

## POSTGRESQL:

Segue il Modello Client-Server dove:

- SERVER  $\Rightarrow$  Daemon Supervisa Tutti i processi
- CLIENT  $\Rightarrow$  Qualsiasi Programma in grado di Gestire l'Interazione con PostMaster

SQL : (Structure Language Query)

Ha varie funzionalità:

- DDL  $\Rightarrow$  Definizione Strutture e Vincoli d'Integrità
- DML  $\Rightarrow$  Manipolare Dati
- DQL/DQ  $\Rightarrow$  Per Interrogare

## COMANDI BASE:

CREATE TABLE [Nome] (attributo dominio ValoreDefault Vincolo)

CREATE TABLE Impiegato (  
Matricola CHARACTER(6) PRIMARY KEY)

### DOMINI ELEMENTARI:

**Tipo Carattere:** Carattere / Stringa con Lunghezza fissa / Variable

CHARACTER(20)

CHARACTER VARYING(20)

Max. 20 char

Se Metto TEXT Allora NON ha Vincoli max. Sulla Lunghezza

**Tipo Booleano:** Valore Singolo di Verità  $\Rightarrow$  T/F + NULL

### Tipo Numerici Esatti:

**Valori Interi  $\Rightarrow$**  SMALLINT: 2 Bytes  
INTEGER: 4 Bytes

**Valori Decimali  $\Rightarrow$**  NUMERIC (#Tot di Cifre, #Cifre dopo la Virgola)  
DECIMAL  
↓ equivalenti

### Tipo Numerici Approssimati:

REAL  $\Rightarrow$  6 cifre decimali

DOUBLE PRECISION  $\Rightarrow$  15 cifre decimali

### INSTANTI TEMPORALI:

DATE  $\Rightarrow$  del Tipo (anno-mese-giorno)

TIME (Precisione)  $\Rightarrow$  del Tipo (ora-min-sec)

↳ #Cifre per la frazione di Secondo

TIME STAMP  $\Rightarrow$  Date + Time

### INTERVALLI TEMPORALI:

CREATI DALL'UTENTE: Tramite 'CREATE DOMAIN' nome AS TipoBase...

CREATE DOMAIN Voto	← Nome dominio
AS SMALLINT	← Tipo di dato (dominio base)
DEFAULT NULL	← Valore di default
CHECK ( value >= 18 AND value <= 30 )	← Vincolo

## VINCOLI INTRA-RELAZIONALI:

**NOT NULL:** Deve Sempre Essere Specificato un Valore, o si Assegna un Valore di DEFAULT.

nome VARCHAR(20) NOT NULL DEFAULT 'VUOTO'

**UNIQUE:** Per (SUPER)CHIAVI

nome VARCHAR(20) UNIQUE oppure UNIQUE (nome, cognome)

**PRIMARY KEY:** Chiave Primaria (implica NOT NULL), una Volta per Tabella

Matricola CHAR(6) PRIMARY KEY o PRIMARY KEY (nome, cognome)

**CHECK:** Vincolo Generico che devono SODDISFARE le TUPLE.

Stipendio NUMERIC(8,2) CHECK (Stipendio >= 0.0)

## VINCOLI INTER-RELAZIONALI:

Integrità Referenziale Tramite REFERENCES e FOREING KEY

Si usa per il Singolo ATTRIBUTO

in fase di DEFINIZIONE



Dipart CHAR(2) REFERENCES Dip(NomeDip)

Si usa dopo le def. degli

ATTRIBUTI elencandoli



FOREING KEY (Nome, Cognome)

REFERENCES Anagr. (Nome, Cognome)

## MODIFICA DEGLI SCHEMI :

■ ALTER  $\Rightarrow$  fa Modifiche

ALTER TABLE tabella ADD Colonna attributo TIPO  
DROP Colonna attributo

■ DROP (Domain, Table)  $\Rightarrow$  Rimuove

DROP TABLE tabella

Tuple Nella Tabella Vengono Eliminate Come Segue:

DELETE From tabella WHERE (condizione)

↳ Se non presente Vengono  
Rimosse tutte

## COMANDO SELECT :

Produce una RELAZIONE RISULTANTE che ha:

■ Tutti Gli Attributi elencati Nella Select

DISTINCT Per rimuovere i DUPLICATI

Per CONCATENARE ATTRIBUTI  $\Rightarrow$  cognome || ' ' || nome AS  
cogn\\_Name

SELECT attributo AS "NomeCheVoglioNelRisultato"

## CLAUSOLA FROM:

Può Riferisci a:

Un Risultato DI UNA QUERY INNESTATA

Una Tabella

Alias  $\Rightarrow$  From tabella AS T1

T1

D'ora in poi posso riferirmi a Tabella  
tramite T1

## CLAUSOLA WHERE:

Si Basa Sul Valore di una CONDITION, Nel CASO IN CUI per  
una Riga CONDITION fosse TRUE allora partecipa al RISULTATO

## OPERATORE LIKE:

Per il Confronto Tra Stringhe (PATTERN MATCHING) con Caratteri  
Speciali:

$'_'$   $\Rightarrow$  Un Carattere Qualsiasi

$'%'$   $\Rightarrow$  Ø o più Caratteri Qualsiasi

WHERE città LIKE 'V.%a' (ILIKE Per CASE-SENSITIVE)

## OPERATORE SIMILAR TO:

è un like più Espressivo che Accetta caratteri del like ed in più:

$'^*'$   $\Rightarrow$  Ripetizioni del Precedente Match Ø o più Volte

**'+'**  $\Rightarrow$  Ripetizioni del Precedente Match 1 o più Volte

**'[...]'**  $\Rightarrow$  Elenco Caratteri Ammissibili

**'[n,m]'**  $\Rightarrow$  Ripetizione del precedente Match almeno n Volte ma non più di m.

### OPERATORE BETWEEN:

Appartenenza ad un INTERVALLO  $\Rightarrow$  BETWEEN a AND b

### OPERATORE IN:

Appartenenza di un Valore ad un Insieme  $\Rightarrow$  attr. IN(a,b,c,...)

### OPERATORE IS NULL:

Per TESTARE Se un Valore è NULL o Meno.

IS NULL  $\hookrightarrow$  IS NOT NULL

### OPERATORE ORDER BY:

Ordina le TUPLE del RISULTATO in Base Agli Attributi Specificati,

Penso Specificare ASC / DESC

$\hookrightarrow$  di DEFAULT

## OPERATORI DI AGGREGAZIONE:

Permettono di Considerare Valori OTTENUTI da una SELECT.

### COUNT:

- $*$   $\Rightarrow$  tutto
- DISTINCT  $<\text{expr}>$   $\Rightarrow$  No Ripetizioni di  $<\text{expr}>$
- $\text{expr} \Rightarrow$  dove la Condizione Nelle Tuple è NOT NULL

### Sum/AVERAGE/min/max

Devo Specificare su che ATTRIBUTI faccio le OPERAZIONI.

expr:: NewType Converte il Valore di EXPR. a un Valore del dominio di NewType  $\Rightarrow$  CASTING



CAST(expr AS NewType)

### GROUP BY:

Il Raggiungimento è un Insieme di TUPLE con Valore Uguale su uno o più ATTRIBUTI

NB!

Quando si USA le GROUP BY Nella SELECT posso Inserire Solo gli Attributi Usati Per IL Raggiungimento ed Eventuali funzioni di Aggregazione Sugli Altri Attributi  $\Rightarrow$  ~~SELECT cognome, nome FROM Persone GROUP BY nome~~

Nel GROUP BY non passo Inserire espressioni con Operatori di Aggregazione

Clausole e Significato:

■ WHERE  $\Rightarrow$  Selezione Righe che devono partecipare al Risultato

■ HAVING  $\Rightarrow$  Selezione i Ragggruppamenti (dopo GROUP BY) per il RISULTATO

