

Project: Data Modeling

Erickson Figueroa

Report

Executive Summary:

The project focused on managing unstructured data from a CSV file in PostgreSQL. Tasks included conceptual model construction, table creation, and development of a physical ERD. Configuring permissions to read the file and importing the data records for users, posts, and comments were integral to organizing and managing data effectively within the PostgreSQL environment.

Objective:

Create a logical data modelling starting from an unstructured CSV file, then transfer that logic to physical tables within the PostgreSQL database engine, create a physical modelling or ERD diagram, and import the necessary records from the CSV file.

Brief functionality of the process:

In the project, there was a CSV file containing unstructured data. After understanding the provided CSV file, the construction of a conceptual model diagram began. Subsequently, initiating the process involved creating tables in PostgreSQL and developing the physical diagram or ERD based on the logic of the conceptual diagram. Once the physical ERD diagram was ready, the configuration of necessary permissions for the PostgreSQL user to read the CSV file in the directory where it took place.

Following the process, a new table named `blog_data` inside PostgreSQL was created to import the data from the CSV file. When the imported data was ready in the `blog_data` table, the insertion of records for the users, posts, and comments table commenced.

The biggest challenge of the process:

The most challenging aspect involved inserting records into the comments table, requiring the creation of a complex SQL query to split, clean, and insert the data accurately. The task

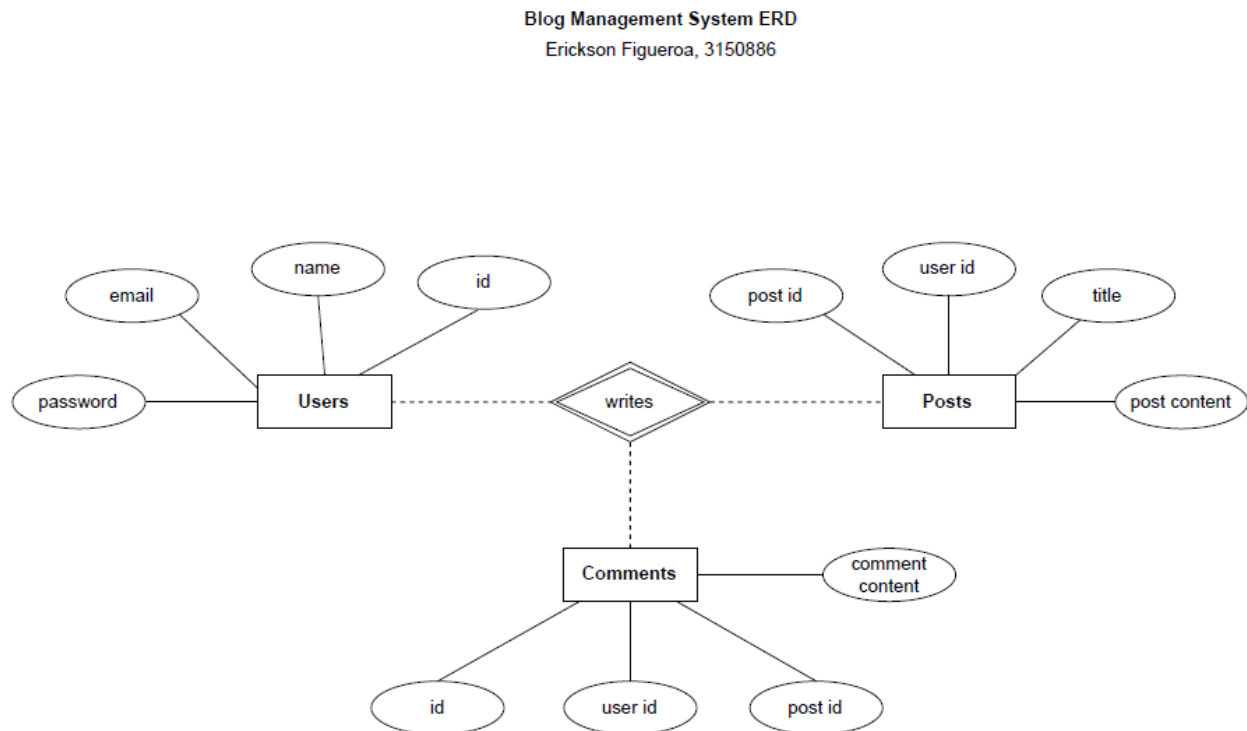
demanded meticulous attention to detail to ensure the correct separation and insertion of multiple comments within a single string for a user into the final comments table.

