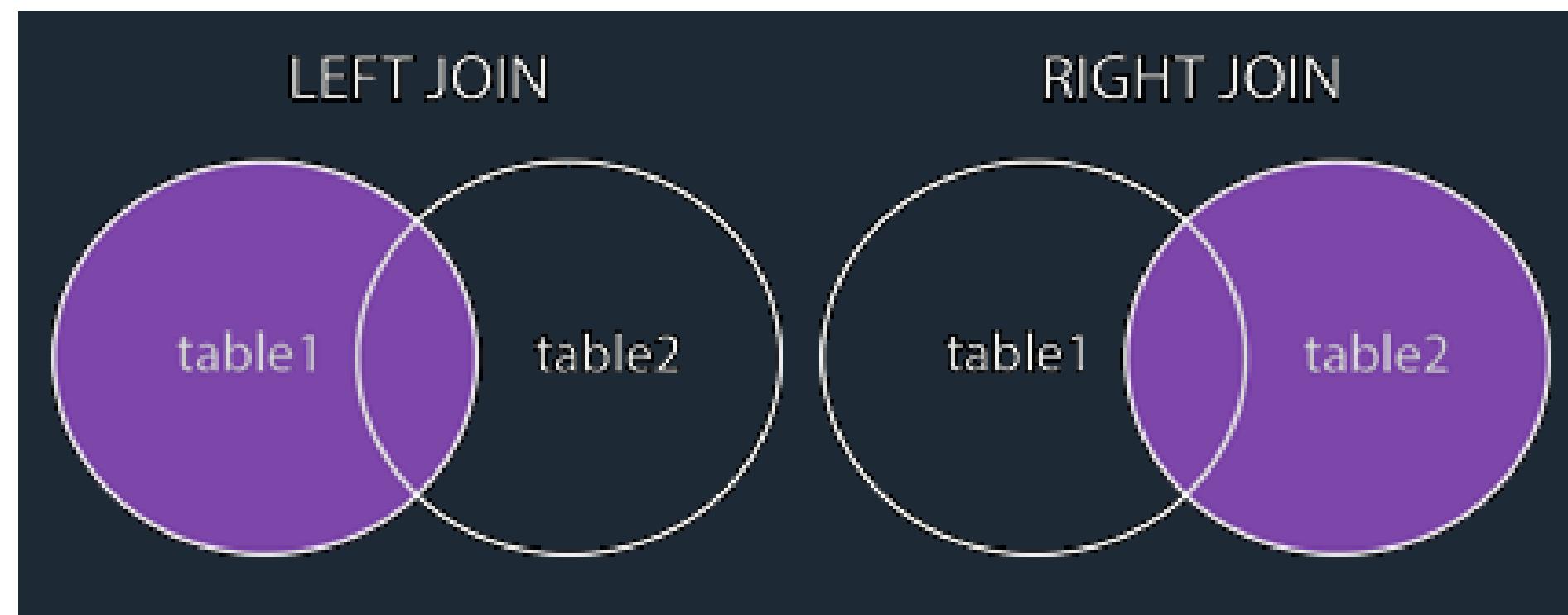
**Esempio di utilizzo con più join (da notare le parentesi):**

```
1. SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName  
2. FROM ((Orders  
3. INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)  
4. INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

# SQL: LEFT JOIN - RIGHT JOIN

Questi join restituiscono tutti i record della tabella di riferimento (Left o Right in base al tipo di join) insieme ai record con i valori corrispondenti dell'altra tabella (Se non c'è alcuna corrispondenza, i valori dalla tabella secondaria sono Null).

