

"Optimizing Logistics Through Data" - Courier Nexus

Chaitali Rajulkumar Thakkar
UB ID:50557808

MS Engineering Science (Data Science)
thakkar6@buffalo.edu

Pranavi Chintala
UB ID: 50563997

MS Engineering Science (Data Science)
pchintal@buffalo.edu

I. PROBLEM STATEMENT

As the scale-up and operations of the logistics company grow increasingly complex, it becomes progressively difficult to manage or keep track of shipment flow. The tools currently used for tracking packages, senders, receivers, and delivery details—mostly Excel—cannot handle the growth in volume and complexity. This eventually leads to inefficiencies in tracking shipment statuses, managing customer feedback, handling delivery failures, and resolving claims. These shortcomings result in delayed deliveries, poor customer satisfaction, and operational bottlenecks.

Besides that, the basic password protection offered by Excel can be easily circumvented, leaving data vulnerable to accidental exposure or overwriting. Moreover, Excel requires manual backups, increasing the risk of data loss and making recovery from file corruption difficult, especially in fast-paced business environments. This highlights the need for the company to transition to a more secure, automated, and highly scalable database solution to meet its growing logistical needs.

We aim to address these issues by implementing a scalable, data-driven logistics management system. This system will provide real-time shipment tracking, efficient monitoring of delivery attempts, and streamlined claims processing. It will capture detailed package dimensions, weight-based shipping rates, and customer feedback trends while ensuring seamless collaboration across departments. The database solution will enable efficient handling of large datasets, support parallel access for multiple users, and allow the company to generate insightful reports to optimize logistics operations. The goal is to enhance data accuracy, improve decision-making, and drive operational efficiency as the company expands.

II. TARGET USERS

This data-driven platform for logistics optimization is designed to serve four key target user groups.

First, core users are the **Logistics Operations Team** who are in charge of managing and tracking shipments. They will add new shipment details, like package dimensions, sender and receiver information, choosing the appropriate methods of shipment. They will also update the delivery status, monitoring the attempts made to ensure transparency and accuracy in the chain of logistics. For instance, Sarah would access the system, a logistics coordinator, input information about packages,

provide a tracking number, and update shipment status as it goes through different stages.

Next, the **Customer Service Representatives** are to use the platform in order to support customers with various inquiries and resolve issues that arise regarding delayed shipments or claims. These representatives are going to use the system to check the status of shipments with a view to informing customers in real-time and initiating claims when necessary. In this way, it smoothes out the process for resolving issues. For instance, John will have no problem pulling shipment details while a customer calls about his or her package to review where his shipment is and its current status, thus giving instant feedback.

The **Warehouse and Delivery Teams** will also interface with the platform to update package statuses, log delivery attempts, and record delivery locations. Members will, in turn, regularly update the system with fresh information regarding the movement of packages. For example, Mike could put it in the system that he couldn't deliver a package because the recipient wasn't available; this would keep the logistics team informed so they can reschedule the delivery as soon as possible.

Finally, the **Administrators and IT Team** maintain security, performance, and track the general care of the platform. This group will be in charge of permissions, data security, different types of system performance checks, and periodic backups to ensure the health of the database. For instance, Maria, as a database administrator, will make sure that sensitive information such as customers' claims is not revealed to unauthorized personnel, but also that the system is in compliance with the established security policies. Such users will ensure smoothness and security for the entire platform, supporting the day-to-day operations of the logistics company.

By going back to the specific needs of these varying user groups, **Courier Nexus** will be able to manage logistics efficiently, improve departmental communication, and deliver a seamless experience across all stakeholders touching the shipment lifecycle.

III. ENTITY - RELATIONSHIP DIAGRAM

A. Entities

1) **Claims**: `claim_id` (Primary Key, INTEGER): Unique identifier for each claim. This cannot be NULL.

claim_status (TEXT): Represents the status of the claim (e.g., "Pending", "Resolved"). Can be NULL.

claim_date (DATE): The date when the claim was filed. Can be NULL.

