CSE 4/560 Data Models and Query Language Semester Project

JON KICK[*], University at Buffalo, USA
MANCY SAXENA[†], University at Buffalo, USA

To prevent the introduction of injurious insects, noxious weeds, and plant diseases into the state, it is imperative to establish a comprehensive database containing pertinent information about licensed nursery growers and greenhouses. By designing and implementing a database for nursery growers, we are aiming to make it easier to manage, store and retrieve data. We are also aiming to ensure integrity constraints by implementing appropriate constraints and normalizing the tables.

Additional Key Words and Phrases: datasets, databases, database design, horticulture dataset

ACM Reference Format:
Jon Kick and Mancy Saxena. 2018. CSE 4/560 Data Models and Query Language Semester Project . In *CSE 4/560 Data Models and Query Language Semester Project*. ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

Having information regarding licensed nurseries is vital to track any pests, noxious weeds, and plant diseases in the ecosystem that are potentially hampering agricultural productivity, plant health, and environmental stability. The database that we have designed is such that it contains useful information about license numbers, license owners, nursery addresses, trade names, contacts etc so that all license-related information can be easily inserted, updated, and retrieved without violating any integrity in the data. This will help various stakeholders such as license offices to maintain this license data and the general public to retrieve this data.

2 USAGE OF DATABASE INSTEAD OF FILES

As noted in the introduction, we need to maintain integrity constraints throughout the data. Eg. NULL values for license numbers should be avoided. As the amount of data grows, it becomes difficult to manage, store, and retrieve the data from text files. Integrity constraints that check that two trades do not share the same license number can be effectively implemented in databases.

2.1 Data Organization

Data in databases are designed in a structural manner, making it easier to manage, store and retrieve data. As the volume of data increases, it becomes difficult to update and manage the data.

[*]UBIT:jonkick
[†]UBIT:mancysax

2.2 Data Integrity

The database is used to ensure data integrity. The objective is to make data consistent. Eg. null license values or nonintegers should not be permitted. Files are more susceptible to data corruption or loss as they lack built data validation and backup mechanism.

2.3 Avoid Redundancy

The introduction of primary keys is done in the database to ensure no two licenses have the same number. Since one license is given to each trader, we want to ensure that there is no data duplicity.

2.4 Scalability

The introduction of primary keys is done in As the data size grows, we may need multiple servers to store such data. The database will ensure that large amounts of data are stored across multiple servers while ensuring data integrity.

3 TARGET USER

There are mainly 2 type of users of this database. Stakeholders include license issuers that maintain this data by inserting, updating, deleting, retrieving values and occasionally altering the schema, adding new constraints, and adding/dropping columns. The other kind of user is the general public that can only retrieve this data to verify if a nursery is licensed or not, or to retrieve information about various nurseries. So to summarize there are two kind of users namely:

- licensing office: To update/insert/delete records of nursery growers
- A potential customer who wants to contact the nursery grower.

4 MILESTONE 1 DATABASE DESIGN

The initial Database had the following tables as per milestone 1

5 TABLE NORMALIZATION

We found that the operation_info table had 3 rows for opid and operation_name, namely:

- opid 1: nursery and greenhouse combined
- opid 2: nursery
- opid 3: greenhouse

Since opid 1 is cummulation of opid 2 and opid 3, therefore, opid 1 was removed and subsequently changes were made. In contact_info table, for rows with opid value = 1, were changed to multivalued attributes opid "2,3". This led to 1NF violation as 1NF requires that each attribute (column) in a table should hold only atomic (indivisible) values. The contact_info table was further decomposed with opid removed:

A new table consisting the relation opid_lid_info was implemented to fix 1NF violation

6 VERIFYING BCNF

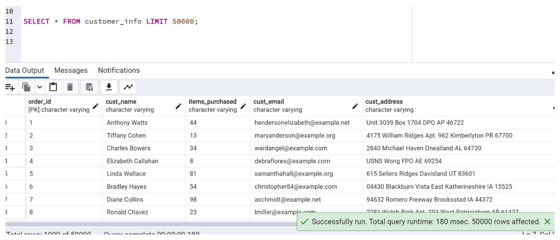
- city_info relation: Let city_id = A, city_name = B R = R(A,B) with A⟶B and B⟶A A+ = A,B and B+ = A,B Therefore, City is in BCNF.
- Contact relation: Contact(lid, business_phone, street_number, street_name, address_line_2, address_line_3, zipcode, city_id, state_id)

