Northeastern University
College of Engineering

IE 6750: DATA WAREHOUSING & INTEGRATION

Milestone #7

Group No. 7

Student 1: Rahul Daruka
(daruka.r@northeastern.edu)

Student 2: Sharvari Deshpande
(deshpande.sha@northeastern.edu)

Percentage of effort contributed by student 1: 50%
Percentage of effort contributed by student 2: 50%
Signature of student 1: Rahul Daruka
Signature of student 2: Sharvari Deshpande
Submission Date: 24th November 2024

Table of Contents

Project Title: “PARKeasy: Parking Management System”

1. Introduction:

The increasing pace of urbanization and the subsequent expansion of metropolitan areas have significantly magnified the challenges faced in managing urban parking systems. These challenges are multifaceted, involving the optimal utilization of limited space, ensuring efficient traffic flow, and providing a satisfactory user experience in increasingly congested urban centers.

The PARKeasy: Parking Management System emerges as a transformative solution specifically engineered to meet these modern demands by radically enhancing the management of parking facilities. This system represents a leap forward in parking management technology, employing automation and sophisticated solutions to refine and streamline parking operations.

Designed with the dual goals of boosting operational efficiency and reducing urban congestion, PARKeasy leverages cutting-edge technologies to achieve several key improvements:

- Enhanced Operational Efficiency: By automating key aspects of parking management, including space allocation, ticketing, and payments, PARKeasy significantly reduces the need for manual intervention. This not only speeds up operations but also minimizes the scope for human error, leading to smoother and more reliable parking management.
- Congestion Mitigation: One of the system's pivotal roles is its contribution to alleviating urban traffic congestion. By efficiently guiding drivers to the nearest available parking spots, PARKeasy minimizes the typical circling behavior of drivers looking for parking. This, in turn, contributes to a smoother flow of traffic around parking facilities and reduces overall congestion in urban areas.
- Improvement of Urban Mobility: With the implementation of PARKeasy, cities can better manage their transportation networks, adapting more dynamically to the ebb and flow of daily and seasonal traffic patterns. This improved capability for managing urban mobility is crucial for cities looking to enhance their livability and sustainability in the face of continued growth and urbanization.

In summary, the PARKeasy: Parking Management System offers a sophisticated, data-driven approach to parking management that meets the contemporary needs of urban environments. By addressing the critical pain points of traditional parking systems with innovative solutions, PARKeasy not only enhances the efficiency of parking operations but also plays a significant role in improving urban mobility and reducing the environmental impact of urban traffic.

2. Problem Statement:

Urban parking systems are currently facing a myriad of issues that significantly detract from their efficiency and effectiveness. These challenges, arising from outdated methodologies and insufficient use of technology, severely impact urban mobility and contribute to the daily frustrations of city dwellers. Here is a more detailed breakdown of the primary issues plaguing urban parking management:

Inefficient Utilization of Space

One of the most pressing concerns in urban parking management is the inefficient utilization of available parking space. Many parking facilities are not optimized for space management, leading to substantial disparities in space usage across the facility. This inefficiency manifests in two critical ways:

- **Overcrowding in Popular Areas:** High-demand areas often experience severe overcrowding, leading to traffic build-ups and increased stress for drivers searching for available spots.
- **Underutilization in Less Popular Areas:** Simultaneously, less frequented parts of the facility remain largely unused, representing a wasted resource that could alleviate pressure on more congested areas. This imbalance not only affects the revenue potential of the facility but also leads to a suboptimal allocation of city resources.

Security Concerns

Security is a critical issue in parking management that is often compromised in traditional systems:

- **Vulnerability to Unauthorized Access:** Many parking facilities do not employ advanced security measures, making them susceptible to unauthorized access and vehicle theft.
- **Lack of Surveillance:** Insufficient monitoring and surveillance within the facility can lead to security breaches and does not provide a deterrent against criminal activities.

These multifaceted challenges highlight the urgent need for an innovative solution that embraces technology and automation. A modernized parking management system, capable of adapting to the complexities of urban environments, is essential for addressing these inefficiencies. By integrating data analytics, automated processes, and enhanced security measures, such systems can transform urban parking into a more efficient, secure, and user-friendly component of city infrastructure.

3. Project Objective:

The PARKeasy system is designed to revolutionize urban parking management by addressing key challenges through several focused objectives. Each objective aims to not only enhance the functionality and efficiency of parking operations but also to provide a framework for continuous improvement and adaptation to changing urban needs. Below is a detailed discussion of each objective:

Optimization of Parking Space Allocation

One of the core objectives of the PARKeasy system is the optimization of parking space allocation. The system will:

- **Dynamic Space Allocation:** Utilize sophisticated algorithms to dynamically allocate parking spaces, ensuring optimal utilization of every available spot. This involves assessing current parking space usage in real-time and adjusting allocations based on fluctuating demand throughout the day.
- **Preference-Based Allocation:** Consider user preferences such as proximity to destinations, preferred zones, and accessibility requirements to tailor parking space allocation, thereby enhancing user convenience and satisfaction.
- **Congestion Management:** By efficiently directing vehicles to available spots, the system will minimize the typical circling of drivers looking for parking, thereby reducing local traffic congestion and associated pollution.

Enhancement of User Experience

Enhancing the user experience is a key objective, with the system providing:

- **Reduced Wait Times:** Through the system, users can reserve parking spaces in advance, which significantly reduces the time spent searching for parking upon arrival.
- **Customer Support:** Implement a responsive customer support system within the app, facilitating quick resolution of any issues and providing a seamless user experience from start to finish.

Data-Driven Decision Making

Leveraging data to inform decision-making processes is crucial:

- **Real-Time Analytics:** Implement real-time analytics to monitor parking patterns and user behavior, allowing for immediate adjustments to pricing, space allocation, and operational procedures.
- **Long-Term Planning:** Use accumulated data to understand trends, predict future demands, and plan for expansions or enhancements in service offerings.
- **Feedback Integration:** Regularly collect and analyze user feedback to continuously refine and improve the parking experience, aligning it more closely with user needs and expectations.

4. Business Requirements:

ParkEasy requires comprehensive data analytics across various dimensions to optimize parking operations, improve customer experience, and increase revenue. The following business use cases drive the design of the data warehouse:

1. **Parking Slot Utilization**: Identifying which slots are used most frequently and during what times. This allows for peak-time pricing adjustments and better resource allocation.
2. **Revenue Tracking**: Monitoring revenue from parking bookings, memberships, and discounts. This helps in forecasting and measuring the performance of different lots.
3. **Incident Tracking**: Analyzing parking-related incidents to improve safety and reduce risks. The severity and frequency of incidents can be tracked to improve operational procedures.
4. **Customer Membership Analysis**: Tracking membership usage, including renewals, discounts, and membership levels to tailor services to high-value customers.

5. **Location and Region Insights**: Understanding which geographic areas are generating the most revenue, experiencing the highest booking rates, or facing the most incidents.

5. About the data

The success and operational excellence of the PARKeasy system are fundamentally reliant on its sophisticated data architecture, which is designed to handle two key types of data: Static Reference Data and Transactional Data. Each type plays a crucial role in supporting the system's functionalities, from real-time operations to strategic decision-making. Below is a detailed explanation of how these data types are utilized within PARKeasy:

5a) Static Reference Data

Static reference data serves as the backbone for the system's operational framework, providing stable and consistent information that helps structure and categorize the more dynamic transactional data. This data is essential for defining the system's operational parameters and user interface:

- **Membership Categories:** These are predefined categories that classify users into different segments such as Corporate, Individual, or Family memberships. This classification helps in tailoring services and pricing to different user needs and preferences. For example, corporate accounts might be eligible for bulk space bookings or dedicated areas, whereas family memberships could include additional facilities like nearby playground access.
- **Parking Zones and Spaces:** Each parking facility managed by PARKeasy is divided into distinct zones and designated spaces. This data includes identifiers for each space and zone, which are critical for navigating users to the right location. The system uses this information to dynamically allocate parking spaces based on current availability and user preferences, thereby optimizing the use of the facility's capacity.
- **Facility Locations and Pricing Models:** Information about the geographical locations of parking facilities and their corresponding pricing models is crucial. This includes static details about the facility's address, the number of available spaces, special amenities, and the pricing rate per hour or day. Pricing models may vary by location depending on local demand, special events, or peak hours, and this data allows the system to provide transparent pricing information to users.

5b) Transactional Data

Transactional data is dynamic and continuously updated, capturing the details of daily operations and user interactions within the PARKeasy system. This data type is vital for the real-time functionality of the system:

- **Parking Reservations and Payments:** Every time a user makes a reservation or payment, the system records detailed transaction data, including the time of booking, the duration of the parking, vehicle details, user identification, and the amount paid. This information is used not only for operational purposes but also for analyzing patterns in user behavior, determining peak times, and adjusting pricing models accordingly.
- **Vehicle Check-ins and Check-outs:** Logs of vehicle entries and exits are meticulously recorded, providing real-time data on space occupancy and turnover rates. This data is

essential for monitoring the flow of traffic into and out of the facility, ensuring that space allocation is optimized and that users can find parking with minimal delay.

5c) Slowly- and not so slowly- changing dimensional data

The dataset includes both slowly and not-so-slowly changing data:

- Slowly Changing Data: Dimensional data such as ParkingLot and ParkingSlot are relatively stable and represent static information that rarely changes.
- Not-So-Slowly Changing Data: Transactional data like Bookings, Payments, and Incidents change frequently. New bookings and payments occur in real-time, and incidents are reported regularly.

The integration and effective management of both static and transactional data enable the PARKeasy system to perform complex tasks such as dynamic space allocation, real-time user guidance, and operational analytics. By leveraging this comprehensive data structure, PARKeasy not only enhances immediate operational efficiency but also establishes a robust foundation for ongoing improvements and adaptations to meet future urban demands and technological advancements. This data-driven approach ensures that the system remains at the forefront of parking management technology, continuously evolving to better serve its users and the urban environment.

