Database Management Systems - UE19CS301 Assignment-4

Date: 6/12/2021
Section: H

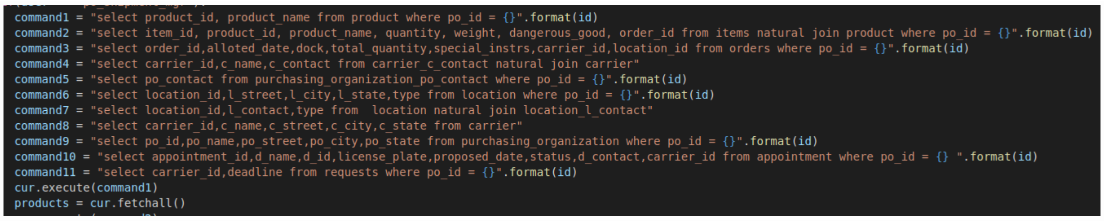
Sumukh Raju Bhat	PES1UG19CS519
Shreyas VBKS	PES1UG19CS469
Sowmya M	PES1UG19CS497

1. Database connectivity:

We use html+bootstrap css as our frontend and flask+postgresqlDB as our backend. We used psycopg2 in python3 for our database connectivity. Psycopg is the most popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety (several threads can share the same connection). It was designed for heavily multi-threaded applications that create and destroy lots of cursors and make a large number of concurrent "INSERT"s or "UPDATE"s.

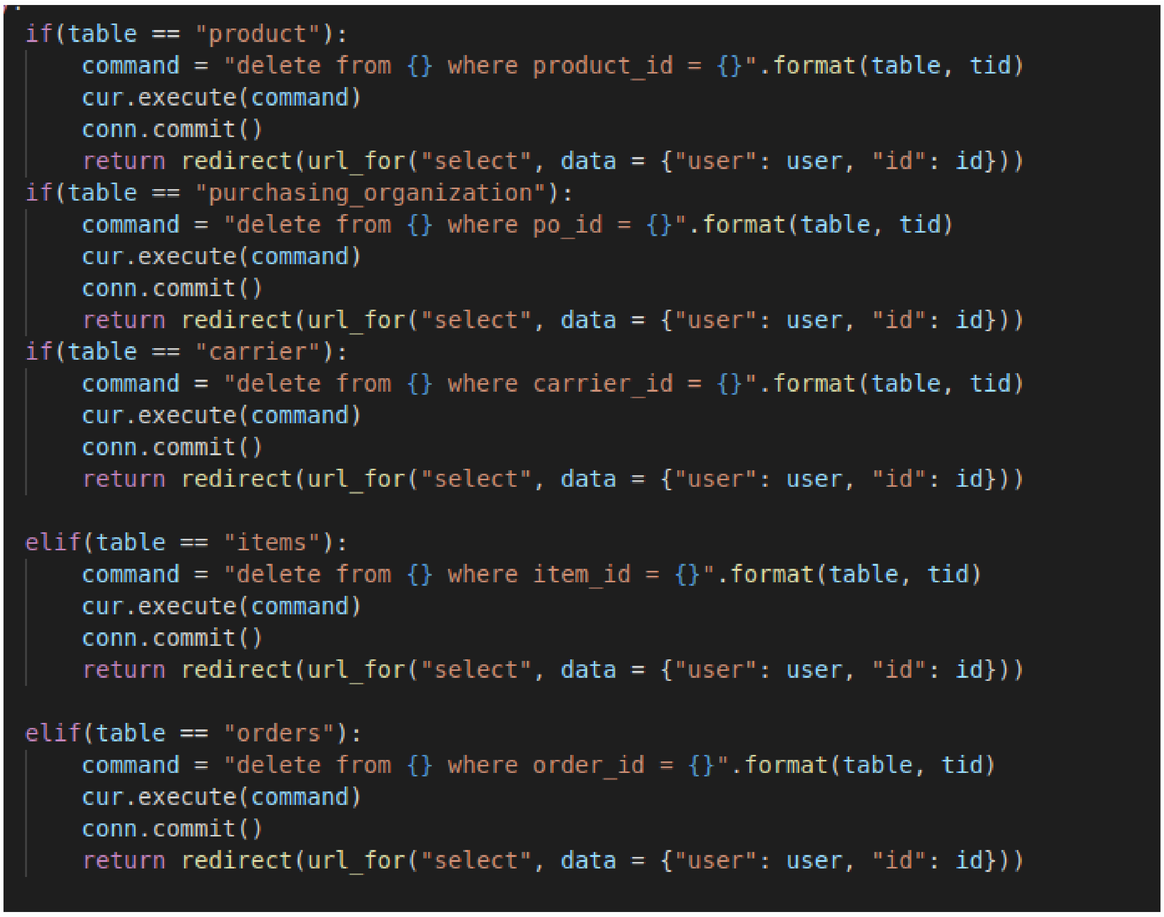
2. Queries from frontend:

a. Some of the simple queries:



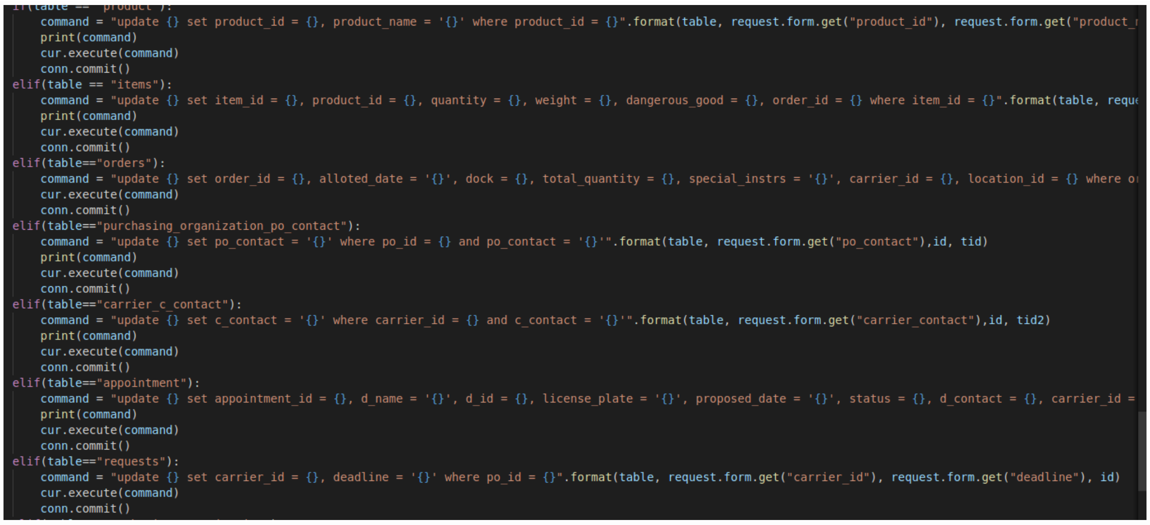
```
command1 = "select product_id, product_name from product where po_id = {}".format(id)
command2 = "select item_id, product_id, product_name, quantity, weight, dangerous_good, order_id from items natural join product where po_id = {}".format(id)
command3 = "select order_id,alloted_date,dock,total_quantity,special_instrs,carrier_id,location_id from orders where po_id = {}".format(id)
command4 = "select carrier_id,c_name,c_contact from carrier_c_contact natural join carrier"
command5 = "select po_contact from purchasing_organization_po_contact where po_id = {}".format(id)
command6 = "select location_id,l_street,l_city,l_state,type from location where po_id = {}".format(id)
command7 = "select location_id,l_contact,type from  location natural join location_l_contact"
command8 = "select carrier_id,c_name,c_street,c_city,c_state from carrier"
command9 = "select po_id,po_name,po_street,po_city,po_state from purchasing_organization where po_id = {}".format(id)
command10 = "select appointment_id,d_name,d_id,license_plate,proposed_date,status,d_contact,carrier_id from appointment where po_id = {} ".format(id)
command11 = "select carrier_id,deadline from requests where po_id = {}".format(id)
cur.execute(command1)
products = cur.fetchall()
```