## IL JOIN:

■ [INNER] JOIN rappresenta il  $\Theta$ -Join dell'Algebra Relazionale,  
Quindi Solo le TUPLE Che Soddisfano La condizione Partecipano  
Al Risultato.  $\Rightarrow$  È Simmetrico

■ LEFT [OUTER] JOIN  $\Rightarrow$  Si Esegue il Classico JOIN e Si  
Aggiungono Anche Tutte le RIGHE di t1 assegnando Agli Attributi  
senza Valore NULL. (tengo tutti i valori di T1 + faccio JOIN), da  
ricordare che non è Simmetrica

■ RIGHT [OUTER] JOIN  $\Rightarrow$  Si Esegue il Classico JOIN e Si  
Aggiungono Anche Tutte le RIGHE di t2 assegnando Agli Attributi  
senza Valore NULL. (tengo tutti i valori di T2 + faccio JOIN), da  
ricordare che non è Simmetrica

■ FULL [outer] JOIN  $\Rightarrow$  Non Equivalente al CROSS JOIN ma a  
INNER JOIN + LEFT JOIN + RIGHT JOIN

## INTERROGAZIONI NIDIFICATE:

Se l'Interrogazione è presente in un'Altra, può essere nel FROM o nel WHERE

Non posso più usare gli OPERATORI DI CONFRONTO ( $>$ ,  $<$ ,  $=$ , ...) ma devo usare:

- [NOT] EXISTS
- [NOT] IN
- [ALL]
- ANY/SOME

Perché non confronto più con un Valore Costante ma con un Insieme di Possibili Valori

**EXISTS:** falso se non ci sono righe. C'è DATA-BINDING se ho attributi uguali nella SELECT e nella SUB-QUERY.

In presenza di DATA-BINDING viene valutata per ogni RIGA della SELECT Principale

**IN:** expr IN (SUBQUERY)

La SUBQUERY deve restituire #Colonne = #Colonne di Expr ed è TRUE se c'è almeno un MATCH

**ANY/SOME:** expr Operator ANY/SOME SubQuery

Coinvolge Attributi della Select Principale

di Confronto, come  $>$ ,  $<$ ,  $=$ ,  $\neq$ , ...

SELECT che deve restituire una sola colonna

Ritorna Vero se expr. è operatore Rispetto al Valore di una Qualsiasi Riga del Risultato di subquery

**ALL:** Ritorna Vero se c'è match su TUTTE le Tuple della subquery

## INTERROGAZIONI INSIEMISTICHE:

**UNION**

**INTERSECT**

**EXCEPT**

} Applicabili SSE Query1 e Query2 hanno lo stesso Numero di Colonne e Tipo Compatibile fra di loro

Tutti Gli Operatori Eliminano i DUPLICATI a Meno che non Specifichi ALL

**UNION**  $\Rightarrow$  fa il MERGE dei 2 Risultati

**INTERSECT**  $\Rightarrow$  Tuple che Stanno in Entrambi i RISULTATI

**EXCEPT**  $\Rightarrow$  Differenza, Risultato di Query1 - Query2

## LE VISTE:

Tabelle "VIRTUALI" il cui Contenuto dipende dal contenuto di Altre Tabelle, ovvero sono l'ESECUZIONE di un INTERROGAZIONE

SQL NON Ammette:

- Definizioni in Termini di Se Stessa o Ricorsive
- dipendenze Circolari

CREATE [TEMP] VIEW nome AS query

↓  
Temporanea, alla  
disconnessione Si  
Perde il tutto

## INDICI:

Strutture Ausiliarie, con Costo di Aggiornamento Per Aumentare le prestazioni d'Accesso

Vogliamo Evitare la Scansione Seg. della Tabella

Con il Comando \Timing Possa Vedere le Statistiche della Query

### COMANDO:

CREATE INDEX [nome] ON NomeTab [USING method]

[{ nomeAttr / expr }] [ASC / DESC]