5d) Data Source:

The data warehouse for this project was designed to integrate data from two primary sources, ensuring a comprehensive and accurate data environment for analysis. The data sources used are as follows:

OLTP Database:

- The primary data source for the ETL process was an operational OLTP (Online Transaction Processing) database. This OLTP system contains transactional data generated manually using unput statements, including entities such as Admin, ParkingLot, ParkingSlot, Booking, Payment, Incident, and Member. This source provides transactional records that form the basis for the fact tables in the data warehouse.

```
INSERT INTO Admin (AdminID, Name, Address, Email, PhoneNo, Username)
VALUES
(1, 'John Smith', '123 Main St', 'john.smith@email.com', '555-1234', 'john_smith'),
(2, 'Alice Johnson', '456 Oak St', 'alice.johnson@email.com', '555-5678', 'alice_johnson'),
(3, 'Bob Brown', '789 Pine St', 'bob.brown@email.com', '555-4321', 'bob_brown'),
(4, 'Emily Davis', '101 Elm St', 'emily.davis@email.com', '555-8765', 'emily_davis'),
(5, 'David White', '202 Maple St', 'david.white@email.com', '555-9876', 'david_white'),
(6, 'Samantha Miller', '303 Birch St', 'samantha.miller@email.com', '555-3456', 'samantha_miller'),
(7, 'Michael Wilson', '404 Cedar St', 'michael.wilson@email.com', '555-6543', 'michael_wilson'),
(8, 'Olivia Moore', '505 Pine St', 'olivia.moore@email.com', '555-7890', 'olivia_moore'),
(9, 'William Taylor', '606 Oak St', 'william.taylor@email.com', '555-2345', 'william_taylor'),
(10, 'Emma Harris', '707 Elm St', 'emma.harris@email.com', '555-5432', 'emma_harris'),
(41, 'Christopher Hall', '111 Cedar St', 'christopher.hall@email.com', '555-9876', 'christopher_hall'),
```

```sql
INSERT INTO ParkingLot (LotID, Name, Location, Capacity)
VALUES
(1111, 'Downtown Parking', 'Main Street', 150),
(1112, 'Central Plaza Parking', 'Center Avenue', 200),
(1113, 'Greenfield Park Parking', 'Park Street', 100),
(1114, 'Metro Mall Parking', 'Shopping Mall', 250),
(1115, 'Tech Hub Parking', 'Innovation Avenue', 120),
(1116, 'Riverside Parking', 'Riverfront Drive', 180);
```

```sql
INSERT INTO Booking (BookingID, Date, StartTime, EndTime, Status, TransactionID, SlotID)
VALUES
(21, '2023-12-18', '15:30:00', '19:30:00', 'Confirmed', 100021, 10401),
(22, '2023-12-19', '12:15:00', '16:15:00', 'Pending', 100022, 10402),
(23, '2023-12-20', '10:00:00', '14:00:00', 'Cancelled', 100023, 10403),
(24, '2023-12-21', '14:45:00', '18:45:00', 'Confirmed', 100024, 10404),
(25, '2023-12-22', '11:30:00', '15:30:00', 'Pending', 100025, 10405),
(26, '2023-12-23', '09:15:00', '13:15:00', 'Cancelled', 100026, 10406),
(27, '2023-12-24', '13:00:00', '17:00:00', 'Confirmed', 100027, 10407),
(28, '2023-12-25', '10:45:00', '14:45:00', 'Pending', 100028, 10408),
```

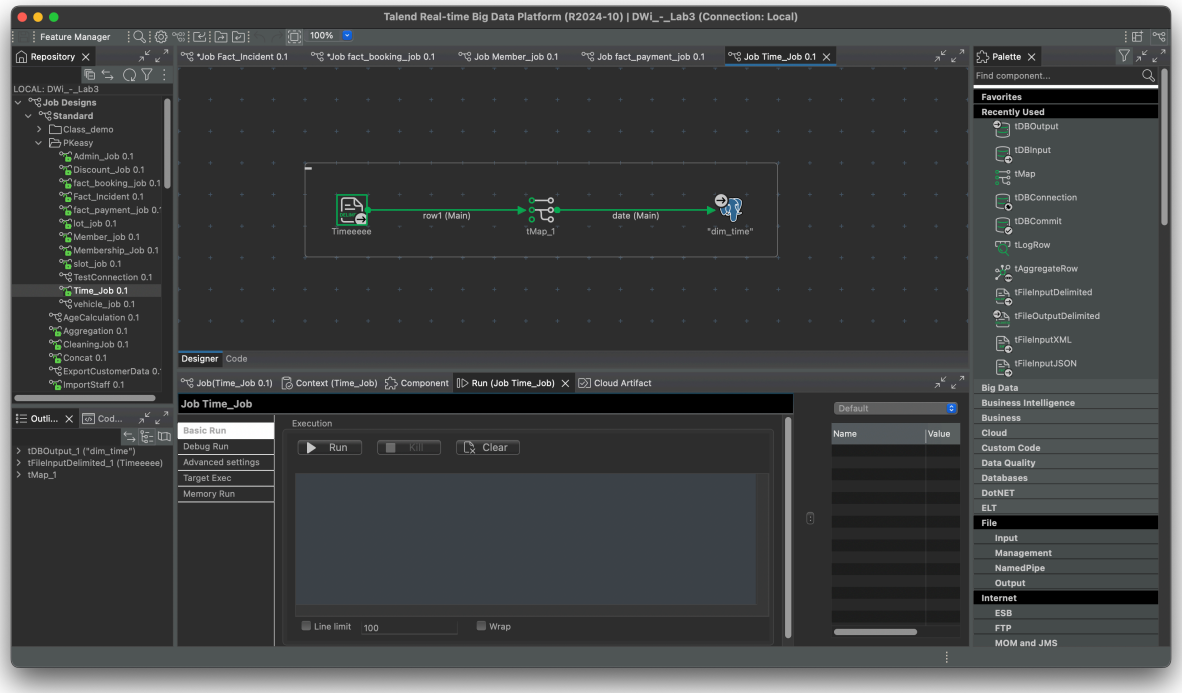
And so on…

Prepopulated Time Dimension:

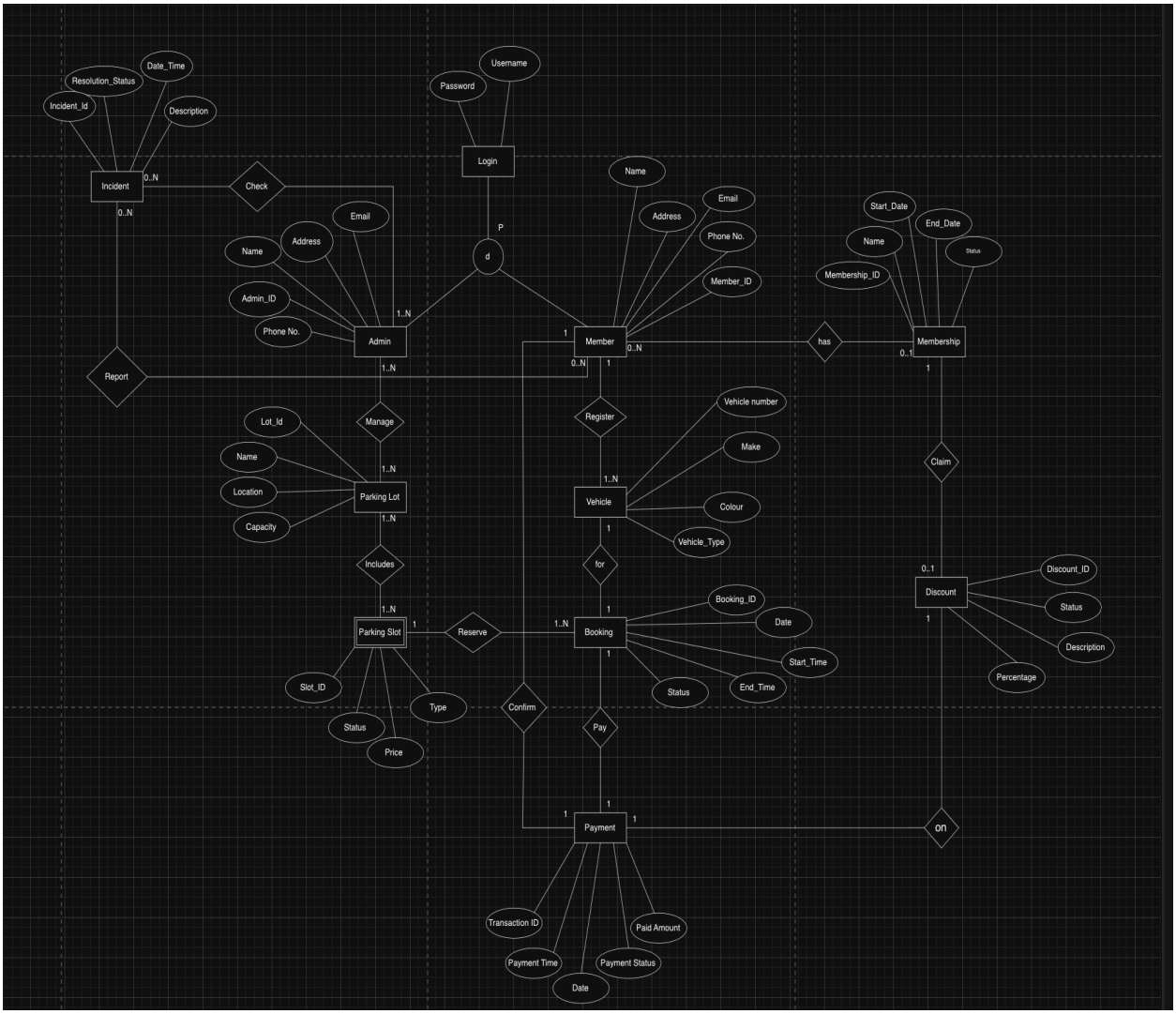
- The time dimension in the data warehouse was prepopulated as a static reference table, providing detailed temporal attributes such as year, quarter, month, day, and time of day. This prepopulated time dimension serves as a reference that is essential for time-based analysis across the data warehouse.
- Using a prepopulated time dimension provides flexibility and efficiency, as it allows for consistent and easy querying of time-related data without needing frequent updates. The time dimension is fundamental for creating time-based reports, tracking trends over periods, and supporting calculations like monthly revenue, booking durations, and incident resolutions over time.

