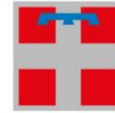




Cofinanziato
dall'Unione europea



REGIONE
PIEMONTE

FinTech Software Developer- 2023/2025

Basi di dati SQL

Docente: Roi Davide Simone

Ripasso:

DB vs File System

- sicurezza,
- accesso,
- contemporaneità,
- velocità

Titolo argomento: Architettura Postgresql – Constraints, rules & Defaults

SQL: linguaggio DB di tipo relazionale, universale (DDL, DML, DCL, TCL)

NoSQL invece prende altre altre realtà, ma limitazioni

l'uso di nosql è applicato su social, wikipedia, chat...

Roi Davide
Dispense

Constraints

Alcuni tipi di vincoli (constraints) che si possono utilizzare in PostgreSQL NB Indice: elemento creato alla nascita di una PK, per migliorare la prestazione nella ricerca di elementi nella tabella

- **PRIMARY KEY** - una constraint che definisce una colonna o un insieme di colonne che identificano univocamente ogni riga della tabella. Può esserci solo una PRIMARY KEY per ogni tabella. rischio duplicazione
- **FOREIGN KEY** - una constraint che definisce una relazione tra due tabelle. La FOREIGN KEY specifica che i valori in una colonna (o insieme di colonne) di una tabella devono corrispondere ai valori di una colonna (o insieme di colonne) in un'altra tabella.
- **UNIQUE** - una constraint che impedisce l'inserimento di valori duplicati in una colonna (o insieme di colonne) di una tabella.
- **CHECK** - una constraint che definisce una condizione che deve essere verificata per ogni riga inserita o aggiornata nella tabella. Ad esempio, è possibile utilizzare una constraint CHECK per assicurarsi che un valore numerico sia maggiore di un certo valore.
- **NOT NULL** - una constraint che impedisce l'inserimento di valori NULL in una colonna. vuoto

L'utilizzo di constraints è un'ottima pratica per garantire l'integrità dei dati in una tabella e prevenire errori o problemi di coerenza. Le constraints vengono specificate durante la creazione della tabella o mediante l'aggiunta successiva di una constraint a una tabella esistente utilizzando il comando ALTER TABLE

Constraints

Esempio di aggiunta di vincolo di primary_key su due colonne della tabella students:

ALTER TABLE students **ADD CONSTRAINT** pk_students **PRIMARY KEY** (codice_fiscale, cognome);

Esempio di aggiunta di vincolo di foreign_key su due colonne della tabella orders, che puntano alla primary_key della tabella customers:

ALTER TABLE orders **ADD CONSTRAINT** fk_orders **FOREIGN KEY** (cust_codice_fiscale, cust_cognome)
REFERENCES customers(codice_fiscale, cognome);

Esempio di aggiunta di vincolo di univocità su due colonne della tabella users:

ALTER TABLE users **ADD CONSTRAINT** uni_email_surname **UNIQUE** (email, surname);

Esempio di aggiunta di vincolo CHECK che impedisce alla colonna data_iscrizione di registrare valori al di fuori dell'anno 2023:

ALTER TABLE studente **ADD CONSTRAINT** check_stud_data_iscrizione **CHECK** (data_iscrizione >= '2023-01-01' AND data_iscrizione <= '2023-12-31')

NOTA: In PostgreSQL, ogni tabella può avere al massimo un **indice clustered**. L'indice clustered in PostgreSQL è creato automaticamente quando viene creata una tabella con una colonna definita come **PRIMARY KEY**

Default & NOT NULL

Esempio di aggiunta di vincoli di DEFAULT e vincoli di valorizzazione NOT NULL:

```
CREATE TABLE utente (  
    id serial PRIMARY KEY NOT NULL, --valorizzazione obbligatoria  
    nome varchar(100),  
    cognome varchar(100) NOT NULL, --valorizzazione obbligatoria  
    eta int, --valorizzazione facoltativa  
    data_inserimento date DEFAULT CURRENT_DATE, --setto automaticamente la data corrente  
    codice_fiscale char(16) NOT NULL DEFAULT '0000000000000000' UNIQUE --setto  
    automaticamente il codice fiscale con una serie di caratteri 16 volte 0, rendo la valorizzazione obbligatoria e  
    univoca  
);
```

Aggiungo il vincolo di univocità:

```
ALTER TABLE utente ADD CONSTRAINT uni_cognome_codfisc UNIQUE (cognome, codice_fiscale);
```

Rules

Le regole (**rules** in inglese) **sono uno strumento potente per definire comportamenti personalizzati che si verificano quando vengono eseguite determinate operazioni sul database**, come ad esempio l'inserimento, l'aggiornamento o l'eliminazione di dati.

Supponiamo di avere due tabelle: "clienti" e "ordini". La tabella "ordini" ha una colonna «codice_cliente» che fa riferimento alla chiave primaria «codice» della tabella "clienti". Vogliamo creare una regola che, quando un cliente viene eliminato dalla tabella "clienti", elimini anche tutti gli ordini associati a quel cliente dalla tabella "ordini".

Ecco come creare tale regola:

CREATE OR REPLACE RULE elimina_ordini_cliente

AS ON DELETE TO clienti

DO ALSO

DELETE FROM ordini **WHERE** codice_cliente = **OLD**.codice;

Se invece vogliamo cancellare la regola:

DROP RULE elimina_ordini_cliente **ON** clienti;

Fonti:

- Documentazione ufficiale di Postgresql
<https://www.postgresql.org/docs/>
- SQL online tutorial.org
<https://www.sqltutorial.org/>
- SQL online documentazione w3schools
<https://www.w3schools.com/>
- Autore: Serena Sensini (2021)
Titolo: Dasi di Dati - Tecnologie, architetture e linguaggi per database
Editore: Apogeo
ISBN: 9788850335534

Fine della Presentazione