Osamah AlBahnasi ICS 424-01 201970750

Assignment 2 - Project 1
Database Administration Report

3/30/2024

I initially have created the database within my database server ‘ OzyDBServer’ within pgAdmin4:

This screenshot represents the 2 Databases within the server:

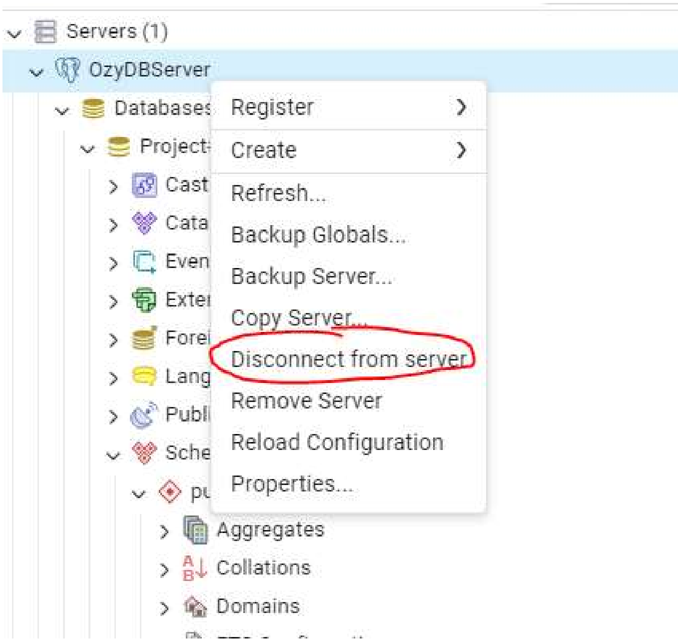
1- Project#1

2- postgres (default administrative connection database)

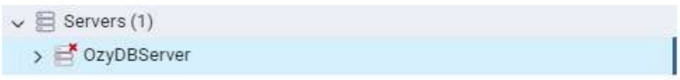
1) Demonstrate startup and shutdown options and procedures.

Shutdown option:

In order to disconnect from/shutdown the server, I must kill the kill the connection with the option as such:

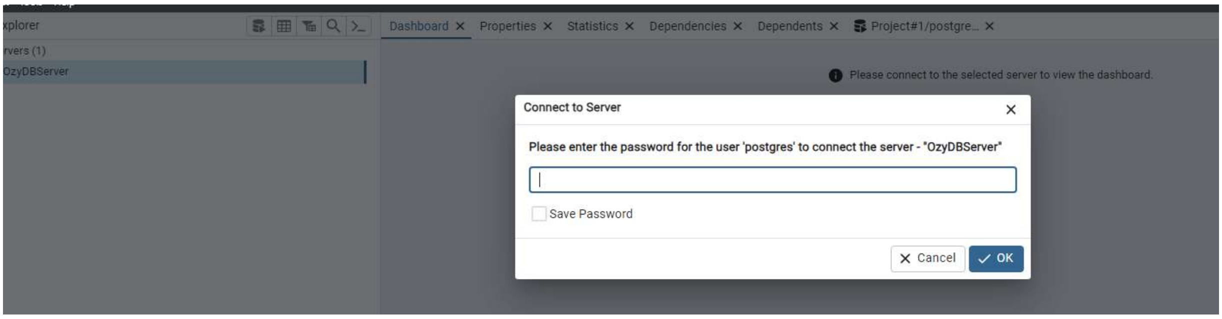


When doing so, there are no ways of seeking any analytics or a checking on the dashboard of the database -- despite it being hosted in the 'localhost'.

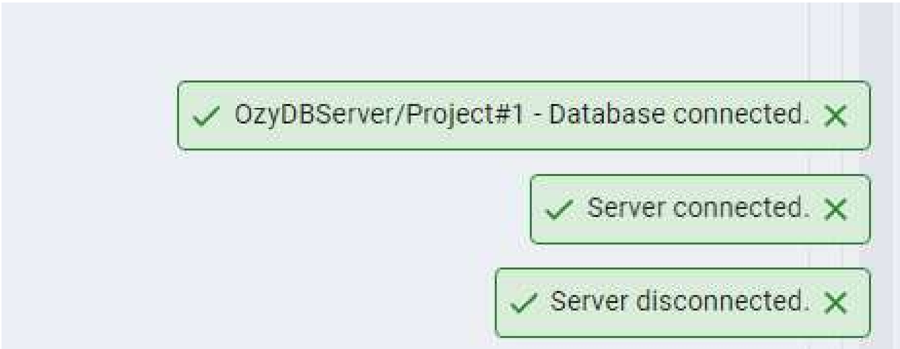


Startup option:

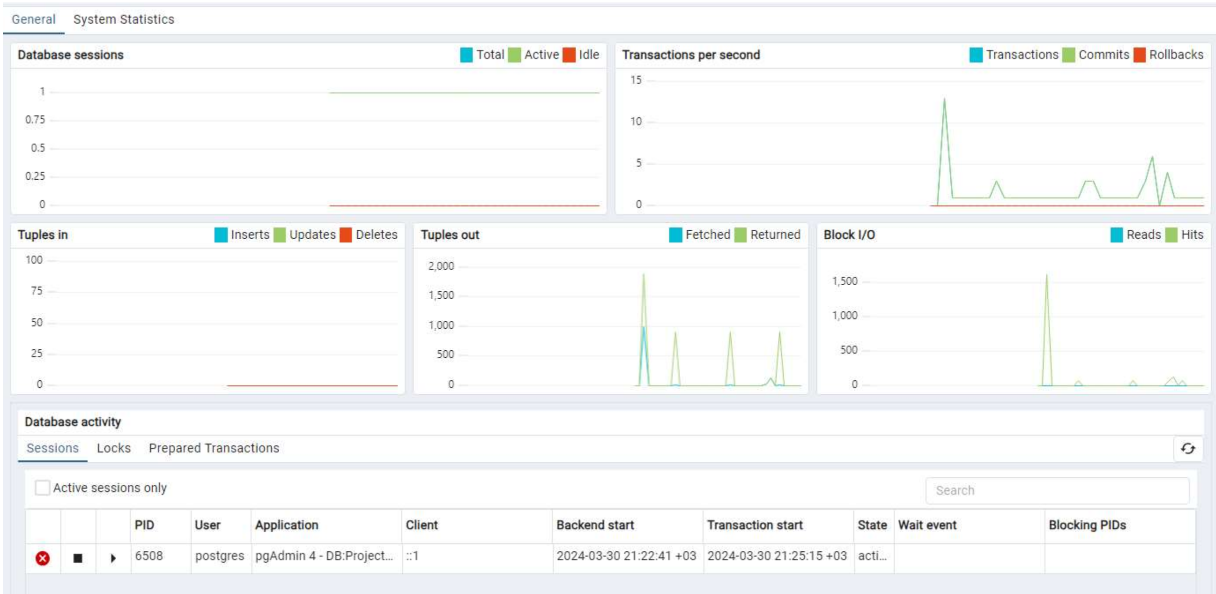
To regain back my connection, I must click the serverDB and login with the dedicated password security protocol embedded within postgresSQL's application.