TimeID	Year	Quarter	Month	Day	Date
20230101	2023	1	1	1	01-01-2023
20230102	2023	1	1	2	02-01-2023
20230103	2023	1	1	3	03-01-2023
20230104	2023	1	1	4	04-01-2023
20230105	2023	1	1	5	05-01-2023
20230106	2023	1	1	6	06-01-2023
20230107	2023	1	1	7	07-01-2023
20230108	2023	1	1	8	08-01-2023
20230109	2023	1	1	9	09-01-2023
20230110	2023	1	1	10	10-01-2023

20251221	2025	4	12	21	21-12-2025
20251222	2025	4	12	22	22-12-2025
20251223	2025	4	12	23	23-12-2025
20251224	2025	4	12	24	24-12-2025
20251225	2025	4	12	25	25-12-2025
20251226	2025	4	12	26	26-12-2025
20251227	2025	4	12	27	27-12-2025
20251228	2025	4	12	28	28-12-2025
20251229	2025	4	12	29	29-12-2025
20251230	2025	4	12	30	30-12-2025
20251231	2025	4	12	31	31-12-2025


Talend Real-time Big Data Platform (R2024-10) | DWi_-_Lab3 (Connection: Local)
Feature Manager
100%
Repository
LOCAL: DWi_-_Lab3
Job Designs
Standard
Class_demo
PKeasy
Admin_Job 0.1
Discount_Job 0.1
fact_booking_job 0.1
Fact_Incident 0.1
fact_payment_job 0.1
lot_job 0.1
Member_job 0.1
Membership_Job 0.1
slot_job 0.1
TestConnection 0.1
Time_Job 0.1
vehicle_job 0.1
AgeCalculation 0.1
Aggregation 0.1
CleaningJob 0.1
Concat 0.1
ExportCustomerData 0.1
ImportStaff 0.1
*Job Fact_Incident 0.1
*Job fact_booking_job 0.1
Job Member_job 0.1
Job fact_payment_job 0.1
Job Time_Job 0.1
Timeeeee
row1 (Main)
tMap_1
date (Main)
"dim_time"
Designer
Code
Job(Time_Job 0.1)
Context (Time_Job)
Component
Run (Job Time_Job)
Cloud Artifact
Job Time_Job
Basic Run
Debug Run
Advanced settings
Target Exec
Memory Run
Execution
Run
Kill
Clear
Line limit
100
Wrap
Default
Name
Value
Outli...
Cod...
tDBOutput_1 ("dim_time")
tFileInputDelimited_1 (Timeeeee)
tMap_1
Palette
Find component...
Favorites
Recently Used
tDBOutput
tDBInput
tMap
tDBConnection
tDBCommit
tLogRow
tAggregateRow
tFileInputDelimited
tFileOutputDelimited
tFileInputXML
tFileInputJSON
Big Data
Business Intelligence
Business
Cloud
Custom Code
Data Quality
Databases
DotNET
ELT
File
Input
Management
NamedPipe
Output
Internet
ESB
FTP
MOM and JMS

6. Extended Entity-Relational Model:



6a) Relational Model:

Primary Key- PK
Foreign Key- FK

-**Admin** (<u>AdminID</u>, Name, Address, Email, PhoneNo, <u>Username</u>)
PK: AdminID
FK: Username refers to Login – Not NULL

-ParkingLot (<u>LotID</u>, Name, Location, Capacity)
PK: LotID

-Manage (<u>AdminID</u>, <u>LotID</u>)

FK: AdminID refers to Admin – Not NULL
LotID refers to ParkingLot – Not NULL

-Includes (SlotID, LotID)
FK: SlotID refers to ParkingSlot – Not NULL
LotID refers to ParkingLot – Not NULL

-ParkingSlot (SlotID, Status, Price, Type)
PK: SlotID

-Booking (BookingID, Date, StartTime, EndTime, Status, TransactionID, SlotID)
PK: BookingID
FK: TransactionID refers to Payment – Not NULL
SlotID refers to ParkingSlot – Not NULL

-Login (Username, Password)
PK: Username

-Vehicle (VehicleNumber, Make, Color, VehicleType, MemberID, BookingID)
PK: VehicleNumber
FK: MemberID refers to Member – Not NULL
BookingID refers to Booking – Not NULL
-Member (MemberID, PhoneNo, Email, Address, Name, MembershipID, Username)
PK: MemberID
FK: MembershipID refers to Membership – NULL Allowed
Username refers to Login- Not NULL

-Membership (MembershipID, Name, StartDate, EndDate, Status, DiscountID)
PK: MembershipID
FK: DiscountID refers to Discount – NULL Allowed

-Discount (DiscountID, Status, Description, Percentage)
PK: DiscountID

-Payment (TransactionID, Date, Time, PaymentStatus, Amount, MemberID)
PK: TransactionID
FK: MemberID refers to Member – Not NULL

-Incident (IncidentID, ResolutionStatus, Date_Time, Description, MemberID)
PK: IncidentID

FK: MemberID refers to Member – NULL Allowed

-Inspect (IncidentID, AdminID)
FK: IncidentID refers to Incident – Not NULL
AdminID refers to Admin – Not NULL

-Report (IncidentID, MemberID)
FK: IncidentID refers to Incident – Not NULL
MemberID refers to Member – Not NULL

*The above relational model is normalized

7. Conceptual Datawarehouse Model:

The conceptual model for the ParkEasy Data Warehouse lays the foundation for understanding the main entities and processes the system needs to analyze. It is the high-level design that provides the overall structure of the data warehouse, focusing on what information needs to be stored and how it relates to business activities.

Key Components of the Conceptual Model:

7a) Fact Tables: Fact tables contain measurable, quantitative data and foreign keys that reference dimension tables. They form the core of the data warehouse as they store the events or transactional data.

1. Booking Fact Table

- **Purpose**: Stores information related to parking bookings.
- **Measures**:
  - Booking ID (PK): Unique identifier for each booking.
  - Transaction ID (FK): Links to the payment fact table.
  - Status: The status of the booking (e.g., active, completed, canceled).
- **Cardinality**:
  - 1, N bookings can occur for each Parking Slot and Member.
- **Relationships**:
  - Linked with dimensions: Parking Slot, Vehicle, Time, Member.
- **Example Query**: Total number of bookings per parking lot by month.

2. **Payment Fact Table**

- **Purpose**: Stores payment details for parking bookings.
- **Measures**:
  - Transaction ID (PK): Unique identifier for each payment.
  - Payment Status: Status of the payment (e.g., completed, pending).
  - Amount: The total amount paid.
- **Cardinality**:
  - 1, N payments are linked to each Member and Booking.
- **Relationships**:
  - Linked with dimensions: Time, Member.
- **Example Query**: Total revenue collected by year from all members.

3. **Incident Fact Table**

- **Purpose**: Logs incidents related to parking activities.
- **Measures**:
  - Incident ID (PK): Unique identifier for each incident.
  - Resolution Status: Indicates if the incident is resolved.

    - Incident Count: Tracks the number of incidents per period.
- **Cardinality**:
    - 1, N incidents can occur for each Admin and Parking Slot.
- **Relationships**:
    - Linked with dimensions: Admin, Parking Slot, Incident Details, Time.
- **Example Query**: Monthly count of incidents by parking lot.

7b) Dimension Tables: Dimension tables store descriptive information about the business entities involved in the fact events. They provide context to the facts and allow the users to slice, dice, and roll-up data for analysis.

1. **Time Dimension**
- Attributes:
    - Date_Time, Day_of_Week, Time
- Cardinality:
    - 1, N bookings and payments can be associated with each time period.
- Hierarchy:
    - Time → Year → Quarter → Month → Day.
- Example Query: Number of bookings per quarter for each parking lot.

2. **Member Dimension**
- Attributes:
    - Member ID (PK): Unique identifier for each member.
    - Member_Name, Phone No, Email, Address.
    - Membership ID (FK): Links to the Membership dimension.
- Cardinality:
    - 1, N bookings and payments can be made by each member.
- Example Query: Number of bookings made by premium members in 2024.

3. **Parking Slot Dimension**
- Attributes:
    - Slot ID (PK): Unique identifier for each parking slot.
    - Type: Type of parking slot
    - Status: Availability of the slot (free, occupied).
    - Price: Pricing for the parking slot.
- Cardinality:
    - 1, N bookings can be associated with each parking slot.
- Example Query: Parking slot usage analysis by status and price.

4. **Parking Lot Dimension**
- Attributes:
    - Lot ID (PK): Unique identifier for each parking lot.
    - Location ID (FK): Links to the Location dimension.
    - Name: Name of the lot
    - Capacity: Number of parking slots available.
- Cardinality:

    - 1, N parking slots are contained in each parking lot.
- Example Query: Total revenue per parking lot per year.

5. **Vehicle Dimension**
- Attributes:
    - Vehicle Number (PK): Unique identifier for each vehicle.
    - Make ID (FK): Links to the vehicle Make dimension.
    - Color
- Cardinality:
    - 1, N bookings can be associated with each vehicle.
- Example Query: Popular vehicle types used in the downtown parking lot.

6. **Admin Dimension**
- Attributes:
    - Admin ID (PK): Unique identifier for each administrator.
    - Admin_Name, Email, Phone No., Address
- Cardinality:
    - 1, N incidents can be managed by each admin.
- Example Query: Number of incidents resolved by each administrator.

7. **Incident Details Dimension**
- Attributes:
    - Type, Description, Severity.
- Cardinality:
    - 1, N incidents can be associated with each incident type.
- Example Query: Severity analysis of incidents reported in 2023.

8. **Membership Dimension**
- Attributes:
    - Membership ID (PK): Unique identifier for each membership.
    - Membership_Name, Status, Membership Level.
- Cardinality:
    - 1, N members can hold a membership.
- Example Query: Booking analysis based on membership level (e.g., premium vs. basic).

9. **Discount Dimension**
- Attributes:
    - Discount ID (PK): Unique identifier for each discount.
    - Percentage, Description, Status
- Cardinality:
    - 1, N memberships may have a discount applied.
- Example Query: Total discount provided to premium members in 2024.

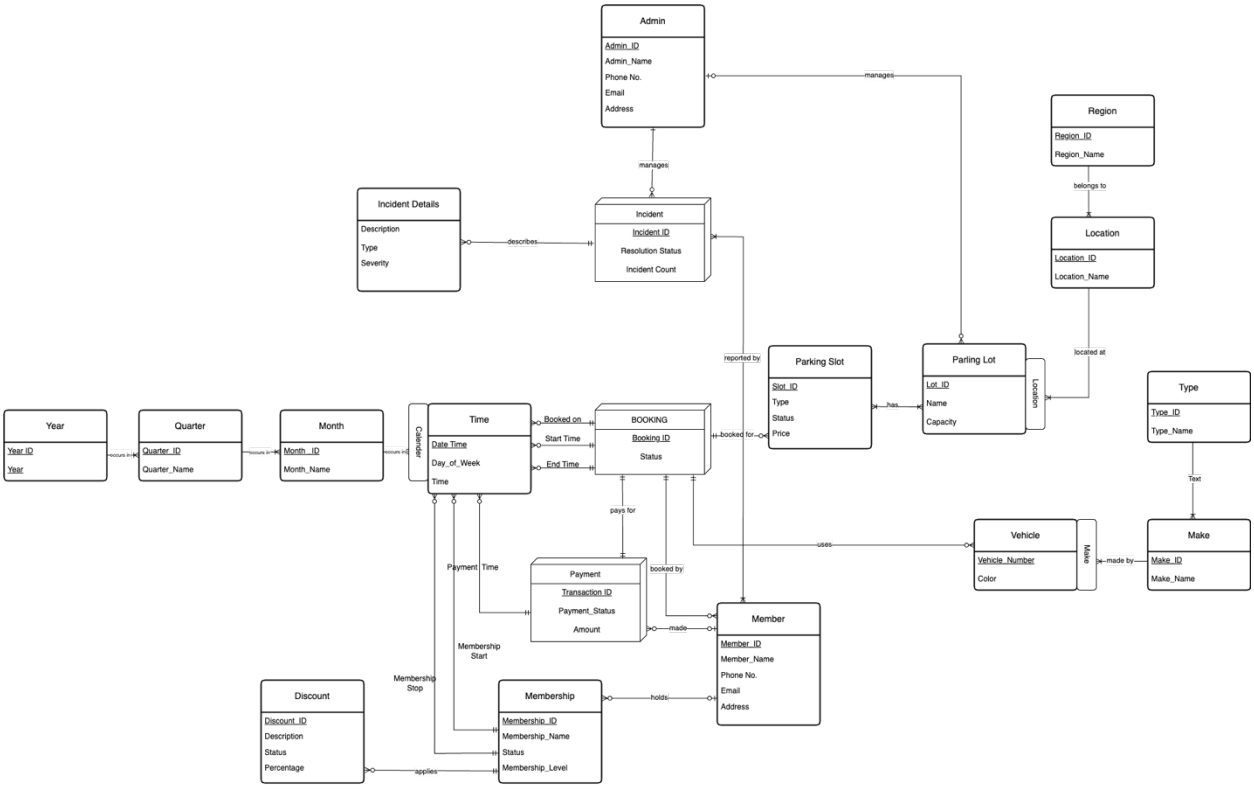
10. **Location and Region Dimension**
- Attributes:
    - Location ID (PK): Unique identifier for each location.

