

Appunti su Basic SQL

Basato sulle slide del Prof. Danilo Montesi

15 maggio 2025

Indice

1	Introduzione a SQL	2
1.1	Cos'è SQL	2
1.2	Caratteristiche Principali	2
1.3	Standard vs. Dialetti	2
1.4	Storia	2
2	DDL (Data Definition Language) - Definire la Struttura	2
2.1	Database e Schemi	2
2.2	Tabelle	3
2.3	Tipi di Dati	3
2.3.1	Tipi Base	3
2.3.2	Tipi Personalizzati (Domini)	4
2.4	Vincoli (Constraints)	4
2.4.1	Vincoli comuni	4
2.4.2	FOREIGN KEY e Integrità Referenziale	4
2.4.3	Azioni Referenziali Triggerate	5
2.4.4	CHECK	5
2.5	Modificare la Struttura	5
2.6	Indici	5
3	DML (Data Manipulation Language) - Interrogare e Modificare i Dati	5
3.1	Interrogazioni (Query) - SELECT	5
3.1.1	Ordine Concettuale di Esecuzione	6
3.1.2	Clausole base e alias	6
3.1.3	Condizioni WHERE	6
3.1.4	DISTINCT	6
3.1.5	JOINS	6
3.1.6	Espressioni nella Target List	7
3.1.7	Ordinamento	7
3.1.8	Operazioni sugli Insiemi (Set Operations)	7
3.2	Subquery (Query Annidate)	7
3.2.1	Nelle clausole WHERE	7
3.2.2	Visibilità (Scope)	8
3.2.3	Nelle clausole FROM (Derived Tables)	8
3.2.4	Nelle clausole SELECT (Scalar Subqueries)	8
3.3	Funzioni Aggregate e Raggruppamento	8
3.3.1	Funzioni Aggregate	8
3.3.2	GROUP BY lista_colonne_raggruppamento	8
3.3.3	HAVING condizione_filtro_gruppi	9
3.3.4	NULLs e Raggruppamento	9
3.4	Modifica dei Dati	9
3.4.1	INSERT	9
3.4.2	UPDATE	9
3.4.3	DELETE	9
4	Concetti Chiave da Ricordare	10

1 Introduzione a SQL

1.1 Cos'è SQL

- Acronimo di “Structured Query Language”, oggi considerato un “nome proprio”.

1.2 Caratteristiche Principali

- Implementa sia **DDL (Data Definition Language)**: comandi per definire la struttura del database (tabelle, schemi, indici, ecc.).
- Implementa sia **DML (Data Manipulation Language)**: comandi per interrogare e modificare i dati.

1.3 Standard vs. Dialetti

- Esiste uno standard ISO, ma ogni DBMS (PostgreSQL, MySQL, SQL Server, Oracle, SQLite) ha le sue piccole variazioni ed estensioni grammaticali.
- *Esempio Pratico*: La sintassi per l'auto-incremento di un ID può variare (SERIAL in PostgreSQL, AUTO_INCREMENT in MySQL).

1.4 Storia

- Predecessore: SEQUEL (1974).
- Prime implementazioni: SQL/DS e Oracle (1981).
- Standard “de facto” dal 1983, con molte evoluzioni (SQL-86, SQL-89, SQL-92, SQL:1999, ecc.) che hanno introdotto:
 - Integrità referenziale (SQL-89)
 - Funzioni come COALESCE, NULLIF, CASE (SQL-92)
 - Concetti object-relational, trigger, funzioni esterne (SQL:1999)
 - Supporto per Java e XML (SQL:2003, SQL:2006)

2 DDL (Data Definition Language) - Definire la Struttura

2.1 Database e Schemi

- `CREATE DATABASE db_name;`
 - Crea un nuovo database, che è un contenitore per tabelle, viste, trigger, ecc.
 - *Esempio Pratico (SQLite)*: Quando esegui `sqlite3 miadatabase.db`, stai creando un file che funge da database.
 - *Nota*: In alcuni DBMS come MySQL, `CREATE SCHEMA` è un sinonimo di `CREATE DATABASE`.
- `CREATE SCHEMA schema_name [AUTHORIZATION 'user_name'];`
 - Uno schema è uno spazio dei nomi all'interno di un database. Serve a organizzare gli oggetti del database.
 - L'utente che esegue il comando diventa il proprietario dello schema, a meno che non sia specificato con `AUTHORIZATION`.
 - *Esempio Pratico (PostgreSQL)*: Spesso si usa lo schema `public` di default, ma potresti creare schemi come `accounting`, `sales` per separare logicamente le tabelle.