We can verify that the server is back up from the following notifications:

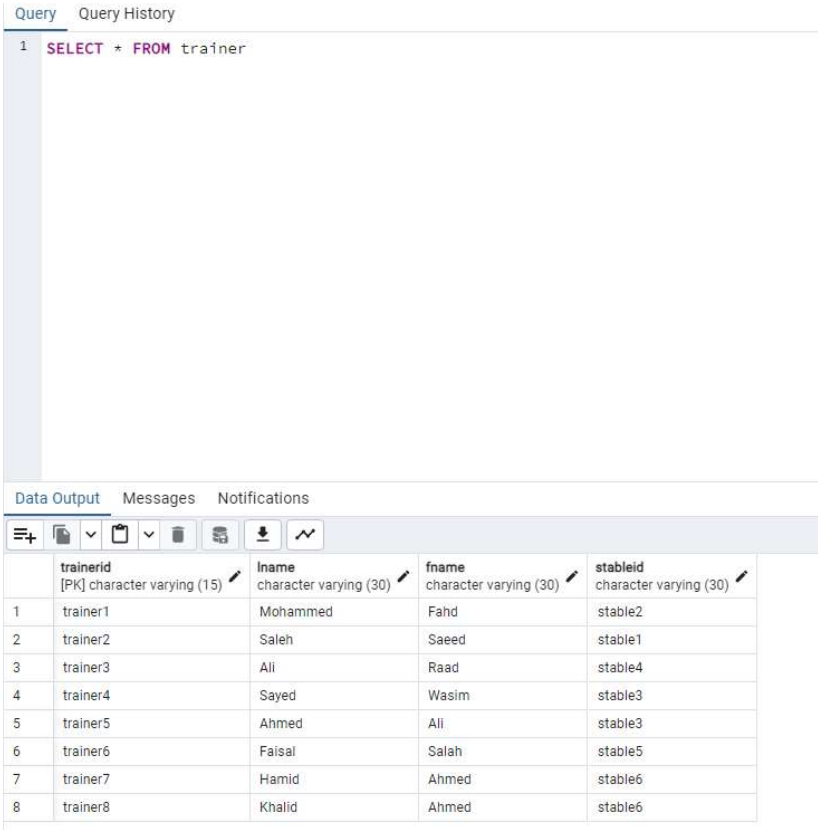


As well as the dashboard sessions have been reset and is slowly initializing the data to gain quicker access to the DB:



2) Demonstrate both the logical and physical structures of a database.

Logical Structure:



	trainerid [PK] character varying (15)	lname character varying (30)	fname character varying (30)	stableid character varying (30)
1	trainer1	Mohammed	Fahd	stable2
2	trainer2	Saleh	Saeed	stable1
3	trainer3	Ali	Raad	stable4
4	trainer4	Sayed	Wasim	stable3
5	trainer5	Ahmed	Ali	stable3
6	trainer6	Faisal	Salah	stable5
7	trainer7	Hamid	Ahmed	stable6
8	trainer8	Khalid	Ahmed	stable6

We can view the tables as a part of the logical structure of our DB with the use of SQL queries.

Physical Structures:

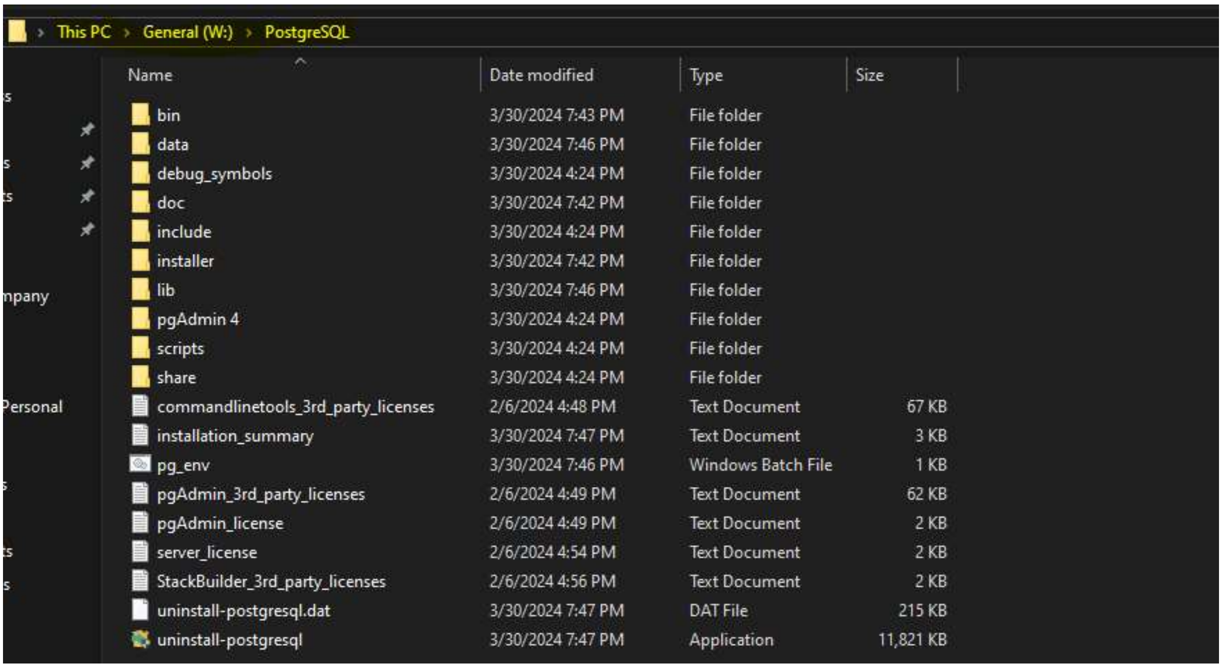
Data is stored on disk in files known as data files, which contain the actual data stored in tables, indexes, and other database objects.

By default, these data files are in the 'base' directory within the specified data directory, such as 'W:\PostgreSQL\data'.

However, administrators can manage the physical location of database objects using tablespaces.

Tablespaces allow administrators to specify custom locations for storing database objects, optimizing storage performance, and managing disk space efficiently.

Administrators can create additional tablespaces and specify their locations on the file system, providing flexibility in organizing and storing data within the PostgreSQL database environment.



3) Create a materialized view with a complex join.

Query Query History

```
-- Create a materialized view joining the "race" and "track" tables
CREATE MATERIALIZED VIEW race_track_mv AS
SELECT r.*, t.location
FROM race r
JOIN track t ON r.trackname = t.trackname;
```

- We're creating a materialized view named "race_track_mv".
- We're selecting all columns from the "race" table (r.*) along with the "location" column from the "track" table (t.location).
- We're joining the "race" table (r) with the "track" table (t) using the "trackname" column from the "race" table and the "name" column from the "track" table.

Query Query History

```
-- Refresh the materialized view to populate it with current data
REFRESH MATERIALIZED VIEW race_track_mv;
```

- After creating the materialized view, we're refreshing it to populate it with the current data from the underlying tables.