b. Some of the delete queries:



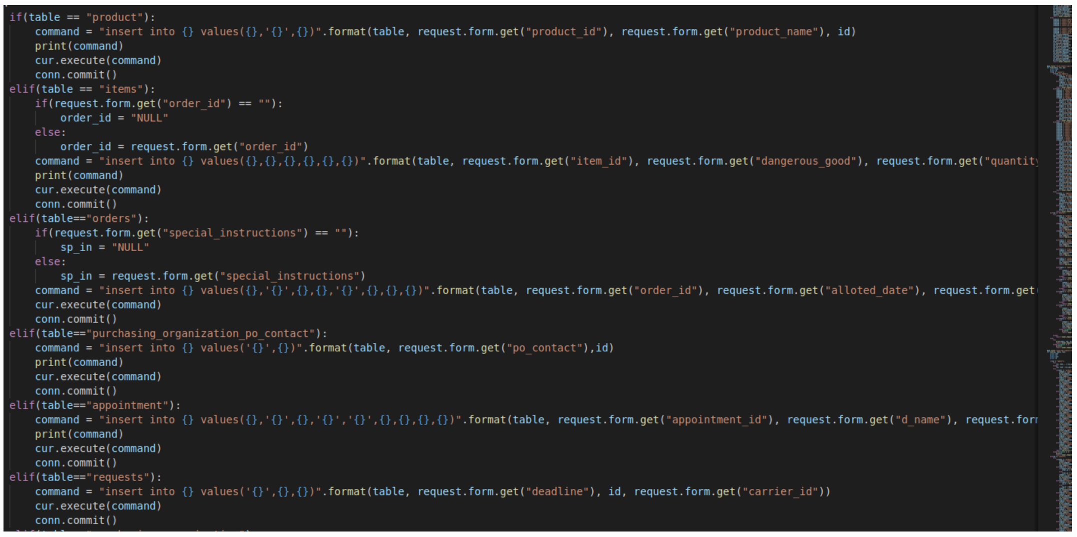
```
if(table == "product"):
    command = "delete from {} where product_id = {}".format(table, tid)
    cur.execute(command)
    conn.commit()
    return redirect(url_for("select", data = {"user": user, "id": id}))
if(table == "purchasing_organization"):
    command = "delete from {} where po_id = {}".format(table, tid)
    cur.execute(command)
    conn.commit()
    return redirect(url_for("select", data = {"user": user, "id": id}))
if(table == "carrier"):
    command = "delete from {} where carrier_id = {}".format(table, tid)
    cur.execute(command)
    conn.commit()
    return redirect(url_for("select", data = {"user": user, "id": id}))

elif(table == "items"):
    command = "delete from {} where item_id = {}".format(table, tid)
    cur.execute(command)
    conn.commit()
    return redirect(url_for("select", data = {"user": user, "id": id}))

elif(table == "orders"):
    command = "delete from {} where order_id = {}".format(table, tid)
    cur.execute(command)
    conn.commit()
    return redirect(url_for("select", data = {"user": user, "id": id}))
```

c. Some of the update queries:



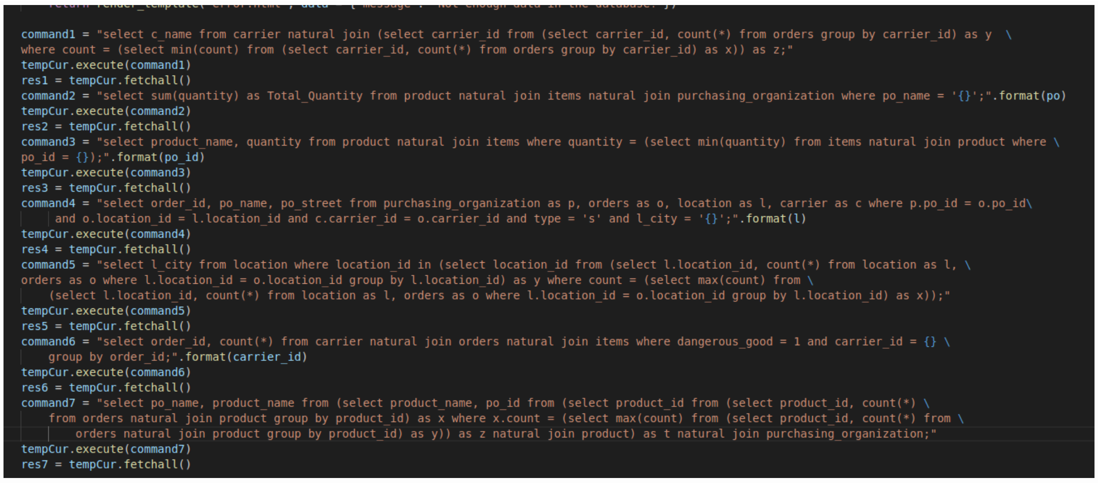
```
if(table == "product"):
    command = "update {} set product_id = {}, product_name = '{}' where product_id = {}".format(table, request.form.get("product_id"), request.form.get("product_
    print(command)
    cur.execute(command)
    conn.commit()
elif(table == "items"):
    command = "update {} set item_id = {}, product_id = {}, quantity = {}, weight = {}, dangerous_good = {}, order_id = {} where item_id = {}".format(table, reque
    print(command)
    cur.execute(command)
    conn.commit()
elif(table=="orders"):
    command = "update {} set order_id = {}, alloted_date = '{}', dock = {}, total_quantity = {}, special_instrs = '{}', carrier_id = {}, location_id = {} where o
    cur.execute(command)
    conn.commit()
elif(table=="purchasing_organization_po_contact"):
    command = "update {} set po_contact = '{}' where po_id = {} and po_contact = '{}'".format(table, request.form.get("po_contact"),id, tid)
    print(command)
    cur.execute(command)
    conn.commit()
elif(table=="carrier_c_contact"):
    command = "update {} set c_contact = '{}' where carrier_id = {} and c_contact = '{}'".format(table, request.form.get("carrier_contact"),id, tid2)
    print(command)
    cur.execute(command)
    conn.commit()
elif(table=="appointment"):
    command = "update {} set appointment_id = {}, d_name = '{}', d_id = {}, license_plate = '{}', proposed_date = '{}', status = {}, d_contact = {}, carrier_id =
    print(command)
    cur.execute(command)
    conn.commit()
elif(table=="requests"):
    command = "update {} set carrier_id = {}, deadline = '{}' where po_id = {}".format(table, request.form.get("carrier_id"), request.form.get("deadline"), id)
    cur.execute(command)
    conn.commit()
```