2.2 Tabelle

- Sintassi base:

```
CREATE TABLE table_name (  
  colonna1 TIPO_DATI [vincoli],  
  colonna2 TIPO_DATI [vincoli],  
  ...  
);
```

- Definisce una nuova tabella (relazione) con le sue colonne (attributi), i tipi di dato per ciascuna colonna e i vincoli iniziali.
- *Esempio Pratico (corrispettivo User con Prisma):*

```
// Prisma Schema  
model User {  
  id      Int      @id @default(autoincrement())  
  email   String   @unique  
  name    String?  
}
```

Equivalente a:

```
-- SQL (es. PostgreSQL)  
CREATE TABLE User (  
  id SERIAL PRIMARY KEY,  
  email VARCHAR(255) UNIQUE NOT NULL,  
  name VARCHAR(255)  
);
```

- Esempio dalla slide:

```
CREATE TABLE EMPLOYEE (  
  Number CHARACTER(6) PRIMARY KEY,  
  Name CHARACTER(20) NOT NULL,  
  Surname CHARACTER(20) NOT NULL,  
  Dept CHARACTER(15),  
  Wage NUMERIC(9) DEFAULT 0,  
  FOREIGN KEY(Dept) REFERENCES DEPARTMENT(Dept)  
);
```

2.3 Tipi di Dati

2.3.1 Tipi Base

- CHARACTER(n), VARCHAR(n): Stringhe di caratteri a lunghezza fissa o variabile.
- NUMERIC(p,s), INTEGER, SMALLINT, DECIMAL: Numeri interi o decimali.
- DATE, TIME, TIMESTAMP, INTERVAL: Per date e orari.
- BOOLEAN: Valori vero/falso.
- BLOB, CLOB: (Binary/Character Large Object) Per grandi quantità di dati binari o testuali.

2.3.2 Tipi Personalizzati (Domini)

- CREATE DOMAIN domain_name AS tipo_base [DEFAULT valore_default] [CHECK (condizione)];
- Permette di definire un tipo di dato riutilizzabile con vincoli e valori di default specifici.
- Esempio dalla slide:

```
CREATE DOMAIN Grade
AS SMALLINT DEFAULT NULL
CHECK (value >= 18 AND value <= 30);
```

Questo Grade può poi essere usato come tipo di dato per una colonna.

2.4 Vincoli (Constraints)

Servono a garantire l'integrità e la coerenza dei dati.

2.4.1 Vincoli comuni

- NOT NULL: La colonna non può contenere valori NULL.
- UNIQUE: I valori nella colonna (o combinazione di colonne) devono essere unici.
- PRIMARY KEY:
 - Identifica univocamente ogni riga. Implica NOT NULL e UNIQUE.
 - Solo una per tabella. Può essere su colonna singola o multipla.
 - Esempio (inline): Number CHARACTER(6) PRIMARY KEY
 - Esempio (standalone): PRIMARY KEY (Number)
- **Attenzione (Slide 23):**
 - UNIQUE (Surname, Name): La *combinazione* di cognome e nome deve essere unica.
 - Surname CHARACTER(20) UNIQUE, Name CHARACTER(20) UNIQUE: Il cognome deve essere unico e il nome deve essere unico (indipendentemente).

2.4.2 FOREIGN KEY e Integrità Referenziale

- FOREIGN KEY (colonna_fk) REFERENCES tabella_riferita (colonna_pk_riferita)
- Garantisce che i valori nella colonna_fk esistano nella colonna_pk_riferita della tabella_riferita.
- *Esempio Pratico (Relazione Post-User con Prisma):*

```
// Prisma Schema
model User {
  id      Int      @id @default(autoincrement())
  posts   Post[]
}
model Post {
  id      Int      @id @default(autoincrement())
  author   User    @relation(fields: [authorId],
    ↪ references: [id])
  authorId Int // Foreign Key
}
```

2.4.3 Azioni Referenziali Triggerate

Cosa succede se un record referenziato viene cancellato o aggiornato: ON DELETE | ON UPDATE

- CASCADE: Propaga l'azione (es. se cancello un utente, cancella anche i suoi post).
- SET NULL: Imposta la foreign key a NULL.
- SET DEFAULT: Imposta la foreign key al suo valore di default.
- NO ACTION / RESTRICT: Impedisce l'operazione (spesso il default).

2.4.4 CHECK

- CHECK (condizione): Specifica una condizione che deve essere vera per ogni riga.
- Esempio: CHECK (Wage > 0)

2.5 Modificare la Struttura

- ALTER DOMAIN domain_name [...opzioni...];
- ALTER TABLE table_name [...opzioni...];
 - Opzioni: ADD COLUMN, DROP COLUMN col_name [RESTRICT|CASCADE], ALTER COLUMN, ADD CONSTRAINT, DROP CONSTRAINT.
- DROP DOMAIN domain_name;
- DROP TABLE table_name; (cancella la tabella e tutti i suoi dati!)