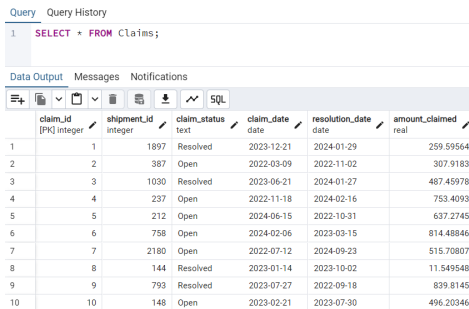
resolution_date (DATE): The date when the claim was resolved. Can be NULL.

amount_claimed (REAL): The amount claimed in the process. Default value is 0.0.

shipment_id (Foreign Key, INTEGER): References the shipment_id in the Shipments table to associate a claim with a specific shipment. Cannot be NULL.

Primary Key Justification: The claim_id serves as the primary key to ensure that each claim is uniquely identifiable within the system. No duplicate claims are allowed, and every claim needs a distinct ID for traceability.

Foreign Key Action: When a shipment (referenced by shipment_id) is deleted, a cascade delete action can be applied, meaning the associated claims would also be deleted.



The screenshot shows a database query result for the Claims table. The query is 'SELECT * FROM Claims;'. The table has 10 rows and 7 columns: claim_id (PK integer), shipment_id integer, claim_status text, claim_date date, resolution_date date, amount_claimed real, and an unnamed column. The data is as follows:

	claim_id [PK] integer	shipment_id integer	claim_status text	claim_date date	resolution_date date	amount_claimed real	
1	1	1897	Resolved	2023-12-21	2024-01-29	259.59564	
2	2	387	Open	2022-03-09	2022-11-02	307.9183	
3	3	1030	Resolved	2023-06-21	2024-01-27	487.45978	
4	4	237	Open	2022-11-18	2024-02-16	753.4093	
5	5	212	Open	2024-06-15	2022-10-31	637.2745	
6	6	758	Open	2024-02-06	2023-03-15	814.48846	
7	7	2180	Open	2022-07-12	2024-05-23	515.70807	
8	8	144	Resolved	2023-01-14	2023-10-02	11.549548	
9	9	793	Resolved	2023-07-27	2022-09-18	839.8145	
10	10	148	Open	2023-02-21	2023-07-30	496.20346	

Fig. 1. Table Claims

2) **Customer Feedback** : feedback_id (Primary Key, INTEGER): Unique identifier for customer feedback. Cannot be NULL.

customer_id (INTEGER): Represents the customer providing feedback. Can be NULL.

rating (INTEGER): The rating given by the customer (e.g., 1–5). Can be NULL.

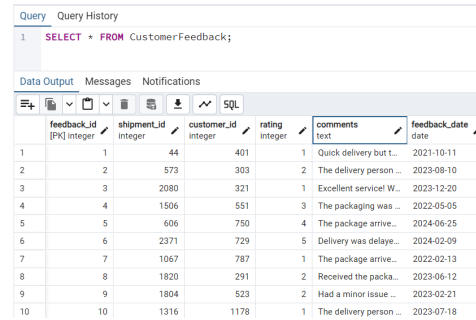
comments (TEXT): Feedback or comments from the customer. Can be NULL.

feedback_date (DATE): The date when the feedback was provided. Can be NULL.

shipment_id (Foreign Key, INTEGER): References the shipment_id in the Shipments table to tie feedback to a specific shipment. Cannot be NULL.

Primary Key Justification: The feedback_id uniquely identifies each feedback entry, ensuring no duplicates and allowing feedback to be tracked individually.

Foreign Key Action: On deletion of a referenced shipment, the associated customer feedback can be set to NULL to maintain feedback records without shipment information.



The screenshot shows a database query result for the CustomerFeedback table. The query is 'SELECT * FROM CustomerFeedback;'. The table has 10 rows and 7 columns: feedback_id (PK integer), shipment_id integer, customer_id integer, rating integer, comments text, and feedback_date date. The data is as follows:

	feedback_id [PK] integer	shipment_id integer	customer_id integer	rating integer	comments text	feedback_date date
1	1	44	401	1	Quick delivery but t...	2021-10-11
2	2	573	303	2	The delivery person ...	2023-08-10
3	3	2080	321	1	Excellent service! W...	2023-12-20
4	4	1506	551	3	The packaging was ...	2022-05-05
5	5	606	750	4	The package arrive...	2024-06-25
6	6	2371	729	5	Delivery was delaye...	2024-02-09
7	7	1067	787	1	The package arrive...	2022-02-13
8	8	1820	291	2	Received the packa...	2023-06-12
9	9	1804	523	2	Had a minor issue ...	2023-02-21
10	10	1316	1178	1	The delivery person ...	2023-07-18

Fig. 2. Table Customer Feedback

3) **Delivery Attempts** : attempt_id (Primary Key, INTEGER): Unique identifier for each delivery attempt. Cannot be NULL.

attempt_date (DATE): The date of the delivery attempt. Can be NULL.