    - Location Name, Region ID (FK): Links to the Region dimension.
- Cardinality:
    - 1, N parking lots belong to a location.
    - 1, N locations belong to a region.
- Hierarchy:
    - Region → Location → Parking Lot → Parking Slot.
- Example Query: Total bookings by region for each year.

11. **Vehicle Make and Type Dimension**
- Attributes:
    - Make ID (PK): Unique identifier for each vehicle make.
    - Make_Name, Type ID (FK): Links to the Type dimension.
- Cardinality:
    - 1, N vehicle makes are of a specific type.
- Example Query: Analyze vehicle types used for parking in different locations.

Hierarchies:
- **Time Hierarchy**: Year → Quarter → Month → Day.
- **Location Hierarchy**: Region → Location → Parking Lot → Parking Slot.
- **Vehicle Hierarchy**: Type → Make → Vehicle.

7c) Conceptual model diagram:



8. Logical Datawarehouse Model:

The logical model refines the conceptual model by introducing primary keys (PK), foreign keys (FK), and relationships between the tables. It also defines the surrogate keys to uniquely identify each record, even when natural keys (like booking IDs or member IDs) exist.

Key Components of the Logical Model:

8a) Fact Tables (with Surrogate Keys):

1. Booking Fact Table

- Primary Key:
  - Booking_ID (PK) (Surrogate Key)
- Attributes:
  - Booking_ID (PK)
  - BookingDate (PK)
  - TransactionID (FK)
  - SlotID (FK)
  - VehicleID (FK)
  - MemberID (FK)
  - StartTime (FK) (from Time dimension)
  - EndTime (FK) (from Time dimension)
  - Status

2. **Payment Fact Table**

- Primary Key:
  - Payment_ID (PK) (Surrogate Key)
- Attributes:
  - Payment_ID (PK)
  - Transaction_ID (FK)
  - Time (FK) (from Time dimension)
  - MemberID (FK)
  - Status
  - Amount

3. **Incident Fact Table**

- Primary Key:
  - Incident_ID (PK) (Surrogate Key)
- Attributes:
  - Incident_ID (PK)
  - Time (FK) (from Time dimension)
  - AdminID (FK)
  - MemberID (FK)
  - Status
  - Count

8b) Dimension Tables (with Surrogate Keys):

1. **Time Dimension**

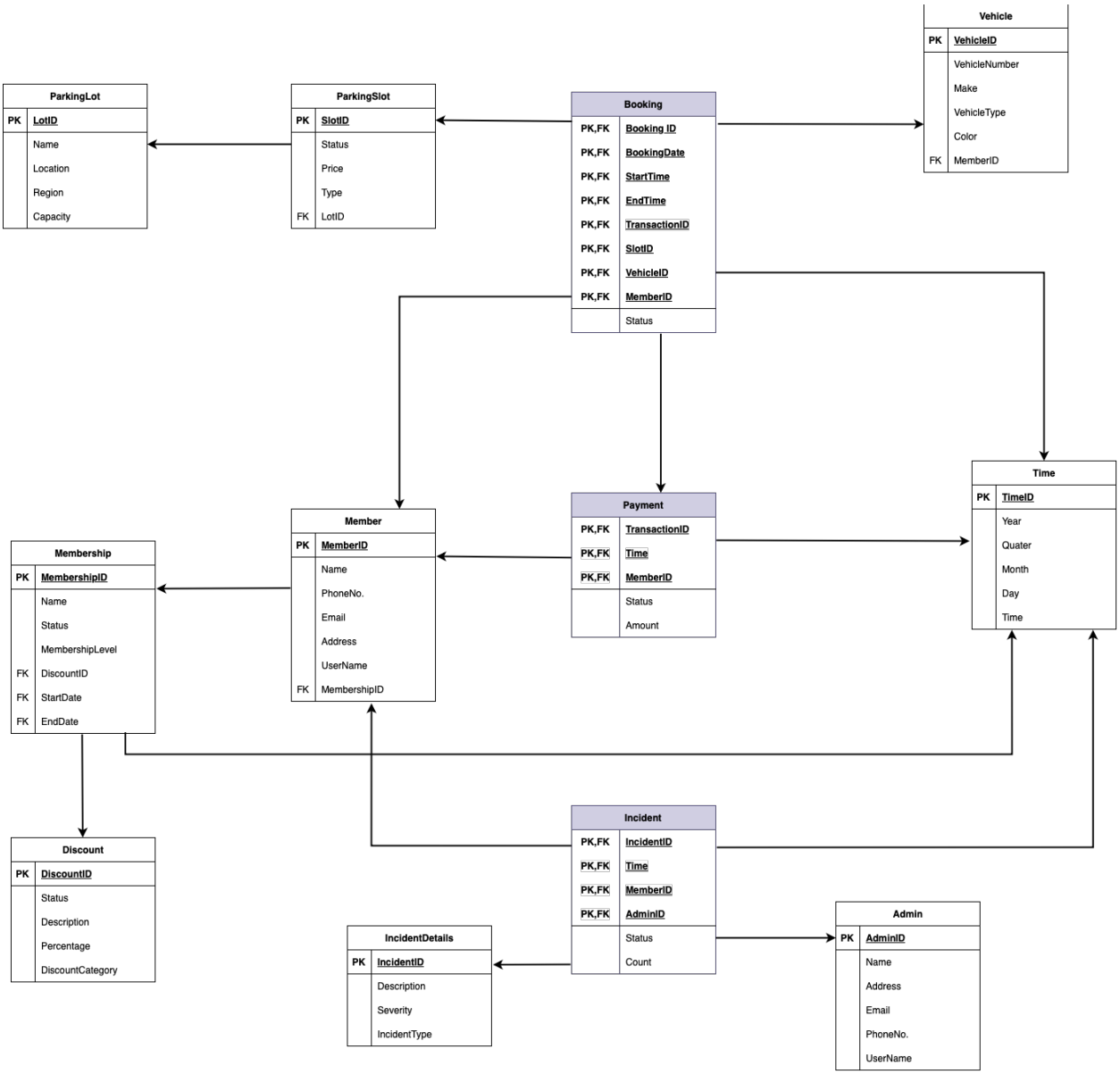
- Primary Key:
    - TimeID (PK) (Surrogate Key)
- Attributes:
    - TimeID (PK)
    - Year
    - Quarter
    - Month
    - Day
    - Time

2. **Member Dimension**
    - Primary Key:
        - MemberID (PK)
    - Attributes:
        - Name
        - PhoneNo
        - Email
        - Address
        - UserName
        - Membership_ID (FK)
3. **Parking Slot Dimension**
    - Primary Key:
        - SlotID (PK)
    - Attributes:
        - Status
        - Price
        - Type
        - Lot_ID (FK)
4. **Parking Lot Dimension**
    - Primary Key:
        - LotID (PK)
    - Attributes:
        - Name
        - Location
        - Region
        - Capacity
5. **Vehicle Dimension**
    - Primary Key:
        - VehicleID (PK)
    - Attributes:
        - VehicleNumber
        - Color
        - MakeID
        - VehicleType
        - MemberID (FK)
6. **Admin Dimension**
    - Primary Key:

    - AdminID (PK)
- Attributes:
    - Name
    - Address
    - Email
    - PhoneNo
    - UserName

7. **Incident Details Dimension**
    - Primary Key:
        - IncidentID (PK)
    - Attributes:
        - IncidentType
        - Severity
        - Description
8. **Membership Dimension**
    - Primary Key:
        - MembershipID (PK)
    - Attributes:
        - Name
        - Status
        - MembershipLevel
        - Discount_ID (FK)
        - StartDate (FK)
        - EndDate (FK)
9. **Discount Dimension**
    - Primary Key:
        - DiscountID (PK)
    - Attributes:
        - Status
        - Description
        - Percentage
        - DiscountCategory

8c) Logical Model diagram:


Vehicle
PK VehicleID
VehicleNumber
Make
VehicleType
Color
FK MemberID
ParkingLot
PK LotID
Name
Location
Region
Capacity
ParkingSlot
PK SlotID
Status
Price
Type
FK LotID
Booking
PK,FK Booking ID
PK,FK BookingDate
PK,FK StartTime
PK,FK EndTime
PK,FK TransactionID
PK,FK SlotID
PK,FK VehicleID
PK,FK MemberID
Status
Time
PK TimeID
Year
Quater
Month
Day
Time
Payment
PK,FK TransactionID
PK,FK Time
PK,FK MemberID
Status
Amount
Member
PK MemberID
Name
PhoneNo.
Email
Address
UserName
FK MembershipID
Membership
PK MembershipID
Name
Status
MembershipLevel
FK DiscountID
FK StartDate
FK EndDate
Discount
PK DiscountID
Status
Description
Percentage
DiscountCategory
Incident
PK,FK IncidentID
PK,FK Time
PK,FK MemberID
PK,FK AdminID
Status
Count
IncidentDetails
PK IncidentID
Description
Severity
IncidentType
Admin
PK AdminID
Name
Address
Email
PhoneNo.
UserName

9. Database Implementation:

1) Fact Tables creation:

```sql
CREATE TABLE FACT_Payment (
    PaymentID SERIAL PRIMARY KEY,
    TimeID INT NOT NULL REFERENCES DIM_Time(TimeID),
    PaymentStatus VARCHAR(50),
    Amount FLOAT NOT NULL CHECK (Amount >= 0),
    MemberID INT NOT NULL REFERENCES DIM_Member(MemberID)
);

Select * from FACT_Payment;
```

Data Output | Messages | Notifications

	paymentid [PK] integer	timeid integer	paymentstatus character varying (50)	amount double precision	memberid integer

```sql
CREATE TABLE FACT_Incident (
    IncidentID SERIAL PRIMARY KEY,
    ResolutionStatus VARCHAR(50),
    TimeID INT NOT NULL REFERENCES DIM_Time(TimeID),
    IncidentCount INT NOT NULL CHECK (IncidentCount >= 0),
    MemberID INT REFERENCES DIM_Member(MemberID),
    AdminID INT NOT NULL REFERENCES DIM_Admin(AdminID)
);

Select * from FACT_Incident;
```