d. Some of the insert queries:



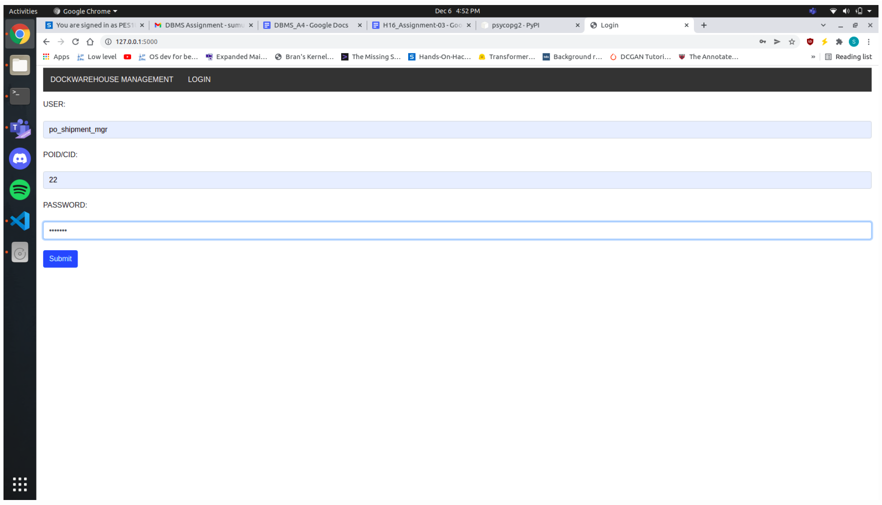
```python
if(table == "product"):
    command = "insert into {} values({},'{}',{})".format(table, request.form.get("product_id"), request.form.get("product_name"), id)
    print(command)
    cur.execute(command)
    conn.commit()
elif(table == "items"):
    if(request.form.get("order_id") == ""):
        order_id = "NULL"
    else:
        order_id = request.form.get("order_id")
    command = "insert into {} values({},{},{},{},{},{})".format(table, request.form.get("item_id"), request.form.get("dangerous_good"), request.form.get("quantity
    print(command)
    cur.execute(command)
    conn.commit()
elif(table=="orders"):
    if(request.form.get("special_instructions") == ""):
        sp_in = "NULL"
    else:
        sp_in = request.form.get("special_instructions")
    command = "insert into {} values({},'{}',{},{},'{}',{},{},{})".format(table, request.form.get("order_id"), request.form.get("alloted_date"), request.form.get
    cur.execute(command)
    conn.commit()
elif(table=="purchasing_organization_po_contact"):
    command = "insert into {} values('{}',{})".format(table, request.form.get("po_contact"),id)
    print(command)
    cur.execute(command)
    conn.commit()
elif(table=="appointment"):
    command = "insert into {} values({},'{}',{},'{}','{}',{},{},{},{})".format(table, request.form.get("appointment_id"), request.form.get("d_name"), request.forr
    print(command)
    cur.execute(command)
    conn.commit()
elif(table=="requests"):
    command = "insert into {} values('{}',{},{})".format(table, request.form.get("deadline"), id, request.form.get("carrier_id"))
    cur.execute(command)
    conn.commit()
```

e. Some of the complex queries:



```python
command1 = "select c_name from carrier natural join (select carrier_id from (select carrier_id, count(*) from orders group by carrier_id) as y  \
where count = (select min(count) from (select carrier_id, count(*) from orders group by carrier_id) as x)) as z;"
tempCur.execute(command1)
res1 = tempCur.fetchall()
command2 = "select sum(quantity) as Total_Quantity from product natural join items natural join purchasing_organization where po_name = '{}';".format(po)
tempCur.execute(command2)
res2 = tempCur.fetchall()
command3 = "select product_name, quantity from product natural join items where quantity = (select min(quantity) from items natural join product where \
po_id = {});".format(po_id)
tempCur.execute(command3)
res3 = tempCur.fetchall()
command4 = "select order_id, po_name, po_street from purchasing_organization as p, orders as o, location as l, carrier as c where p.po_id = o.po_id\
     and o.location_id = l.location_id and c.carrier_id = o.carrier_id and type = 's' and l_city = '{}';".format(l)
tempCur.execute(command4)
res4 = tempCur.fetchall()
command5 = "select l_city from location where location_id in (select location_id from (select l.location_id, count(*) from location as l, \
orders as o where l.location_id = o.location_id group by l.location_id) as y where count = (select max(count) from \
    (select l.location_id, count(*) from location as l, orders as o where l.location_id = o.location_id group by l.location_id) as x));"
tempCur.execute(command5)
res5 = tempCur.fetchall()
command6 = "select order_id, count(*) from carrier natural join orders natural join items where dangerous_good = 1 and carrier_id = {} \
    group by order_id;".format(carrier_id)
tempCur.execute(command6)
res6 = tempCur.fetchall()
command7 = "select po_name, product_name from (select product_name, po_id from (select product_id from (select product_id, count(*) \
    from orders natural join product group by product_id) as x where x.count = (select max(count) from (select product_id, count(*) from \
        orders natural join product group by product_id) as y)) as z natural join product) as t natural join purchasing_organization;"
tempCur.execute(command7)
res7 = tempCur.fetchall()
```

f. Resultant frontend:



DOCKWAREHOUSE MANAGEMENT LOGIN

USER:

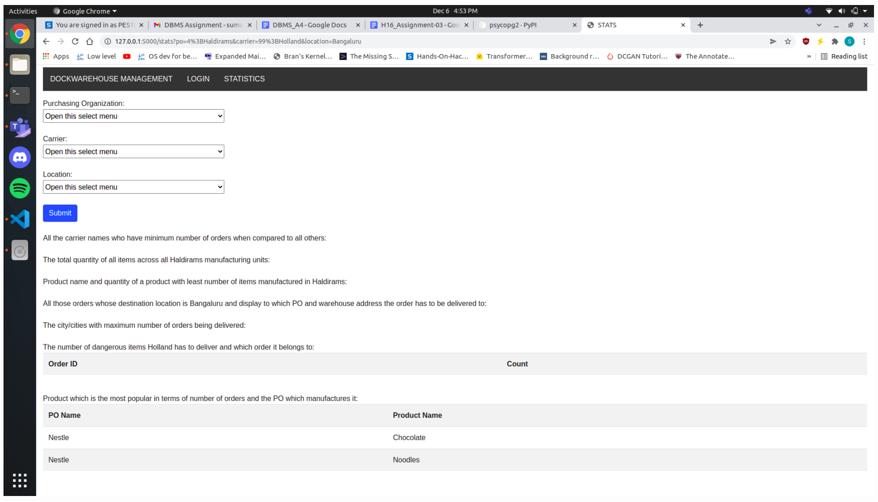
po_shipment_mgr

POID/CID:

22

PASSWORD:

•••••••


DOCKWAREHOUSE MANAGEMENT LOGIN STATISTICS

Purchasing Organization:
Open this select menu

Carrier:
Open this select menu

Location:
Open this select menu

Submit

All the carrier names who have minimum number of orders when compared to all others:

The total quantity of all items across all Haldirams manufacturing units:

Product name and quantity of a product with least number of items manufactured in Haldirams:

All those orders whose destination location is Bangaluru and display to which PO and warehouse address the order has to be delivered to:

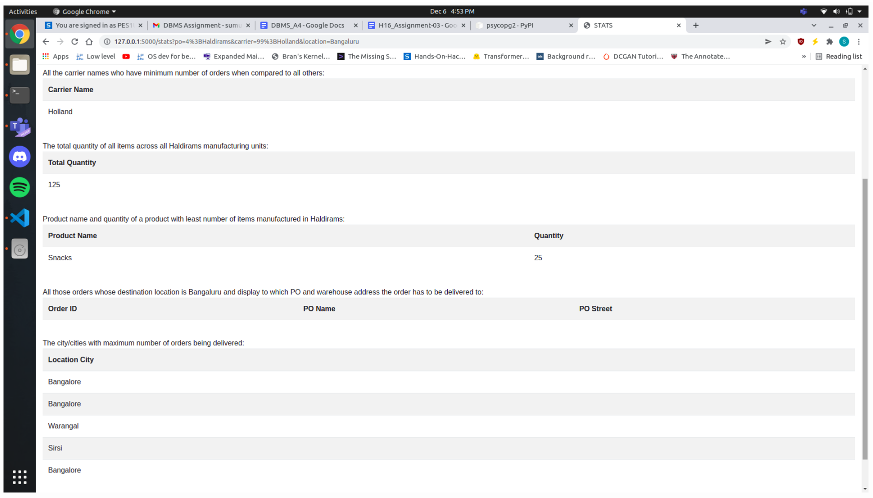
The city/cities with maximum number of orders being delivered:

The number of dangerous items Holland has to deliver and which order it belongs to:

Order ID	Count

Product which is the most popular in terms of number of orders and the PO which manufactures it:

PO Name	Product Name
Nestle	Chocolate
Nestle	Noodles


All the carrier names who have minimum number of orders when compared to all others:

Carrier Name
Holland

The total quantity of all items across all Haldirams manufacturing units:

Total Quantity
125

Product name and quantity of a product with least number of items manufactured in Haldirams:

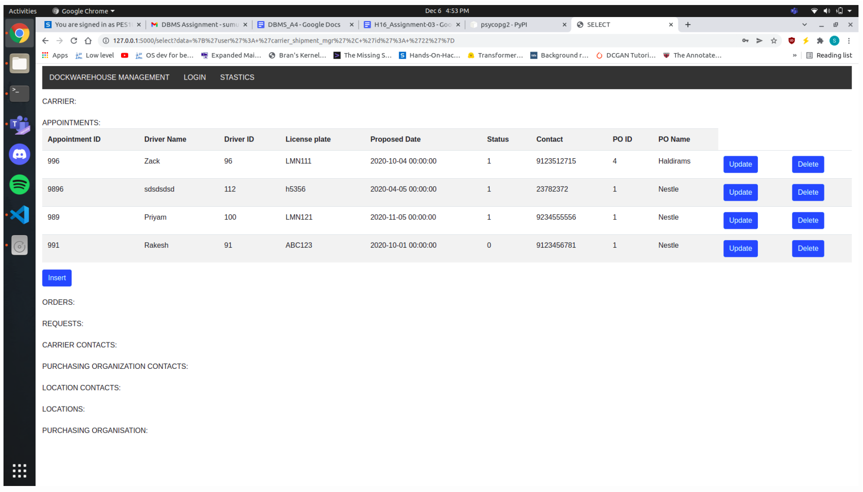
Product Name	Quantity
Snacks	25

All those orders whose destination location is Bangaluru and display to which PO and warehouse address the order has to be delivered to:

Order ID	PO Name	PO Street

The city/cities with maximum number of orders being delivered:

Location City
Bangalore
Bangalore
Warangal
Sirsi
Bangalore


DOCKWAREHOUSE MANAGEMENT LOGIN STASTICS

CARRIER:

APPOINTMENTS:

Appointment ID	Driver Name	Driver ID	License plate	Proposed Date	Status	Contact	PO ID	PO Name		
996	Zack	96	LMN111	2020-10-04 00:00:00	1	9123512715	4	Haldirams	Update	Delete
9896	sdsdsdsd	112	h5356	2020-04-05 00:00:00	1	23782372	1	Nestle	Update	Delete
989	Priyam	100	LMN121	2020-11-05 00:00:00	1	9234555556	1	Nestle	Update	Delete
991	Rakesh	91	ABC123	2020-10-01 00:00:00	0	9123456781	1	Nestle	Update	Delete

Insert

ORDERS:

REQUESTS:

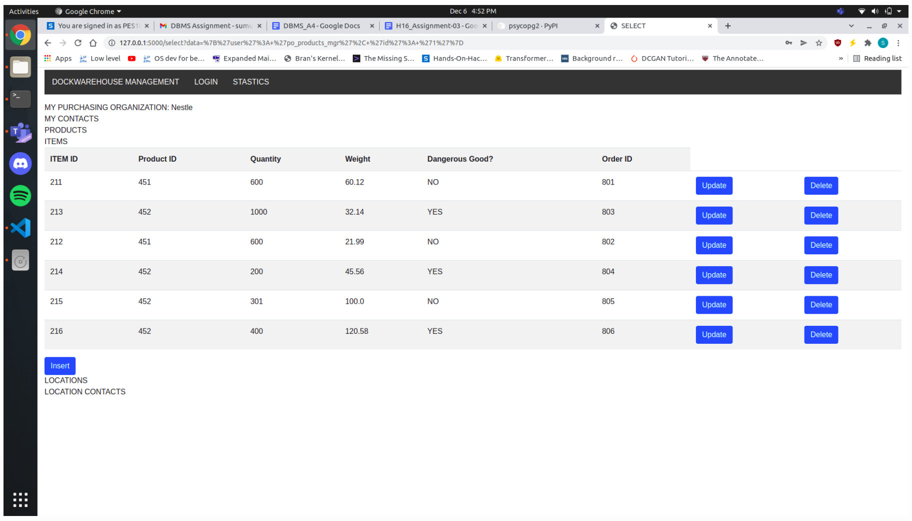
CARRIER CONTACTS:

PURCHASING ORGANIZATION CONTACTS:

LOCATION CONTACTS:

LOCATIONS:

PURCHASING ORGANISATION:

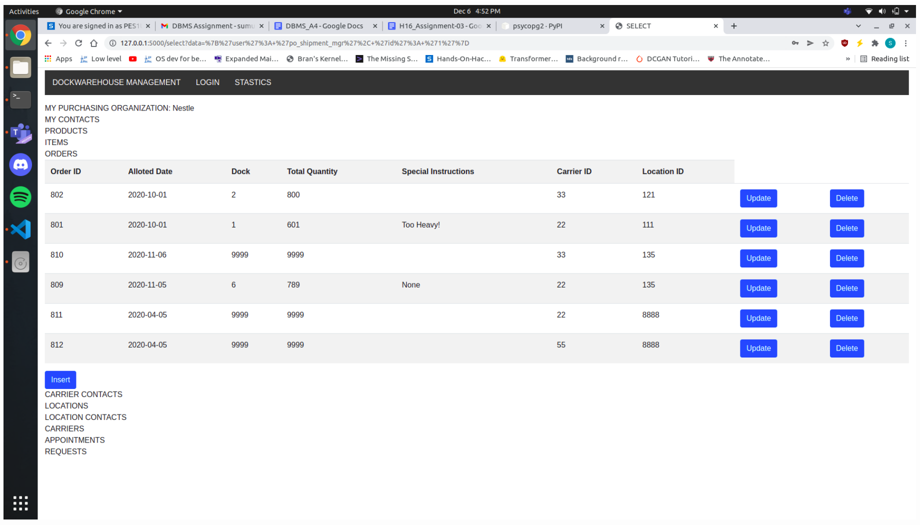


DOCKWAREHOUSE MANAGEMENT LOGIN STASTICS

MY PURCHASING ORGANIZATION: Nestle
MY CONTACTS
PRODUCTS
ITEMS

ITEM ID	Product ID	Quantity	Weight	Dangerous Good?	Order ID		
211	451	600	60.12	NO	801	Update	Delete
213	452	1000	32.14	YES	803	Update	Delete
212	451	600	21.99	NO	802	Update	Delete
214	452	200	45.56	YES	804	Update	Delete
215	452	301	100.0	NO	805	Update	Delete
216	452	400	120.58	YES	806	Update	Delete

Insert
LOCATIONS
LOCATION CONTACTS



DOCKWAREHOUSE MANAGEMENT LOGIN STASTICS

MY PURCHASING ORGANIZATION: Nestle
MY CONTACTS
PRODUCTS
ITEMS
ORDERS

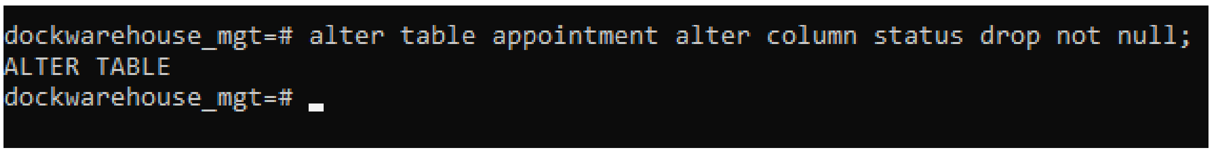
Order ID	Alloted Date	Dock	Total Quantity	Special Instructions	Carrier ID	Location ID		
802	2020-10-01	2	800		33	121	Update	Delete
801	2020-10-01	1	601	Too Heavy!	22	111	Update	Delete
810	2020-11-06	9999	9999		33	135	Update	Delete
809	2020-11-05	6	789	None	22	135	Update	Delete
811	2020-04-05	9999	9999		22	8888	Update	Delete
812	2020-04-05	9999	9999		55	8888	Update	Delete

Insert
CARRIER CONTACTS
LOCATIONS
LOCATION CONTACTS
CARRIERS
APPOINTMENTS
REQUESTS

3. Schema Update changes:

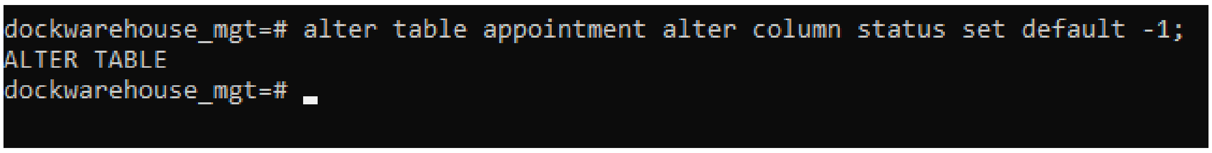
Po name makes it unique for purchasing organizations.

Dropping Not null constraint on status field of appointment:

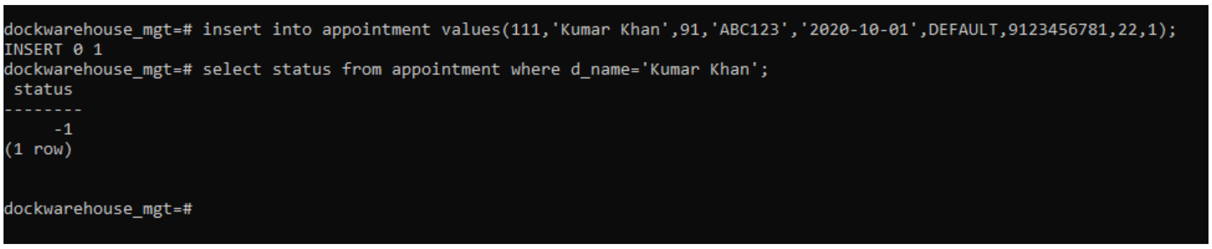


```
dockwarehouse_mgt=# alter table appointment alter column status drop not null;
ALTER TABLE
dockwarehouse_mgt=# _
```