B ⟶ C, D, E, F, G, H, I, J D, E, F, G ⟶ H, I, J
C, D, E, F, G, H, I, J ⟶ B
B,+ = C, B, C, D, E, F, G, H, I, J - in BCNF
D, E, F, G+ = D, E, F, G, H, I, J, B- in BCNF
C, D, E, F, G, H, I, J+ = C, D, E, F, G, H, I, J, B - in BCNF
Therefore, this relation is in BCNF

- Contact relation: Contact(lid, business_phone, street_number, street_name, address_line_2, address_line_3, zipcode, city_id, state_id)
  B ⟶ C, D, E, F, G, H, I, J D, E, F, G ⟶ H, I, J
  C, D, E, F, G, H, I, J ⟶ B
  B,+ = C, B, C, D, E, F, G, H, I, J - in BCNF
  D, E, F, G+ = D, E, F, G, H, I, J, B- in BCNF
  C, D, E, F, G, H, I, J+ = C, D, E, F, G, H, I, J, B - in BCNF
  Therefore, this relation is in BCNF
- Operation_info relation: Operation_info(opid, operation_name)) A ⟶ B B ⟶ A A+ = A, B B+ = A, B Therefore, operation_info is in BCNF
- state_info relation: State_info(state_id, state) A ⟶ B B ⟶ A A+ = A, B B+ = B, A Therefore, state_info is in BCNF
- License_info relation: License_info(lid, license_num, owner_name, trade_name, expiration_year) B ⟶ C, D, E, A A ⟶ C, D, E C, D, E ⟶ B B+ = B, C, D, E, A - in BCNF A+ = A, C, D, E, B - in BCNF C, D, E+ = C, D, E, B, A - in BCNF Therefore, this relation is in BCNF
- Customer relation: Customer(order_id, cust_name, items_purchased, cust_email, cust_address, cust_phone, employees_needed)
  A ⟶ B, C, D, E, F, G B, C ⟶ D, E, F, G, A A+ = A, B, C, D, E, F, G - in BCNF B, C+ = B, C, D, E, F, G, A - in BCNF
  Therefore, this relation is in BCNF

7 INDEXING

The customers purchasing from licensed growers will grow exponentially compared to the licensed growers. Therefore, in our DB, we have created a table customer_info consisting of information regarding the customer. We have generated 50,000 rows for customers. Without indexing the primary key, i.e. order_id the retrieval was 190ms. After adding indexing to the attribute order_id, it was found that the execution time for retrieval reduced to 89ms, reducing it to half the speed. The following screenshots are before and after indexing is implemented. The data is stored such that it results in faster data retrieval and reduced I//O time.



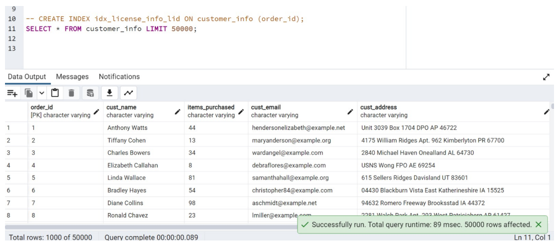
```sql
SELECT * FROM customer_info LIMIT 50000;
```

	order_id [PK] character varying	cust_name character varying	items_purchased character varying	cust_email character varying	cust_address character varying
1	1	Anthony Watts	44	hendersonelizabeth@example.net	Unit 3039 Box 1704 DPO AP 46722
2	2	Tiffany Cohen	13	maryanderson@example.org	4175 William Ridges Apt. 962 Kimberlyton PR 67700
3	3	Charles Bowers	34	wardangel@example.com	2840 Michael Haven Onealland AL 64730
4	4	Elizabeth Callahan	8	debraflores@example.com	USNS Wong FPO AE 69254
5	5	Linda Wallace	81	samanthahall@example.org	615 Sellers Ridges Davisland UT 83601
6	6	Bradley Hayes	54	christopher84@example.com	04430 Blackburn Vista East Katherineshire IA 15525
7	7	Diane Collins	98	aschmidt@example.net	94632 Romero Freeway Brooksstad IA 44372
8	8	Ronald Chavez	23	lmiller@examp	2281 Walsh Park Apt. 293 West Patriciaborg AP 61427

✓ Successfully run. Total query runtime: 180 msec. 50000 rows affected.