1. **SELECT column\_name(s)**
2. **FROM table1**
3. **LEFT JOIN table2**
4. **ON table1.column\_name = table2.column\_name;**



# SQL: LEFT JOIN - RIGHT JOIN

Un esempio dove recuperiamo e ordiniamo tutti i clienti in aggiunta recuperiamo anche l'eventuale id dell'ordine:

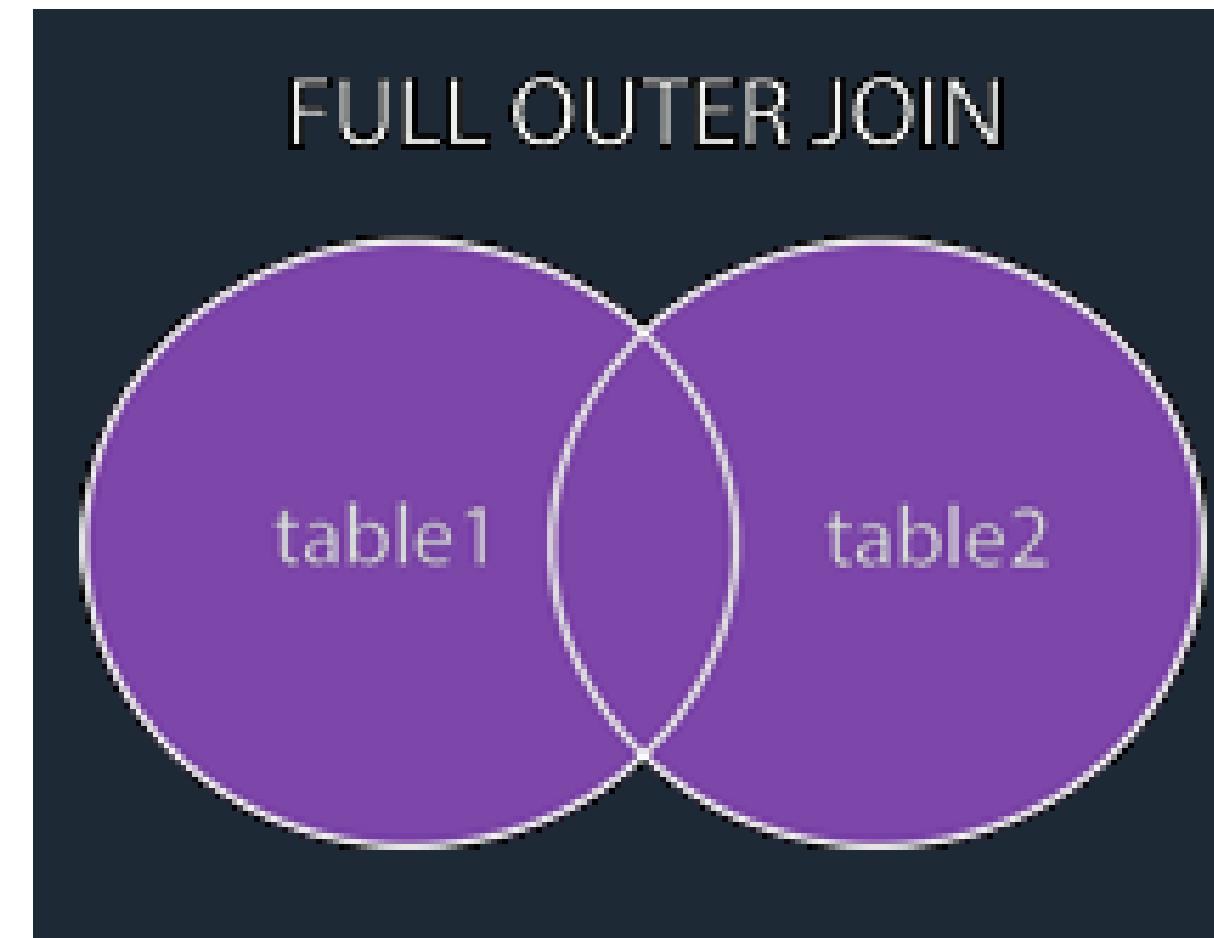
```
1. SELECT Orders.OrderID, Customers.CustomerName  
2. FROM Customers  
3. LEFT JOIN Orders  
4. ON Customers.CustomerID=Orders.CustomerID  
5. ORDER BY Customers.CustomerName;
```

OrderID	CustomerName
<i>null</i>	Alfreds Futterkiste
10308	Ana Trujillo Emparedados y helados
10365	Antonio Moreno Taquería

# SQL: CROSS JOIN

**CROSS JOIN restituisce tutti i record da entrambe le tavole.**

- 1. SELECT column\_name(s)**
- 2. FROM table1**
- 3. CROSS JOIN table2;**



# SQL: UNION

---

**L'operatore UNION viene utilizzato per combinare il result-set di risultati distinti di due o più istruzioni. Vanno però rispettate queste condizioni:**

- **Ogni istruzione SELECT all'interno deve avere lo stesso numero di colonne;**
- **Le colonne devono inoltre avere tipi di dati simili;**
- **Le colonne devono inoltre essere nello stesso ordine.**

**Sintassi:**

- 1. SELECT column\_name(s) FROM table1**
- 2. UNION**
- 3. SELECT column\_name(s) FROM table2;**

# SQL: UNION ALL

---

**È identico all'operatore UNION solo che include tutti i valori anche i duplicati.**

- 1. SELECT column\_name(s) FROM table1**
- 2. UNION ALL**
- 3. SELECT column\_name(s) FROM table2;**

**Esempio:**

- SELECT City FROM Customers**
- UNION ALL**
- SELECT City FROM Suppliers**
- ORDER BY City;**

# Esercizi 2

---

**Esercizi:**

- 1) Si vogliono recuperare dal database "world" la lingua e la nazione di ogni città.**
  
  - 2) Si vuole recuperare il numero di città per nazione dal database "world" mostrando anche il nome della nazione e ordinarli in base al numero di città.**
  