Final output:

Query Query History

```
1 SELECT * FROM race_track_mv;
```

Data Output Messages Notifications

	raceid character varying (15)	racename character varying (30)	trackname character varying (30)	racedate date	racetime time without time zone	location character varying (30)
5	race5	Claiming Stake	Sharjah	2007-05-03	12:30:00	UE
6	race6	3-year-old fillies	Jubail	2007-06-02	12:30:00	SA
7	race7	Handicap	Jubail	2007-06-02	09:30:00	SA
8	race8	2-year-old colts	Real Runs	2007-06-02	10:30:00	SA
9	race9	2-year-old fillies	Jubail	2007-06-02	11:30:00	SA
10	race10	Claiming Stake	Sharjah	2007-06-02	12:30:00	UE
11	race11	3-year-old fillies	Deira	2007-04-02	10:30:00	UE
12	race12	Handicap	Yanbu	2007-05-03	11:30:00	SA
13	race13	3-year-old fillies	Yanbu	2007-05-03	11:00:00	SA
14	race14	Handicap	Dhahran	2007-05-10	10:00:00	SA
15	race15	3-year-old colts	Deira	2007-05-12	15:00:00	UE
16	race16	Claiming Stake	Yanbu	2007-05-20	14:30:00	SA
17	race17	Handicap	Doha Park	2007-05-20	13:00:00	QT
18	race18	3-year-old fillies	Sharjah	2007-05-21	08:00:00	UE
19	race19	2-year-old colts	Dhahran	2007-05-25	11:00:00	SA
20	race20	Claiming Stake	Blue Hills	2007-05-25	08:30:00	SA
21	race21	3-year-old colts	Real Runs	2007-03-19	14:30:00	SA
22	race22	Handicap	Dhahran	2007-03-27	15:00:00	SA
23	race23	3-year-old fillies	Blue Hills	2007-03-28	09:30:00	SA
24	race24	3-year-old colts	Jubail	2007-03-28	13:30:00	SA
25	race25	Claiming Stake	Blue Hills	2007-03-29	10:00:00	SA
26	race26	3-year-old colts	Yanbu	2007-03-30	12:30:00	SA
27	race27	Handicap	Deira	2007-04-03	14:00:00	UE
28	race28	2-year-old fillies	Blue Hills	2007-04-04	08:30:00	SA
29	race29	3-year-old colts	Bahrain	2007-04-05	08:00:00	BH
30	race30	Claiming Stake	Dhahran	2007-04-08	09:30:00	SA
31	race31	Handicap	Dhahran	2007-04-08	09:00:00	SA
32	race32	2-year-old colts	Jubail	2007-04-09	11:00:00	SA

4) Develop and demonstrate one stored procedure and one trigger.

Stored Procedure:

Query Query History

```
CREATE OR REPLACE FUNCTION calculate_horse_birth_year(age_in_years INTEGER) RETURNS INTEGER AS $$
DECLARE
    current_year INTEGER;
    birth_year INTEGER;
BEGIN
    -- Get the current year
    SELECT EXTRACT(YEAR FROM CURRENT_DATE) INTO current_year;

    -- Calculate the birth year based on the age
    birth_year := current_year - age_in_years;

    RETURN birth_year;
END;
```

- This procedure will calculate the year the horse was born.
- Utilizing it as such:

Query Query History

```
SELECT
    horseid,
    horsename,
    age,
    calculate_horse_birth_year(age) AS birth_year
FROM
    horse;
```

Data Output Messages Notifications

	horseid [PK] character varying (15)	horsename character varying (15)	age integer	birth_year integer
1	horse1	Warrior	2	2022
2	horse2	Conquerer	2	2022
3	horse3	Dove of Peace	3	2021
4	horse4	Ever Faster	3	2021
5	horse5	Slow Winner	2	2022
6	horse6	Windrunner	2	2022
7	horse7	Catapult	4	2020
8	horse8	Flying Force	2	2022
9	horse9	Laggard	2	2022
10	horse10	Formula One	6	2018
11	horse11	Frisky Frolic	3	2021
12	horse12	Fantastic	3	2021
13	horse13	Midnight	2	2022
14	horse14	Running Wild	4	2020
15	horse15	FastOffMyFeet	3	2021
16	horse16	Slow Poke	2	2022
17	horse17	Slinger	3	2021
18	horse18	Sublime	5	2019
19	horse19	Front Runner	4	2020
20	horse20	Night	3	2021
21	horse21	Negative	3	2021
22	horse22	Lightening	2	2022
23	horse23	Lazy Loser	4	2020

Trigger Procedure:

Query Query History

```sql
CREATE OR REPLACE FUNCTION update_horse_birth_year()
RETURNS TRIGGER AS $$
BEGIN
    NEW.birth_year := EXTRACT(YEAR FROM CURRENT_DATE) - NEW.age;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

Data Output Messages Notifications

```
CREATE FUNCTION

Query returned successfully in 358 msec.
```

- Similar functionality to the previous, except applied via trigger.

Query Query History

```sql
CREATE TRIGGER update_birth_year_trigger
BEFORE INSERT OR UPDATE OF age ON horse
FOR EACH ROW
EXECUTE FUNCTION update_horse_birth_year();
```

Data Output Messages Notifications

