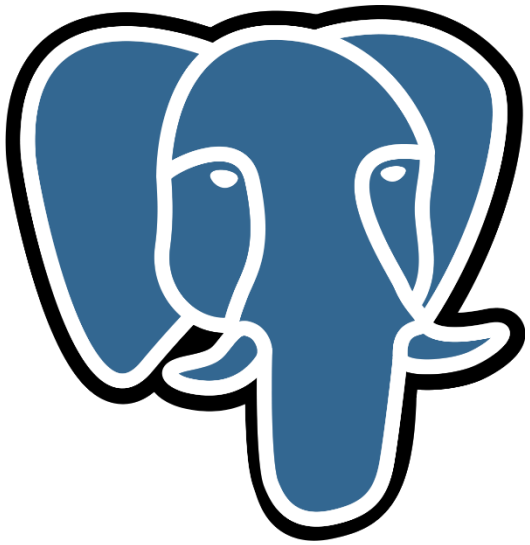
SQL Queries

Shan-Hung Wu & DataLab
CS, NTHU



PostgreSQL

- Download and install
- For Mac users, try PostgreSQL.app

Today's Example Code

- You can find the example we use today from here:
  - https://shwu10.cs.nthu.edu.tw/courses/databases/2020-spring/sql_query/blob/master/01-postgresql-example.md

Using PostgreSQL

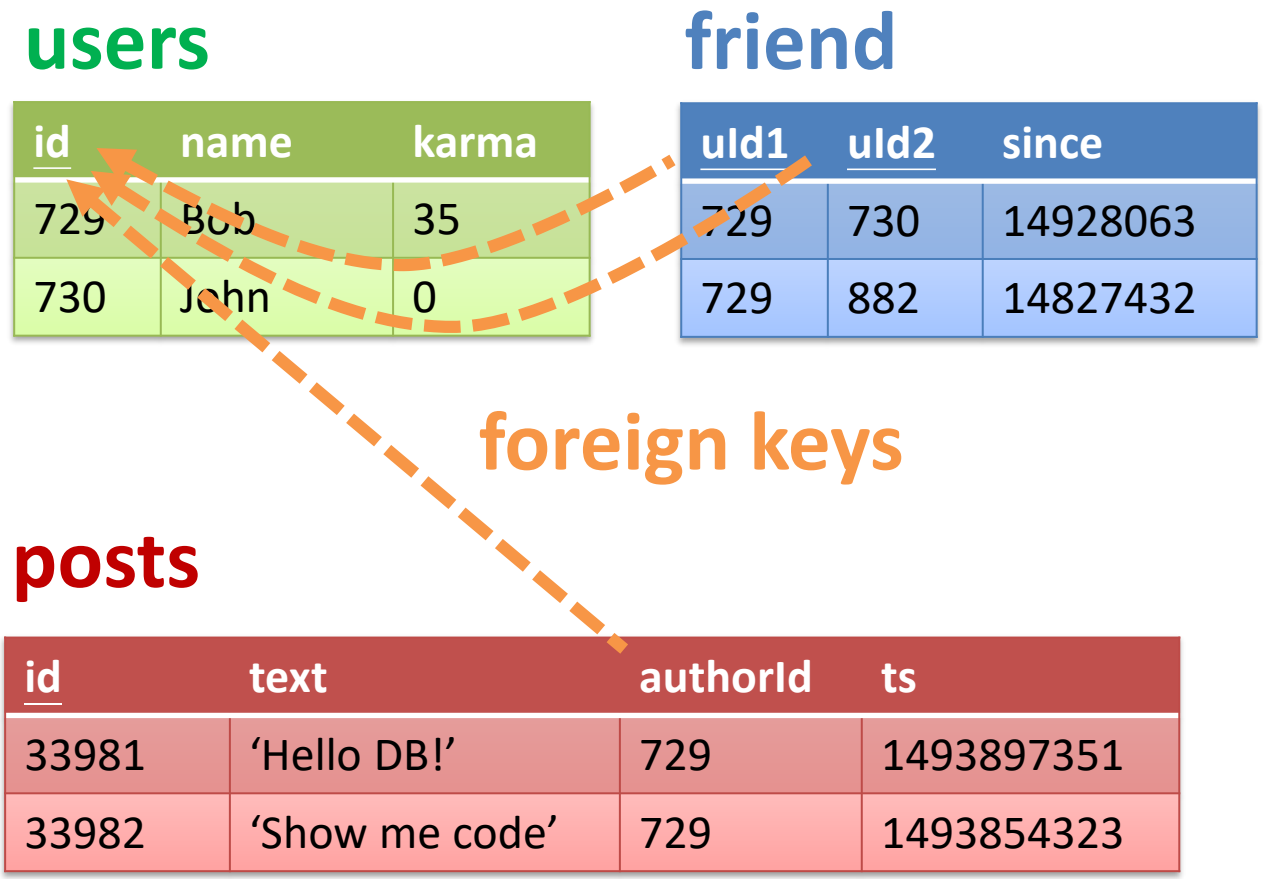
```
$ createdb <db>
$ psql <db> [user]
> \h or \?
> SELECT now(); -- SQL commands
```

