1) Cosa si intende per database?
Un database è un insieme di dati organizzati e strutturati in modo tale da poter essere facilmente gestiti, aggiornati e interrogati. È come un grande archivio digitale dove le informazioni sono memorizzate in tabelle collegate tra loro.

2) DBMS sta per Database Management System (Sistema di Gestione di Basi di Dati). È un software che permette di creare, gestire e interagire con un database. In pratica, fa da intermediario tra l'utente e il database, consentendo di inserire, modificare, cancellare e recuperare i dati in modo semplice e sicuro.

3) Indica le principali clausole di uno statement SELECT in ordine di esecuzione logica. Descrivi per ciascuna delle clausole indicate la logica di funzionamento.

1. FROM: Per prima cosa, il sistema individua la tabella (o le tabelle) da cui prendere i dati.
2. WHERE: Poi applica le condizioni per filtrare le righe, scartando quelle che non rispettano i criteri specificati.
3. GROUP BY: Successivamente, raggruppa le righe che hanno gli stessi valori nelle colonne specificate in questa clausola, in modo da poter eseguire funzioni di aggregazione su ciascun gruppo.
4. HAVING: Dopo aver raggruppato, applica eventuali filtri ai gruppi creati.
5. SELECT: A questo punto, seleziona le colonne da mostrare nel risultato finale.
6. ORDER BY: Infine, ordina le righe del risultato in base a una o più colonne, in modo crescente o decrescente.

4) Per spiegare il concetto di GROUP BY, userò uno scenario pratico basato su un catalogo musicale di Afrobeats.

La clausola GROUP BY in SQL ha lo scopo di raggruppare righe che hanno qualcosa in comune per poter poi fare dei calcoli su questi gruppi.

Come scenario pratico del suo utilizzo propongo una tabella di brani musicali del genere afrobeats che chiamo TracceMusicali:

| ID_Traccia | Titolo | Artista | Nazione_Artista |
|---|---|---|---|
| 101 | Last Last | Burna Boy | Nigeria |
| 102 | Essence | Wizkid ft. Tems | Nigeria |
| 103 | Ku Lo Sa | Oxlade | Nigeria |
| 104 | Sugarcane | Camidoh | Ghana |
| 105 | Calm Down | Rema | Nigeria |
| 106 | Peru | Fireboy DML | Nigeria |
| 107 | Terminator | King Promise | Ghana |

Volendo rispondere alla domanda: "Quante canzoni ho in catalogo per ogni nazione?"

Uso una query con GROUP BY come segue:

```
SELECT
    Nazione_Artista,
    COUNT(ID_Traccia) AS NumeroTracce
FROM
    TracceMusicali
GROUP BY
    Nazione_Artista;
```

Quello che fa questa query per arrivare al risultato è:

1. Con il FROM TracceMusicali: La query prende la tabella TracceMusicali come punto di partenza.
2. Con il GROUP BY Nazione_Artista: La query crea dei gruppi. Mette insieme tutte le righe che hanno lo stesso valore nella colonna Nazione_Artista. Basandosi sulla mia tabella, formerà due gruppi:
    1. Un gruppo "Nigeria" (con 5 canzoni).
    2. Un gruppo "Ghana" (con 2 canzoni).
3. Con il COUNT(ID_Traccia): dopo aver creato i gruppi, la funzione COUNT, conta quante righe (e quindi quante canzoni) ci sono all'interno di ciascun gruppo.
4. Con il SELECT Nazione_Artista: Ila clausola SELECT costruisce la tabella finale che vedremo come risultato. Mostra quindi il nome del gruppo (Nazione_Artista) e accanto il conteggio calcolato nel passaggio precedente (con l'etichetta NumeroTracce).

L'output della query che risponde quindi alla domanda "Quante canzoni ho in catalogo per ogni nazione?” è:

| Nazione_Artista | NumeroTracce |
|---|---|
| Ghana | 2 |
| Nigeria | 5 |

GROUP BY ha permesso di trasformare una lista di dati grezzi in un report aggregato, dando una visione d'insieme immediata e utile per analisi.