Total rows: 1000 of 50000   Query complete 00:00:00.180   Ln 7, Col

Fig. 1. Before Indexing : Execution Time : 180ms



```sql
-- CREATE INDEX idx_license_info_lid ON customer_info (order_id);
SELECT * FROM customer_info LIMIT 50000;
```

	order_id [PK] character varying	cust_name character varying	items_purchased character varying	cust_email character varying	cust_address character varying
1	1	Anthony Watts	44	hendersonelizabeth@example.net	Unit 3039 Box 1704 DPO AP 46722
2	2	Tiffany Cohen	13	maryanderson@example.org	4175 William Ridges Apt. 962 Kimberlyton PR 67700
3	3	Charles Bowers	34	wardangel@example.com	2840 Michael Haven Onealland AL 64730
4	4	Elizabeth Callahan	8	debraflores@example.com	USNS Wong FPO AE 69254
5	5	Linda Wallace	81	samanthahall@example.org	615 Sellers Ridges Davisland UT 83601
6	6	Bradley Hayes	54	christopher84@example.com	04430 Blackburn Vista East Katherineshire IA 15525
7	7	Diane Collins	98	aschmidt@example.net	94632 Romero Freeway Brooksstad IA 44372
8	8	Ronald Chavez	23	lmiller@example	2281 Walsh Park Apt. 293 West Patriciaborg AP 61427

✓ Successfully run. Total query runtime: 89 msec. 50000 rows affected.

Total rows: 1000 of 50000   Query complete 00:00:00.089   Ln 11, Col 1

Fig. 2. After Indexing : Execution Time : 89ms

8 ER DIAGRAM

There are 7 tables in our database. 6 of these tables are decomposed from the main table horticulture_master table.

city_info	contact
customer_info	license_info
operation_info	opid_lid_info
state_info	

Attribute Name	Is PK	Is FK
lid	Yes	No
owner_name	No	No
trade_name	No	No

Attribute Name	Is PK	Is FK
order_id	Yes	No
cust_name	No	No
cust_phone	No	No
items_purchased	No	No
cust_email	No	No
cust_address	No	No

Table 1. Table Name: customer_info

Attribute Name	Is PK	Is FK
lid	Yes	Yes license_info(lid)
business_phone	No	No
street_number	No	No
street_name	No	No
address_line_2	No	No
address_line_3	No	No
zipcode	No	No
city_id	No	Yes city_info(city_id)
state_id	No	Yes state_info(state_id)

Table 2. Table Name: contact_info

Attribute Name	Is PK	Is FK
city_id	Yes	No
city_name	No	No

Table 3. Table Name: city_info

Attribute Name	Is PK	Is FK
state_id	Yes	No
state_name	No	No

Table 4. Table Name: state_info

Attribute Name	Is PK	Is FK
operation_id	Yes	No
operation_name	No	No

Table 5. Table Name: operation_info

Attribute Name	Is PK	Is FK
operation_id	Yes	No
lid_name	No	No

Table 6. Table Name: opid_lid_info

All the tables are composed from a master table except for customer_info table, that is generated using Python's faker library. The DB contains 50k records of that table. Other table have been decomposed from horticulture_master table. The primary keys are mentioned in the table above, along with the foreign keys. Below is the schema for the database.