↳ tipo di ORDINE

↓  
tipo di  
INDICE

DBMS Li Crea in Automatico Per Gli Attributi: PRIMARY KEY

Con ANALYZE tabella Forza DBMS ad Aggiornare Le Statistiche dopo la CREAZIONE degli Indici

## TIPO B-TREE:

È di DEFAULT e Viene Considerato per TUTTI gli Operatori di CONFRONTO ma Anche per BETWEEN - IN - NULL - LIKE

## TIPO HASH:

È limitato ai CONFRONTI di Uguaglianza

## INDICI MULTIATTRIBUTO:

Se si hanno QUERY con Condizioni Su COPPIE/TERNE/... allora può essere più utile CREARNE uno unico Multiattributo Rispetto a Molti MONO-ATTRIBUTO

### COSTO e REGOLA PRATICA:

mantenerli Aggiornati

Si COSTRUISCONO in Base alle Query Più frequenti

### EXPLAIN:

Potrò Vedere il PIANO D'ESECUZIONE, per ogni Riga il TIPO DI OPERAZIONE + Stima del Costo

## CONTROLLO CONCORRENZA:

Transazione è una Seq. di Istruzioni che può Essere Eseguita in Concorrenza Con Altre Transazioni in Modo Isolato

### Possibili Anomalie:

- Perdita di Aggiornamento
- Lettura Sporca
- lettura Inconsistente
- Aggiornamento fantasma
- Inserimento fantasma

PostgreSQL Mantiene la CONSISTENZA dei DATI usando MVCC  
Piu Versatile del 2PL Stricto ↩

In Generale Ogni Transazione:

- Vede un Istantanea della Base di DATI
- Letture Sempre Possibili
- Scritture possono Essere Bloccate Se in PARALLELO un'altra Ti ha Modificato la Sorgente dei DATI che Si Vuole Aggiornare

## LIVELLI DI ISOLAMENTO:

- ① SERIALIZZABILE  $\Rightarrow$  Nessun Anomalia Possibile, Lvl Max. di isolamento
- ② REPEATABLE READ  $\Rightarrow$  I dati letti non Cambieranno a Causa di Altre Transazioni.  
Possibile Anomalia è la mancata serializzazione
- ③ READ COMMITTED  $\Rightarrow$  Ogni SELECT Agisce solo sui DATI Confermati, prima che Inizi  
Lvl di ISOLAMENTO di DEFAULT e Posso Avere:  
 $\text{m}$  lettura Inconsistente  
 $\text{m}$  Aggiornamento/Inserimento fantasma  
 $\text{m}$  Mancata Serializzazione
- ④ READ UNCOMMITTED  $\Rightarrow$  In PostgreSQL non Esiste.

Comando: `SET TRANSACTION ISOLATION LEVEL mode`

- $\text{m}$  SERIALIZZABILE
- $\text{m}$  READ COMMITTED
- $\text{m}$  READ UNCOMMITTED

Deve Esserci più di una Istruzione, Altrimenti non ha senso e non Necessita Nemmeno una Transazione

## READ COMMITTED: (rivaluta/Blocca a Lvl di Riga)

È il lvl **di DEFAULT**, le SELECT vede solo i dati committed.

Anche UPDATE/DELETE vedono gli stessi dati della SELECT e se i dati che devono essere aggiornati sono stati modificati da Ti che non hanno ancora eseguito COMMIT/ROLLBACK allora il comando deve attendere e poi rielaborare le righe per vedere se soddisfano ancora i criteri.