  - 3) Si vuole conoscere la lista di repubbliche con aspettativa di vita maggiore dei 70 anni, inoltre si vuole visualizzare anche la lingua parlata.**
-

# SQL: HAVING

---

**La clausola HAVING in SQL viene utilizzata con la clausola GROUP BY per filtrare i risultati di una query basandosi su condizioni aggregate. In altre parole, viene utilizzata per filtrare i risultati di gruppi di righe, spesso dopo l'applicazione di funzioni di aggregazione come COUNT, SUM, AVG, ecc.**

- 1. SELECT column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE condition**
- 4. GROUP BY column\_name(s)**
- 5. HAVING condition**
- 6. ORDER BY column\_name(s);**

**NOTA: deve essere usato prima dell'ORDER By.**

---

# SQL: EXISTS

L'operatore **EXISTS** viene utilizzato per verificare l'esistenza di qualsiasi record in una sottoquery, restituisce TRUE se la sottoquery restituisce uno o più record.

- 1. SELECT column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE EXISTS**
- 4. (SELECT column\_name FROM table\_name WHERE condition);**

L'esempio seguente restituisce TRUE ed elenca i fornitori con un prezzo del prodotto inferiore a 20:

- 1. SELECT SupplierName**
- 2. FROM Suppliers**
- 3. WHERE EXISTS**
- 4. (SELECT ProductName FROM Products WHERE Products.SupplierID =**
- 5. Suppliers.supplierID AND Price < 20);**

# SQL: ANY

**L'operatore ANY restituisce TRUE se UNO qualsiasi dei valori della sottoquery soddisfa la condizione dell'operatore.**

- 1. SELECT column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE column\_name operator ANY**
- 4. (SELECT column\_name FROM table\_name WHERE condition);**

**L'esempio seguente restituisce TRUE se qualsiasi prodotto ha ordini superiori a 90:**

- 1. SELECT ProductName**
- 2. FROM Products**
- 3. WHERE ProductID = ANY**
- 4. (SELECT ProductID FROM OrderDetails WHERE Quantity > 90);**

# SQL: ALL

L'operatore **ALL** restituisce TRUE se TUTTI i valori della sottoquery soddisfa la condizione dell'operatore, si può utilizzare in: **SELECT, WHERE, HAVING**.