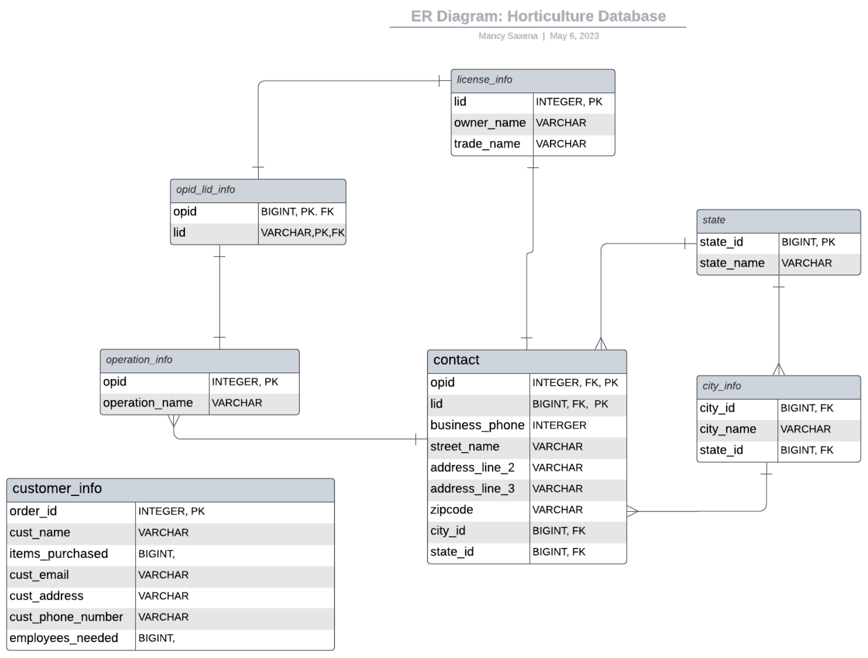


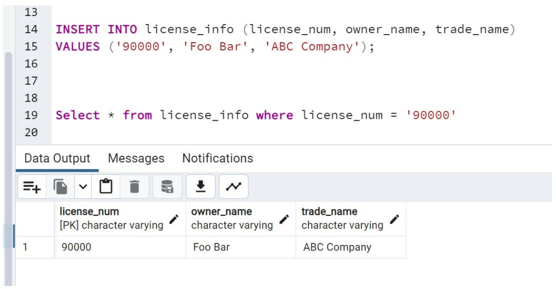
Fig. 3. Horticulture Database Schema

9 QUERIES

The following queries were run successfully in the PGADMIN:

- insert into license_info(license_num, owner_name, trade_name) values (’90000’, ’Foo Bar’, ’ABC Company’)
- insert into contact (lid,business_phone,street_number,street_name, adressline_2,adressline_3,zipcode,city_id,state_id) values (’90000’,’1234567890’,’123’,’Main Street’,”,”,’12345’,1,1)
- update contact set street_number = ’999’
- select * from license_info where license_num = ’90000’
- delete from opid_lid where lid = ’100047’
- select c.lid, c.business_phone, ci.city_name from contact c join city_info ci on c.city_id = ci.city_id where c.city_id = 644
- select c.state_id, s.state, count(distinct c.lid) as state_count from contact c join state_info s on c.state_id = s.state_id group by c.state_id , s.state order by state_count desc
- select l.license_num, ci.city_name from license_info l join contact co on l.license_num = co.lid join city_info ci on co.city_id = ci.city_id where l.license_num in (select license_num from license_info where cast(license_num AS BIGINT) > 100600 limit 10)

- select ci.city_name, count(*) AS contact_count from contact co join city_info ci ON co.city_id = ci.city_id group by ci.city_name having count (*) > 5 order by contact_count desc;

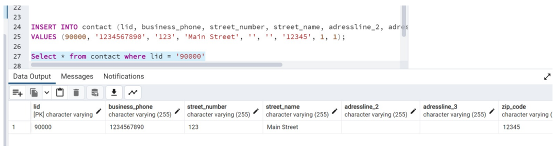


```
INSERT INTO license_info (license_num, owner_name, trade_name)
VALUES ('90000', 'Foo Bar', 'ABC Company');


Select * from license_info where license_num = '90000'
```

	license_num [PK] character varying	owner_name character varying	trade_name character varying
1	90000	Foo Bar	ABC Company

Fig. 4. Inserting Values

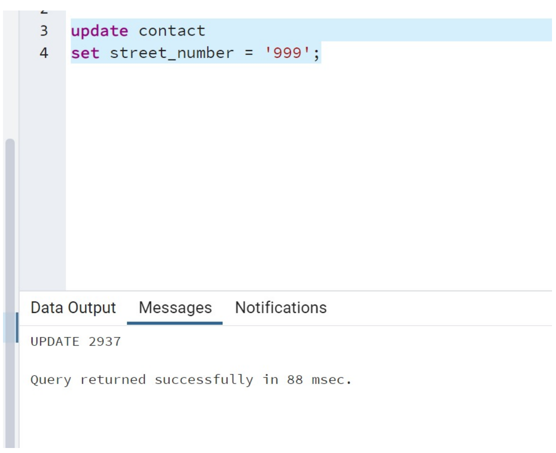


```
INSERT INTO contact (lid, business_phone, street_number, street_name, adressline_2, adres
VALUES (90000, '1234567890', '123', 'Main Street', '', '', '12345', 1, 1);

Select * from contact where lid = '90000'
```

	lid [PK] character varying	business_phone character varying (255)	street_number character varying (255)	street_name character varying (255)	adressline_2 character varying (255)	adressline_3 character varying (255)	zip_code character varying (
1	90000	1234567890	123	Main Street			12345

