

Appunti su SQL Avanzato

Basato sulle slide del Prof. Danilo Montesi

15 maggio 2025

Indice

1	Vincoli (Constraints)	2
1.1	CHECK	2
1.2	ASSERTION	2
2	Viste (Views)	2
2.1	Aggiornamento delle Viste e WITH CHECK OPTION	3
2.2	Interrogare le Viste	3
3	Query Ricorsive (WITH RECURSIVE)	4
4	Funzioni Scalari	4
4.1	Temporal	4
4.2	Stringhe	4
4.3	Casting	4
4.4	Condizionali	5
5	Sicurezza del Database	5
5.1	Privilegi	5
5.2	GRANT e REVOKE	5
5.3	Discussione sui Privilegi	5
6	Autorizzazioni: RBAC (Role-Based Access Control)	6
7	Transazioni	6

1 Vincoli (Constraints)

1.1 CHECK

- **Concetto:** Specifica vincoli sui valori che una tupla (riga) può assumere. È una forma di validazione dei dati a livello di database.

- **Sintassi:** CHECK (Predicate)

- **Esempi:**

- Semplice:

```
Gender CHARACTER NOT NULL CHECK (Gender IN ('M', 'F'))
```

- Semplice:

```
Salary INTEGER CHECK (Salary >= 0)
```

- Complesso (con subquery):

```
-- Assicura che lo stipendio di un impiegato non superi
-- quello del suo supervisore.
-- Nota: le subquery nei CHECK non sono supportate da tutti i DBMS.
CHECK (Salary <= (SELECT Salary
FROM EMPLOYEE J
WHERE Supervisor = J.Number))
```

- Derivato:

```
-- Assicura la coerenza per campi calcolati.
CHECK (Net = Salary - Withholding)
```

- **Importanza:** Se un INSERT o UPDATE viola un vincolo CHECK, l'operazione fallisce, mantenendo l'integrità dei dati.

1.2 ASSERTION

- **Concetto:** Definisce vincoli a livello di schema, cioè che coinvolgono potenzialmente più tabelle o l'intero database, non solo una singola tupla.

- **Sintassi:** CREATE ASSERTION NomeAsserzione CHECK (Predicate)

- **Esempio:**

```
-- Questa asserzione garantisce che la tabella EMPLOYEE
-- non sia mai completamente vuota.
CREATE ASSERTION AtLeastOneEmployee
CHECK (1 <= (SELECT COUNT(*) FROM EMPLOYEE));
```

- *Nota Pratica:* Anche qui, il supporto completo (specialmente con subquery complesse) varia tra i DBMS.

2 Viste (Views)

- **Concetto:** Una vista è una tabella virtuale il cui contenuto è definito da una query. Non memorizza dati fisicamente (generalmente), ma esegue la sua query sottostante ogni volta che viene interrogata.

- **Sintassi:** CREATE VIEW NomeVista [(ListaAttributi)] AS SelectStatement
- **Esempio:**

```
CREATE VIEW ADMINEMPLOYEES (Name, Surname, Salary) AS
SELECT Name, Surname, Salary
FROM EMPLOYEE
WHERE Dept = 'Administration' AND Salary > 10;
```

- **Utilizzi:**
 - **Semplificazione:** Nascondere la complessità di query complesse.
 - **Sicurezza:** Limitare l'accesso a determinate colonne o righe di una tabella.
 - **Indipendenza logica dei dati:** Se la struttura delle tabelle sottostanti cambia, la vista può essere modificata per mantenere la stessa interfaccia per gli utenti/applicazioni.

2.1 Aggiornamento delle Viste e WITH CHECK OPTION

- Le viste possono essere aggiornabili (tramite INSERT, UPDATE, DELETE) se definite su una singola tabella e soddisfano certe condizioni.
- **WITH CHECK OPTION:** Se specificato, qualsiasi INSERT o UPDATE eseguito tramite la vista deve soddisfare la clausola WHERE della vista stessa.
- **Esempio:**

```
CREATE VIEW POORADMINEMPLOYEES AS
SELECT *
FROM ADMINEMPLOYEES -- Supponiamo sia una vista o tabella
WHERE Salary < 50
WITH CHECK OPTION;
```

Se si tenta di fare UPDATE POORADMINEMPLOYEES SET Salary = 60 WHERE Name = 'Ann', l'operazione fallirà.

