POSTGRESQL RIFERIMENTO RAPIDO

Tabelle, query, join, indici, JSON, ruoli

Connessione

Riga di Comando

```
psql -U postgres
psql -h localhost -p 5432 -U user -d mydb
psql "postgresql://user:pass@host:5432/mydb"
```

Meta-Comandi psql

`\l`	Elenca i database
`\c dbname`	Connetti al database
`\dt`	Elenca le tabelle
`\d tablename`	Descrive la struttura della tabella
`\dn`	Elenca gli schema
`\du`	Elenca i ruoli
`\q`	Esci da psql
`\i file.sql`	Esegue un file SQL

Tabelle e Schema

Crea Tabella

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  email TEXT UNIQUE,
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Operazioni sullo Schema

```
CREATE SCHEMA app;
CREATE TABLE app.users (id SERIAL PRIMARY KEY);
SET search_path TO app, public;
DROP SCHEMA app CASCADE;
```

Modifica Tabella

```
ALTER TABLE users ADD COLUMN age INT;
ALTER TABLE users ALTER COLUMN name TYPE TEXT;
ALTER TABLE users DROP COLUMN age;
ALTER TABLE users RENAME TO customers;
```

Tipi di Dato

Numerici

`INTEGER / INT`	Intero a 4 byte
`BIGINT`	Intero a 8 byte
`SERIAL`	Intero auto-incrementante
`NUMERIC(p,s)`	Numerico esatto (es. NUMERIC(10,2))
`REAL / DOUBLE PRECISION`	Virgola mobile (4 / 8 byte)
`BOOLEAN`	true / false / null

Stringhe e Binari

`TEXT`	Testo variabile senza limite
`VARCHAR(n)`	Testo variabile fino a n caratteri
`CHAR(n)`	Testo a lunghezza fissa
`BYTEA`	Dati binari
`UUID`	Identificatore unico universale a 128 bit

Date, JSON e Array

`DATE`	Data del calendario
`TIMESTAMPTZ`	Timestamp con fuso orario
`INTERVAL`	Intervallo di tempo (es. '2 days')
`JSONB`	JSON binario (indicizzabile)
`INT[] / TEXT[]`	Tipi array

Query

Insert

```
INSERT INTO users (name, email)
VALUES ('Alice', 'alice@example.com')
RETURNING id;

INSERT INTO users (name, email) VALUES
('Bob', 'bob@ex.com'),
('Carol', 'carol@ex.com');
```

Select

```
SELECT * FROM users WHERE id = 1;
SELECT name, email FROM users
ORDER BY name LIMIT 10 OFFSET 20;
```

Update

```
UPDATE users SET email = 'new@ex.com'
WHERE id = 1 RETURNING *;
```

Upsert

```
INSERT INTO users (email, name)
VALUES ('a@ex.com', 'Alice')
ON CONFLICT (email) DO UPDATE
SET name = EXCLUDED.name;
```

Delete

```
DELETE FROM users WHERE id = 1 RETURNING *;
TRUNCATE TABLE users RESTART IDENTITY;
```

Join e Sottoquery

Tipi di Join

`INNER JOIN`	Righe corrispondenti in entrambe le tabelle
`LEFT JOIN`	Tutte le righe sinistra + corrispondenti destra
`RIGHT JOIN`	Tutte le righe destra + corrispondenti sinistra
`FULL OUTER JOIN`	Tutte le righe da entrambe le tabelle
`CROSS JOIN`	Prodotto cartesiano
`LATERAL JOIN`	Sottoquery che fa riferimento alla riga esterna

CTE (Common Table Expression)

```
WITH active AS (
  SELECT * FROM users WHERE active = true
)
SELECT a.name, o.total
FROM active a
JOIN orders o ON a.id = o.user_id;
```

Sottoquery

```
SELECT name FROM users
WHERE id IN (
  SELECT user_id FROM orders
  WHERE total > 100
);
```