Fig. 5. Inserting Values into Contact

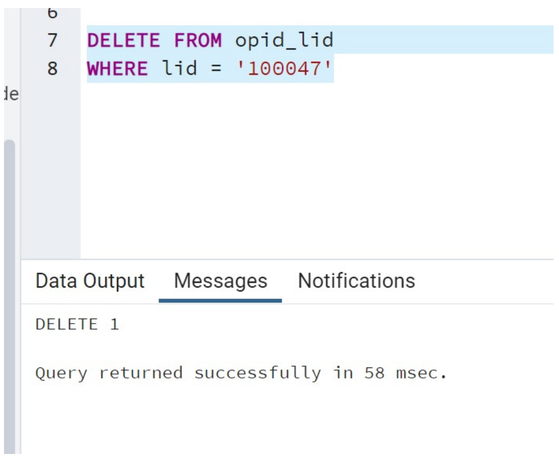


```
update contact
set street_number = '999';
```

```
UPDATE 2937

Query returned successfully in 88 msec.
```

Fig. 6. Updating Values

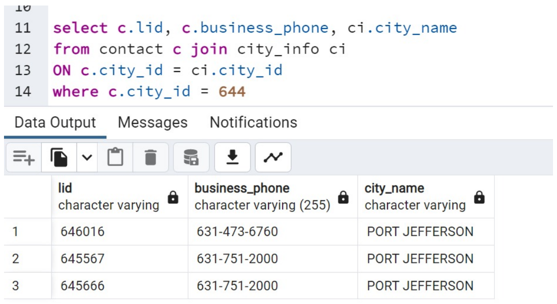


```
DELETE FROM opid_lid
WHERE lid = '100047'
```

```
DELETE 1

Query returned successfully in 58 msec.
```

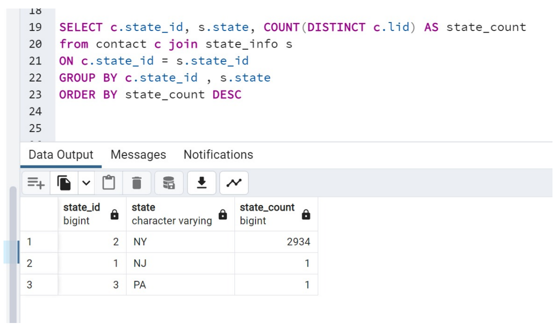
Fig. 7. Deleting Values into Contact



```
select c.lid, c.business_phone, ci.city_name
from contact c join city_info ci
ON c.city_id = ci.city_id
where c.city_id = 644
```

	lid character varying	business_phone character varying (255)	city_name character varying
1	646016	631-473-6760	PORT JEFFERSON
2	645567	631-751-2000	PORT JEFFERSON
3	645666	631-751-2000	PORT JEFFERSON

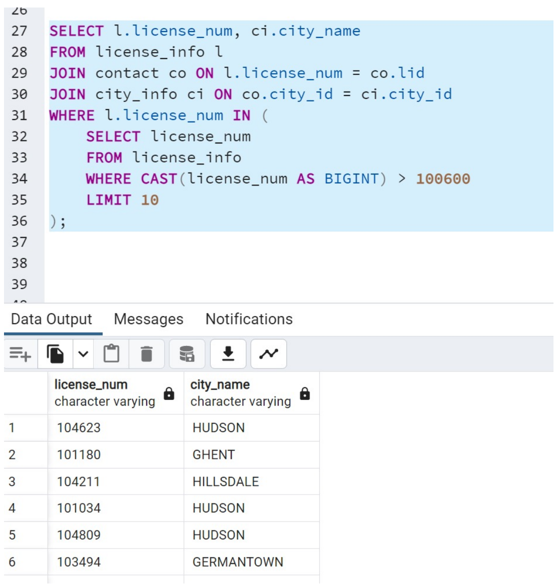
Fig. 8. Select with Join



```
SELECT c.state_id, s.state, COUNT(DISTINCT c.lid) AS state_count
from contact c join state_info s
ON c.state_id = s.state_id
GROUP BY c.state_id , s.state
ORDER BY state_count DESC
```

	state_id bigint	state character varying	state_count bigint
1	2	NY	2934
2	1	NJ	1
3	3	PA	1