- **LOCAL vs CASCADED (per viste su viste):**
 - * **LOCAL:** Il CHECK OPTION si applica solo alla definizione della vista corrente.
 - * **CASCADED:** Il CHECK OPTION si applica alla vista corrente E a tutte le viste sottostanti.

2.2 Interrogare le Viste

- Si interrogano come normali tabelle. Il DBMS sostituisce la vista con la sua definizione.
- **Utilità per query complesse:**
 - **Problema:** "Calcolare la media del numero di uffici distinti per dipartimento". Una query come SELECT AVG(COUNT(DISTINCT Office)) FROM EMPLOYEE GROUP BY Dept è errata perché non si possono annidare funzioni aggregate direttamente.
 - **Soluzione con Vista:**

```
CREATE VIEW DEPTOFFICES (NameDept, OffNum) AS
SELECT Dept, COUNT(DISTINCT Office)
FROM EMPLOYEE
GROUP BY Dept;

SELECT AVG(OffNum) FROM DEPTOFFICES;
```

3 Query Ricorsive (WITH RECURSIVE)

- **Concetto:** Permettono di interrogare dati gerarchici o grafi. SQL:1999 ha introdotto le Common Table Expressions (CTE) ricorsive.

- **Sintassi Base:**

```
WITH RECURSIVE NomeCTE (colonne) AS (  
    -- Membro Ancora (non ricorsivo, caso base)  
    SELECT ...  
    UNION ALL  
    -- Membro Ricorsivo (richiama NomeCTE)  
    SELECT ... FROM NomeCTE JOIN ...  
)  
SELECT * FROM NomeCTE;
```

- **Esempio (Trovare tutti gli antenati):** Data una tabella FATHERHOOD(Father, Child)

```
WITH RECURSIVE ANCESTORS (Ancestor, Descendant) AS (  
    -- Caso base: padri diretti  
    SELECT Father, Child FROM FATHERHOOD  
    UNION ALL  
    -- Passo ricorsivo: il padre di un antenato  
    -- è anche un antenato  
    SELECT FH.Father, A.Descendant  
    FROM FATHERHOOD FH, ANCESTORS A  
    WHERE FH.Child = A.Ancestor  
)  
SELECT * FROM ANCESTORS;
```

4 Funzioni Scalari

Funzioni che operano su valori singoli e restituiscono un singolo valore per tupla.

4.1 Temporal

- `CURRENT_DATE()`: Data corrente.
- `EXTRACT(parte FROM espressione_data)`: Estrae una parte da una data (es. `EXTRACT(YEAR FROM OrderDate)`).
- Esempio:

```
SELECT EXTRACT(YEAR FROM OrderDate) AS OrderYear  
FROM ORDERS  
WHERE DATE(OrderDate) = CURRENT_DATE();
```

4.2 Stringhe

- `CHAR_LENGTH(stringa)`: Lunghezza della stringa.
- `LOWER(stringa)`: Stringa in minuscolo.

4.3 Casting

- `CAST(espressione AS NuovoTipo)`: Converte un valore in un altro tipo di dato.