```
CREATE TRIGGER

Query returned successfully in 392 msec.
```

- No new updates will be found here as this documentation is being done within the same day.

5) Demonstrate locking and timestamping.

Locking:

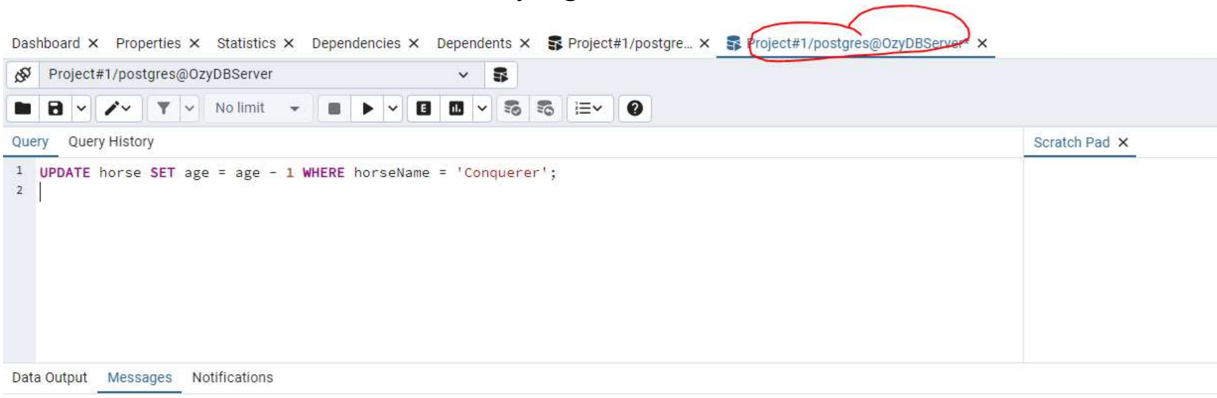
- We first would want to lock the table (horse) from transaction#1:

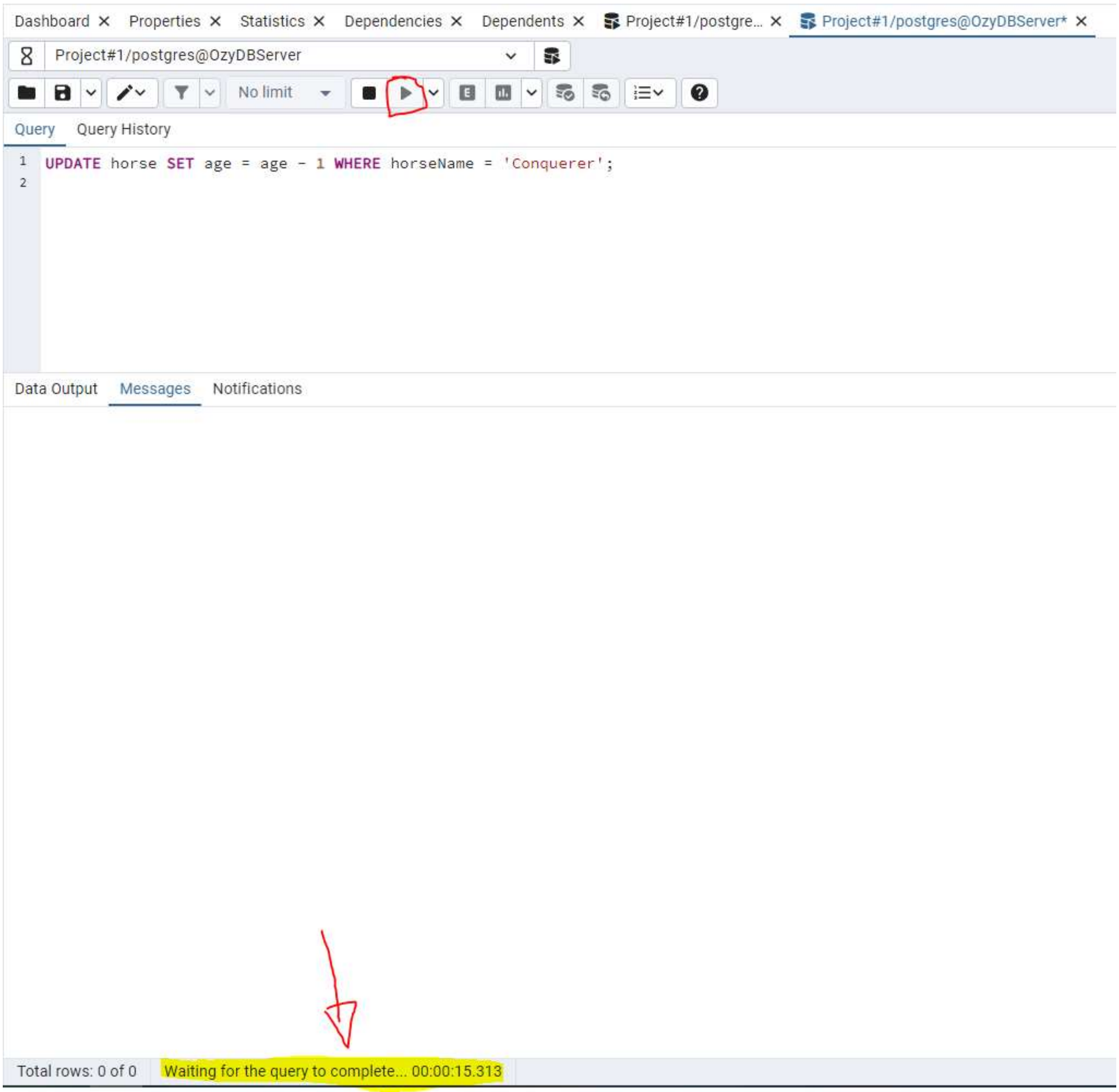
- 

- 

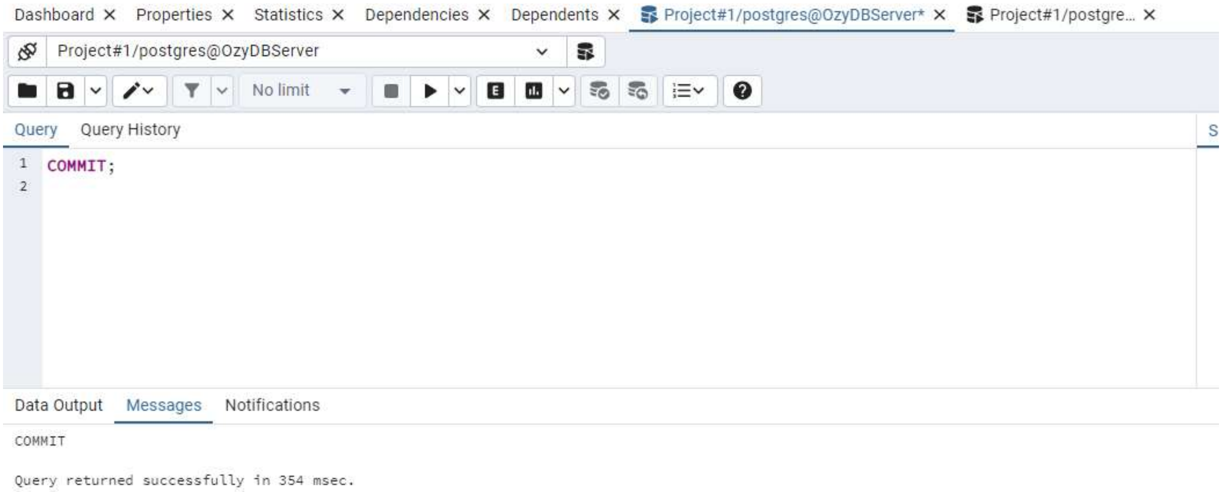
- 

- Then showcase the transaction#2 trying to access the locked table:

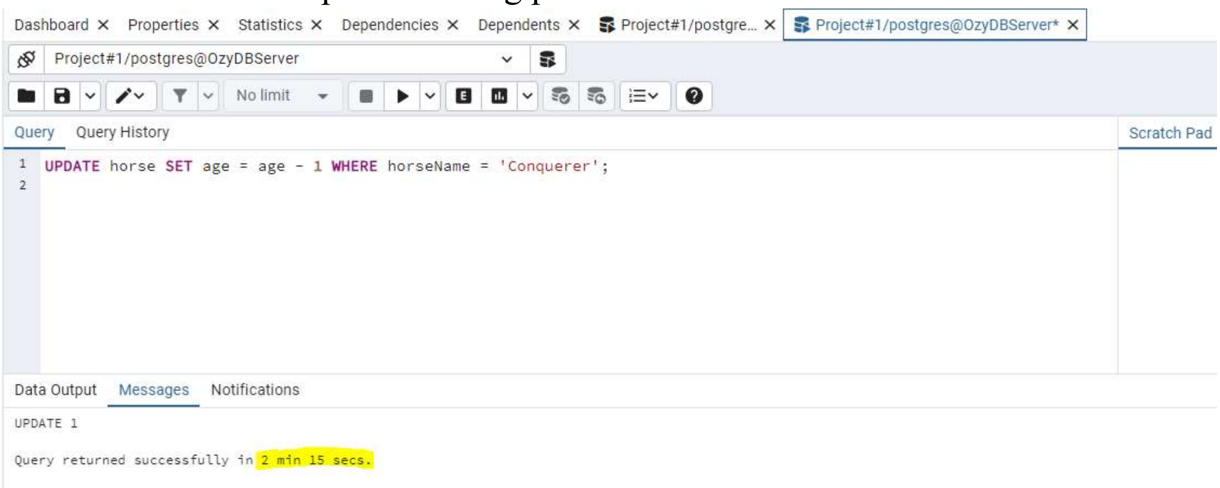




- What’s being shown here is that the query in transaction #2 is waiting for the table to be unlocked to complete the query. To do so, we must unlock the table from transaction#1, as such:



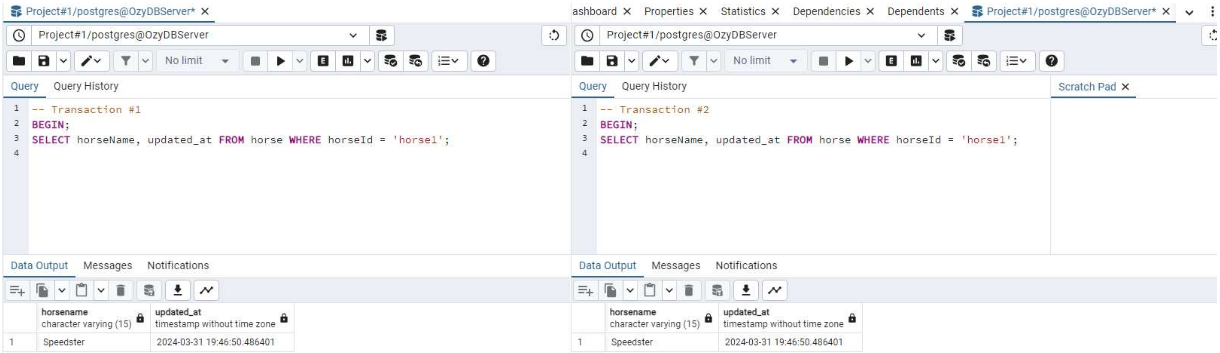
- Now that the transaction has been committed, the table will also unlock as it follows the strict two-phase locking protocol.



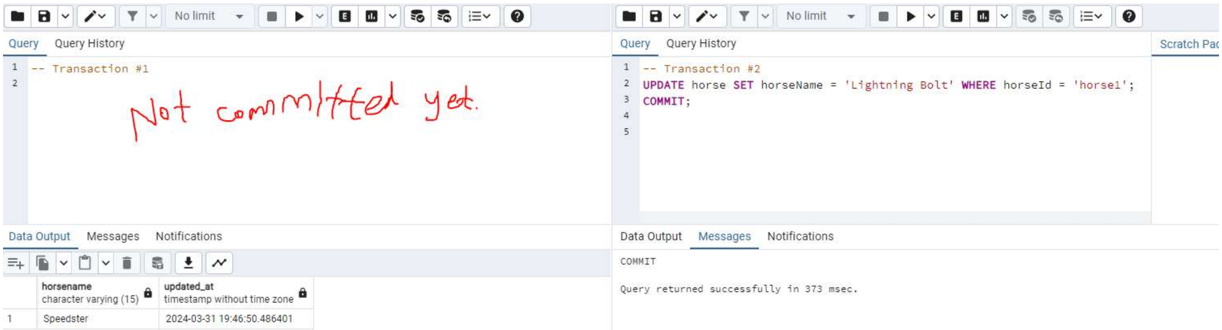
- Now the query in transaction#2 has been complete (and committed) into our table/DB.

Timestamping:

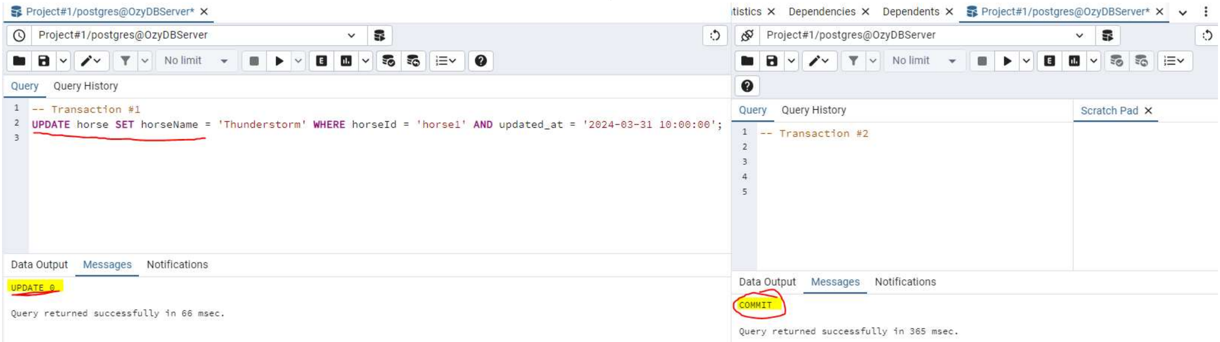
- Showcase timestamping procedures and the violations that occurs within it:



-



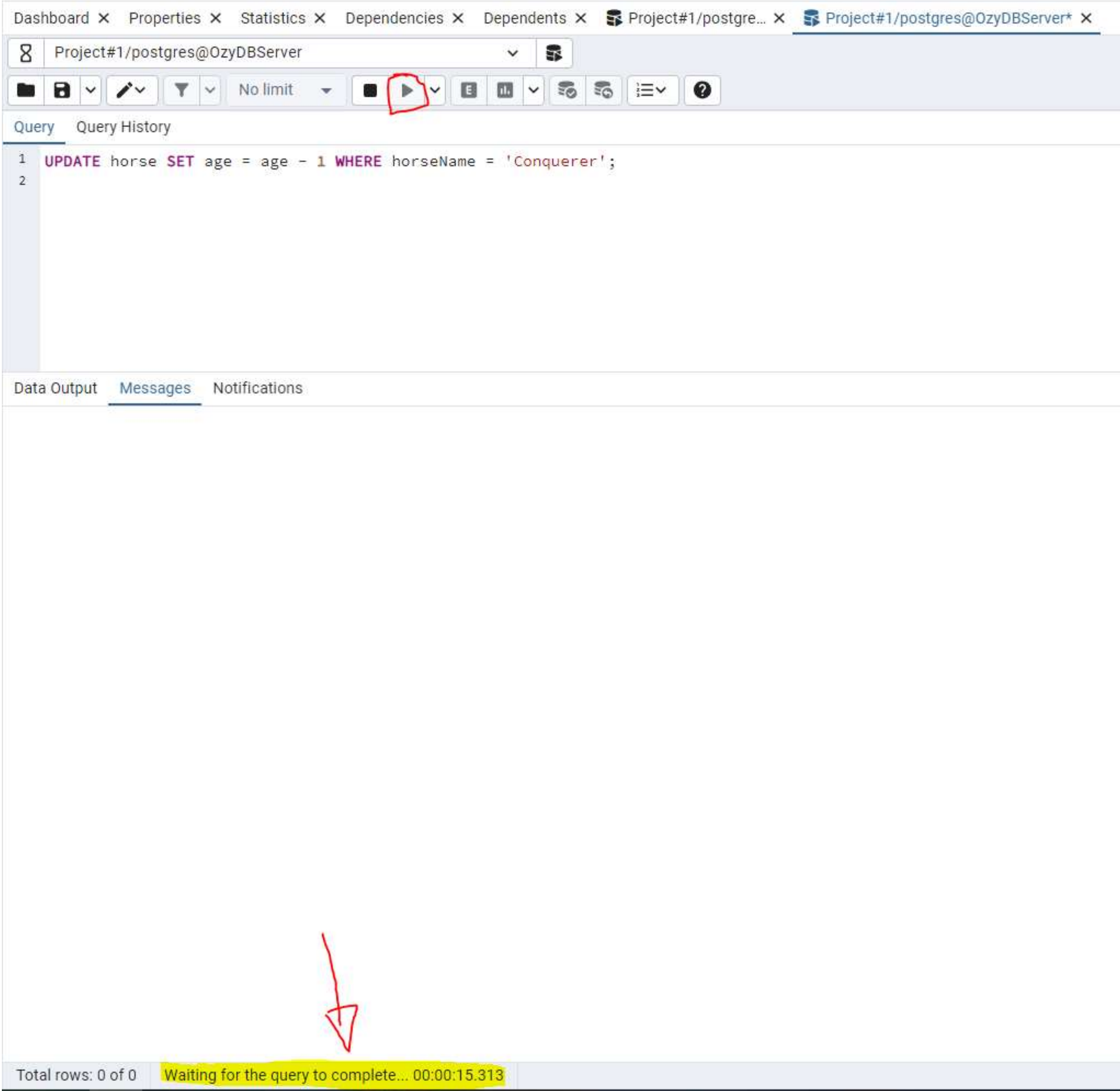
-



-
- As shown, T1's attempt to update the row based on an outdated timestamp (its last read time) violates the timestamp-based concurrency control mechanism. The update fails because the actual ‘updated_at’ timestamp in the database has been changed by T2, indicating that the data T1 read is no longer current.
- Note: ‘UPDATE 0’ indicates that the operation was completed successfully, but no rows met the criteria specified in the ‘UPDATE’ statement, so no changes were made to the database.

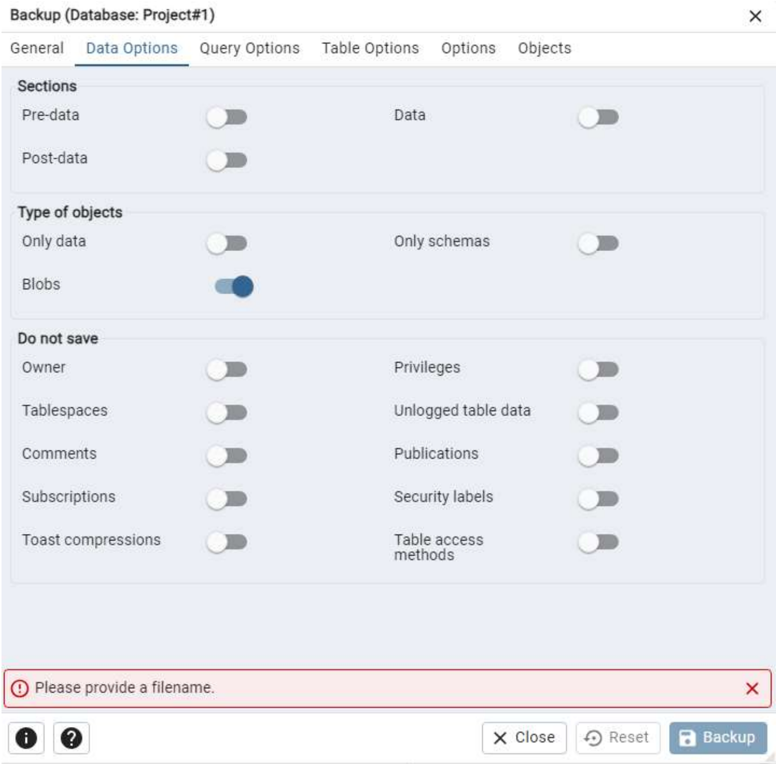
6) Diagnose and resolve locking conflicts.