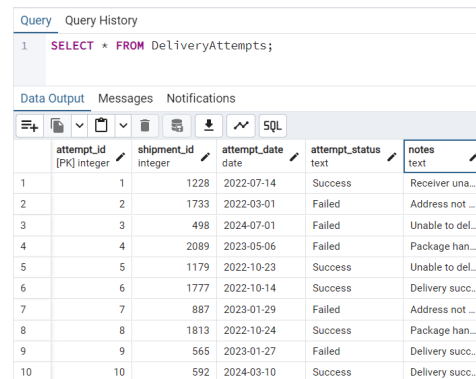
attempt_status (TEXT): The result or status of the delivery attempt (e.g., "Failed", "Successful"). Can be NULL.

notes (TEXT): Additional information about the attempt. Can be NULL.

shipment_id (Foreign Key, INTEGER): References shipment_id in the Shipments table. Cannot be NULL.

Primary Key Justification: The attempt_id is necessary to uniquely identify each attempt made to deliver a shipment, ensuring that all records are traceable.

Foreign Key Action: On deletion of a shipment, a cascade delete can be applied, removing all associated delivery attempts for that shipment.



The screenshot shows a database query result for the DeliveryAttempts table. The query is 'SELECT * FROM DeliveryAttempts;'. The table has 10 rows and 6 columns: attempt_id (PK integer), shipment_id integer, attempt_date date, attempt_status text, and notes text. The data is as follows:

	attempt_id [PK] integer	shipment_id integer	attempt_date date	attempt_status text	notes text
1	1	1228	2022-07-14	Success	Receiver una...
2	2	1733	2022-03-01	Failed	Address not ...
3	3	498	2024-07-01	Failed	Unable to del...
4	4	2089	2023-05-06	Failed	Package han...
5	5	1179	2022-10-23	Success	Unable to del...
6	6	1777	2022-10-14	Success	Delivery succ...
7	7	887	2023-01-29	Failed	Address not ...
8	8	1813	2022-10-24	Success	Package han...
9	9	565	2023-01-27	Failed	Delivery succ...
10	10	592	2024-03-10	Success	Delivery succ...

Fig. 3. Table Delivery Attempts

4) **Delivery Locations**: location_id (Primary Key, INTEGER): Unique identifier for each delivery location. Cannot be NULL.

longitude (REAL): The longitude of the delivery location. Can be NULL.

latitude (REAL): The latitude of the delivery location. Can be NULL.

delivery_date (DATE): The date of delivery at this location. Can be NULL.

shipment_id (Foreign Key, INTEGER): References shipment_id in the Shipments table. Cannot be NULL.
Primary Key Justification: The location_id is used to ensure that every delivery location is unique, which is crucial for tracking where shipments were delivered.
Foreign Key Action: On deletion of a shipment, all associated delivery locations should also be deleted, enforcing a cascade delete policy.

	location_id [PK] integer	shipment_id integer	longitude real	latitude real	delivery_date date
1	1	1373	-99.89181	34.38017	2023-08-15
2	2	1476	-41.30496	24.38021	2023-03-24
3	3	1178	82.0588	-3.6846325	2023-02-13
4	4	754	56.149853	43.25911	2022-12-05
5	5	1630	97.22757	56.849693	2022-06-01
6	6	1782	160.31514	-86.59653	2022-01-08
7	7	1210	111.70607	-66.451126	2022-01-13
8	8	941	-109.08024	4.1975765	2021-11-01
9	9	2023	116.70572	-59.936333	2023-04-08
10	10	1915	119.57497	82.997406	2023-06-27