- Default schema: `public`
  - `\dn` for listing all schemas
- 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
friend
id
name
karma
729
Bob
35
730
John
0
uId1
uId2
since
729
730
14928063
729
882
14827432
foreign keys
posts
id
text
authorId
ts
33981
'Hello DB!'
729
1493897351
33982
'Show me code'
729
1493854323

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())),
  ...
);
```

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

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())),
  ...
);
```

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

Inserting Rows

```
INSERT INTO posts(text, "authorId", ...)
VALUES ('Today is a good day!', 5, ...);
```

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

```
INSERT INTO posts(text, "authorId")
SELECT
  'Dummy word ' || i || '.',
  round(random() * 10) + 1
FROM generate_series(1, 20) AS s(i);
```

Queries

```
SELECT *
FROM posts
WHERE ts > 147988213 AND text ILIKE '%good%'
ORDER BY ts DESC, id ASC
LIMIT 2;
```

- To see how a query is processed:

```
EXPLAIN ANALYZE -- show plan tree
SELECT *
FROM posts
WHERE ts > 147988213 AND text ILIKE '%good%'
ORDER BY ts DESC, id ASC
LIMIT 2;
```

(Batch) Updating Rows

```
UPDATE posts SET ts = ts + 3600 WHERE "authorId" = 10;
```

- ***All*** rows satisfying the WHERE clause will be updated
- ts + 3600 is an ***expression***
  - Can be evaluated to a single value

Handling “Big” Data

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

- Some queries will be slow:

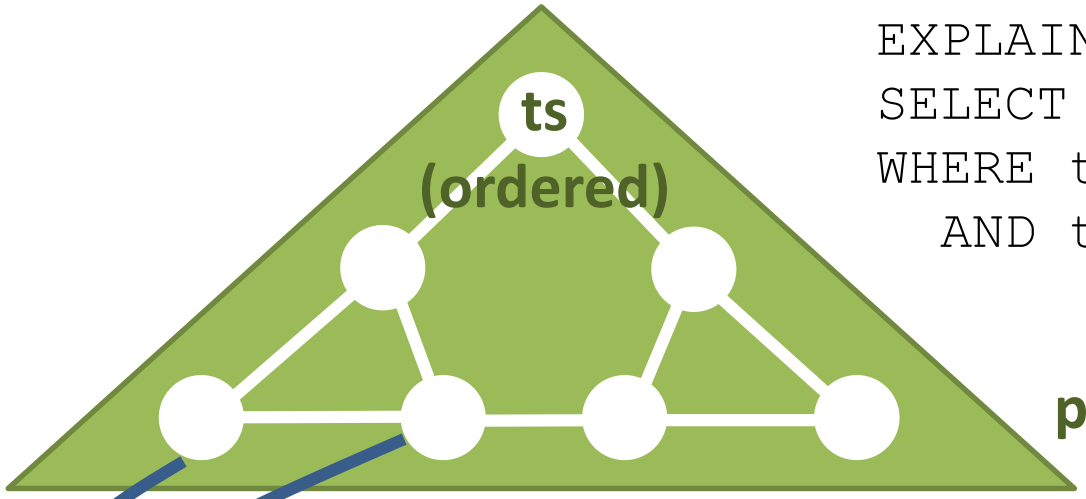
```
EXPLAIN ANALYZE SELECT * FROM posts
WHERE id > 500000 AND id < 501000; -- 1ms

EXPLAIN ANALYZE SELECT * FROM posts
WHERE ts > 1400000000 AND ts < 1403600000; -- 230ms
```

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

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%'; -- 1.5s
```

- 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
```