5) Descrivi la differenza tra uno schema OLTP e uno schema OLAP.

La differenza principale sta nello scopo per cui sono progettati.

- OLTP (Online Transaction Processing): È ottimizzato per gestire un gran numero di piccole transazioni in tempo reale, come inserimenti, modifiche e cancellazioni. L'obiettivo è la velocità e l'efficienza nelle operazioni quotidiane. Ad esempio il database di un sito di e-commerce che deve registrare tanti ordini velocemente. La struttura è normalizzata per evitare ridondanze.

- OLAP (Online Analytical Processing): È progettato per l'analisi di grandi volumi di dati. L'obiettivo è eseguire query complesse per estrarre informazioni e supportare le decisioni aziendali (Business Intelligence).

6) Dato un medesimo scenario di analisi, qual è la differenza in termini di risultato ottenibile tra una join e una subquery?

In molti casi, una JOIN e una subquery possono portare allo stesso identico risultato. La differenza sta più nel modo in cui si scrive la query.

- Una JOIN combina righe da due o più tabelle basandosi su una colonna in comune. Il risultato è un'unica tabella "virtuale" con le colonne di entrambe.
- Una subquery è una query dentro un'altra query. Può essere usata in varie clausole (WHERE, FROM, SELECT) per filtrare o per generare dati su cui la query principale lavorerà. Una subquery è più leggibile se la logica è complessa.

7) Cosa si intende per DML e DDL?
Sono due "sottolinguaggi" di SQL.

- DDL (Data Definition Language): Serve per definire e gestire la struttura del database. Sono i comandi che si usano per creare, modificare o eliminare oggetti come tabelle, indici e viste. I comandi principali sono CREATE, ALTER e DROP.

- DML (Data Manipulation Language): Serve per manipolare i dati all'interno delle tabelle. Sono i comandi che si usano più frequentemente per interrogare e aggiornare i dati. I comandi principali sono SELECT, INSERT, UPDATE e DELETE.

8) Si possono usare le funzioni YEAR e EXTRACT

1. Funzione YEAR()

Esempio: Per selezionare l'anno dalla colonna DataOrdine della tabella Ordini.

SELECT YEAR(DataOrdine) FROM Ordini;

Esempio con WHERE: Per trovare tutti gli ordini dell'anno 2024.

SELECT * FROM Ordini WHERE YEAR(DataOrdine) = 2024;

2. Funzione EXTRACT()

Esempio sulla stessa tabella Ordini:

SELECT EXTRACT(YEAR FROM DataOrdine) FROM Ordini;
Entrambe le funzioni restituiscono lo stesso risultato. Per MySQL, YEAR() è spesso preferita per la sua semplicità.

9) Qual è la differenza tra gli operatori logici AND e OR?

Sono usati nella clausola WHERE per combinare più condizioni.

- AND: Restituisce VERO solo se tutte le condizioni collegate sono vere.
- OR: Restituisce VERO se almeno una delle condizioni collegate è vera.

10) È possibile innestare una query nella clausola SELECT?

Sì, è possibile. Si chiama subquery scalare e deve restituire un solo valore (una riga e una colonna)

11) Qual è la differenza tra l’operatore logico OR e l’operatore logico IN?

Entrambi servono a verificare se un valore appartiene a un insieme di valori, ma IN è più conciso e spesso più efficiente.

- Usare OR richiede di ripetere il nome della colonna per ogni valore ad esempio:
    - WHERE citta = 'Bergamo' OR citta = 'Milano' OR citta = 'Napoli'
- L'operatore IN permette di specificare una lista di valori in modo più pulito:
    - WHERE citta IN ('Bergamo', 'Milano', 'Napoli')

Il risultato è lo stesso, ma la seconda versione è più leggibile e, se la lista di valori è lunga, è decisamente meglio.

12) L’operatore logico BETWEEN include anche gli estremi del range specificato?