Fig. 4. Table Delivery Locations

5) **Shipping Rates** : rate_id (Primary Key, INTEGER): Unique identifier for each shipping rate. Cannot be NULL.
shipping_method (TEXT): The shipping method used (e.g., "Air", "Ground"). Can be NULL.
weight_limit (REAL): The maximum weight allowed for this rate. Default value is 0.0.
base_price (REAL): The base price for this shipping rate. Default value is 0.0.
additional_cost_per_kg (REAL): Additional cost per kilogram over the weight limit. Default value is 0.0.
Primary Key Justification: The rate_id ensures each shipping rate is unique, as different rates may apply based on method and weight.
Foreign Key Action: No foreign keys are present, so no specific action is required for deletions.

	rate_id [PK] integer	shipping_method text	weight_limit real	base_price real	additional_cost real
1	1	Sea	9.288893	67.30307	3.0308886
2	2	Ground	18.316689	59.710064	7.6476617
3	3	Air	62.37874	20.341097	6.700099
4	4	Air	52.873016	66.149124	3.5533419
5	5	Sea	39.385883	23.125923	8.530161
6	6	Air	35.220295	31.24818	4.7629867
7	7	Ground	35.649265	33.091633	9.789519
8	8	Ground	10.081452	87.70735	1.4218218
9	9	Air	26.561792	75.10101	5.698608
10	10	Air	95.30269	62.07692	7.74137

Fig. 5. Table Shipping Rates

6) **Packages** : package_id (Primary Key, INTEGER): Unique identifier for each package. Cannot be NULL.
weight (REAL): Weight of the package. Default value is 0.0.
length (REAL): Length of the package. Default value is 0.0.
width (REAL): Width of the package. Default value is 0.0.
height (REAL): Height of the package. Default value is 0.0.
content_description (TEXT): Description of the contents of the package. Can be NULL.
Primary Key Justification: The package_id uniquely identifies each physical package, allowing shipments to be tied to specific packages.
Foreign Key Action: No foreign keys are present, so no specific action is required for deletions.

	package_id [PK] integer	weight real	length real	width real	height real	content_description text
1	1	7.9755473	92.16988	23.074186	43.144604	Toys - Children
2	2	1.4266732	49.89054	31.036566	48.295486	Documents - Confidential
3	3	3.5929816	18.250854	68.93615	40.849167	Household Items
4	4	2.3161578	34.769405	63.15338	61.250557	Electronics - Fragile
5	5	5.787209	69.9619	37.626244	34.959892	Toys - Children
6	6	5.270032	55.13593	50.392223	33.89992	Clothing - Casual Wear
7	7	3.3712265	43.226013	83.558556	87.29519	Sports Equipment
8	8	3.4808204	54.05882	86.01713	52.93781	Jewelry - Delicate
9	9	1.5713879	75.32122	69.65692	53.45917	Clothing - Casual Wear
10	10	6.2093225	53.09072	37.73148	45.184784	Jewelry - Delicate

Fig. 6. Table Packages

7) **Senders** : sender_id (Primary Key, INTEGER): Unique identifier for each sender. Cannot be NULL.
name (TEXT): Name of the sender. Can be NULL.
address (TEXT): Address of the sender. Can be NULL.
contact (TEXT): Contact information for the sender. Can be NULL.
Primary Key Justification: The sender_id uniquely identifies each sender, allowing packages to be tracked by who sent them.
Foreign Key Action: No foreign keys are present, so no specific action is required for deletions.

	sender_id [PK] integer	name text	address text	contact_number text
1	1	Ryan Dunn	4055 Monica Key	9372572007
2	2	Aaron Clements	2641 Proctor Trail Apt. 482	(459)757-3054x08142
3	3	Jason Martinez	8047 Welch Tunnel	309-323-8006x367
4	4	Lindsey Hawkins	83902 Smith Island	7456199563
5	5	Kevin Foster	762 Ana Islands	+1-684-247-7873x11655
6	6	Michelle Peters	944 Murray Alley Suite 169	(584)506-6020x3322
7	7	Michael Thomas	448 Castillo Route Apt. 713	836 601 1128
8	8	Stanley White	9221 Gregory Trail Suite 548	919-379-4623x74771
9	9	Natalie Jenkins	999 Compton Knoll	(338)539-6116x6420
10	10	Caleb Ferrell	2474 Kevin Gardens Apt. 305	(467)232-0044

Fig. 7. Table Senders

8) **Receivers** : receiver_id (Primary Key, INTEGER): Unique identifier for each receiver. Cannot be NULL.

name (TEXT): Name of the receiver. Can be NULL.
address (TEXT): Address of the receiver. Can be NULL.
contact (TEXT): Contact information for the receiver.
Can be NULL.