Status should be set to -1 by default

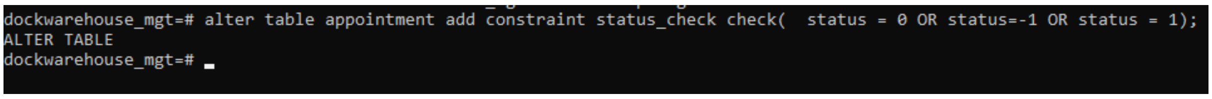


```
dockwarehouse_mgt=# alter table appointment alter column status set default -1;
ALTER TABLE
dockwarehouse_mgt=# _
```

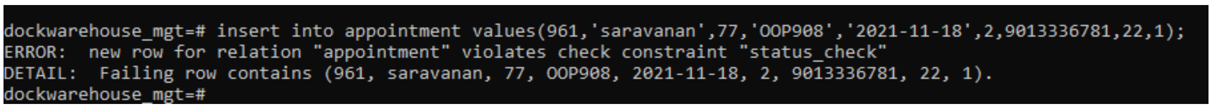


```
dockwarehouse_mgt=# insert into appointment values(111,'Kumar Khan',91,'ABC123','2020-10-01',DEFAULT,9123456781,22,1);
INSERT 0 1
dockwarehouse_mgt=# select status from appointment where d_name='Kumar Khan';
 status
--------
     -1
(1 row)


dockwarehouse_mgt=#
```

Status field in the appointment should be either 1,0 or -1

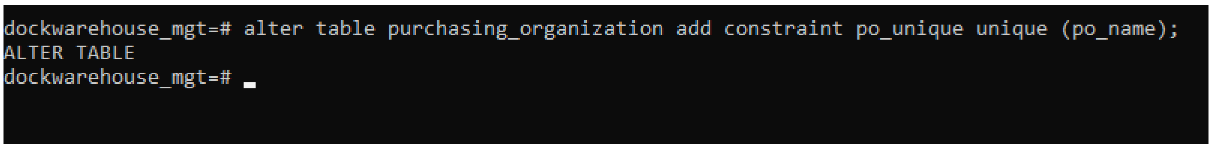


```
dockwarehouse_mgt=# alter table appointment add constraint status_check check(  status = 0 OR status=-1 OR status = 1);
ALTER TABLE
dockwarehouse_mgt=#
```

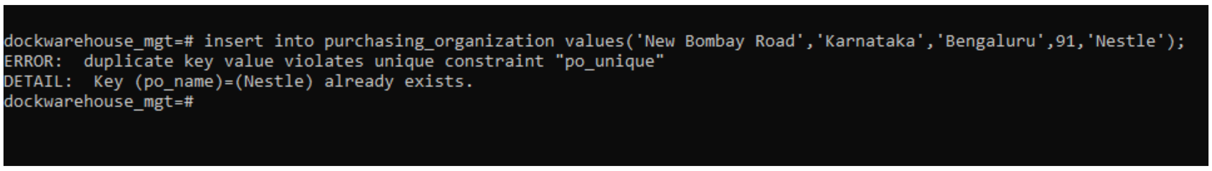


```
dockwarehouse_mgt=# insert into appointment values(961,'saravanan',77,'OOP908','2021-11-18',2,9013336781,22,1);
ERROR:  new row for relation "appointment" violates check constraint "status_check"
DETAIL:  Failing row contains (961, saravanan, 77, OOP908, 2021-11-18, 2, 9013336781, 22, 1).
dockwarehouse_mgt=#
```

Make po_name as unique in purchasing organization:

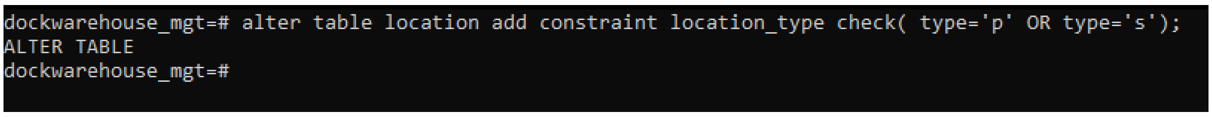


```
dockwarehouse_mgt=# alter table purchasing_organization add constraint po_unique unique (po_name);
ALTER TABLE
dockwarehouse_mgt=#
```

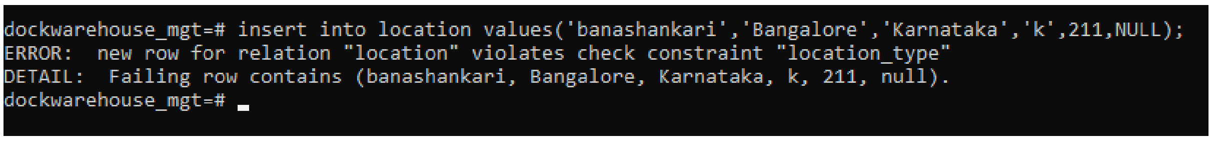


```
dockwarehouse_mgt=# insert into purchasing_organization values('New Bombay Road','Karnataka','Bengaluru',91,'Nestle');
ERROR:  duplicate key value violates unique constraint "po_unique"
DETAIL:  Key (po_name)=(Nestle) already exists.
dockwarehouse_mgt=#
```

Location type should be either p or s

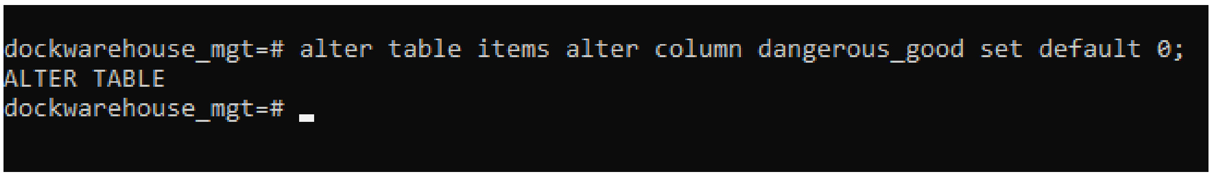


```
dockwarehouse_mgt=# alter table location add constraint location_type check( type='p' OR type='s');
ALTER TABLE
dockwarehouse_mgt=#
```