Data Output | Messages | Notifications

	incidentid [PK] integer	resolutionstatus character varying (50)	timeid integer	incidentcount integer	memberid integer	adminid integer

```sql
CREATE TABLE FACT_Booking (
    BookingID SERIAL PRIMARY KEY,
    BookingDateID INT NOT NULL REFERENCES DIM_Time(TimeID),
    StartTimeID INT NOT NULL REFERENCES DIM_Time(TimeID),
    EndTimeID INT NOT NULL REFERENCES DIM_Time(TimeID),
    Status VARCHAR(50),
    TransactionID INT REFERENCES FACT_Payment(PaymentID),
    SlotID INT NOT NULL REFERENCES DIM_ParkingSlot(SlotID),
    VehicleID INT REFERENCES DIM_Vehicle(VehicleID),
    MemberID INT NOT NULL REFERENCES DIM_Member(MemberID)
);

Select * from FACT_Booking;
```

Data Output | Messages | Notifications

bookingid [PK] integer	bookingdateid integer	starttimeid integer	endtimeid integer	status character varying (50)	transactionid integer	slotid integer	vehicleid integer	memberid integer

2) Dimension Tables creation:

```sql
CREATE TABLE DIM_Time (
    TimeID SERIAL PRIMARY KEY,
    Year INT NOT NULL CHECK (Year > 0),
    Quarter INT CHECK (Quarter >= 1 AND Quarter <= 4),
    Month INT CHECK (Month >= 1 AND Month <= 12),
    Day INT CHECK (Day >= 1 AND Day <= 31),
    TimeOfDay TIME
);

Select * from DIM_Time;
```

Data Output | Messages | Notifications

timeid [PK] integer	year integer	quarter integer	month integer	day integer	timeofday time without time zone

```sql
CREATE TABLE DIM_Admin (
    AdminID SERIAL PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Address VARCHAR(255),
    Email VARCHAR(100) NOT NULL UNIQUE,
    PhoneNo VARCHAR(15),
    Username VARCHAR(50) NOT NULL UNIQUE
);

Select * from DIM_Admin;
```

Data Output | Messages | Notifications

adminid [PK] integer	name character varying (100)	address character varying (255)	email character varying (100)	phoneno character varying (15)	username character varying (50)

```sql
CREATE TABLE DIM_ParkingLot (
    LotID SERIAL PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Location VARCHAR(100),
    Region VARCHAR(100),
    Capacity INT NOT NULL CHECK (Capacity >= 0)
);

Select * from DIM_ParkingLot;
```

Data Output | Messages | Notifications

lotid [PK] integer	name character varying (100)	location character varying (100)	region character varying (100)	capacity integer

```sql
CREATE TABLE DIM_ParkingSlot (
    SlotID SERIAL PRIMARY KEY,
    Status VARCHAR(50),
    Price FLOAT NOT NULL CHECK (Price >= 0),
    Type VARCHAR(50),
    LotID INT NOT NULL REFERENCES DIM_ParkingLot(LotID)
);

Select * from DIM_ParkingSlot;
```

Data Output | Messages | Notifications

	slotid [PK] integer	status character varying (50)	price double precision	type character varying (50)	lotid integer

```sql
CREATE TABLE DIM_Discount (
    DiscountID SERIAL PRIMARY KEY,
    Status VARCHAR(50),
    Description TEXT,
    Percentage FLOAT CHECK (Percentage >= 0 AND Percentage <= 100),
    DiscountCategory VARCHAR(50)
);

Select * from DIM_Discount;
```

Data Output | Messages | Notifications

	discountid [PK] integer	status character varying (50)	description text	percentage double precision	discountcategory character varying (50)

Query | Query History

```sql
CREATE TABLE DIM_Membership (
    MembershipID SERIAL PRIMARY KEY,
    Name VARCHAR(100) NOT NULL,
    Status VARCHAR(50),
    MembershipLevel VARCHAR(50),
    DiscountID INT REFERENCES DIM_Discount(DiscountID),
    StartDateID INT REFERENCES DIM_Time(TimeID),
    EndDateID INT REFERENCES DIM_Time(TimeID)
);

Select * from DIM_Membership;
```

Data Output | Messages | Notifications

membershipid [PK] integer	name character varying (100)	status character varying (50)	membershiplevel character varying (50)	discountid integer	startdateid integer	enddateid integer

```sql
CREATE TABLE DIM_Member (
    MemberID SERIAL PRIMARY KEY,
    PhoneNo VARCHAR(15),
    Email VARCHAR(100) NOT NULL UNIQUE,
    Address VARCHAR(255),
    Name VARCHAR(100) NOT NULL,
    MembershipID INT REFERENCES DIM_Membership(MembershipID),
    Username VARCHAR(50) NOT NULL UNIQUE
);

Select * from DIM_Member;
```

Data Output | Messages | Notifications

memberid [PK] integer	phoneno character varying (15)	email character varying (100)	address character varying (255)	name character varying (100)	membershipid integer	username character varying (50)

```sql
CREATE TABLE DIM_Vehicle (
    VehicleID SERIAL PRIMARY KEY,
    VehicleNumber VARCHAR(20) NOT NULL UNIQUE,
    Make VARCHAR(50),
    VehicleType VARCHAR(50),
    Color VARCHAR(50),
    MemberID INT REFERENCES DIM_Member(MemberID)
);

Select * from DIM_Vehicle;
```

Data Output | Messages | Notifications

vehicleid [PK] integer	vehiclenumber character varying (20)	make character varying (50)	vehicletype character varying (50)	color character varying (50)	memberid integer

```sql
CREATE TABLE DIM_IncidentDetails (
    IncidentID SERIAL PRIMARY KEY,
    Description TEXT,
    Severity VARCHAR(50),
    IncidentType VARCHAR(50)
);

Select * from DIM_IncidentDetails;
```

Data Output | Messages | Notifications

incidentid [PK] integer	description text	severity character varying (50)	incidenttype character varying (50)

10. Primary Events

1. **Event: Vehicle Parking Booking**
   - **Description:** A customer reserves a parking slot for a vehicle in a specific parking lot.
   - **Data Captured:** Customer details (Member ID), Vehicle information (Vehicle ID, Type), Parking Slot (Slot ID), Parking Lot (Lot ID), Booking Time, Duration, and Payment Details.
   - **Tables Involved:** Booking Fact, Member, Vehicle, ParkingSlot, ParkingLot, Payment, Time.
   - **Dependencies:** A valid member and vehicle should exist in the system. The parking slot and lot must be available.
   - **Outcome:** A new record in the Booking fact table, capturing the transaction details.

2. **Event: Payment Processing for Parking Booking**
   - **Description**: Payment is processed for the booking of a parking slot, including any discounts applied and the payment method used.
   - **Data Captured**: Payment Amount, Discount, Payment Method, Booking ID, Member ID, Payment Time.
   - **Tables Involved**: Payment Fact, Booking Fact, Member, Time.
   - **Dependencies**: A valid booking must exist in the Booking Fact table, and payment details must correspond to the member.
   - **Outcome**: A new record in the Payment Fact table.

3. **Event: Parking Slot Assignment**

- **Description**: When a parking slot is assigned to a booking, and the vehicle occupies the slot.
- **Data Captured**: Slot ID, Booking ID, Member ID, Vehicle ID, Parking Start Time, Parking Duration.
- **Tables Involved**: ParkingSlot, Booking Fact, Member, Vehicle, Time.
- **Dependencies**: Valid parking slots and bookings must exist. Slots must be available for assignment.
- **Outcome**: The parking slot is assigned, and the status is updated in the ParkingSlot table.

4. **Event: Membership Registration and Renewal**
   - **Description**: A customer registers for a parking membership or renews an existing one.
   - **Data Captured**: Member ID, Membership Level, Start Date, End Date, Discount Applied, Payment Details.
   - **Tables Involved**: Membership Dimension, Member, Payment Fact, Time.
   - **Dependencies**: Valid member details must exist, and the membership type must be valid.
   - **Outcome**: The member's membership record is updated in the Membership table, and the corresponding payment is recorded.

5. **Event: Incident Reporting**
   - **Description**: An incident (such as vehicle damage or theft) is reported during parking, which is inspected by an admin.
   - **Data Captured**: Incident ID, Incident Type, Severity, Incident Date, Admin Assigned, Resolution Status.
   - **Tables Involved**: Incident Fact, Admin, Member, Time, IncidentDetails.
   - **Dependencies**: The member and admin involved must exist, and the parking incident details must be recorded.
   - **Outcome**: A new record in the Incident Fact table with relevant details, and an incident is reported for further investigation.

6. **Event: Vehicle Registration**
   - **Description**: A member registers a vehicle in the system to use in future parking bookings.
   - **Data Captured**: Vehicle ID, Vehicle Type, License Plate Number, Member ID, Registration Date.
   - **Tables Involved**: Vehicle, Member.
   - **Dependencies**: A valid member must exist, and vehicle details should be correctly registered.
   - **Outcome**: A new vehicle record is added to the Vehicle table.

7. **Event: Parking Lot or Slot Maintenance**
   - **Description**: Maintenance is conducted on parking slots or parking lots, which temporarily makes them unavailable.
   - **Data Captured**: Lot ID, Slot ID, Maintenance Start Date, Maintenance End Date, Admin Assigned.
   - **Tables Involved**: ParkingSlot, ParkingLot, Admin, Time.

- **Dependencies**: Valid parking slots or lots must be in the system, and an admin must be responsible for the maintenance.
- **Outcome**: Slot or lot availability is updated, and maintenance status is recorded.

11. OLAP Operations:

In order to analyze the above primary events, OLAP (Online Analytical Processing) operations help in slicing and dicing the data for better insights. Below is a list of common OLAP operations that will be used in ParkEasy, along with explanations:

1.Drill-Down
Description:
Drill-down allows users to view data at more detailed levels. For example, instead of viewing yearly booking totals, users can drill down to see monthly, weekly, or daily booking data.

Example 1:
From the total number of bookings in 2023, drill down to view monthly totals for each parking lot.

Drill-Down Query:
DRILLDOWN(Bookings, Time → Month, Year = 2023, ParkingLot)

Example 2:
From the total payments made in 2024, drill down to view the payment totals for each member on a monthly basis.

Drill-Down Query:
DRILLDOWN(Payments, Time → Month, Year = 2024, Member)

---

2. **Slice**
Description:
Slice allows users to focus on a subset of the data based on one dimension.

Example 1:
Slicing the data to view only bookings made by premium members in 2024.

Slice Query:
SLICE(Bookings, MembershipLevel = 'Premium', Year = 2024)
Example 2:
Slicing the data to view only incidents that were resolved in 2023.