- 1. SELECT ALL column\_name(s)**
- 2. FROM table\_name**
- 3. WHERE column\_name operator ALL**
- 4. (SELECT column\_name FROM table\_name WHERE condition);**

L'esempio seguente elenca ProductName se TUTTI i record nella tabella OrderDetails ha Quantity uguale a 10:

- 1. SELECT ProductName**
- 2. FROM Products**
- 3. WHERE ProductID = ALL (SELECT ProductID FROM OrderDetails WHERE Quantity = 10);**

# Esercizi 3

---

## ESERCIZIO 1:

**Si vuole recuperare dal database WORLD le lingue parlate per nazione con la rispettiva percentuale di utilizzo;**

## ESERCIZIO 2:

**Si vuole recuperare dal database WORLD le nazioni e la percentuale della lingua più parlata;**

## ESERCIZIO 3:

**Si vuole recuperare dal database WORLD il nome dei paesi con la lingua ufficiale più parlata in assoluto (100%).**

---

# SQL: INSERT INTO SELECT

---

**INSERT INTO con SELECT copia i record da una tabella e li aggiunge ad un'altra tabella.**

**L'operazione richiede che i tipi di dati delle colonne nelle tabelle di origine e di destinazione corrispondono.**

- 1. INSERT INTO table2 (column1, column2, column3, ...)**
- 2. SELECT column1, column2, column3, ...**
- 3. FROM table1**
- 4. WHERE condition;**

# SQL: CASE

---

**CASE passa attraverso dei WHEN restituisce con THEN un valore quando trova la prima condizione vera. Quindi, una volta che una condizione è vera, si fermerà la lettura del CASE e restituirà un valore. Se nessuna condizione è vera, restituisce il valore indicato nella clausola ELSE.**

- 1. CASE**
- 2. WHEN condition1 THEN result1**
- 3. WHEN condition2 THEN result2**
- 4. WHEN conditionN THEN resultN**
- 5. ELSE result**
- 6. END;**

**Possiamo quindi considerarlo un if-elif-else oppure uno switch di java**

---

# SQL: CASE

---

**Un esempio che passa un valore in base alla condizione vera trovata:**

```
1. SELECT OrderID, Quantity,  
2. CASE  
3. WHEN Quantity > 30 THEN 'The quantity is greater than 30'  
4. WHEN Quantity = 30 THEN 'The quantity is 30'  
5. ELSE 'The quantity is under 30'  
6. END AS QuantityText  
7. FROM OrderDetails;
```

# SQL: CASE

---

**Un esempio che sceglie in base alla condizione per quale colonna ordinare:**

- 1. SELECT CustomerName, City, Country**
- 2. FROM Customers**
- 3. ORDER BY**
- 4. (CASE**
- 5. WHEN City IS NULL THEN Country**
- 6. ELSE City**
- 7. END);**

# SQL: IFNULL() - COALESCE()

---

**IFNULL() è una funzione che restituisce un valore alternativo se l'espressione è NULL:**

**1. IFNULL(expression, alt\_value)**

**COALESCE() è una funzione che restituisce il primo valore non NULL:**

**1. COALESCE(val1, val2, ...., val\_n)**

# SQL: VIEW

---

**Una VIEW è un tabella virtuale basata sul result-set di una QUERY, contiene righe e colonne come una tabella. I campi di una VIEW sono campi di una o più tabelle presenti nel database.**

- 1. CREATE VIEW view\_name AS**
- 2. SELECT column1, column2, ...**
- 3. FROM table\_name**
- 4. WHERE condition;**

**Nota: una VIEW mostra sempre dati aggiornati, il di database ricrea la VIEW ad ogni QUERY.**

- 1.-- Creazione di una vista che mostra il nome e la popolazione dei paesi**
- 2. CREATE VIEW CountryView AS**
- 3. SELECT Name AS CountryName, Population**
- 4. FROM country;**

# SQL: VIEW

---

**Una VIEW si può aggiornare modificandola con il comando CREATE OR REPLACE VIEW:**

- 1. CREATE OR REPLACE VIEW view\_name AS**
- 2. SELECT column1, column2, ...**
- 3. FROM table\_name**
- 4. WHERE condition;**

**Per eliminare una VIEW invece bisogna usare DROP VIEW:**

- 1. DROP VIEW view\_name;**

# Esercizi 4

---

## ESERCIZIO 1:

Create una vista chiamata **TopCountries** che mostri il nome del paese e la sua popolazione per i paesi con una popolazione superiore a 50 milioni di abitanti;

## ESERCIZIO 2:

Create una vista chiamata **PopulationByContinent** che mostri il nome del continente e la popolazione totale per ciascun continente.

## ESERCIZIO 3:

Create una vista chiamata **CapitalCities** che mostri il nome del paese e il nome della sua capitale.

---