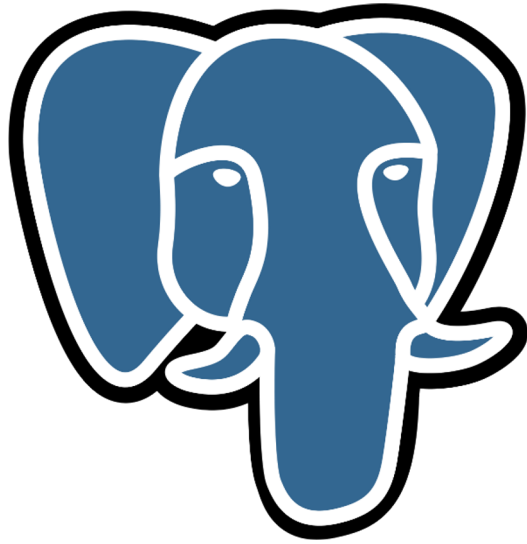


SQL Queries

Database Systems
DataLab, CS, NTHU
Spring, 2021



PostgreSQL

- [Download and install](#)
- For Mac users, try [PostgreSQL.app](#)

Using PostgreSQL

```
$ createdb <db>  
$ psql <db> [user]  
> \h or \?  
> SELECT now(); -- SQL commands
```

- Multiple lines until ‘;’
- ‘--’ for comments
- ***Case insensitive***
 - Use “” to distinguish lower and upper cases
 - E.g., `SELECT "authorId" FROM posts;`

Structured Query Language (SQL)

- Data Definition Language (DDL) on schema
 - CREATE TABLE ...
 - ALTER TABLE ...
 - DROP TABLE ...
- Data Manipulation Language (DML) on records
 - INSERT INTO ... VALUES ...
 - SELECT ... FROM ... WHERE ...
 - UPDATE ... SET ... WHERE ...
 - DELETE FROM ... WHERE ...

Schema

users

<u>id</u>	name	karma
729	Bob	35
730	John	0

friend

<u>uld1</u>	<u>uld2</u>	since
729	730	14928063
729	882	14827432

posts

<u>id</u>	text	authorId	ts
33981	'Hello DB!'	729	1493897351
33982	'Show me code'	729	1493854323

Creating Tables/Relations

```
CREATE TABLE users (  
    id          serial PRIMARY KEY NOT NULL,  
    name        varchar(50) NOT NULL,  
    Karma       integer NOT NULL  
);
```

- Column types:
 - Integer, bigint, real, double, etc.
 - varchar(10), text, etc.
- Non-null constraint

Creating Tables/Relations

```
CREATE TABLE users (  
    id          serial PRIMARY KEY NOT NULL,  
    name        varchar(50) NOT NULL,  
    Karma       integer NOT NULL  
);
```

- Primary key:
 - Unique (no duplicate values among rows)
 - Usually of type “serial” (auto-filled integer)
 - Index automatically created

Creating Tables/Relations

```
CREATE TABLE posts (  
  id          serial PRIMARY KEY NOT NULL,  
  text        text NOT NULL,  
  "authorId"  integer NOT NULL  
               REFERENCES users ON DELETE CASCADE,  
  ts          bigint NOT NULL  
               DEFAULT (extract(epoch from now()))  
);
```

- Foreign key: posts.authorId must be a valid users.id
- When deleting a user (row):
 - NO ACTION (default): user not deleted, error raised
 - CASCADE: user **and all referencing posts** deleted

Schema

users

<u>id</u>	name	karma
729	Bob	35
730	John	0

friend

<u>uld1</u>	<u>uld2</u>	since
729	730	14928063
729	882	14827432

posts

<u>id</u>	text	authorId	ts
33981	'Hello DB!'	729	1493897351
33982	'Show me code'	729	1493854323

Inserting Rows

```
INSERT INTO users (name, karma)
VALUES ('Bob', 35);
```

```
INSERT INTO posts (text, "authorId", ts) VALUES
('Today is a good day!', 1, 123456789);
```

- String values should be *single* quoted
- Inserting dummy rows:

```
INSERT INTO users (name, karma)
SELECT 'User ' || s, round(random() * 100)
FROM generate_series(1, 10) AS s;
```

Queries

```
SELECT * FROM users;
```

- **Aggregate function:**

```
SELECT COUNT(*) FROM users;
```

```
SELECT AVG(karma) FROM users;
```

```
SELECT MIN(karma) FROM users;
```

- **Often used with the GROUP BY**

Queries

```
SELECT *  
FROM users  
WHERE id<5 AND name ILIKE '%User%'  
ORDER BY id DESC  
LIMIT 2;
```

- To see how a query is processed:

```
EXPLAIN ANALYZE -- show plan tree  
SELECT *  
FROM users  
WHERE id<5 AND name ILIKE '%User%'  
ORDER BY id DESC  
LIMIT 2;
```

(Batch) Updating Rows

```
UPDATE users SET karma = karma + 10 WHERE name =  
'Bob';
```

- ***All*** rows satisfying the WHERE clause will be updated

Handling “Big” Data

```
INSERT INTO posts(text, "authorId")
SELECT
    'Dummy word ' || i || '.',
    round(random() * 10) + 1
FROM generate_series(1, 1000000) AS s(i);
```

- Some queries will be slow:

```
EXPLAIN ANALYZE
SELECT * FROM posts
WHERE id > 500000 AND id < 501000; -- 1ms
```

```
EXPLAIN ANALYZE
SELECT * FROM posts
WHERE ts > 14000000000 AND ts < 14036000000; -- 100ms
```

Using Index

```
CREATE INDEX posts_idx_ts  
ON posts  
USING btree(ts);
```

```
\di -- list indices
```

```
EXPLAIN ANALYZE  
SELECT * FROM posts  
WHERE ts > 1400000000  
AND ts < 1403600000; -- 2ms
```

posts_idx_ts

(ordered)

posts

id	text	ts
1	'Good day'	1493880220
...
33981	'Hello DB!'	1493897351
33982	'Show me code'	1493904323

Index for ILIKE?

```
CREATE INDEX posts_idx_text ON posts  
USING btree(text);
```

```
EXPLAIN ANALYZE SELECT * FROM posts  
WHERE text ILIKE '% word 500000%'; -- 300ms
```

- B-tree indices are **not** helpful for text searches
- Use GIN (generalized inverted index) instead:

```
CREATE EXTENSION pg_trgm;  
\dx -- list extensions
```

```
CREATE INDEX posts_idx_text_trgm ON posts  
USING gin(text gin_trgm_ops);
```

```
EXPLAIN ANALYZE SELECT * FROM posts  
WHERE text ILIKE '%word 500000%'; -- 50ms
```


Assigned Reading

- [SQL Tutorial](#)