Sì, l'operatore BETWEEN è inclusivo. Quando si scrive WHERE valore BETWEEN x AND y, il sistema seleziona tutte le righe in cui "valore" è maggiore o uguale a x e minore o uguale a y.

13) Case study

Task 1: Progettazione Concettuale e Logica

Analizzando lo scenario, ho supposto che bastino le 3 entità di partenza (Product, Region, Sales), perché ci sono delle gerarchie. Mettere la categoria dentro la tabella dei prodotti o la regione dentro quella degli stati creerebbe un sacco di dati ripetuti.

La cosa migliore è "normalizzare", cioè separare queste informazioni in tabelle specifiche.

Quindi, ho pensato di usare 5 tabelle in tutto:

- Una per le Categorie dei prodotti.
- Una per i Prodotti, che si collega alla sua categoria.
- Una per le Regioni di vendita.
- Una per gli Stati, che si collega alla sua regione.
- Infine, una tabella per le Vendite che lega un prodotto a uno stato in una certa data.

Progettazione Concettuale (Schema E/R a parole, non ho saputo usare lo strumento per costruire i grafici)

- Entità Categorie: Contiene le categorie.
    - Chiave: ID_Categoria
    - Attributi: NomeCategoria
- Entità Prodotti: Contiene i singoli giocattoli.
    - Chiave: ID_Prodotto

    - Attributi: NomeProdotto, PrezzoUnitario
- Entità Regioni: Contiene le macro-aree di vendita.
    - Chiave: ID_Regione
    - Attributi: NomeRegione
- Entità Stati: Contiene gli stati o nazioni.
    - Chiave: ID_Stato
    - Attributi: NomeStato
- Entità Vendite: registra ogni transazione.
    - Chiave: ID_Vendita
    - Attributi: DataVendita, Quantita

Relazioni:

- Prodotti e Categorie: Ogni prodotto appartiene a UNA sola categoria, ma una categoria può avere TANTI prodotti. (Relazione uno-a-molti)
- Stati e Regioni: Ogni stato appartiene a UNA sola regione, ma una regione ha TANTI stati. (Relazione uno-a-molti)
- Vendite, Prodotti e Stati: Ogni vendita riguarda UN solo prodotto e avviene in UN solo stato. Un prodotto e uno stato possono apparire in TANTE vendite.

Progettazione Logica

Questo è come ho immaginato la struttura delle tabelle con le loro colonne e i collegamenti.

- Categorie
  - ID_Categoria (PK, Intero)
  - NomeCategoria (Testo)
- Prodotti
  - ID_Prodotto (PK, Intero)
  - NomeProdotto (Testo)
  - PrezzoUnitario (Numero decimale)
  - ID_Categoria (FK che punta a Categorie)
- Regioni
  - ID_Regione (PK, Intero)
  - NomeRegione (Testo)
- Stati
  - ID_Stato (PK, Intero)
  - NomeStato (Testo)
  - ID_Regione (FK che punta a Regioni)
- Vendite
  - ID_Vendita (PK, Intero)
  - DataVendita (Data)
  - Quantita (Intero)
  - ID_Prodotto (FK che punta a Prodotti)
  - ID_Stato (FK che punta a Stati)

Task 2: Creazione delle Tabelle

Tabella per le categorie dei prodotti
```
CREATE TABLE Categorie (
    ID_Categoria INT PRIMARY KEY,
    NomeCategoria VARCHAR(100)
);
```

Tabella per le regioni geografiche
```
CREATE TABLE Regioni (
    ID_Regione INT PRIMARY KEY,
    NomeRegione VARCHAR(100)
);
```

Tabella per gli stati, collegata alle regioni
```
CREATE TABLE Stati (
    ID_Stato INT PRIMARY KEY,
    NomeStato VARCHAR(100),
    ID_Regione INT,
    FOREIGN KEY (ID_Regione) REFERENCES Regioni(ID_Regione)
);
```