- Within the previous part (#5), we showcased the locking protocol, showcasing how the locking functions within our DB:



- As shown, we resolve the locking conflict by 'waits'; waiting for the data to be unlocked by another transaction to progress with the execution.
- Whilst when it comes to timestamping, we adapt to the roll-back of transactions that are unable to execute an immediate update when it comes to the conflict of timestamping.
-

7) Evaluate the various backup options available.



Sections

- Pre-data: This includes commands to set up the database schema such as creating tables, without inserting any data.
- Data: This represents the actual data within the tables.
- Post-data: This can include definitions that must be applied after the data is inserted, such as indexes, triggers, and constraints.

Type of Objects

- Only data: If toggled on, the backup will include only the data, not the schema (table definitions, functions, etc.).
- Only schema: If toggled on, the backup will include only the database schema.
- Blobs: Binary Large OBjects, usually used to store large data such as images, audio, or other multimedia objects.

Do Not Save

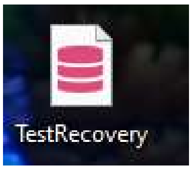
- Owner: Excludes ownership information from the backup.
- Privileges: Excludes privilege (GRANT/REVOKE) information.
- Tablespaces: Excludes tablespace assignments for database objects.
- Unlogged Table Data: Excludes data from unlogged tables, which are not recorded in the Write-Ahead Log.
- Comments: Excludes comments.
- Publications: Excludes publication information used in logical replication.
- Security Labels: Excludes security label assignments for database objects.
- Subscriptions: Excludes subscription information used in logical replication.
- Toast Compressions: Excludes storage settings for large field values (TOASTed values).