2.6 Indici

- CREATE INDEX index_name ON table_name (colonna1, [colonna2, ...]);
- Migliorano le performance delle query.
- Strutture dati fisiche, non logiche.
- Le PRIMARY KEY e le colonne UNIQUE creano automaticamente un indice.
- *Esempio Pratico:* Se fai spesso ricerche di utenti per email, un indice su User(email) velocizzerà molto.

3 DML (Data Manipulation Language) - Interrogare e Modificare i Dati

3.1 Interrogazioni (Query) - SELECT

La struttura base è:

```
SELECT [DISTINCT] {* | lista_colonne | espressioni [AS alias_colonna]}
FROM tabella1 [AS alias_tabella1]
[, tabella2 [AS alias_tabella2] ... |
JOIN_TYPE tabella2 ON condizione_join]
[WHERE condizione_filtro_righe]
[GROUP BY lista_colonne_raggruppamento]
[HAVING condizione_filtro_gruppi]
[ORDER BY lista_colonne_ordinamento [ASC|DESC]];
```

3.1.1 Ordine Concettuale di Esecuzione

1. FROM (e JOINS)
2. WHERE
3. GROUP BY
4. HAVING
5. SELECT
6. DISTINCT
7. ORDER BY

3.1.2 Clausole base e alias

- **SELECT ***: Seleziona tutte le colonne.
- Rinominare Colonne e Tabelle (Alias): **AS nome_alias**

```
SELECT P.Name AS GivenName FROM PEOPLE AS P;
```

3.1.3 Condizioni WHERE

- Operatori logici: AND, OR, NOT.
- Operatori di confronto: =, <>, <, >, <=, >=.
- LIKE: Pattern matching (% per zero o più caratteri, _ per un singolo carattere).

```
WHERE Name LIKE 'J_m%';
```

- IS NULL / IS NOT NULL: Per verificare valori NULL.

3.1.4 DISTINCT

- Rimuove le righe duplicate dal risultato.

3.1.5 JOINS

Combinano righe da due o più tabelle.

- **Implicit JOIN** (sconsigliato):

```
SELECT ... FROM TableA, TableB WHERE TableA.id = TableB.a_id;
```

- **Explicit JOIN** (preferito):

- **INNER JOIN** (o solo JOIN): Solo righe con corrispondenza in entrambe.

```
SELECT ... FROM TableA INNER JOIN TableB ON TableA.id  
↪ = TableB.a_id;
```

- **LEFT [OUTER] JOIN**: Tutte le righe da sinistra, e le corrispondenti da destra (o NULL).
- **RIGHT [OUTER] JOIN**: Tutte le righe da destra, e le corrispondenti da sinistra (o NULL).
- **FULL [OUTER] JOIN**: Tutte le righe da entrambe; NULL dove non c'è corrispondenza.

- NATURAL JOIN: Join automatico su colonne con lo stesso nome (usare con cautela).
- *Esempio Pratico (Left Join):* Trovare tutti gli utenti e i loro post.

```
SELECT U.name, P.title
FROM User U LEFT JOIN Post P ON U.id = P.authorId;
```

3.1.6 Espressioni nella Target List

```
SELECT Income / 2 AS halvedIncome FROM PEOPLE;
```

3.1.7 Ordinamento

- ORDER BY colonna [ASC|DESC]; (ASC è il default).

3.1.8 Operazioni sugli Insiemi (Set Operations)

Le query devono avere lo stesso numero di colonne e tipi compatibili.

- UNION: Combina risultati, rimuovendo duplicati.
- UNION ALL: Come UNION, ma mantiene i duplicati.
- INTERSECT: Righe presenti in entrambi i risultati.
- EXCEPT (o MINUS): Righe nel primo risultato ma non nel secondo.
- *Nota sulla denominazione delle colonne:* I nomi sono presi dalla prima query SELECT.

3.2 Subquery (Query Annidate)

Una query all'interno di un'altra.

3.2.1 Nelle clausole WHERE

- Con operatori di confronto: la subquery deve restituire un valore scalare.

```
SELECT Name FROM PEOPLE WHERE Income = (SELECT MAX(Income) FROM
↪ PEOPLE);
```

- IN: Verifica se un valore è nel set di risultati della subquery.

```
SELECT Name FROM PEOPLE WHERE Name IN (SELECT Father FROM
↪ FATHERHOOD);
```

- ANY / SOME, ALL: Usati con operatori di confronto.
 - valore > ANY (subquery): vero se valore > di almeno un valore della subquery.
 - valore > ALL (subquery): vero se valore > di tutti i valori della subquery.
- EXISTS: Vero se la subquery restituisce almeno una riga.

```
SELECT Name FROM PEOPLE P
WHERE EXISTS (SELECT * FROM FATHERHOOD F WHERE F.Father = P.Name);
```

- NOT EXISTS: Vero se la subquery non restituisce righe.