Tabella per i prodotti, collegata alle categorie
```
CREATE TABLE Prodotti (
    ID_Prodotto INT PRIMARY KEY,
    NomeProdotto VARCHAR(100),
    PrezzoUnitario DECIMAL(10, 2),
    ID_Categoria INT,
    FOREIGN KEY (ID_Categoria) REFERENCES Categorie(ID_Categoria)
);
```

Infine, la tabella delle vendite che lega tutto

```
CREATE TABLE Vendite (
    ID_Vendita INT PRIMARY KEY,
    DataVendita DATE,
    Quantita INT,
    ID_Prodotto INT,
    ID_Stato INT,
    FOREIGN KEY (ID_Prodotto) REFERENCES Prodotti(ID_Prodotto),
    FOREIGN KEY (ID_Stato) REFERENCES Stati(ID_Stato)
);
```

Task 3: Popolamento delle Tabelle

Popolo le categorie

```
INSERT INTO Categorie (ID_Categoria, NomeCategoria) VALUES
(1, 'Biciclette'),
(2, 'Abbigliamento'),
(3, 'Action Figures');
```

Popolo le regioni

```
INSERT INTO Regioni (ID_Regione, NomeRegione) VALUES
(10, 'Europa del Sud'),
(20, 'Europa Occidentale');
```

Popolo gli stati

```
INSERT INTO Stati (ID_Stato, NomeStato, ID_Regione) VALUES
(101, 'Italia', 10),
(102, 'Grecia', 10),
(201, 'Francia', 20),
(202, 'Germania', 20);
```

Popolo i prodotti

```
INSERT INTO Prodotti (ID_Prodotto, NomeProdotto, PrezzoUnitario,
ID_Categoria) VALUES
(1001, 'Bikes-100', 250.00, 1),
(1002, 'Bikes-200', 350.50, 1),
(2001, 'Bike Glove M', 25.00, 2),
(2002, 'Bike Gloves L', 25.00, 2),
(3001, 'Super Robot X', 45.00, 3);
```

Popolo le vendite (con date diverse per i test)
INSERT INTO Vendite (ID_Vendita, DataVendita, Quantita, ID_Prodotto, ID_Stato) VALUES
(1, '2024-11-10', 2, 1001, 101), -- Italia, recente
(2, '2024-03-15', 1, 1002, 201), -- Francia, più di 180gg fa
(3, '2025-01-20', 10, 2001, 101),-- Italia, recente
(4, '2025-02-05', 5, 1001, 202), -- Germania, recente
(5, '2024-05-22', 3, 3001, 102); -- Grecia, più di 180gg fa

Task 4: Query di Analisi

Ecco le query per rispondere alle varie domande.

1. Verificare l'univocità delle Chiavi Primarie

Per controllare che non ci siano duplicati nelle chiavi primarie, cerco le chiavi che compaiono più di una volta. Se queste query non restituiscono risultati, allora la PK di Prodotti è univoca

SELECT ID_Prodotto, COUNT(*) FROM Prodotti GROUP BY ID_Prodotto HAVING COUNT(*) > 1;

Faccio lo stesso per le altre tabelle
SELECT ID_Vendita, COUNT(*) FROM Vendite GROUP BY ID_Vendita HAVING COUNT(*) > 1;
etc. per Categorie, Regioni, Stati

2. Elenco delle transazioni con dettagli

Questa query unisce tutte le tabelle per avere una vista completa. Per il campo booleano uso una CASE che controlla la differenza in giorni tra la data della vendita e oggi.

```
SELECT
    v.ID_Vendita,
    v.DataVendita,
    p.NomeProdotto,
    c.NomeCategoria,
    s.NomeStato,
    r.NomeRegione,
    CASE
        WHEN DATEDIFF(day, v.DataVendita, CURDATE()) > 180 THEN 'True'
        ELSE 'False'
    END AS Oltre180Giorni

FROM Vendite v
JOIN Prodotti p ON v.ID_Prodotto = p.ID_Prodotto
JOIN Categorie c ON p.ID_Categoria = c.ID_Categoria
JOIN Stati s ON v.ID_Stato = s.ID_Stato
JOIN Regioni r ON s.ID_Regione = r.ID_Regione;
```