Primary Key Justification: The receiver_id ensures that each receiver is unique, allowing the system to track who receives which shipment.

Foreign Key Action: No foreign keys are present, so no specific action is required for deletions.

receiver_id [PK] integer	name text	address text	contact_number text
1	218	Christopher Ferguson	727.263.824x86692
2	1	Abigail Garcia	226 Joshua Street
3	2	Paul Mccoy	0986 Smith Rapids
4	3	Danny Haynes	55592 Gibbs Summit
5	4	Mr. Cody Smith	823 Lisa Station Suite 345
6	5	Jacob Prince	92252 Lisa Harbor
7	6	Jennifer Hernandez	15114 Peter Ranch Suite 198
8	7	Samuel Valenzuela	93029 Angelica Drive Apt. 798
9	8	Dominique Gomez	70044 Wood Villages
10	9	Sheri Vincent	8340 Ryan Rapids Suite 313
11	10	David Perez	8634 Hernandez Knoll

Fig. 8. Table Receivers

9) **Service Areas** : area_id (Primary Key, INTEGER): Unique identifier for each service area. Cannot be NULL.
service_area_name (TEXT): Name of the service area.
Can be NULL.

coverage_description (TEXT): Description of the area covered by the service. Can be NULL.

is_active (BOOLEAN): Indicates whether the service area is active. Default value is TRUE.

Primary Key Justification: The area_id uniquely identifies each service area, ensuring distinct coverage zones.

Foreign Key Action: No foreign keys are present, so no specific action is required for deletions.

id [PK] integer	service_area_name text	coverage_description text	is_active boolean
1	East Nicholas	Full coverage of residential areas.	false
2	New Jasonberg	Offers next-day delivery across the city.	false
3	Kennedytown	Delivery limited to commercial areas only.	true
4	North Jared	Covers coastal cities with express delivery.	true
5	Port Danielfurt	Next-day delivery across all urban zones.	true
6	East Corey	Primarily services rural and hard-to-reach areas.	true
7	Snydermouth	Full coverage of residential areas.	true
8	Bryanfurt	Covers downtown and neighboring suburbs.	true
9	Murillochester	Full coverage of residential areas.	true
10	East Jenniferport	Covers downtown and neighboring suburbs.	false

Fig. 9. Table Service Areas

10) **Shipments** : shipment_id (Primary Key, INTEGER): Unique identifier for each shipment. Cannot be NULL.
package_id (Foreign Key, INTEGER): References package_id in the Packages table. Cannot be NULL.
sender_id (Foreign Key, INTEGER): References sender_id in the Senders table. Cannot be NULL.

receiver_id (Foreign Key, INTEGER): References receiver_id in the Receivers table. Cannot be NULL.

shipment_method (TEXT): The method used for shipment. Can be NULL.

tracking_number (TEXT): Tracking number for the shipment. Can be NULL.

status (TEXT): The status of the shipment (e.g., "In Transit", "Delivered"). Can be NULL.

Primary Key Justification: The shipment_id ensures that each shipment is uniquely identified, allowing detailed tracking of every shipment in the system.

Foreign Key Action: For package_id, sender_id, and receiver_id, when the primary key is deleted, all associated shipments should be deleted using cascade delete.

Query Query History

1 SELECT * FROM Shipments;

Data Output Messages Notifications

Fig. 10. Table Shipments

B. Relations and Attributes

Table I below gives an overview of some of the important entities that describe the relationships and attributes of the logistics enterprise, including claims, shipments, and customer feedback. It provides full elaboration, featuring specific details for each entity on its main attributes and how those entities relate to one another through foreign keys. The following table shows the structure of this system.

C. Relationships Between Entities

Fig. 11 illustrates the relationship of the logistics system entities. This shows how shipments, packages, senders, and receivers—the major entities in this database—relate to one another. The relationships between different entities are established through foreign keys, which ensure that the flow of data within the system is smooth for tracking and managing purposes. The relations between each entity are given as follows:

1) **Shipments** → **Packages**: There is a one-to-one relationship between Shipments and Packages, meaning that each shipment must have precisely one package, and every package has to be shipped once. This ensures that each package's information is attached to a specific shipment.