- Table Access Methods: Excludes information about access methods used by tables.

When **evaluating** these options, you should consider the specific needs of your backup:

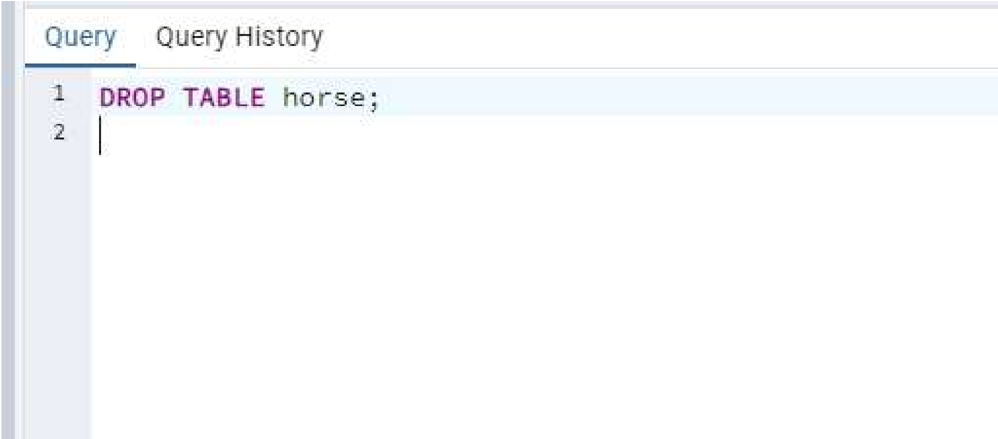
- **Full Backup:** To perform a full backup, include pre-data, data, and post-data. Include blobs if you have binary data in your database. Leave the 'Do not save' options turned off to ensure a complete backup.
- **Schema-Only Backup:** For a schema-only backup (no data), enable 'Only schema'. This is useful for creating a database structure without the data, possibly for setting up test environments.
- **Data-Only Backup:** For a data-only backup, enable 'Only data'. This might be used when you need to refresh the data in another environment where the schema is already deployed.

8) Recover a database.

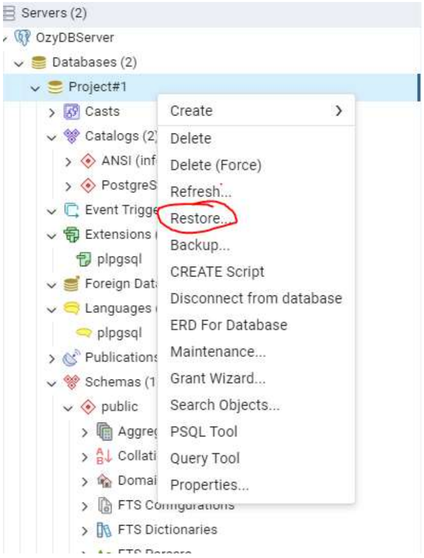
In this case we will need to first backup our data (which I have with a full-backup):

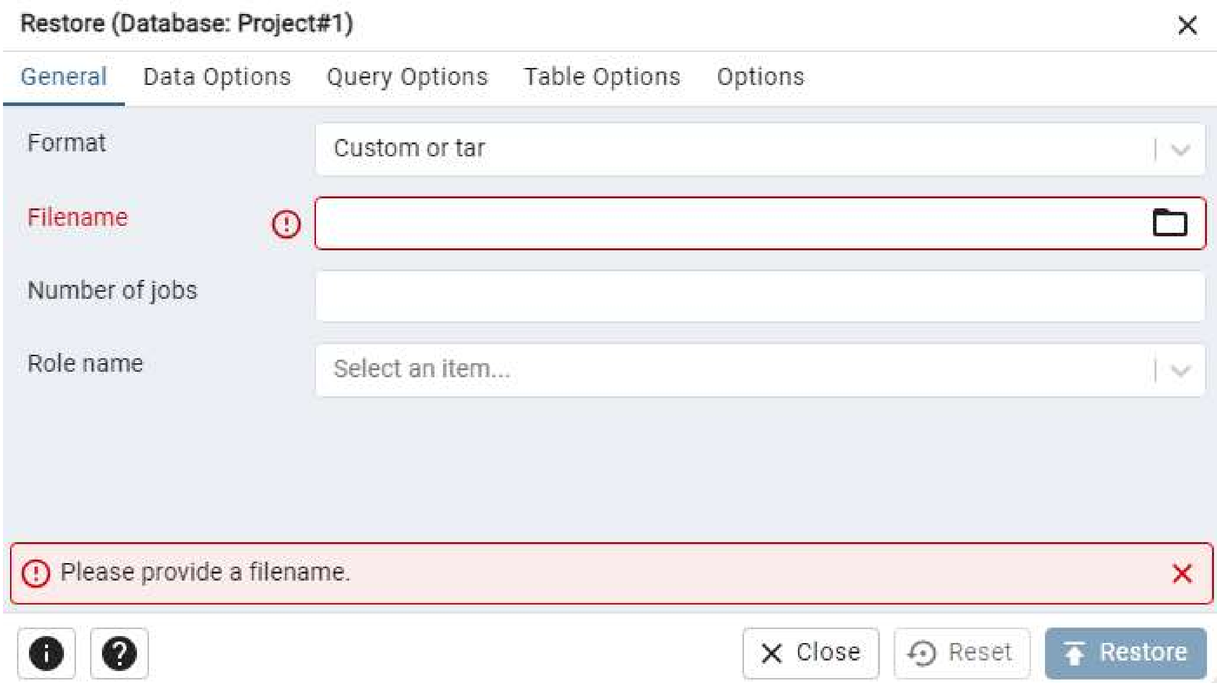


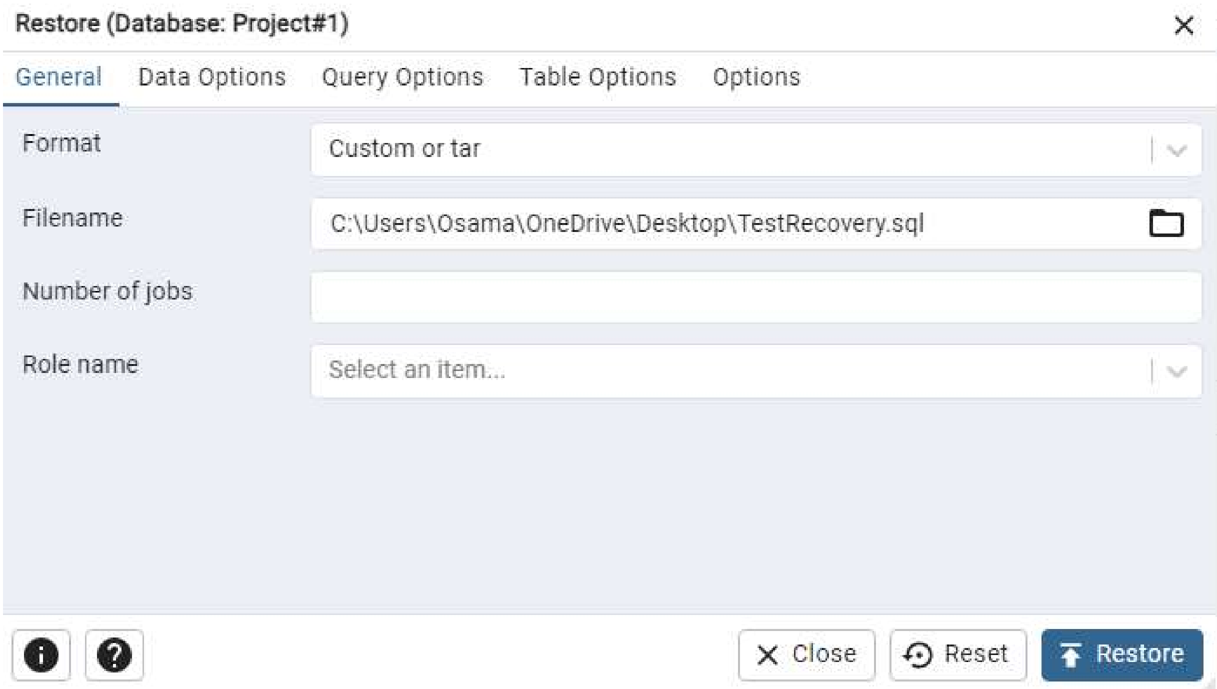
Now that we have backed-up our data, we will work on dropping a table as an example and try to recover it with our backed-up data.



Now that the table 'horse' is no longer in our DB, we will recover it with our 'TestRecovery' SQL file.







Now we have recovered our table ‘Horse’ back to our DB, with its recent changes.

Data Output Messages Notifications

	horseid [PK] character varying (15)	horsename character varying (15)	age integer	gender character	registration integer	stableid character varying (30)	created_at timestamp without time zone	updated_at timestamp without time zone	birth_year integer
1	horse3	Dove of Peace	3	C	33333	stable1	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
2	horse4	Ever Faster	3	F	44444	stable3	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