3. Prodotti con vendite totali maggiori della media dell'ultimo anno

Qui uso le subquery: una per trovare l'ultimo anno in cui ci sono state vendite e un'altra per calcolare la quantità media venduta in quell'anno.

```
SELECT
    p.NomeProdotto,
    SUM(v.Quantita) as TotaleVenduto
FROM Vendite v
JOIN Prodotti p ON v.ID_Prodotto = p.ID_Prodotto
GROUP BY p.NomeProdotto
HAVING SUM(v.Quantita) > (
```

Sottoquery per la media delle vendite nell'ultimo anno censito

```
  SELECT AVG(Quantita)
    FROM Vendite
    WHERE YEAR(DataVendita) = (
```

Sottoquery per trovare l'ultimo anno

```
SELECT MAX(YEAR(DataVendita)) FROM Vendite
    )
);
```

4. Fatturato totale per prodotto per anno

Raggruppo per nome del prodotto e per anno, e calcolo il fatturato.

```
SELECT
    p.NomeProdotto,
    YEAR(v.DataVendita) AS Anno,
    SUM(v.Quantita * p.PrezzoUnitario) AS FatturatoTotale
```

```
FROM Vendite v
JOIN Prodotti p ON v.ID_Prodotto = p.ID_Prodotto
GROUP BY p.NomeProdotto, YEAR(v.DataVendita)
ORDER BY p.NomeProdotto, Anno;
```

5. Fatturato totale per stato per anno
Simile alla precedente, ma raggruppo per stato invece che per prodotto.

```
SELECT
    s.NomeStato,
    YEAR(v.DataVendita) AS Anno,
    SUM(v.Quantita * p.PrezzoUnitario) AS FatturatoTotale
FROM Vendite v
JOIN Prodotti p ON v.ID_Prodotto = p.ID_Prodotto
JOIN Stati s ON v.ID_Stato = s.ID_Stato
GROUP BY s.NomeStato, YEAR(v.DataVendita)
ORDER BY Anno, FatturatoTotale DESC;
```

6. Categoria di articoli più richiesta

Raggruppo per categoria, conto la quantità totale venduta e ordino in modo decrescente, prendendo solo il primo risultato

```
SELECT TOP 1
    c.NomeCategoria,
    SUM(v.Quantita) AS QuantitaTotaleVenduta
FROM Vendite v
JOIN Prodotti p ON v.ID_Prodotto = p.ID_Prodotto
JOIN Categorie c ON p.ID_Categoria = c.ID_Categoria
GROUP BY c.NomeCategoria
ORDER BY QuantitaTotaleVenduta DESC;
```

7. Prodotti invenduti

- LEFT JOIN
  Faccio una LEFT JOIN da Prodotti a Vendite e cerco le righe dove non c'è una corrispondenza nella tabella delle vendite (quindi il campo della vendita è NULL).
  SQL

  SELECT p.NomeProdotto
- FROM Prodotti p
- LEFT JOIN Vendite v ON p.ID_Prodotto = v.ID_Prodotto
- WHERE v.ID_Vendita IS NULL;

8. Creazione di una vista per i prodotti

Una vista è come una tabella virtuale. Questa unisce prodotti e categorie per avere le info pronte all'uso.

```
CREATE VIEW VistaProdotti AS
SELECT
    p.ID_Prodotto,
    p.NomeProdotto,
    c.NomeCategoria
FROM Prodotti p
JOIN Categorie c ON p.ID_Categoria = c.ID_Categoria;
```

Poi per usarla basta:
SELECT * FROM VistaProdotti;

9. Creazione di una vista per le informazioni geografiche

Stessa cosa per le informazioni geografiche.

```
CREATE VIEW VistaGeografica AS
SELECT
    s.ID_Stato,
    s.NomeStato,
    r.NomeRegione
FROM Stati s
JOIN Regioni r ON s.ID_Regione = r.ID_Regione;
```

Per usarla:

```
SELECT * FROM VistaGeografica;
```