Indici

Crea e Rimuovi

```
CREATE INDEX idx_name ON users(name);
CREATE UNIQUE INDEX idx_email ON users(email);
CREATE INDEX idx_gin ON posts USING GIN(tags);
DROP INDEX idx_name;
```

Tipi di Indice

`B-tree`	Default, ottimo per =, <, >, BETWEEN
`Hash`	Solo confronti di uguaglianza
`GIN`	Invertito generalizzato — array, JSONB, full-text
`GiST`	Ricerca generalizzata — geometrica, range
`BRIN`	Intervallo blocco — tabelle ordinate grandi

Analisi delle Query

```
EXPLAIN ANALYZE
SELECT * FROM users WHERE name = 'Alice';
```

Funzioni e Procedure

Funzione SQL

```
CREATE FUNCTION active_count()
RETURNS INTEGER AS $$
  SELECT COUNT(*)::INT FROM users
  WHERE active = true;
$$ LANGUAGE sql;
SELECT active_count();
```

Funzione PL/pgSQL

```
CREATE FUNCTION greet(name TEXT)
RETURNS TEXT AS $$
BEGIN
  RETURN 'Hello, ' || name;
END;
$$ LANGUAGE plpgsql;
```

Funzioni Integrate Utili

`NOW() / CURRENT_TIMESTAMP`	Timestamp corrente con TZ
`AGE(ts1, ts2)`	Intervallo tra timestamp
`COALESCE(a, b)`	Primo valore non-null
`NULLIF(a, b)`	NULL se a = b
`GENERATE_SERIES(1,10)`	Genera righe di valori sequenziali
`STRING_AGG(col, ',')`	Concatena valori con separatore

Ruoli e Permessi

Gestione dei Ruoli

```
CREATE ROLE app LOGIN PASSWORD 'secret';
ALTER ROLE app SET search_path TO myapp;
DROP ROLE app;
```

Grants

```
GRANT ALL ON DATABASE mydb TO app;
GRANT SELECT, INSERT ON users TO reader;
GRANT USAGE ON SCHEMA public TO app;
REVOKE INSERT ON users FROM reader;
```

Row-Level Security

```
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
CREATE POLICY user_own ON users
FOR ALL USING (id = current_setting('app.uid')::INT);
```

Supporto JSON

Operatori JSONB

`->'key'`	Ottieni valore JSON per chiave (come JSON)
`->>'key'`	Ottieni valore JSON per chiave (come testo)
`#>'{a,b}'`	Ottieni valore annidato per percorso
`@>`	Contiene (sinistro include destro)
`?`	La chiave esiste
`||`	Concatena valori JSONB

Query JSONB

```
SELECT data->>'name' FROM profiles
WHERE data @> '{"active": true}';

SELECT * FROM profiles
WHERE data ? 'email';
```

Funzioni JSONB

```
SELECT jsonb_each(data) FROM profiles;
SELECT jsonb_array_elements('[1,2,3]');
SELECT jsonb_set(data, '{name}', '"Alice"')
FROM profiles WHERE id = 1;
```

Pattern Comuni

Transazioni

```
BEGIN;
UPDATE accounts SET balance = balance - 100
WHERE id = 1;
UPDATE accounts SET balance = balance + 100
WHERE id = 2;
COMMIT;  -- oppure ROLLBACK;
```

Window Function

```
SELECT name, salary,
  RANK() OVER (ORDER BY salary DESC),
  AVG(salary) OVER (PARTITION BY dept)
FROM employees;
```

Copia Dati

```
COPY users TO '/tmp/users.csv'
WITH (FORMAT csv, HEADER);
COPY users FROM '/tmp/users.csv'
WITH (FORMAT csv, HEADER);
```

Backup con pg_dump

```
pg_dump -U postgres mydb > backup.sql
pg_dump -Fc mydb > backup.dump
pg_restore -d mydb backup.dump
```