2) **Shipments** → **Senders / Receivers**: Shipments are related to both Senders and Receivers on a many-to-one basis: one sender can send many shipments, and similarly, one receiver can receive many shipments; each shipment is linked with one sender and one receiver.

TABLE I
ENTITY RELATIONS AND ATTRIBUTES

Entity	Attributes
Claims	claim_id (primary key), claim_status, claim_date, resolution_date, amount_claimed, shipment_id (foreign key)
Customer Feedback	feedback_id (primary key), customer_id, rating, comments, feedback_date, shipment_id (foreign key)
Delivery Attempts	attempt_id (primary key), attempt_date, attempt_status, notes, shipment_id (foreign key)
Delivery Locations	location_id (primary key), longitude, latitude, delivery_date, shipment_id (foreign key)
Shipping Rates	rate_id (primary key), shipping_method, weight_limit, base_price, additional_cost_per_kg
Packages	package_id (primary key), weight, length, width, height, content_description
Senders	sender_id (primary key), name, address, contact
Receivers	receiver_id (primary key), name, address, contact
Service Areas	area_id (primary key), service_area_name, coverage_description, is_active (Boolean)
Shipments	shipment_id (primary key), package_id (foreign key), sender_id (foreign key), receiver_id (foreign key), shipment_method, tracking_number, status

3) **Shipments** → **Claims** : It can be said that the relationship between Shipments and Claims is one-to-one because a shipment can only have one claim, and a claim can only be linked to one shipment. This ensures that a claim is directly related to an individual shipment.

4) **Shipments** → **Customer Feedback**: This is a one-to-many relationship; many entries regarding customer feedback are associated with one shipment. A customer can leave multiple pieces of feedback, such as providing updates about experiences or following up after the receipt of delivery.

5) **Shipments** → **Delivery Attempts**: The relationship between Shipments and Delivery Attempts is one-to-many because for each shipment, there can be more than one attempt at its delivery. This models real-world logistics where multiple delivery attempts may be required.

6) **Shipments** → **Delivery Locations**: This also represents a one-to-many relationship. There might be several delivery locations logged against one shipment, especially if it was moved to another distribution center or if multiple delivery attempts were made.

7) **Service Areas** → **Shipments**: While not directly connected in the diagram, Service Areas define the

geographical zones in which shipments can be delivered. Shipping Rates depend on the shipment method and the package's weight, helping to estimate costs.

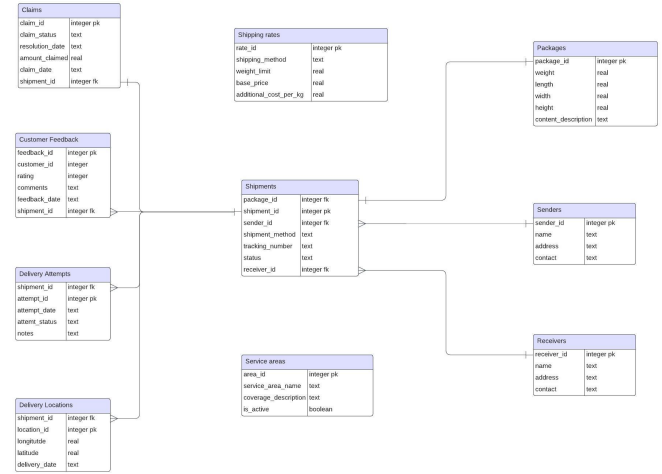


Fig. 11. Entity Relationship Diagram for Logistics Enterprise

IV. DATABASE GENERATION

In the project, Python scripts were written to create a mock dataset representative of an imaginary domain. The idea behind creating this dataset was to develop a large but manageable database for SQL-based applications. The dataset needs to involve queries, trend analysis, and updates in the specified domain. As the data is synthetic, we have the flexibility to shape and modify it so that the database efficiently meets its intended purpose.