3	horse5	Slow Winner	2	C	55555	stable3	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
4	horse6	Windrunner	2	F	66666	stable2	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
5	horse7	Catapult	4	M	77777	stable6	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
6	horse8	Flying Force	2	C	88888	stable4	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
7	horse9	Laggard	2	F	99999	stable4	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
8	horse10	Formula One	6	G	10101	stable2	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
9	horse11	Frisky Frolic	3	C	11011	stable4	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
10	horse12	Fantastic	3	F	12121	stable2	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
11	horse13	Midnight	2	C	13131	stable3	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
12	horse14	Running Wild	4	S	14141	stable2	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
13	horse15	FastOffMyFeet	3	C	15151	stable1	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
14	horse16	Slow Poke	2	C	16161	stable3	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
15	horse17	Slinger	3	F	17171	stable2	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
16	horse18	Sublime	5	M	18181	stable6	2024-03-30 22:09:13.520637	2024-03-30 22:09:13.520637	[null]
17	horse19	Front Runner	4	G	19191	stable4	2024-03-30 22:09:13.520637	2024-	
18	horse20	Night	3	C	20200	stable1	2024-03-30 22:09:13.520637	2024-	
19	horse21	Negative	3	F	21210	stable3	2024-03-30 22:09:13.520637	2024-	
20	horse22	Lightening	2	C	22220	stable6	2024-03-30 22:09:13.520637	2024-	
21	horse23	Lazy Loser	4	G	23230	stable1	2024-03-30 22:09:13.520637	2024-	

Process started

Restoring backup on the server 'OzyDBServer (localhost:5432)'

View Processes

9) Create and manage indexes.

Query | Query History