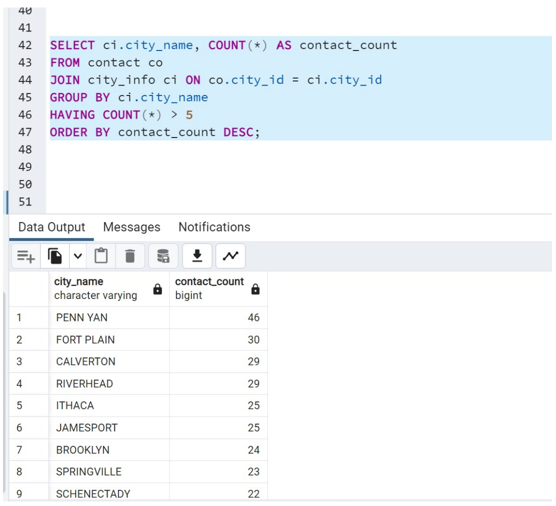
Fig. 9. Groupby and Join



```
SELECT l.license_num, ci.city_name
FROM license_info l
JOIN contact co ON l.license_num = co.lid
JOIN city_info ci ON co.city_id = ci.city_id
WHERE l.license_num IN (
    SELECT license_num
    FROM license_info
    WHERE CAST(license_num AS BIGINT) > 100600
    LIMIT 10
);
```

Data Output Messages Notifications

	license_num character varying	city_name character varying
1	104623	HUDSON
2	101180	GHENT
3	104211	HILLSDALE
4	101034	HUDSON
5	104809	HUDSON
6	103494	GERMANTOWN



```
SELECT ci.city_name, COUNT(*) AS contact_count
FROM contact co
JOIN city_info ci ON co.city_id = ci.city_id
GROUP BY ci.city_name
HAVING COUNT(*) > 5
ORDER BY contact_count DESC;
```

Data Output Messages Notifications

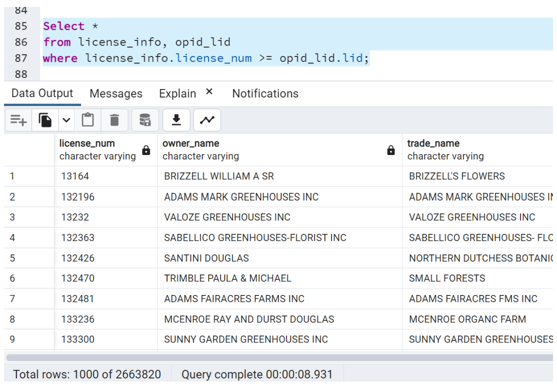
	city_name character varying	contact_count bigint
1	PENN YAN	46
2	FORT PLAIN	30
3	CALVERTON	29
4	RIVERHEAD	29
5	ITHACA	25
6	JAMESPORT	25
7	BROOKLYN	24
8	SPRINGVILLE	23
9	SCHENECTADY	22

Fig. 10. Groupby and Join

10 QUERY OPTIMIZATIONS

There are 3 queries that were optimized majorly.

- Selecting all columns instead of required columns can delay the execution of the query
  Inefficient query : Select * from license_info, customer where license_info.employee_count >= customer.employees_needed;
  Efficient Query : Here we only project the columns that are required by us Select license_info.trade_name from license_info inner join customer on license_info.employee_count >= customer.employees_needed; Here execution time reduces to 3s from 8s on optimizing query



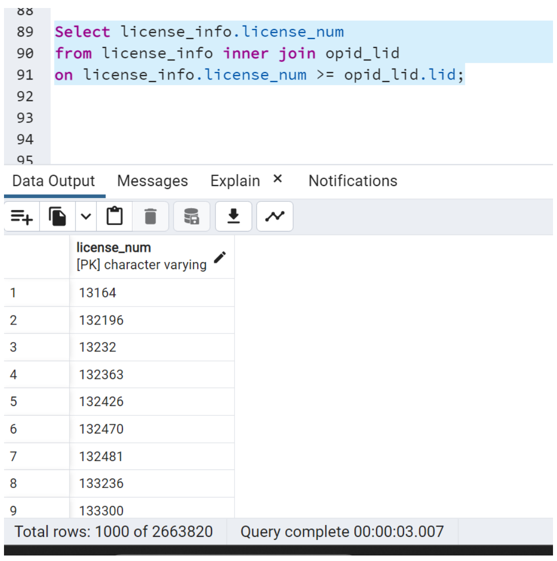
```
Select *
from license_info, opid_lid
where license_info.license_num >= opid_lid.lid;
```