3.2.2 Visibilità (Scope)

- Una subquery può fare riferimento a colonne della query esterna (subquery correlata).
- La query esterna non può fare riferimento a colonne definite solo nella subquery.
- Se un nome di colonna è ambiguo, si assume quello dello scope più interno.

3.2.3 Nelle clausole FROM (Derived Tables)

La subquery agisce come una tabella temporanea e deve avere un alias.

```
SELECT P.Name, J.Child
FROM PEOPLE P, (SELECT Child FROM FATHERHOOD WHERE Father='Jim') AS J
WHERE P.Name = J.Child;
```

3.2.4 Nelle clausole SELECT (Scalar Subqueries)

La subquery deve restituire un singolo valore per ogni riga della query esterna.

```
SELECT C.Num, (SELECT COUNT(*) FROM ORDERS O WHERE O.CustomerNum = C.Num) AS
↪ OrderCount
FROM CUSTOMER C;
```

3.3 Funzioni Aggregate e Raggruppamento

3.3.1 Funzioni Aggregate

- COUNT(), SUM(), AVG(), MIN(), MAX().
- Operano su un insieme di righe e restituiscono un singolo valore.
 - COUNT(*): conta tutte le righe.
 - COUNT(colonna): conta le righe dove colonna non è NULL.
 - COUNT(DISTINCT colonna): conta i valori unici non NULL.
 - Le altre funzioni ignorano i NULL.
- **Attenzione:** Non mischiare colonne non aggregate con funzioni aggregate nella SELECT list a meno che le colonne non aggregate non siano nella GROUP BY.
 - Errato: SELECT Name, MAX(Income) FROM PEOPLE;
 - Corretto: SELECT MAX(Income) FROM PEOPLE;

3.3.2 GROUP BY lista_colonne_raggruppamento

- Raggruppa le righe che hanno gli stessi valori nelle colonne specificate.
- Le funzioni aggregate vengono applicate a ciascun gruppo.
- Esempio:

```
SELECT Dept, AVG(Wage) FROM EMPLOYEE GROUP BY Dept;
```

3.3.3 HAVING condizione_filtro_gruppi

- Filtra i gruppi creati da GROUP BY. La condizione in HAVING di solito coinvolge funzioni aggregate.
- WHERE filtra le righe *prima* del raggruppamento, HAVING filtra i gruppi *dopo*.
- Esempio:

```
SELECT Dept, AVG(Wage) FROM EMPLOYEE
GROUP BY Dept
HAVING AVG(Wage) > 50000;
```

3.3.4 NULLs e Raggruppamento

- I valori NULL in una colonna di raggruppamento formano un gruppo a sé stante.

3.4 Modifica dei Dati

3.4.1 INSERT

- ```
INSERT INTO table_name [(colonna1, colonna2, ...)]
VALUES (valore1, valore2, ...);
```
- Aggiunge una nuova riga. Se la lista colonne è omessa, fornire valori per tutte le colonne nell'ordine definito.

- ```
INSERT INTO table_name [(colonna1, ...)]
SELECT query_che_restituisce_righe_compatibili;
```

3.4.2 UPDATE

- ```
UPDATE table_name
SET colonna1 = valore1, colonna2 = valore2, ...
[WHERE condizione];
```
- Modifica righe che soddisfano la condizione. **ATTENZIONE:** Senza WHERE, aggiorna tutte le righe!
- Il valore può essere un'espressione, NULL, DEFAULT, o una subquery scalare.

---

```
UPDATE PEOPLE SET Income = Income * 1.1 WHERE Age < 30;
```

---

### 3.4.3 DELETE

- ```
DELETE FROM table_name [WHERE condizione];
```
- Cancella righe che soddisfano la condizione. **ATTENZIONE:** Senza WHERE, cancella tutte le righe!
- Può innescare azioni referenziali.

4 Concetti Chiave da Ricordare

1. **SQL è Dichiarativo:** Dici *cosa* vuoi, non *come* ottenerlo.
2. **Integrità dei Dati:** I vincoli sono fondamentali.
3. **NULL è Speciale:** Rappresenta assenza di valore. Va trattato con `IS NULL` / `IS NOT NULL`.
4. **JOINS sono Potenti:** Cuore delle query relazionali. Comprendere `INNER` vs `OUTER JOINS` è cruciale.
5. **Aggregazione e Raggruppamento:** `GROUP BY`, funzioni aggregate e `HAVING` permettono calcoli sui dati.
6. **Subquery:** Offrono flessibilità per query complesse.