Potrei avere:

- Lettura Inconsistente
- Aggiornamento/Inserimento fantasma
- Mancata Serializzazione

PRE-CONDIZIONE:

ci devono essere operatori aggregati

## REPEATABLE READ:

PostgreSQL associa alla transazione un istantanea della base di dati e rimarrà sempre quella durante tutta l'EXEC della transazione

2 Select consecutive vedranno SEMPRE gli stessi dati.

Anche UPDATE/DELETE vedono gli stessi dati della SELECT e se i dati che devono essere aggiornati sono stati modificati da Ti che non hanno ancora eseguito COMMIT/ROLLBACK allora il comando deve attendere e in caso di:

■ ROLLBACK  $\Rightarrow$  UPDATE/DELETE possono procedere

Commit  $\Rightarrow$  Dati Sono Cambiati Quindi UPDATE/DELETE Vengono Bloccati con ERRORE

Quindi REPEATABLE READ è più Restrittivo di READ COMMITTED

Questo Lvl Cattura TUTTE le Anomalie Tcenne la Mancata Serializzazione Però Bisogna Prevedere i Possibili ABORT per Aggiornamenti Concorrenti

SERIALIZABLE: (rivaluta/Blocca a Lvl di Tabella)

È il Più Restrittivo, Nessuna Anomalia Permetta ed Anche Qui Bisogna Prevedere i Possibili ABORT per Aggiornamenti Concorrenti

Limita il THROUGHTPUT del DBMS.

Se una Ti viene Bloccata con Errore allora si Prova a rieseguirla

Specificare Sempre il LVL di PostgreSQL oppure Rimanere Generici a Lvl Teorico MA VA SPECIFICATO

# POSTRESQL CON PYTHON: (DB-API, psycopg2)

Oggetto Connection: Tramite Metodo .connect() così:

Connector = psycopg2.connect(host, database, user, password)

## METODI PRINCIPALI:

- **CURSOR** ⇒ Cursore della BASE DI DATI per Invia Comandi e Scorre il Risultato
- **Commit** ⇒ Registra la Ti Corrente, se NON Viene fatto si Perdono tutte le Modifiche fatte
- **Rollback** ⇒ Abort della Ti
- **CLOSE** ⇒ Chiude Connessione ed Implica ROLLBACK delle op. Non Registrate (SENZA COMMIT)
- **Auto Commit** ⇒ Proprietà R/W, se TRUE ogni Comando è una Ti Isolata altrimenti serve commit/Rollback
- **READ ONLY** ⇒ Proprietà R/W, di DEFAULT è FALSE
- **ISOLATION-LEVEL** ⇒ Proprietà R/W, Meglio Assegnarla Subito dopo la Creazione della CONNESSIONE

## CURSORI: Gestire Interazione Con le BASE DI DATI

EXECUTE (comando, parametri)  $\Rightarrow$  esegue il Comando SQL, Parametri Passati come DICT o TUPLA

EXECUTEMANY (comando, Parametri)  $\Rightarrow$  esegue un 'comando' per ogni Veloce Nella Lista dei Parametri.

È meno Efficiente di un CICLO FOR di execute

Casting da PYTHON a SQL Garantito per i fondamentali.

Usare Sempre %s invece che la Concatenazione di Stringhe per Evitare l'SQL INJECTION

FETCHONE ()  $\Rightarrow$  Ritorna una TUPLA del Risultato e si usa dopo un EXECUTE

FETCHMANY (<numero>)  $\Rightarrow$  Ritorna al massimo <numero> Tuple

Accesso Tramite CICLO :

```
cur.execute (...)  
for RECORD in cur:  
    print(record, end = ",")
```

RowCount  $\Rightarrow$  #Righe ultimo Comando

StatusMessage  $\Rightarrow$  Stato Ultimo Comando

## SCHEMA DI UTILIZZO:

- ② Aprire Connessione  $\Rightarrow$  conn = psycopg2.connect(...)
- ③ Modifiche sul Isolamento, Auto-commit, ReadOnly
- ③ Creo Cursore  $\Rightarrow$  curr = conn.cursor()
- ④ Esegue Operazioni  $\Rightarrow$  cursor.execute (...)
- ⑤ Se NON è in AUTOCOMMIT allora eseguire conn.commit()
- ⑥ Chiudere Cursore e Connessione  $\Rightarrow$  cur.close() e conn.close()

UTILIZZO DEL WITH: All' Uscita del Blocco viene fatto un commit in Automatico e la Connessione NON Viene Chiusa ma se si usa un CURSORE invece lui Viene Chiuso

connessione = psycopg2.connect(...)

with connessione:

With connessione.cursor() as cursore:

cursore.execute (...)