Data Output Messages Explain × Notifications

	license_num character varying	owner_name character varying	trade_name character varying
1	13164	BRIZZELL WILLIAM A SR	BRIZZELL'S FLOWERS
2	132196	ADAMS MARK GREENHOUSES INC	ADAMS MARK GREENHOUSES II
3	13232	VALOZE GREENHOUSES INC	VALOZE GREENHOUSES INC
4	132363	SABELLICO GREENHOUSES-FLORIST INC	SABELLICO GREENHOUSES- FLC
5	132426	SANTINI DOUGLAS	NORTHERN DUTCHESS BOTANI(
6	132470	TRIMBLE PAULA & MICHAEL	SMALL FORESTS
7	132481	ADAMS FAIRACRES FARMS INC	ADAMS FAIRACRES FMS INC
8	133236	MCENROE RAY AND DURST DOUGLAS	MCENROE ORGANC FARM
9	133300	SUNNY GARDEN GREENHOUSES INC	SUNNY GARDEN GREENHOUSES

Total rows: 1000 of 2663820 Query complete 00:00:08.931

Fig. 11. 8s execution time before optimization



```
Select license_info.license_num
from license_info inner join opid_lid
on license_info.license_num >= opid_lid.lid;
```

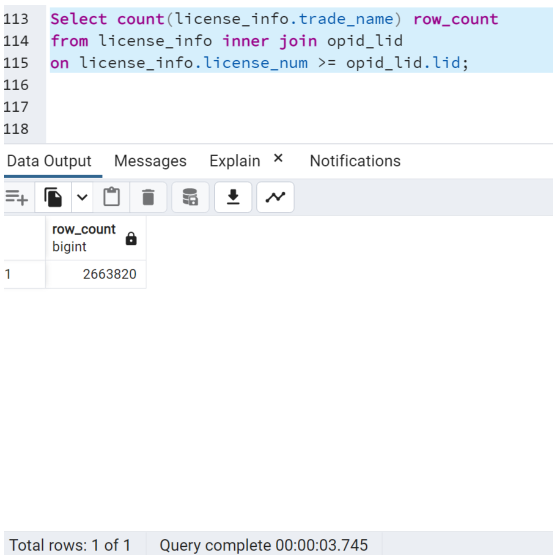
Data Output Messages Explain × Notifications

	license_num [PK] character varying
1	13164
2	132196
3	13232
4	132363
5	132426
6	132470
7	132481
8	133236
9	133300

Total rows: 1000 of 2663820 Query complete 00:00:03.007

Fig. 12. 3s execution time after optimization

- Similarly reducing the extra columns when only count has to be determined will make execution faster. Inefficient query 2: Select count(*) row_count from license_info, customer where license_info.employee_count >= customer.employees_needed;
  An efficient version of query 2: Select count(license_info.trade_name) row_count from license_info inner join customer on license_info.employee_count >= customer.employees_needed; Below are the execution time for same logic

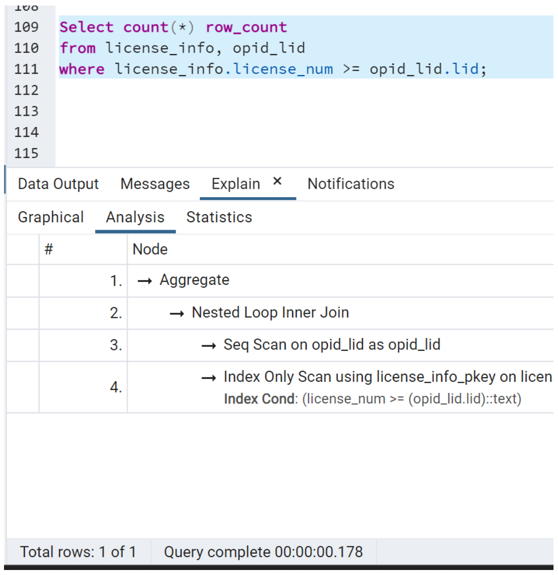


```
Select count(license_info.trade_name) row_count
from license_info inner join opid_lid
on license_info.license_num >= opid_lid.lid;
```