Additional tables:

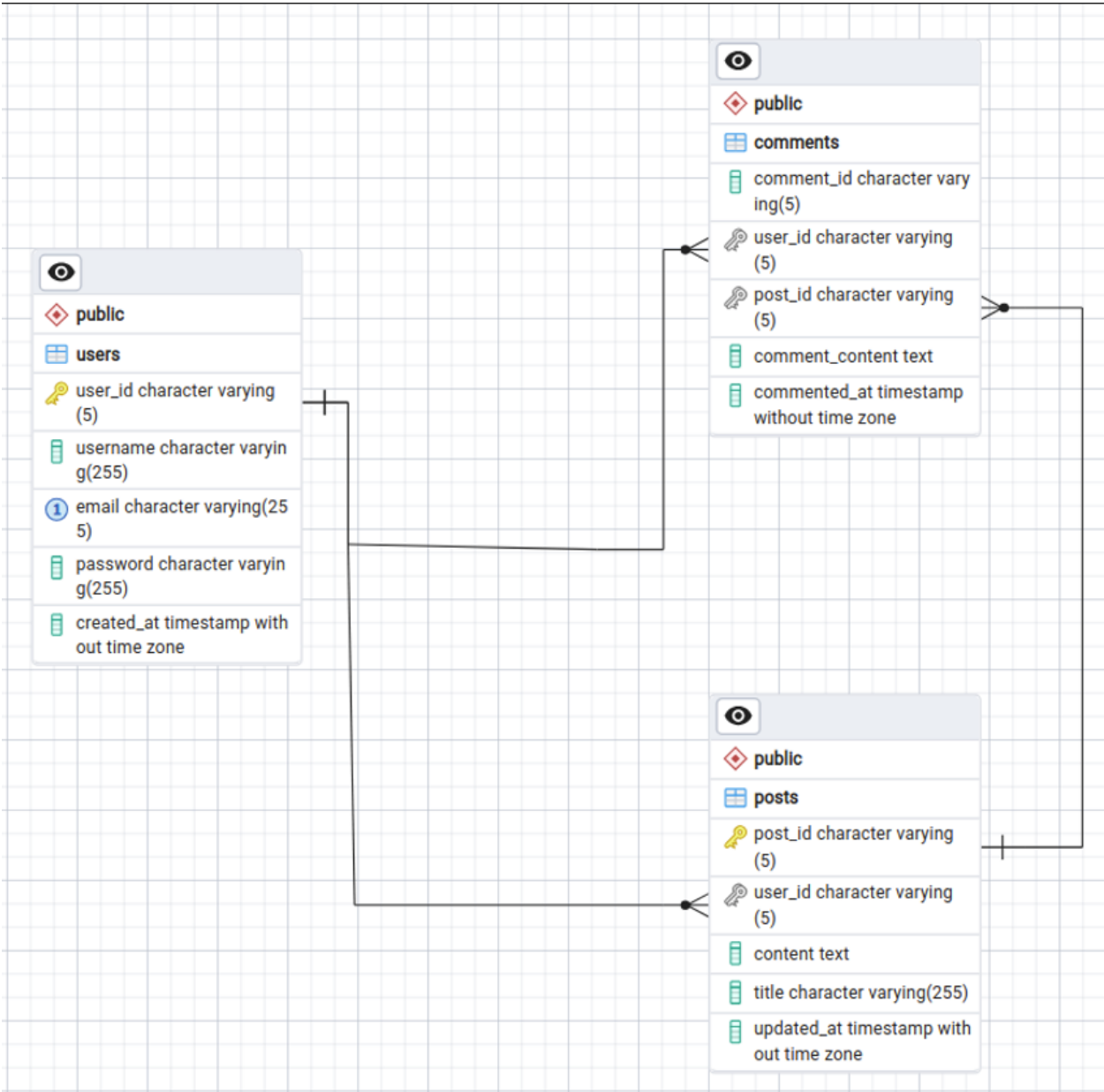
[blog_data](#): To import the records from the CSV file without any transformation.