---

Si usano più CURSORI Sulla Medesima Connessione Quando si fanno Transazioni in auto-Commit o Transazioni In Sole Lettura

# MONGODB:

No-SQL ma DOCUMENT BASED.

Base di Dati formata da **COLLEZIONI DI DOCUMENTI** dove ogni **Collezione** è il **Contenuto di Tabella** mette i **DOCUMENTI**. Sono le **TUPLE**.

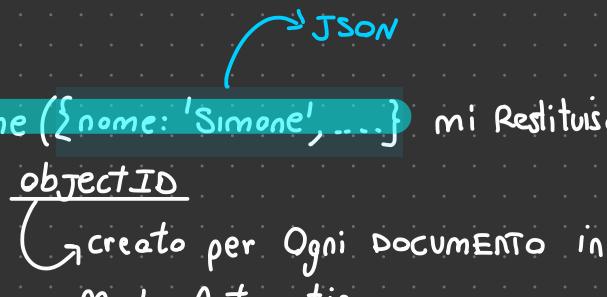
È Schema-Less:

- Struttura dei DATI si Evolve Nel Tempo
- Dati (non o semi) Strutturati

Tool:

- GUI ⇒ MongoDB Compass
- CLI ⇒ MongoSH
- mongoStat ⇒ Statistiche Sulle Prestazioni;
- MongoDB Atlas per CLUSTER Gratuito

## OPERAZIONI CRUD:

**CREATE:** `db.nome.InsertOne({nome: 'Simone', ...})` mi Restituisce un Oggetto JSON con Stato e objectId  creato per Ogni DOCUMENTO in Modo Automatico

`db.nome.InsertMany([{...}, {...}, ...])`

**READ:** `db.nome.find({nome: 'Simone'})` restituisce oggetti JSON che soddisfano il filtro.

Se non inserisco un filtro Allora Recupero tutti i DOCUMENTI della COLLEZIONE

Se inserisco più Condizioni Saranno Verificate in AND altrimenti per l'OR devo:

`db.nome.find({$or: [{name: 'Simone'}, {cognome: 'Bottesi'}]})`

Per Valori Nulli:

- `{name: null}`
- `{name: {$ne: null}}`

**AGGIORNAMENTO:** `db.nome.UpdateOne(filtro, campi da Aggiornare)`

ad Esempio:

`db.Museo.UpdateOne({città: 'Verona'}, {$set: {price: 30}});`

Ritorna il JSON con:

- MatchedCount  $\Rightarrow$  #documenti che soddisfano il filtro
- ModifiedCount  $\Rightarrow$  #documenti che sono stati modificati

Con UPDATE ONE viene Modificato Sempre un Solo documento, il primo che soddisfa il criterio.

UpdateMany li Aggiorna TUTTI

`replaceOne(filtro, replacement, options)` Sostituisce il Singolo documento mantenendo lo stesso Object ID

**DELETE:** `deleteOne({nome: 'Simone'})` ritorna il JSON con `deletedCount`  
e continua Sempre il Primo...altrimenti c'è `deleteMany()`

Documenti in MONGODB Supportano l'Incapsulamento che Riduce la Necessità di JOIN e facilità il RECUPERO dell' Informazione

Sfrutto i RIFERIMENTI Tramite Object id e le Applicazioni andranno a Risolvere Questi LINK/PUNTATORI per Arrivare ai DATI relativi;

Posso denormalizzare (a metà tra INCAPSULAMENTO e RIFERIMENTO) per Velocizzare l'Accesso a certe Informazioni

MongoDB memorizza l'Informazioni in BSON (Binary Json)

## JOIN:

**■ \$lookup:** {

**FROM:** collezione

**LOCALFIELD:** doc. Corrente

**FOREIGNFIELD:** doc. Esterno

**LET:**

**Pipeline:**

**AS:** Nome Attr. che conterrà l'Elenco dei documenti Collegati