	row_count bigint
1	2663820

Total rows: 1 of 1   Query complete 00:00:03.745

Fig. 13. 8s execution time before optimization



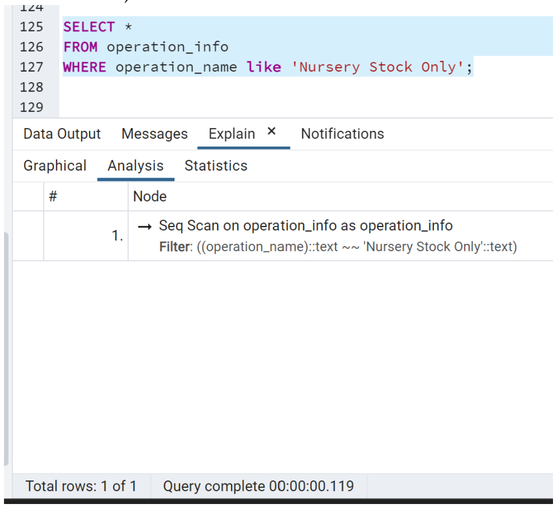
```
Select count(*) row_count
from license_info, opid_lid
where license_info.license_num >= opid_lid.lid;
```

#	Node
1.	→ Aggregate
2.	→ Nested Loop Inner Join
3.	→ Seq Scan on opid_lid as opid_lid
4.	→ Index Only Scan using license_info_pkey on licen Index Cond: (license_num >= (opid_lid.lid)::text)

Total rows: 1 of 1   Query complete 00:00:00.178

Fig. 14. 3s execution time after optimization

- Using equal to operator (=) instead of like for pattern matching can drastically reduce search time (retrieval time)

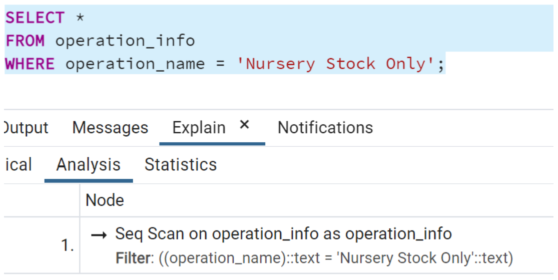


```
SELECT *
FROM operation_info
WHERE operation_name like 'Nursery Stock Only';
```

#	Node
1.	→ Seq Scan on operation_info as operation_info Filter: ((operation_name)::text ~~ 'Nursery Stock Only'::text)

Total rows: 1 of 1   Query complete 00:00:00.119

Fig. 15. 119ms execution time before optimization



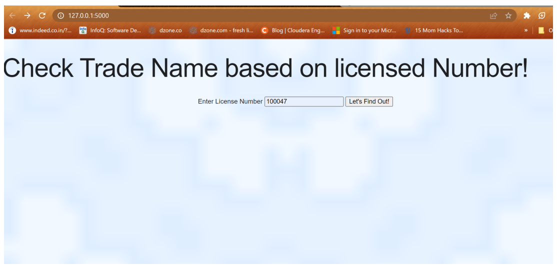
```
SELECT *
FROM operation_info
WHERE operation_name = 'Nursery Stock Only';
```

	Node
1.	→ Seq Scan on operation_info as operation_info Filter: ((operation_name)::text = 'Nursery Stock Only'::text)

Fig. 16. 31ms execution time after optimization

11 BONUS TASK

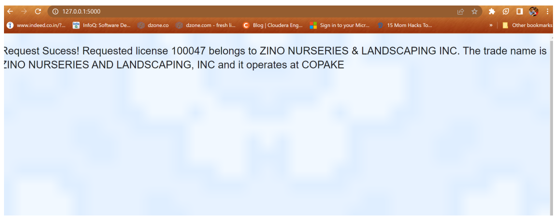
We implemented a website using pgadmin and flask, where in we performed a simple retreival of information specific to a license_id



Check Trade Name based on licensed Number!

Enter License Number 100047

Fig. 17. Enter license to retrieve



Request Sucess! Requested license 100047 belongs to ZINO NURSERIES & LANDSCAPING INC. The trade name is ZINO NURSERIES AND LANDSCAPING, INC and it operates at COPAKE

Fig. 18. Get License, city, trade info

On searching via license_id, we can see if it is present in the database, we will get a success page, else we will get license_not_found page.

12 CONTRIBUTIONS

- Jon Kick: Dataset collection, BCNF Performed, Populated customer_info table to 10000 rows, Report Milestone 1, Query Optimization via explain, Sample Query execution in Milestone1, Created Initial DB in Milestone 1, ER Diagram Milestone 2, bonus task :flask+pgadmin
- Mancy Saxena: Populated extra customer_info dataset to 50,000 rows, Normalized DB in SQL files, Report Milestone 2, Queries for Milestone 2, ER Diagram Milestone 1