[temp_comment_string](#): To split the comments for those users who made more than one comment on the same post.

Conceptual Model:



Physical Model:



Unstructured CSV file:

	A	B	C	D	E	F	G	H	I	J
	user_id	username	email	password	created_at	post_id	content	title	updated_at	comments
1	uid1	sarah_c	sarah@example.com	sarapass	2024-01-16 8:10	pid3	JavaScript is a popular ...	JavaScript Basics	2024-01-16 9:30	JavaScript is amazing! 2024-01-16 08:30 cid5
2	uid2	emily_j	emily@example.com	secure123	2024-01-16 17:30	pid5	Build web applications using ...	Python Web Development	2024-01-16 18:00	Thanks for the tutorial! 2024-01-16 18:00 cid7
3	uid3	jane_s	jane@example.com	passw0rd	2024-01-14 14:45	pid1	Python is a versatile and ...	Getting Started with Python	2024-01-15 10:20	I learned a lot. 2024-01-14 16:00 cid2 Thanks for the tutorial! 2024-01-14 15:30 cid1
4	uid4	michael_b	michael@example.com	p@ssw0rd	2024-01-16 12:20	pid4	Learn about various data ...	Data Structures in C++	2024-01-16 15:45	Great explanation! 2024-01-16 13:00 cid6
5	uid5	john_d	john@example.com	password123	2024-01-15 9:30	pid2	MongoDB is a highly scalable ...	Introduction to MongoDB	2024-01-15 9:35	Well explained. 2024-01-15 10:00 cid4 Great article! 2024-01-15 09:45 cid3

Granting Permission to CSV for user 'postgres' and Importing the Data:

```
adminuser@erickson-lab:~$ sudo setfacl -m u:postgres:r /home/adminuser/Downloads/blogging_dataset.csv
adminuser@erickson-lab:~$
adminuser@erickson-lab:~$
adminuser@erickson-lab:~$ psql -h localhost -U postgres -d db_lab
Password for user postgres:
psql (15.5 (Ubuntu 15.5-0ubuntu0.23.10.1))
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, compression: off)
Type "help" for help.

db_lab=#
db_lab=#
db_lab=#
db_lab=# \COPY blog_data FROM '/home/adminuser/Downloads/blogging_dataset.csv' WITH DELIMITER ',' CSV HEADER;
COPY 5
```

Data Import Result: (blog_data table)

```
db_lab=# SELECT * FROM blog_data;
db_lab=# \x auto
Expanded display is used automatically.
db_lab=# SELECT * FROM blog_data;
```

```
-[ RECORD 1 ]-----
user_id      | uid1
username     | sarah_c
email        | sarah@example.com
password     | sarapass
created_at   | 2024-01-16 08:10:00
post_id      | pid3
content      | JavaScript is a popular
title        | JavaScript Basics
updated_at   | 2024-01-16 09:30:00
comments     | JavaScript is amazing! 2024-01-16 08:30 cid5
-[ RECORD 2 ]-----
user_id      | uid2
username     | emily_j
email        | emily@example.com
password     | secure123
created_at   | 2024-01-16 17:30:00
post_id      | pid5
content      | Build web applications using
title        | Python Web Development
updated_at   | 2024-01-16 18:00:00
comments     | Thanks for the tutorial! 2024-01-16 18:00 cid7
-[ RECORD 3 ]-----
user_id      | uid3
username     | jane_s
email        | jane@example.com
password     | passw0rd
created_at   | 2024-01-14 14:45:00
post_id      | pid1
content      | Python is a versatile and
title        | Getting Started with Python
updated_at   | 2024-01-15 10:20:00
comments     | I learned a lot. 2024-01-14 16:00 cid2 | Thanks for the tutorial! 2024-01-14
15:30 cid1
-[ RECORD 4 ]-----
user_id      | uid4
username     | michael_b
email        | michael@example.com
password     | p@ssw0rd
created_at   | 2024-01-16 12:20:00
post_id      | pid4
content      | Learn about various data
```

Verifying blog_data table result in Pgadmin:

Query Query History Scratch Pad X

1 `SELECT * FROM blog_data;`

	user_id	username	email	password	created_at	post_id	content	title	updated_at	comments
	character varying (50)	character varying (50)	character varying (255)	character varying (50)	timestamp without time zone	character varying (50)	text	character varying (255)	timestamp without time zone	character varying
1	uid1	sarah_c	sarah@example.com	sarapass	2024-01-16 08:10:00	pid3	JavaScript is a popular	JavaScript Basics	2024-01-16 09:30:00	JavaScript is am
2	uid2	emily_j	emily@example.com	secure123	2024-01-16 17:30:00	pid5	Build web applications using	Python Web Development	2024-01-16 18:00:00	Thanks for the tu
3	uid3	jane_s	jane@example.com	passw0rd	2024-01-14 14:45:00	pid1	Python is a versatile and	Getting Started with Python	2024-01-15 10:20:00	I learned a lot. 20
4	uid4	michael_b	michael@example.com	p@ssw0rd	2024-01-16 12:20:00	pid4	Learn about various data	Data Structures in C++	2024-01-16 15:45:00	Great explanatio
5	uid5	john_d	john@example.com	password123	2024-01-15 09:30:00	pid2	MongoDB is a highly scalable	Introduction to MongoDB	2024-01-15 09:35:00	Well explained. 20

Tables Creation Process:

Query Query History

1 `-- Creating the table blog_data to import the CSV file`
2 `CREATE TABLE blog_data (`
3 `user_id VARCHAR(50),`
4 `username VARCHAR(50),`
5 `email VARCHAR(255),`
6 `password VARCHAR(50),`
7 `created_at TIMESTAMP,`
8 `post_id VARCHAR(50),`
9 `content TEXT,`
10 `title VARCHAR(255),`
11 `updated_at TIMESTAMP,`
12 `comments VARCHAR(255)`
13 `);`

Query Query History

1 `CREATE TABLE users (`
2 `user_id character varying(5) PRIMARY KEY,`
3 `username VARCHAR(255),`
4 `email VARCHAR(255) UNIQUE,`
5 `password VARCHAR(255),`
6 `created_at TIMESTAMP`
7 `);`
8
9 `CREATE TABLE posts (`
10 `post_id character varying(5) PRIMARY KEY,`
11 `user_id character varying(5) REFERENCES users(user_id),`
12 `content TEXT,`
13 `title VARCHAR(255),`
14 `updated_at TIMESTAMP`
15 `);`
16
17 `CREATE TABLE comments (`
18 `comment_id character varying(5),`
19 `user_id character varying(5) REFERENCES users(user_id),`
20 `post_id character varying(5) REFERENCES posts(post_id),`
21 `comment_content TEXT,`
22 `commented_at TIMESTAMP`
23 `);`

Materialized Views

Operators

Procedures

1.3 Sequences

Tables (4)

- blog_data
- comments
- posts
- users

Inserting Data Process:

```

Query  Query History
1  -----
2  -- Queries to populate, clean and split the data in each table
3  -----
4
5  -- Inserting data to each table based on "blog_data table"
6  -- The blog_data table previously created has the CSV data rows
7
8  -- Insert into users table
9  INSERT INTO users (user_id, username, email, password, created_at)
10 SELECT user_id, username, email, password, created_at FROM blog_data;
11
12
13 -- Insert into posts table
14 INSERT INTO posts (post_id, user_id, content, title, updated_at)
15 SELECT post_id, user_id, content, title, updated_at FROM blog_data;
16
17
18 -- Create a temporary table to clean and get the comments string by user_id
19 CREATE TEMPORARY TABLE temp_comment_string (
20     original_comment VARCHAR(255) NULL,
21     comment_id VARCHAR(50) NULL,
22     user_id VARCHAR(50) NULL,
23     post_id VARCHAR(50) NULL,
24     comment_content VARCHAR(255) NULL,
25     commented_at VARCHAR(50) NULL
26 );
27
28
29 -- Insert data with more than user_id in the same comment string
30 INSERT INTO temp_comment_string (original_comment, comment_id, user_id, post_id, comment_content, commented_at)
31 WITH SplitComments AS (
32     SELECT
33         comments,
34         user_id,
35         post_id,
36         SPLIT_PART(comments, '|', 1) AS comment_part_1,
37         SPLIT_PART(comments, '|', 2) AS comment_part_2
38     FROM
39         blog_data
40     WHERE
41         comments LIKE '%|%'
42 )
43 SELECT
44     comments AS original_comment,
45     'cid' || SPLIT_PART(comment_part_1, 'cid', 2) AS comment_id,
46     user_id,
47     post_id,
48     REGEXP_REPLACE(comment_part_1, '2024.*$', '') AS comment_content,
49     REGEXP_REPLACE(comment_part_1, '^.*?(\d{4}-\d{2}-\d{2} \d{2}:\d{2}).*$', '\1') AS commented_at
50 FROM
51     SplitComments
52
53 UNION ALL
54
55 SELECT
56     comments AS original_comment,
57     'cid' || SPLIT_PART(comment_part_2, 'cid', 2) AS comment_id,
58     user_id,
59     post_id,
60     REGEXP_REPLACE(comment_part_2, '2024.*$', '') AS comment_content,
61     REGEXP_REPLACE(comment_part_2, '^.*?(\d{4}-\d{2}-\d{2} \d{2}:\d{2}).*$', '\1') AS commented_at
62 FROM
63     SplitComments;
64