4.4 Condizionali

- COALESCE(expr1, expr2, ..., default): Restituisce la prima espressione non-NULL nella lista.
 - Esempio: SELECT COALESCE(Mobile, PhoneHome, 'N/A') FROM EMPLOYEE;
- NULLIF(expr1, expr2): Restituisce NULL se expr1 = expr2, altrimenti restituisce expr1.
 - Esempio: SELECT NULLIF(Dept, 'Unknown') FROM EMPLOYEE;
- CASE: Struttura if-then-else in SQL.
 - Sintassi "Searched":

```
CASE
WHEN condizione1 THEN risultato1
WHEN condizione2 THEN risultato2
...
ELSE risultato_default
END
```

- Esempio: Calcolo tasse veicoli

```
SELECT PlateNum,
(CASE Type
WHEN 'Car' THEN 2.58 * KWatt
WHEN 'Moto' THEN (22.00 + 1.00 * KWatt)
ELSE NULL
END) AS Tax
FROM VEHICLE
WHERE Year > 1975;
```

5 Sicurezza del Database

5.1 Privilegi

- SQL permette di concedere privilegi specifici (es. SELECT, INSERT, UPDATE, DELETE, REFERENCES, USAGE) agli utenti.
- I privilegi possono essere su: intero DB, tabelle, viste, colonne, domini.

5.2 GRANT e REVOKE

- GRANT: Concede privilegi.
 - Sintassi: GRANT <Privilegi | ALL PRIVILEGES> ON Risorsa TO Utenti [WITH GRANT OPTION];
 - WITH GRANT OPTION: Permette all'utente ricevente di propagare quel privilegio ad altri.
 - Esempio: GRANT SELECT ON DEPARTMENT TO Jack;
- REVOKE: Rimuove privilegi.
 - Sintassi: REVOKE Privilegi ON Risorsa FROM Utenti [RESTRICT | CASCADE];
 - RESTRICT (default): La revoca fallisce se altri utenti dipendono da quel grant.
 - CASCADE: La revoca si estende a tutti gli utenti a cui il privilegio è stato propagato.

5.3 Discussione sui Privilegi

- Il sistema dovrebbe nascondere le parti del DB non accessibili senza dare indizi sulla loro esistenza.
- Le **viste** sono uno strumento chiave per la sicurezza: si possono concedere privilegi su una vista che mostra solo certe righe/colonne.

6 Autorizzazioni: RBAC (Role-Based Access Control)

- **Concetto:** SQL-3 introduce RBAC. Un **ROLE** (ruolo) è un contenitore di privilegi.

1. Si creano ruoli.
2. Si concedono privilegi AI RUOLI.
3. Si concedono I RUOLI AGLI UTENTI.

- **Comandi RBAC:**

- `CREATE ROLE NomeRuolo;`
- `GRANT Privilegio ON Risorsa TO NomeRuolo;`
- `GRANT NomeRuolo TO NomeUtente;`
- `SET ROLE NomeRuolo;`

- **Esempio RBAC:**

```
-- 1. Crea il ruolo
CREATE ROLE Employee;
-- 2. Concedi un privilegio al ruolo
GRANT CREATE TABLE TO Employee;
-- 3. Assegna il ruolo a un utente
GRANT Employee TO 'specific_user';
```

7 Transazioni

- **Concetto:** Una transazione è un'unità logica di elaborazione del database, trattata come un'operazione atomica.

- **Proprietà ACID:**

- **Atomicity (Atomicità):** O tutto o niente.
- **Consistency (Consistenza):** Porta il DB da uno stato valido a un altro.
- **Isolation (Isolamento):** Le transazioni concorrenti non interferiscono.
- **Durability (Durabilità):** Le modifiche confermate (`COMMIT`) sono permanenti.

- **Supporto SQL per Transazioni:**

- `START TRANSACTION;` (o `BEGIN TRANSACTION;`)
- `COMMIT [WORK];` : Salva permanentemente le modifiche.
- `ROLLBACK [WORK];` : Annulla tutte le modifiche.
- `AUTOCOMMIT:` Modalità in cui ogni singola istruzione SQL è una transazione.

- **Esempio Transazione:**

```
START TRANSACTION;
UPDATE BANKACCOUNT SET Balance = Balance - 10
WHERE AccountNumber = 42177;
UPDATE BANKACCOUNT SET Balance = Balance + 10
WHERE AccountNumber = 12202;
-- Se tutto va bene:
COMMIT WORK;
-- Se c'è un errore (da verificare in logica applicativa):
-- ROLLBACK WORK;
```
