DDL - Data Definition Language

Creazione di Database

```
CREATE DATABASE nome_database;
USE nome_database;
```

Creazione di Tabelle

```
CREATE TABLE studenti (
    id INT PRIMARY KEY AUTO_INCREMENT,
    nome VARCHAR(50) NOT NULL,
    cognome VARCHAR(50) NOT NULL,
    data_nascita DATE,
    classe VARCHAR(10),
    media_voti DECIMAL(4,2)
);
```

DML - Data Manipulation Language

Inserimento Dati

```
-- Inserimento completo
INSERT INTO studenti (nome, cognome, data_nascita, classe)
VALUES ('Mario', 'Rossi', '2006-05-10', '5A');

-- Inserimento multiplo
INSERT INTO studenti (nome, cognome, classe)
VALUES ('Luigi', 'Bianchi', '4B'),
       ('Anna', 'Verdi', '3C');
```

Aggiornamento Dati

```
-- Aggiornamento di tutti i record che soddisfano una condizione
UPDATE studenti
SET classe = '5B'
WHERE classe = '4B';

-- Aggiornamento con calcoli
UPDATE studenti
SET media_voti = media_voti + 0.5
WHERE media_voti < 6;
```

Eliminazione Dati

```
-- Eliminazione condizionale
DELETE FROM studenti
WHERE media_voti < 5;

-- Eliminazione totale (mantenendo la struttura)
DELETE FROM studenti;
-- oppure (più efficiente per tabelle grandi)
TRUNCATE TABLE studenti;
```

DQL - Data Query Language

Interrogazioni di Base

```
-- Selezione di tutte le colonne
SELECT * FROM studenti;

-- Selezione di colonne specifiche
SELECT nome, cognome, classe FROM studenti;

-- Selezione con condizioni
SELECT * FROM studenti WHERE classe = '5A';

-- Ordinamento
SELECT * FROM studenti ORDER BY cognome ASC, nome DESC;

-- Limitazione dei risultati
SELECT * FROM studenti LIMIT 10;
```

Operatori di Confronto

- `=`, `<>` o `!=`, `<`, `>`, `<=`, `>=`
- `BETWEEN`: intervallo di valori
- `IN`: insieme di valori
- `LIKE`: pattern matching con wildcard (`%` e `_`)
- `IS NULL`, `IS NOT NULL`: controllo valori nulli

```
-- Esempi
SELECT * FROM studenti WHERE media_voti BETWEEN 7 AND 9;
SELECT * FROM studenti WHERE classe IN ('5A', '5B', '5C');
SELECT * FROM studenti WHERE cognome LIKE 'R%'; -- inizia con R
SELECT * FROM studenti WHERE data_nascita IS NULL;
```

Funzioni di Aggregazione

- `COUNT()` : conta il numero di righe
- `SUM()` : somma valori
- `AVG()` : calcola la media
- `MAX()`, `MIN()` : massimo e minimo
- `GROUP BY` : raggruppa risultati
- `HAVING` : filtra gruppi (simile a WHERE ma per gruppi)

```sql
-- Esempi
SELECT COUNT(*) AS totale_studenti FROM studenti;
SELECT classe, AVG(media_voti) AS media_classe
FROM studenti
GROUP BY classe
HAVING AVG(media_voti) > 7;
```

JOIN

Permettono di combinare dati da più tabelle basandosi su relazioni.

```sql
-- INNER JOIN: solo record con corrispondenze in entrambe le tabelle
SELECT s.nome, s.cognome, c.nome AS corso
FROM studenti s
INNER JOIN iscrizioni i ON s.id = i.studente_id
INNER JOIN corsi c ON i.corso_id = c.id;

-- LEFT JOIN: tutti i record dalla tabella di sinistra
SELECT s.nome, s.cognome, c.nome AS corso
FROM studenti s
LEFT JOIN iscrizioni i ON s.id = i.studente_id
LEFT JOIN corsi c ON i.corso_id = c.id;

-- RIGHT JOIN: tutti i record dalla tabella di destra
SELECT s.nome, s.cognome, c.nome AS corso
FROM studenti s
RIGHT JOIN iscrizioni i ON s.id = i.studente_id
RIGHT JOIN corsi c ON i.corso_id = c.id;
```

Subquery

Query annidate all'interno di altre query.

```sql
-- Subquery nella clausola WHERE
SELECT nome, cognome
FROM studenti
WHERE id IN (SELECT studente_id FROM iscrizioni WHERE corso_id = 3);

-- Subquery come tabella
```

```sql
SELECT s.nome, media_classe.valore
FROM studenti s
JOIN (
    SELECT classe, AVG(media_voti) AS valore
    FROM studenti
    GROUP BY classe
) AS media_classe ON s.classe = media_classe.classe;
```