```

```

68 -- Insert data with only one user_id in the same comment string
69 INSERT INTO temp_comment_string (original_comment, comment_id, user_id, post_id, comment_content, commented_at)
70
71 WITH SplitComments AS (
72     SELECT
73         comments,
74         user_id,
75         post_id,
76         SPLIT_PART(comments, '|', 1) AS comment_part_1,
77         SPLIT_PART(comments, '|', 2) AS comment_part_2
78     FROM
79         blog_data
80     WHERE
81         comments NOT LIKE '%|%'
82 )
83
84 SELECT
85     comments AS original_comment,
86     'cid' || SPLIT_PART(comment_part_1, 'cid', 2) AS comment_id,
87     user_id,
88     post_id,
89     REGEXP_REPLACE(comment_part_1, '2024.*$', '') AS comment_content,
90     REGEXP_REPLACE(comment_part_1, '^.*(\d{4}-\d{2}-\d{2} \d{2}:\d{2}).*$', '\1') AS commented_at
91 FROM
92     SplitComments
93
94 UNION ALL
95
96 SELECT
97     comments AS original_comment,
98     'cid' || SPLIT_PART(comment_part_2, 'cid', 2) AS comment_id,
99     user_id,
100    post_id,
101    REGEXP_REPLACE(comment_part_2, '2024.*$', '') AS comment_content,
102    REGEXP_REPLACE(comment_part_2, '^.*(\d{4}-\d{2}-\d{2} \d{2}:\d{2}).*$', '\1') AS commented_at
103 FROM
104     SplitComments;
105
106
107 -- Delete rows where comment_content and commented_at is empty after insert
108 DELETE FROM temp_comment_string WHERE comment_content = '' OR commented_at = ''
109
110
111 -- Finally, insert in the comments table the clean data
112 INSERT INTO comments (comment_id, user_id, post_id, comment_content, commented_at)
113 SELECT
114     comment_id,
115     user_id,
116     post_id,
117     TRIM(BOTH ' ' FROM comment_content) AS comment_content,
118     commented_at::timestamp AS commented_at --> casting to avoid datetime error conversion
119 FROM temp_comment_string;
120

```

Making the following queries:

a) Retrieve all users

```

121
122 -- very data in each table
123 SELECT * FROM users
124 --SELECT * FROM posts
125 --SELECT * FROM comments
126

```

	user_id [PK] character varying (5)	username character varying (255)	email character varying (255)	password character varying (255)	created_at timestamp without time zone
1	uid1	sarah_c	sarah@example.com	sarapass	2024-01-16 08:10:00
2	uid2	emily_j	emily@example.com	secure123	2024-01-16 17:30:00
3	uid3	jane_s	jane@example.com	passw0rd	2024-01-14 14:45:00
4	uid4	michael_b	michael@example.com	p@ssw0rd	2024-01-16 12:20:00
5	uid5	john_d	john@example.com	password123	2024-01-15 09:30:00

b) Retrieve all posts

```

122 -- very data in each table
123 --SELECT * FROM users
124 SELECT * FROM posts
125 --SELECT * FROM comments
126