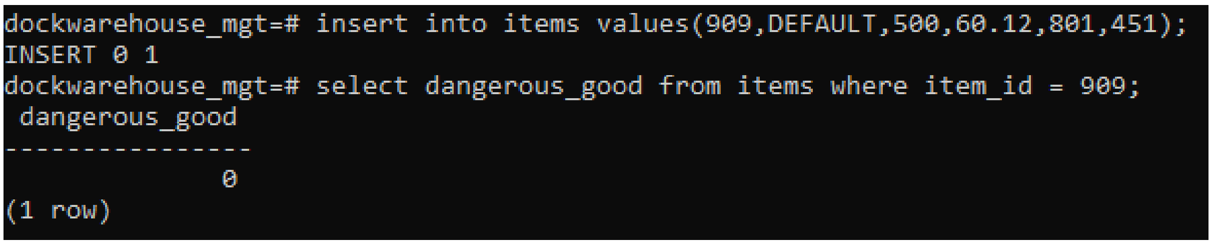


```
dockwarehouse_mgt=# insert into location values('banashankari','Bangalore','Karnataka','k',211,NULL);
ERROR:  new row for relation "location" violates check constraint "location_type"
DETAIL:  Failing row contains (banashankari, Bangalore, Karnataka, k, 211, null).
dockwarehouse_mgt=#
```

Altering table items set dangerous good default value to zero:

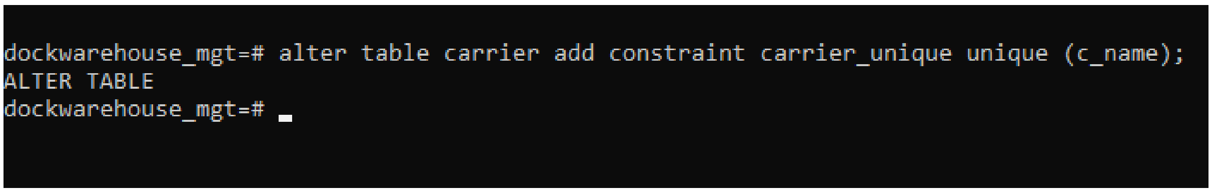


```
dockwarehouse_mgt=# alter table items alter column dangerous_good set default 0;
ALTER TABLE
dockwarehouse_mgt=# _
```

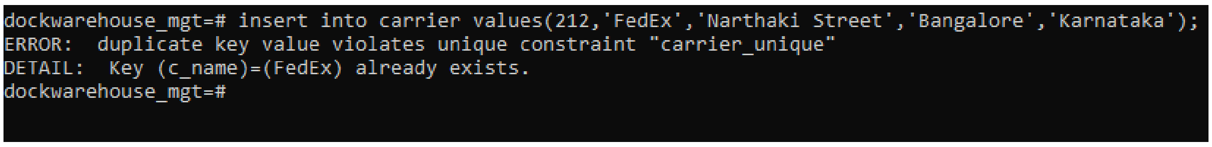


```
dockwarehouse_mgt=# insert into items values(909,DEFAULT,500,60.12,801,451);
INSERT 0 1
dockwarehouse_mgt=# select dangerous_good from items where item_id = 909;
 dangerous_good
----------------
              0
(1 row)
```

Making c_name as unique in carrier table:

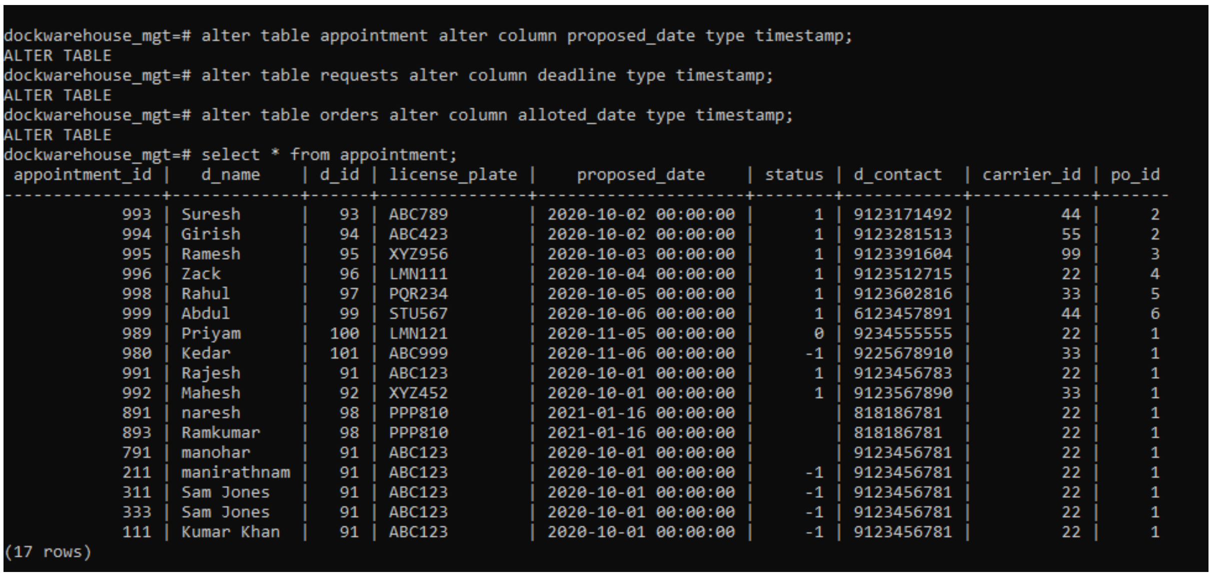


```
dockwarehouse_mgt=# alter table carrier add constraint carrier_unique unique (c_name);
ALTER TABLE
dockwarehouse_mgt=# _
```

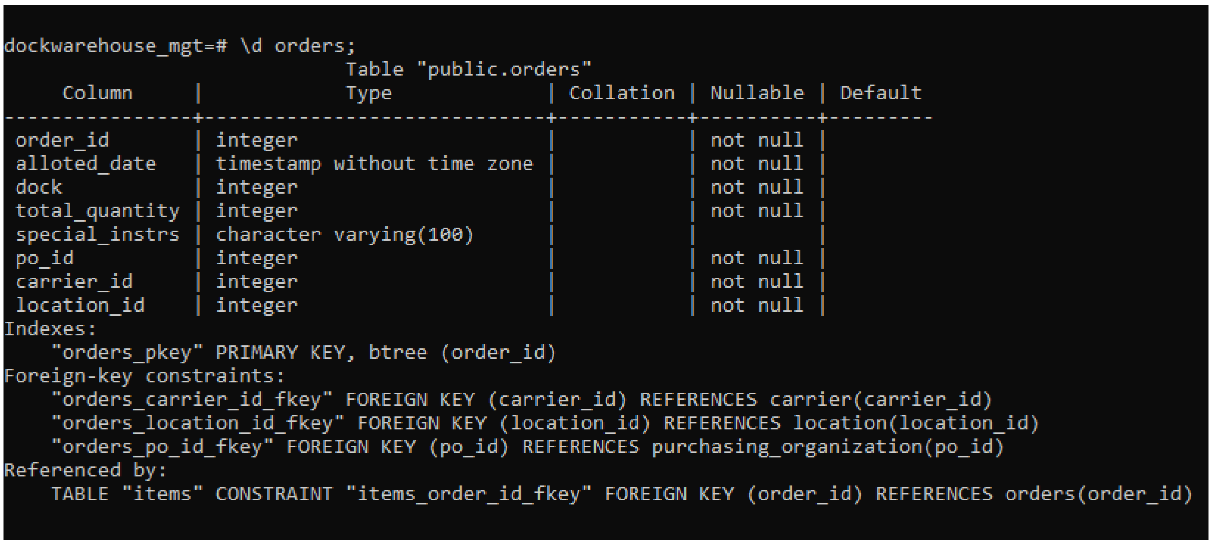


```
dockwarehouse_mgt=# insert into carrier values(212,'FedEx','Narthaki Street','Bangalore','Karnataka');
ERROR:  duplicate key value violates unique constraint "carrier_unique"
DETAIL:  Key (c_name)=(FedEx) already exists.
dockwarehouse_mgt=#
```