A. Generating the table Packages

```
import sqlite3
from faker import Faker
import random

# Initialize Faker
fake = Faker()

# Connect to SQLite database (or create it)
conn = sqlite3.connect('courier_services5.db')
cursor = conn.cursor()

# Create Tables
cursor.execute('''CREATE TABLE IF NOT EXISTS Packages (
    package_id INTEGER PRIMARY KEY,
    weight REAL,
    length REAL,
    width REAL,
    height REAL,
    content_description TEXT
)''')
```

```

# Commit table creation
conn.commit()

# Helper functions to generate random data
def create_packages(n):
    for _ in range(n):
        cursor.execute(''''INSERT INTO Packages
        (weight, length, width, height,
        content_description)
        VALUES (?, ?, ?, ?, ?)''',
        (random.uniform(1, 10),
        random.uniform(10, 100),
        random.uniform(10, 100),
        random.uniform(10, 100),
        fake.text(max_nb_chars=50)))
    conn.commit()

# Generate Data for a larger set
num_packages = 1500

create_packages(num_packages)

# Close the connection
conn.close()

```

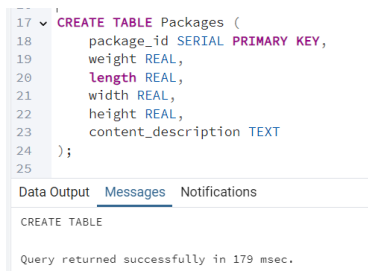


Fig. 12. Creating the table Packages

B. Generating the table Shipments

```

import sqlite3
from faker import Faker
import random

# Initialize Faker
fake = Faker()

# Connect to SQLite database
conn = sqlite3.connect('courier_services5.db')
cursor = conn.cursor()

# Create Tables
cursor.execute(''''CREATE TABLE IF NOT
EXISTS Shipments (
    shipment_id INTEGER PRIMARY KEY,
    package_id INTEGER,
    sender_id INTEGER,
    receiver_id INTEGER,
    shipping_method TEXT,
    tracking_number TEXT,
    status TEXT,
    FOREIGN KEY (package_id) REFERENCES Packages(package_id),
    FOREIGN KEY (sender_id) REFERENCES Senders(sender_id),
    FOREIGN KEY (receiver_id) REFERENCES Receivers(receiver_id)
)''')
conn.commit()

# Helper functions to generate random data
def create_shipments(n):
    for _ in range(n):
        cursor.execute(''''INSERT INTO Shipments
        (package_id, sender_id, receiver_id,
        shipping_method, tracking_number, status)
        VALUES (?, ?, ?, ?, ?, ?)''',
        (random.randint(1, num_packages),
        random.randint(1, num_senders),
        random.randint(1, num_receivers),
        random.choice(['Air', 'Ground', 'Sea']),
        fake.uuid4(),
        random.choice(['Pending', 'Delivered'])))
    conn.commit()

# Generate Data for a larger set
num_shipments = 2500

create_shipments(num_shipments)

# Close the connection
conn.close()

```

```

shipping_method TEXT,
tracking_number TEXT,
status TEXT,
FOREIGN KEY (package_id) REFERENCES
Packages(package_id),
FOREIGN KEY (sender_id) REFERENCES
Senders(sender_id),
FOREIGN KEY (receiver_id) REFERENCES
Receivers(receiver_id)
)'''
)'''

# Helper functions to generate random data
def create_shipments(n):
    for _ in range(n):
        cursor.execute(''''INSERT INTO Shipments
        (package_id, sender_id, receiver_id,
        shipping_method, tracking_number, status)
        VALUES (?, ?, ?, ?, ?, ?)''',
        (random.randint(1, num_packages),
        random.randint(1, num_senders),
        random.randint(1, num_receivers),
        random.choice(['Air', 'Ground', 'Sea']),
        fake.uuid4(),
        random.choice(['Pending', 'Delivered'])))
    conn.commit()

# Generate Data for a larger set
num_shipments = 2500

create_shipments(num_shipments)

# Close the connection
conn.close()

```

```

# Generate Data for a larger set
num_shipments = 2500

create_shipments(num_shipments)

# Close the connection
conn.close()

```

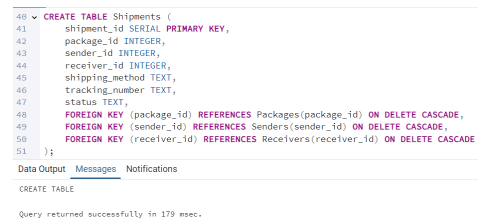


Fig. 13. Creating the table Shipments

As generated above, we have created the remaining tables in the database.