```sql
CREATE INDEX idx_horse_name ON horse (horseName);
```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 681 msec.
```

Query | Query History

```sql
SELECT * FROM pg_stat_user_indexes WHERE relname = 'horse';
```

Data Output | Messages | Notifications

	relid oid	indexrelid oid	schemaname name	relname name	indexrelname name	idx_scan bigint	last_idx_scan timestamp with time zone	idx_tup_read bigint	idx_tup_fetch bigint
1	16480	16483	public	horse	horse_pkey	134	2024-03-31 22:41:02.239246+03	134	134
2	16480	16572	public	horse	idx_horse_name	0	[null]	0	0

We could also use indexes with conditions (Partial indexing):

Query | Query History

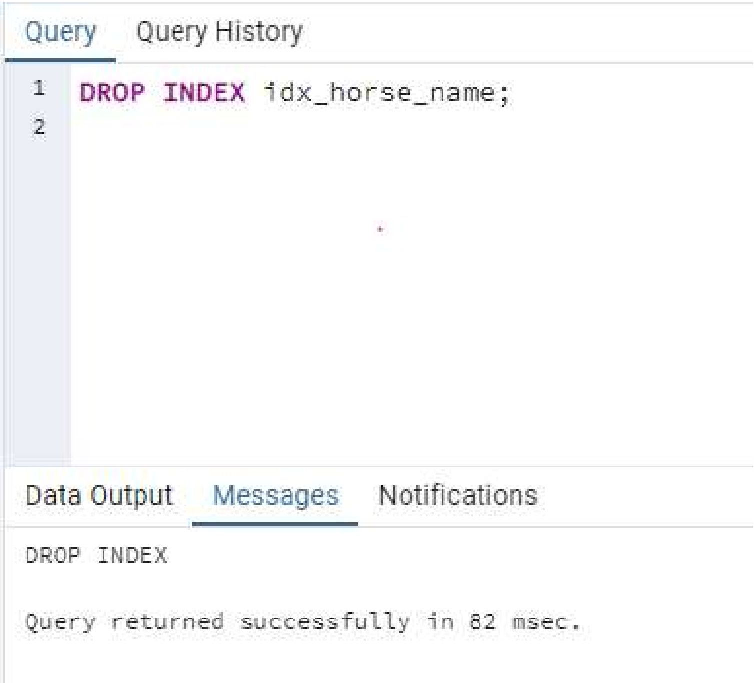
```sql
CREATE INDEX idx_young_horses ON horse (horseName) WHERE age < 5;
```

Data Output | Messages | Notifications

```
CREATE INDEX

Query returned successfully in 60 msec.
```

As well as dropping the index:



Similar to a normal database, indexes can be managed/manipulated in a variety of ways:

- **Index Creation**
  - CREATE INDEX index_name ON table_name (column_name);
    - Creates a new index on the specified column(s) of a table.
  - CREATE UNIQUE INDEX index_name ON table_name (column_name);
    - Creates a unique index, which ensures that the indexed columns do not contain duplicate values.
- **Index Modification**
  - REINDEX INDEX index_name;
    - Rebuilds an existing index; used if the index becomes bloated or is suspected to be corrupted.
  - ALTER INDEX index_name SET TABLESPACE new_tablespace;
    - Moves the index to a different tablespace.
- **Index Deletion**
  - DROP INDEX index_name;
    - Removes an index from the database.

- **Index Monitoring**
  - SELECT * FROM pg_stat_user_indexes WHERE relname = 'table_name';
    - Shows the usage statistics for indexes on a specified table.
  - SELECT * FROM pg_statio_user_indexes WHERE relname = 'table_name';
    - Provides I/O statistics for indexes on a specified table.
- **Index Maintenance**
  - VACUUM (VERBOSE, ANALYZE) table_name;
    - Cleans up dead tuples from the table and associated indexes, and updates statistics used by the query planner.
- **Managing Index Locking**
  - CREATE INDEX CONCURRENTLY index_name ON table_name (column_name);
    - Creates an index without locking out writes to the table.
  - DROP INDEX CONCURRENTLY index_name;
    - Drops an index without locking out writes to the table.
- **Managing Indexes for Constraints**
  - ALTER TABLE table_name ADD CONSTRAINT constraint_name UNIQUE (column_name);
    - Adds a unique constraint to a table (and automatically creates a unique index).
  - ALTER TABLE table_name DROP CONSTRAINT constraint_name;
    - Drops a constraint from a table (and automatically drops the associated index).

10) Collect and analyze relevant database performance information. Display important results of your analysis.

There are a variety of analysis we could represent from our DB's performance, such as the activity & connections, Index usages, table access statistics, I/O statistics, and long running queries.

In our case, we will be using the 'Table access statistics':

Query | Query History

```
SELECT relname, seq_scan, seq_tup_read, idx_scan, n_tup_ins, n_tup_upd, n_tup_del FROM pg_stat_user_tables;
```

Data Output | Messages | Notifications

	relname name	seq_scan bigint	seq_tup_read bigint	idx_scan bigint	n_tup_ins bigint	n_tup_upd bigint	n_tup_del bigint
1	owner	1	0	31	20	0	0
2	trainer	3	16	0	8	0	0
3	stable	1	0	35	6	0	0
4	raceresults	1	0	0	75	0	0
5	track	4	30	36	10	0	0
6	owns	1	0	0	31	0	0
7	horse	11	287	134	26	9	0
8	race_track_mv	1	36	[null]	72	0	0
9	race	4	108	75	36	0	0

- The **horse** table has had 11 sequential scans which read 287 rows in total, and 134 index scans. This may suggest that although some queries are effectively using indexes, there may be opportunities to optimize queries that result in sequential scans.
- The **track** table has four sequential scans reading 30 rows, and 36 index scans. Since the number of sequential scans is low and the number of rows read is also low, the table appears to be well-indexed or not heavily queried.
- The **race** table shows 4 sequential scans with 108 rows read and 75 index scans. This could indicate good use of indexes but may also suggest checking if the queries that lead to sequential scans could be optimized with better indexing.
- The **race_track_mv** materialized view shows index scans with a null value in idx_scan, which might indicate an issue or just a lack of index scan operations if the value isn't being tracked or an index isn't present.