Slice Query:
SLICE(Incidents, ResolutionStatus = 'Resolved', Year = 2023)

---

3. **Dice**
Description:

Dice allows users to select specific ranges across multiple dimensions for detailed analysis.

Example 1:
Dicing to view all bookings made in 2023 for SUV vehicles in the downtown parking lot.

Dice Query:
DICE(Bookings, Year = 2023, VehicleType = 'SUV', ParkingLot = 'Downtown')

Example 2:
Dicing to view all payments made in the first quarter of 2024 for premium members.

Dice Query:
DICE(Payments, Quarter = 'Q1', Year = 2024, MembershipLevel = 'Premium')

---

4. **Pivot**
Description:
Pivot allows users to switch rows and columns to view data from different perspectives.

Example 1:
Pivoting from viewing bookings by time (rows: months, columns: days) to viewing bookings by parking lot.

Pivot Query:
PIVOT(Bookings, Time → ParkingLot)

Example 2:
Pivoting from viewing payments by parking lot (rows: parking lot, columns: payment method) to viewing payments by vehicle type.

Pivot Query:
PIVOT(Payments, ParkingLot → VehicleType)

---

5. **Roll-Up**
Description:
Roll-up aggregates data by climbing up a hierarchy, such as aggregating bookings from daily to monthly to yearly totals.

Example 1:
Rolling up booking data from daily to monthly totals for a specific parking lot.

Roll-Up Query:
ROLLUP(Bookings, Time → Month)

Example 2:
Rolling up incident data from specific incidents to quarterly totals for each parking lot.

Roll-Up Query:
ROLLUP(Incidents, Time → Quarter, ParkingLot)

---

6. Drill Across:
Description:
Drill Across allows comparison of facts from two or more cubes that share common dimensions.

Example 1:
Compare the number of bookings and total payments made across different parking lots for the first quarter of 2023.

Drill Across Query:
DRILLACROSS(Bookings, Payments, ParkingLot, Time → Quarter = 'Q1', Year = 2023)

Example 2:
Compare the number of incidents and total payments across different parking lots for the second quarter of 2023.

Drill Across Query:
DRILLACROSS(Incidents, Payments, ParkingLot, Time → Quarter = 'Q2', Year = 2023)

---

7. Rank:
Description:
The **Rank** operation ranks the values of a dimension based on a specific measure.

Example 1:
Rank the parking lots based on total payments received in 2023.

Rank Query:
RANK(ParkingLot, SUM(Payments → Amount), Year = 2023)

Example 2:
Rank the vehicle types based on the total number of bookings made in 2023.

Rank Query:
RANK(VehicleType, COUNT(Bookings → BookingID), Year = 2023)

---

12. ETL Execution:

For this project, the ETL execution involved a series of steps to ensure successful loading and accurate population of all tables in the OLAP schema. Key aspects of the ETL execution included:

1. Extraction:

- Data was extracted from the source OLTP schema tables, which include entities such as Admin, ParkingLot, ParkingSlot, Member, Booking, Payment, Incident, and related tables.
- The extraction process was designed to retrieve data efficiently, ensuring that no critical information was missed. This phase included both static and transactional data, which allowed us to capture a comprehensive view of the data environment.

2. **Transformation**:
   - The transformation step involved several operations to align the source data with the target OLAP schema's requirements. Key transformations included:
     - **Calculating Measures**: Measures like Total_Payment_Amount were derived based on business logic (e.g., calculating the payment amount based on booking duration and slot price).
     - **Surrogate Key Generation**: Surrogate keys were implemented to maintain unique identifiers in the OLAP schema independently of the source system's keys. This approach allows for better control and management of data in the data warehouse.
     - **SCD (Slowly Changing Dimensions)**: Where applicable, transformations included logic to handle slowly changing dimensions, ensuring historical data integrity while capturing updates.
   - Multiple transformations were applied to ensure data accuracy and consistency, achieving data integration across different dimensions and fact tables.

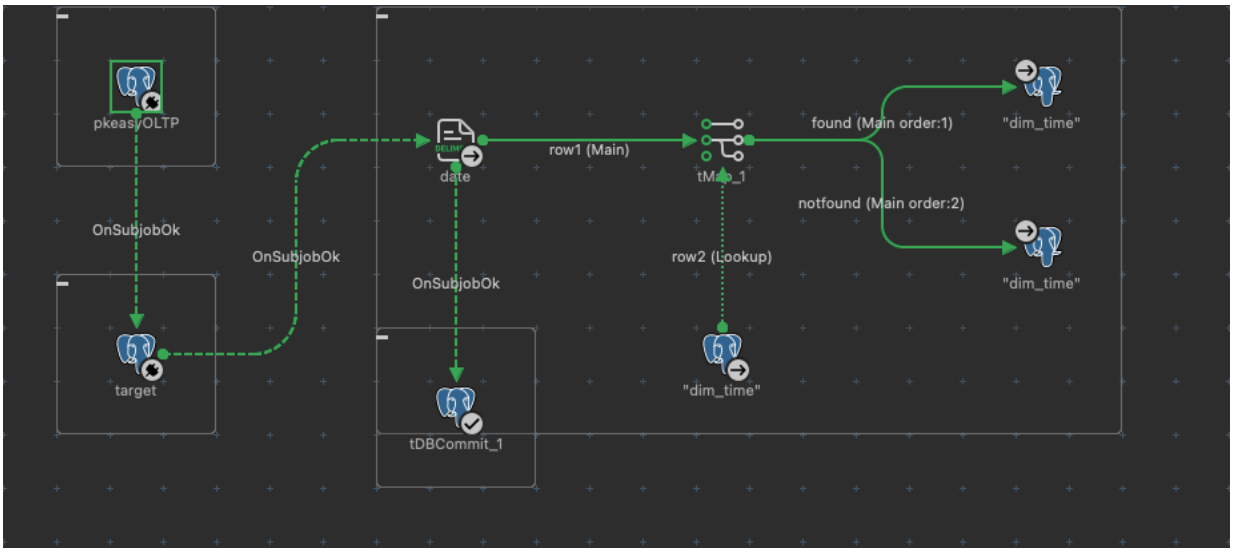
3. **Loading**:
   - The loading process was executed in a defined sequence to respect dependencies and foreign key constraints. For example, dimension tables were loaded before fact tables, and tables like DIM_Time were loaded first to support time-based relationships in other tables.
   - Both **Insert** and **Update** flows were implemented in the ETL pipeline:
     - **Insert Flows**: Initial loading of data was executed to populate empty tables with source data.
     - **Update Flows**: Updates were handled to maintain up-to-date records in cases where data changes over time, especially for transactional data like bookings and payments.
   - A control flow was implemented to handle dependencies and ensure successful execution. This flow checked for successful execution of each step, handling exceptions and ensuring data consistency throughout the process.

4. **Control Flow Execution**:
   - A structured control flow was created to manage the dependencies between tables and ensure that data was loaded in the correct order. The control flow included checks to validate successful completion at each stage of the ETL pipeline.
   - This control mechanism ensured that the data warehouse schema was consistently populated with accurate data and that tables with foreign key dependencies were loaded in the correct sequence.

Implementation –

1. DIM_Time

DIM_Time was created first as it provided the timestamp which were required by other tables



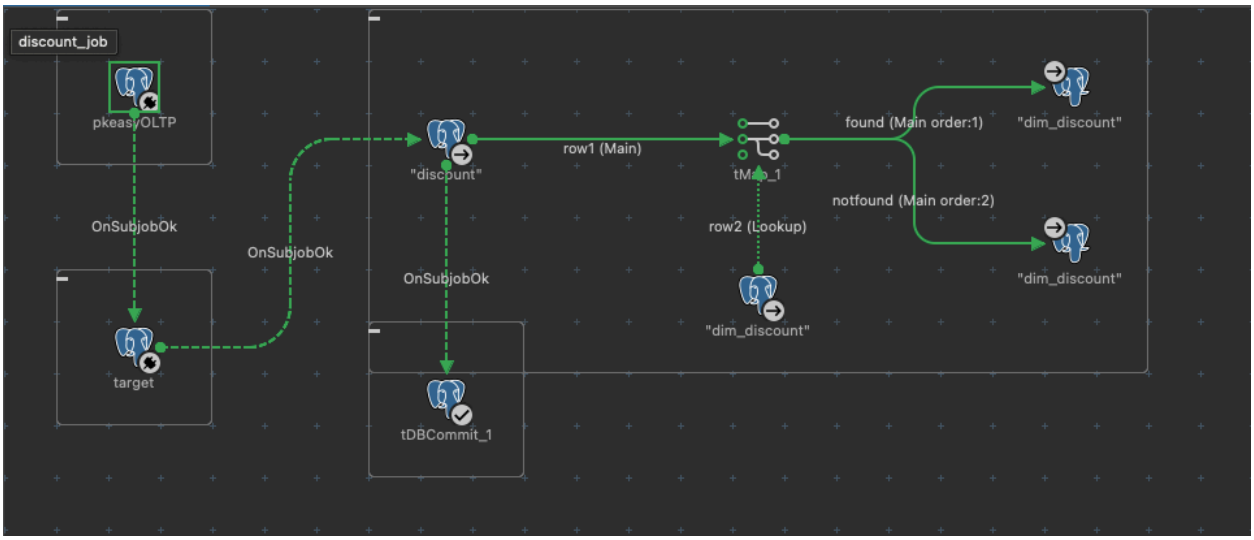

```
11
12  select * from DIM_Time;
13
```

Data Output | Messages | Notifications

	timeid [PK] integer	year integer	quarter integer	month integer	day integer	date date
1	20230101	2023	1	1	1	2023-01-01
2	20230102	2023	1	1	2	2023-01-02
3	20230103	2023	1	1	3	2023-01-03
4	20230104	2023	1	1	4	2023-01-04
5	20230105	2023	1	1	5	2023-01-05
6	20230106	2023	1	1	6	2023-01-06
7	20230107	2023	1	1	7	2023-01-07

2. DIM_Discount

DIM_Discount is referenced by DIM_Membership and the input table used was Discount




```
33  select * from DIM_Discount;
34
```