V. QUERY EXECUTION

Various SQL queries on a logistics database may be designed to retrieve and analyze key information related to shipments, claims, customer feedback, and delivery attempts. SQL can serve potentially powerful capabilities given through commands such as 'GROUP BY', 'JOIN', sub-queries, and

aggregate functions. These queries enable the system to answer essential questions such as, but not limited to, how many shipments are in each status, which shipments have claims, and so on. Following are four different kinds of SQL queries which use GROUP BY, sub-queries, JOIN, and aggregate functions on the Logistics Dataset:

A. Using GROUP BY to Get the Count of Shipments by Status

```
/*This query counts how many shipments are in each status
(e.g., "Pending", "Shipped", "Delivered").*/
SELECT status, COUNT(*) AS shipment_count
FROM Shipments
GROUP BY status;
```

Fig. 14. Using Group by function

Below is the output to the query in Fig. 14:

	status text	shipment_count bigint
1	Delivered	1212
2	Pending	1288

Fig. 15. Output of the Group by function

B. Using a Sub-query to Find Shipments with Claims

```
/*This query retrieves details of all shipments,]
that have had claims filed against them using a sub-query to filter shipments.*/
SELECT shipment_id, package_id, sender_id, receiver_id, status
FROM Shipments
WHERE shipment_id IN (
  SELECT shipment_id FROM Claims
);
```

Fig. 16. Using Sub-query

Below is the output to the above query in Fig. 16:

	shipment_id [PK] integer	package_id integer	sender_id integer	receiver_id integer	status text
1	12	460	632	267	Pending
2	19	1476	157	734	Pending
3	21	654	932	1082	Pending
4	22	941	280	428	Delivered
5	33	413	660	763	Delivered
6	37	110	97	889	Delivered
7	38	1024	483	19	Delivered
8	42	1412	589	330	Pending
9	43	675	519	995	Pending
10	57	1471	7	763	Delivered
Total rows: 459 of 459 Query complete 00:00:00.057 Ln 9735, Col 3					

Fig. 17. Output of the Sub-query

C. Using JOIN to Get Detailed Information About Shipments

```
/*This query uses an inner JOIN to get a detailed view of the shipments,
including the sender's name and receiver's name.*/
SELECT s.shipment_id, p.package_id, s.status, se.name AS sender_name, r.name AS receiver_name
FROM Shipments s
JOIN Packages p ON s.package_id = p.package_id
JOIN Senders se ON s.sender_id = se.sender_id
JOIN Receivers r ON s.receiver_id = r.receiver_id;
```

Fig. 18. Using Join function

Below is the output to the above query in Fig. 18:

	shipment_id integer	package_id integer	status text	sender_name text	receiver_name text
1	1	84	Pending	Joseph Pennington	Shane Williams
2	2	1397	Pending	Christopher Nash Jr.	Kimberly Reynolds
3	3	661	Pending	Brian Alvarez	Scott Booth
4	4	1325	Delivered	Michael Leon	Michael Smith
5	5	201	Pending	Brianna Cooper	Robert Soto
6	6	430	Delivered	Omar Donovan	Brandon Schwartz
7	7	641	Delivered	Ashlee Hamilton	Krista Jackson
8	8	530	Pending	Crystal Blake	Nicole Bailey
9	9	454	Delivered	Heather Mcege	Vanessa Campbell
10	10	528	Delivered	Seth Rivera	Mr. Corey Smith
Total rows: 1000 of 2500 Query complete 00:00:00.047 Ln 9745, Col 51					

Fig. 19. Output of the Join function

D. Using Aggregate Functions with a Sub-query to get Total Claims to specified Status

```
/*This query calculates the total value of all claims for shipments that have a specific status
(e.g., "Delivered") using an aggregate function and sub-query.*/
SELECT SUM(c.amount_claimed) AS total_claimed
FROM Claims c
WHERE c.shipment_id IN (
  SELECT shipment_id FROM Shipments WHERE status = 'Delivered'
);
```

Fig. 20. Using Aggregate function on Sub-query

Below is the output to the above query in Fig. 20:

	total_claimed real
1	124772.97

Fig. 21. Output of the Aggregate function on Sub-query