Changing data type of date to timestamp:



```
dockwarehouse_mgt=# alter table appointment alter column proposed_date type timestamp;
ALTER TABLE
dockwarehouse_mgt=# alter table requests alter column deadline type timestamp;
ALTER TABLE
dockwarehouse_mgt=# alter table orders alter column alloted_date type timestamp;
ALTER TABLE
dockwarehouse_mgt=# select * from appointment;
 appointment_id |   d_name    | d_id | license_plate |    proposed_date     | status | d_contact  | carrier_id | po_id
----------------+-------------+------+---------------+---------------------+--------+------------+------------+-------
            993 | Suresh      |   93 | ABC789        | 2020-10-02 00:00:00 |      1 | 9123171492 |         44 |     2
            994 | Girish      |   94 | ABC423        | 2020-10-02 00:00:00 |      1 | 9123281513 |         55 |     2
            995 | Ramesh      |   95 | XYZ956        | 2020-10-03 00:00:00 |      1 | 9123391604 |         99 |     3
            996 | Zack        |   96 | LMN111        | 2020-10-04 00:00:00 |      1 | 9123512715 |         22 |     4
            998 | Rahul       |   97 | PQR234        | 2020-10-05 00:00:00 |      1 | 9123602816 |         33 |     5
            999 | Abdul       |   99 | STU567        | 2020-10-06 00:00:00 |      1 | 6123457891 |         44 |     6
            989 | Priyam      |  100 | LMN121        | 2020-11-05 00:00:00 |      0 | 9234555555 |         22 |     1
            980 | Kedar       |  101 | ABC999        | 2020-11-06 00:00:00 |     -1 | 9225678910 |         33 |     1
            991 | Rajesh      |   91 | ABC123        | 2020-10-01 00:00:00 |      1 | 9123456783 |         22 |     1
            992 | Mahesh      |   92 | XYZ452        | 2020-10-01 00:00:00 |      1 | 9123567890 |         33 |     1
            891 | naresh      |   98 | PPP810        | 2021-01-16 00:00:00 |        | 818186781  |         22 |     1
            893 | Ramkumar    |   98 | PPP810        | 2021-01-16 00:00:00 |        | 818186781  |         22 |     1
            791 | manohar     |   91 | ABC123        | 2020-10-01 00:00:00 |        | 9123456781 |         22 |     1
            211 | manirathnam |   91 | ABC123        | 2020-10-01 00:00:00 |     -1 | 9123456781 |         22 |     1
            311 | Sam Jones   |   91 | ABC123        | 2020-10-01 00:00:00 |     -1 | 9123456781 |         22 |     1
            333 | Sam Jones   |   91 | ABC123        | 2020-10-01 00:00:00 |     -1 | 9123456781 |         22 |     1
            111 | Kumar Khan  |   91 | ABC123        | 2020-10-01 00:00:00 |     -1 | 9123456781 |         22 |     1
(17 rows)
```


```
dockwarehouse_mgt=# \d orders;
                             Table "public.orders"
     Column     |            Type             | Collation | Nullable | Default
----------------+-----------------------------+-----------+----------+---------
 order_id       | integer                     |           | not null |
 alloted_date   | timestamp without time zone |           | not null |
 dock           | integer                     |           | not null |
 total_quantity | integer                     |           | not null |
 special_instrs | character varying(100)      |           |          |
 po_id          | integer                     |           | not null |
 carrier_id     | integer                     |           | not null |
 location_id    | integer                     |           | not null |
Indexes:
    "orders_pkey" PRIMARY KEY, btree (order_id)
Foreign-key constraints:
    "orders_carrier_id_fkey" FOREIGN KEY (carrier_id) REFERENCES carrier(carrier_id)
    "orders_location_id_fkey" FOREIGN KEY (location_id) REFERENCES location(location_id)
    "orders_po_id_fkey" FOREIGN KEY (po_id) REFERENCES purchasing_organization(po_id)
Referenced by:
    TABLE "items" CONSTRAINT "items_order_id_fkey" FOREIGN KEY (order_id) REFERENCES orders(order_id)
```

4. Future scope of change in:

a. Schema:

- For the columns defined by the miniworld currently, the schema is well-thought of and is air-tight and requires no modification. So whenever new features are introduced there will be a heavy need of schema changes. For example, say we need to also store the driver’s relation as a separate entity. Then the columns in the relation have to be related to most of the relations which are there which concern the carrier. Also, if the organizations suddenly decide to keep a detailed tracking of stopping points, we may have to add a separate relation for it and almost all relations currently existing would have to be related to it. There could always be a need for conversion of relations to higher normal forms for improvement of the quality of relations.

b. Constraints:

- We can add many schematic constraints to the relations as and when external factors deciding the description of the miniworld changes. This could be business/economical or some other changes. For example, if the database is demanded to have only some specific k-orders per day, we can impose that constraint to the relations existing.

c. Database migration:

- If the application demands semi-structured or unstructured data and/or multimedia files to be stored in the database like fee receipts, video footage etc there might be a need to shift to the NO-SQL DB. Otherwise, the application would work fine with the current RDBMS setup.

5. NO-SQL database migration:

There is no need to migrate from SQL based rdbms to a NO-SQL database as our data is highly structured and our multimedia doesn’t concern multimedia file formats. But if we have to migrate, key-value stores are the best choice. One simple reason being, that is least complex NO-SQL databases, easy to understand, learn and maintain. They are lightweight and we can introduce semi-structured data.

6. Team Contributions:

1. Sumukh Bhat
    - Creating login page(30mins), complex queries page(30mins) and all queries(update, insert, select, delete) for the po_product_manager(5 tables) (3 hrs) on the frontend

2. Shreyas VBKS
    - all queries(update, insert, select, delete) for the po_shipment_manager(11 tables) (4 hrs) on the frontend

3. Sowmya M
    - all queries(update, insert, select, delete) for the carrier_shipment_manager(11 tables) (4 hrs) on the frontend