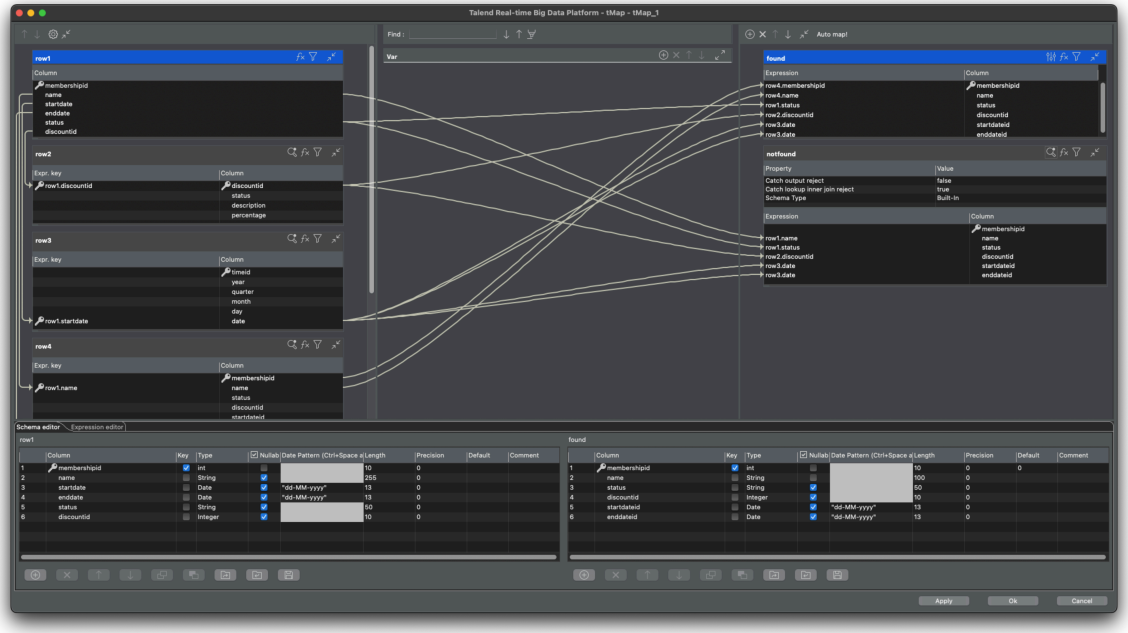
Data Output | Messages | Notifications

SQL

discountid [PK] integer	status character varying (50)	description text	percentage numeric (5,2)
1	Expired	Discount_1	21.27
2	Active	Discount_2	8.52
3	Active	Discount_3	9.89
4	Expired	Discount_4	16.84
5	Expired	Discount_5	15.98

3. DIM_Membership

DIM_Membership has foreign key dependencies on DIM_Discount and DIM_Time (for StartDateID and EndDateID). Input Tables used were Membership, DIM_Discount, DIM_Time




```
50  select * from DIM_Membership;
51
```

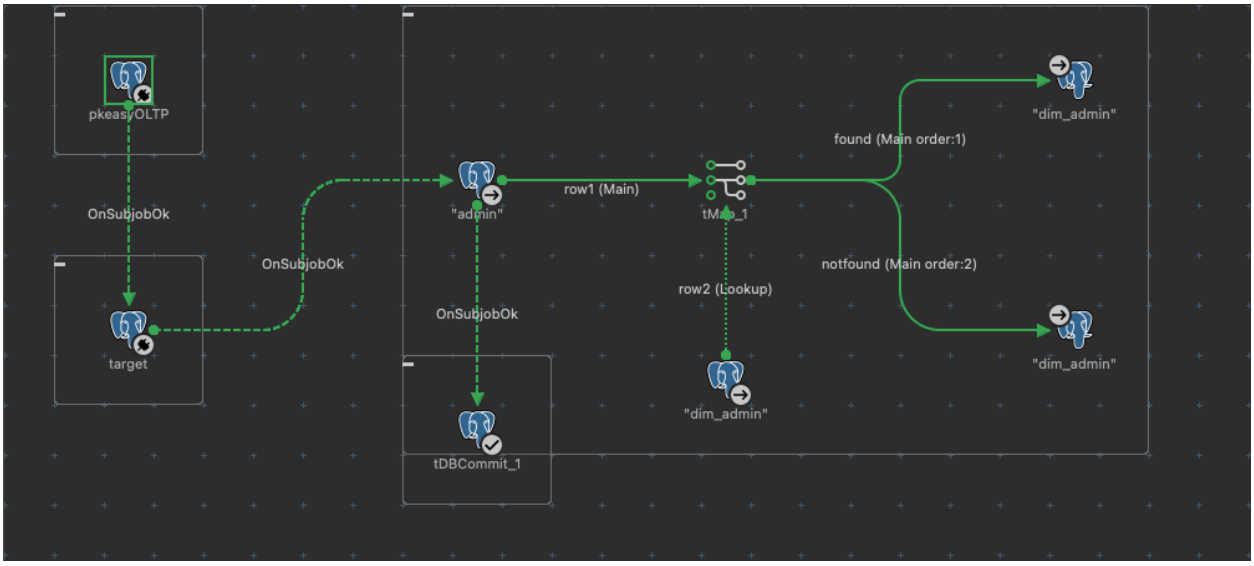
Data Output | Messages | Notifications

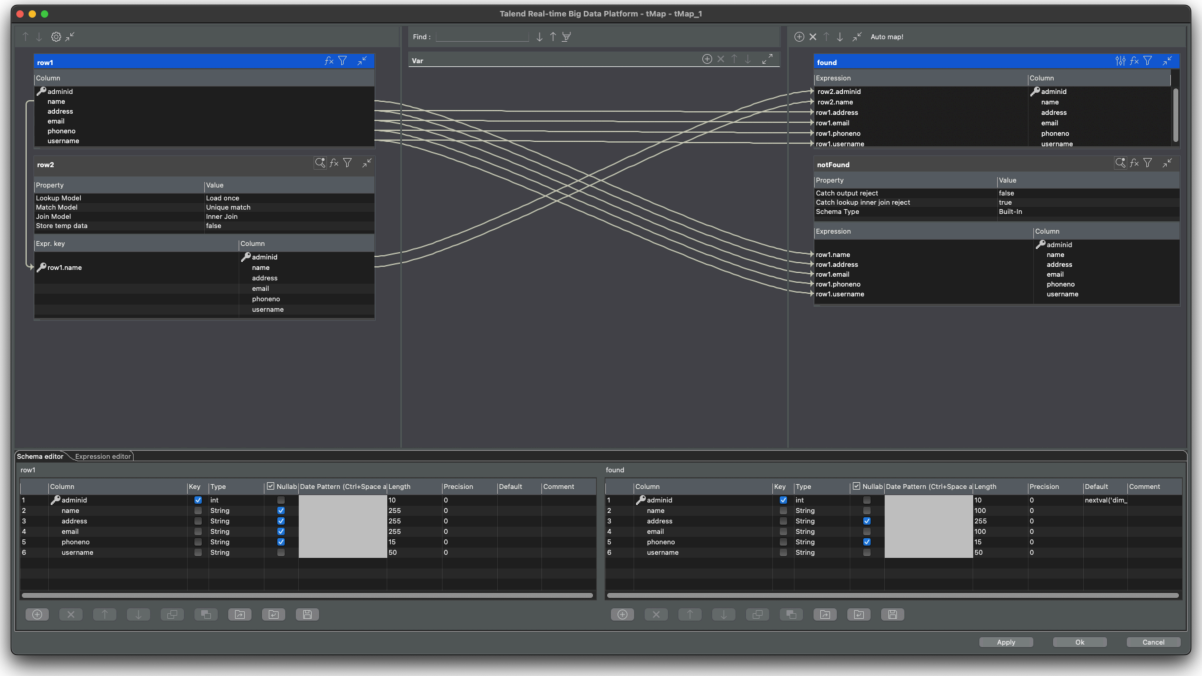
SQL

	membershipid [PK] integer	name character varying (100)	status character varying (50)	discountid integer	startdateid date	enddateid date
1	1	Bronze	Active	2	2023-12-17	2023-12-17
2	2	Silver	Active	1	2023-12-11	2023-12-11
3	3	Gold	Active	2	2024-01-25	2024-01-25
4	4	Gold	Inactive	4	2024-06-16	2024-06-16
5	5	Silver	Active	2	2024-01-12	2024-01-12
6	6	Gold	Inactive	3	2024-06-16	2024-06-16

4. DIM_Admin

Input table used for DIM_Admin was Admin






```
78  select*from DIM_Admin;
79
```

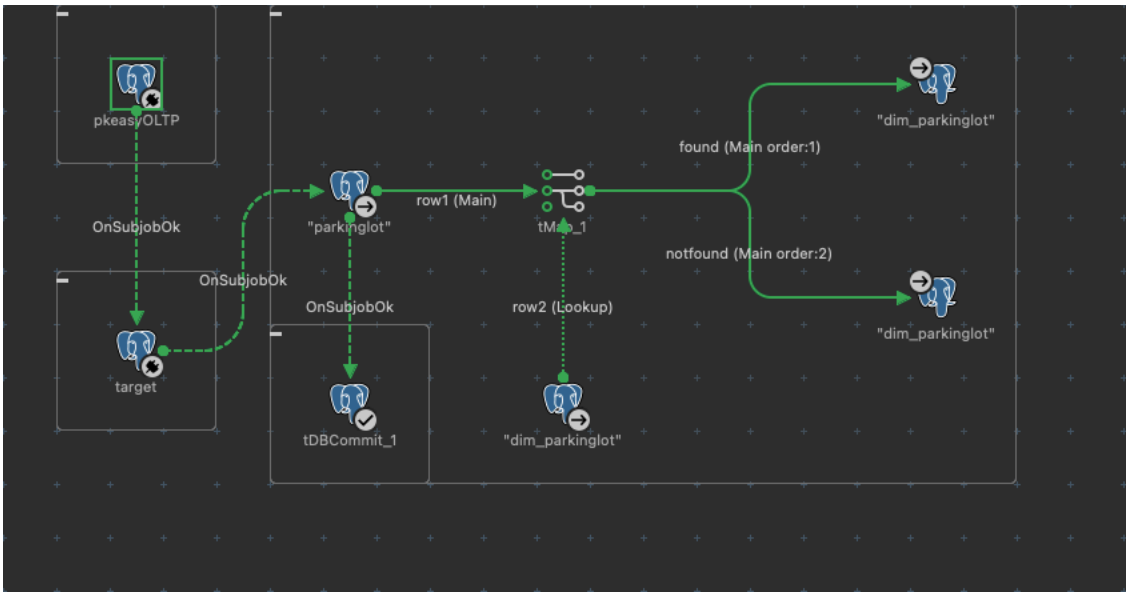
Data Output | Messages | Notifications

SQL

	adminid [PK] integer	name character varying (100)	address character varying (255)	email character varying (100)	phoneno character varying (15)	username character varying (50)
1	1	Admin_1	Address_1	admin1@example.com	+12345678901	admin_user1
2	2	Admin_2	Address_2	admin2@example.com	+12345678902	admin_user2
3	3	Admin_3	Address_3	admin3@example.com	+12345678903	admin_user3
4	4	Admin_4	Address_4	admin4@example.com	+12345678904	admin_user4
5	5	Admin_5	Address_5	admin5@example.com	+12345678905	admin_user5

5. DIM_ParkingLot

Input table used for DIM_ParkingLot was ParkingLot




```
88  select*from DIM_ParkingLot;
89
```