```

	post_id [PK] character varying (5)	user_id character varying (5)	content text	title character varying (255)	updated_at timestamp without time zone
1	pid3	uid1	JavaScript is a popular	JavaScript Basics	2024-01-16 09:30:00
2	pid5	uid2	Build web applications using	Python Web Development	2024-01-16 18:00:00
3	pid1	uid3	Python is a versatile and	Getting Started with Python	2024-01-15 10:20:00
4	pid4	uid4	Learn about various data	Data Structures in C++	2024-01-16 15:45:00
5	pid2	uid5	MongoDB is a highly scalable	Introduction to MongoDB	2024-01-15 09:35:00

c) Retrieve all comments

```

122 -- very data in each table
123 --SELECT * FROM users
124 --SELECT * FROM posts
125 SELECT * FROM comments
126

```

	comment_id character varying (5)	user_id character varying (5)	post_id character varying (5)	comment_content text	commented_at timestamp without time zone
1	cid2	uid5	pid1	I learned a lot.	2024-01-14 16:00:00
2	cid4	uid5	pid2	Well explained.	2024-01-15 10:00:00
3	cid1	uid3	pid1	Thanks for the tutorial!	2024-01-14 15:30:00
4	cid3	uid5	pid2	Great article!	2024-01-15 09:45:00
5	cid5	uid1	pid3	JavaScript is amazing!	2024-01-16 08:30:00
6	cid7	uid2	pid5	Thanks for the tutorial!	2024-01-16 18:00:00
7	cid6	uid4	pid4	Great explanation!	2024-01-16 13:00:00

d) Update the user email of sarah_c to sarab@gmail.com

```

1 -- updating email for sarah
2 UPDATE users
3 SET email = 'sarab@gmail.com'
4 WHERE username = 'sarah_c' and user_id = 'uid1';
5
6 -- Verify the update
7 SELECT * FROM users WHERE user_id = 'uid1';

```

	user_id [PK] character varying (5)	username character varying (255)	email character varying (255)	password character varying (255)	created_at timestamp without time zone
1	uid1	sarah_c	sarab@gmail.com	sarapass	2024-01-16 08:10:00

e) Delete comment id cid4


```

Query    Query History
1  -- deleting comment_id = cid4
2  DELETE FROM comments WHERE TRIM(BOTH ' ' FROM comment_id) = 'cid4';
3
4  -- Verify the comment don't exist anymore!
5  SELECT * FROM comments WHERE TRIM(BOTH ' ' FROM comment_id) = 'cid4';
6

```

Data Output Messages Notifications

comment_id	user_id	post_id	comment_content	commented_at
character varying (5)	character varying (5)	character varying (5)	text	timestamp without time zone



f) What happens if you insert the data below into the user table?

User-id: uid5, Username: John_d, Email: john@go.com , Password: password123

Error of primary key violation, because duplicate **userid = uid5**

```

db_lab
├── Casts
├── Catalogs
├── Event Triggers
├── Extensions
├── Foreign Data Wrappers
├── Languages
├── Publications
└── Schemas (1)
    └── public
        ├── Aggregates
        ├── Collations
        ├── Domains
        ├── FTS Configurations
        ├── FTS Dictionaries
        ├── FTS Parsers
        ├── FTS Templates
        ├── Foreign Tables
        ├── Functions
        └── Materialized Views

```

```

Query    Query History
1
2  -- Inserting a duplicate user_id we will receive an error
3  -- of violation constrain, because is the primary key
4  -- and do not permit duplicates rows in the column: user_id
5  INSERT INTO users (user_id, username, email, password)
6  VALUES ('uid5', 'John_d', 'john@go.com', 'password123');
7
8

```

Data Output Messages Notifications

```

ERROR:  Key (user_id)=(uid5) already exists.duplicate key value violates unique constraint "users_pkey"
ERROR:  duplicate key value violates unique constraint "users_pkey"
SQL state: 23505
Detail: Key (user_id)=(uid5) already exists.

```



The SQL source code

ALL SQL Queries.sql

Dataset

blogging_dataset.csv