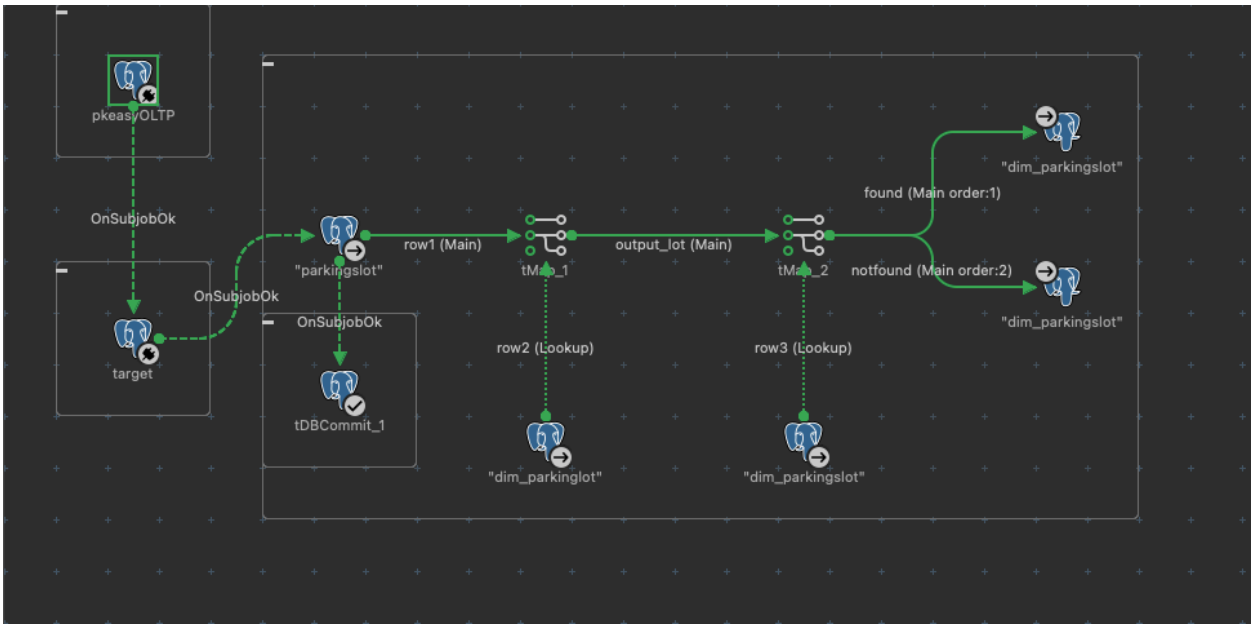
Data Output | Messages | Notifications

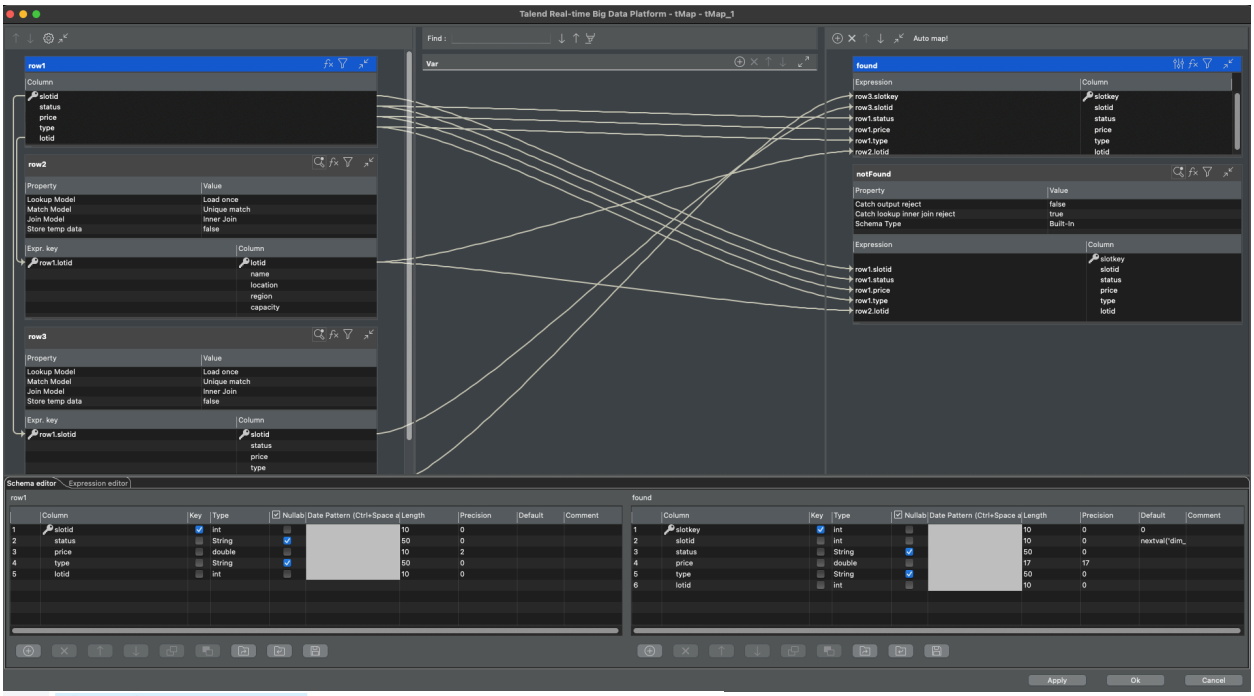
SQL

	lotid [PK] integer	name character varying (100)	location character varying (100)	region character varying (100)	capacity integer
1	1	ParkingLot_1	Location_1	East	100
2	2	ParkingLot_2	Location_2	South	100
3	3	ParkingLot_3	Location_3	West	100
4	4	ParkingLot_4	Location_4	South	100
5	5	ParkingLot_5	Location_5	South	100
6	6	ParkingLot_6	Location_6	West	100
7	7	ParkingLot_7	Location_7	East	100
8	8	ParkingLot_8	Location_8	South	100
9	9	ParkingLot_9	Location_9	North	100
10	10	ParkingLot_10	Location_10	South	100

6. DIM_ParkingSlot

DIM_ParkingSlot has a foreign key reference to DIM_ParkingLot (via LotID). Input Tables used were ParkingSlot, DIM_ParkingLot





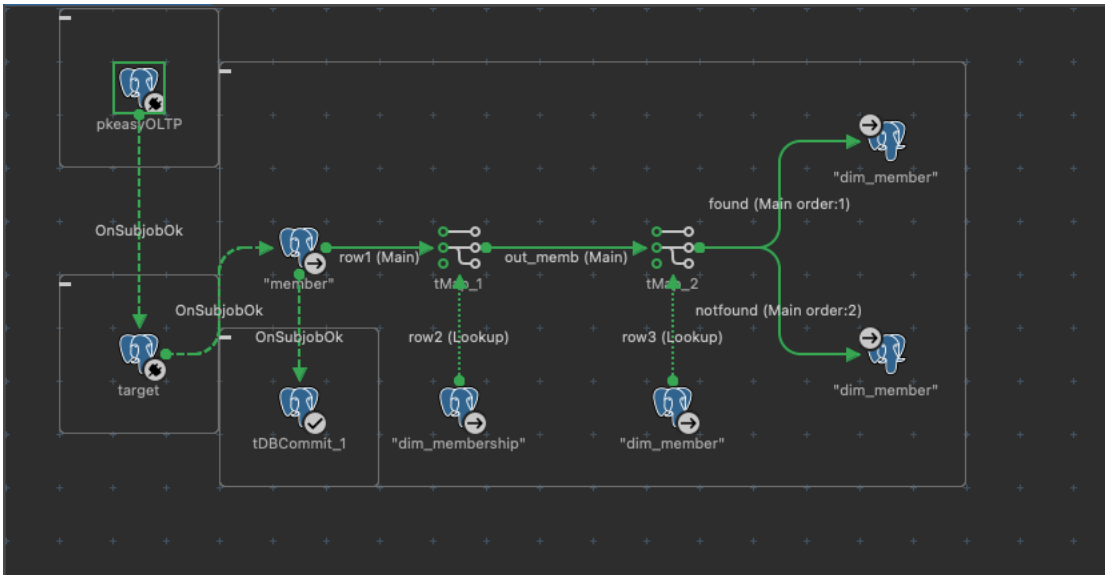

```
101   select * from DIM_ParkingSlot;
```

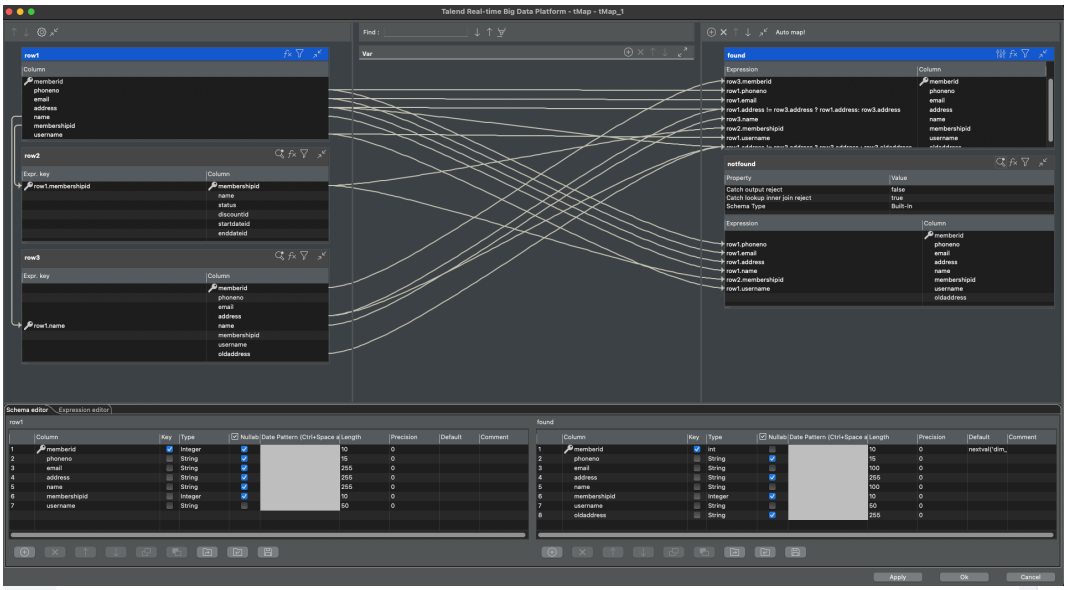
Data Output | Messages | Notifications

	slotid [PK] integer	status character varying (50)	price double precision	type character varying (50)	lotid integer	slotkey integer
1	1	Available	20	Standard	1	1
2	2	Occupied	10	Standard	1	2
3	3	Available	10	Compact	1	3
4	4	Available	20	Standard	1	4
5	5	Available	15	Compact	1	5
6	6	Occupied	20	Compact	1	6

7. DIM_Member

DIM_Member references DIM_Membership, so it was loaded after DIM_Membership. Input Tables used were Member, DIM_Membership





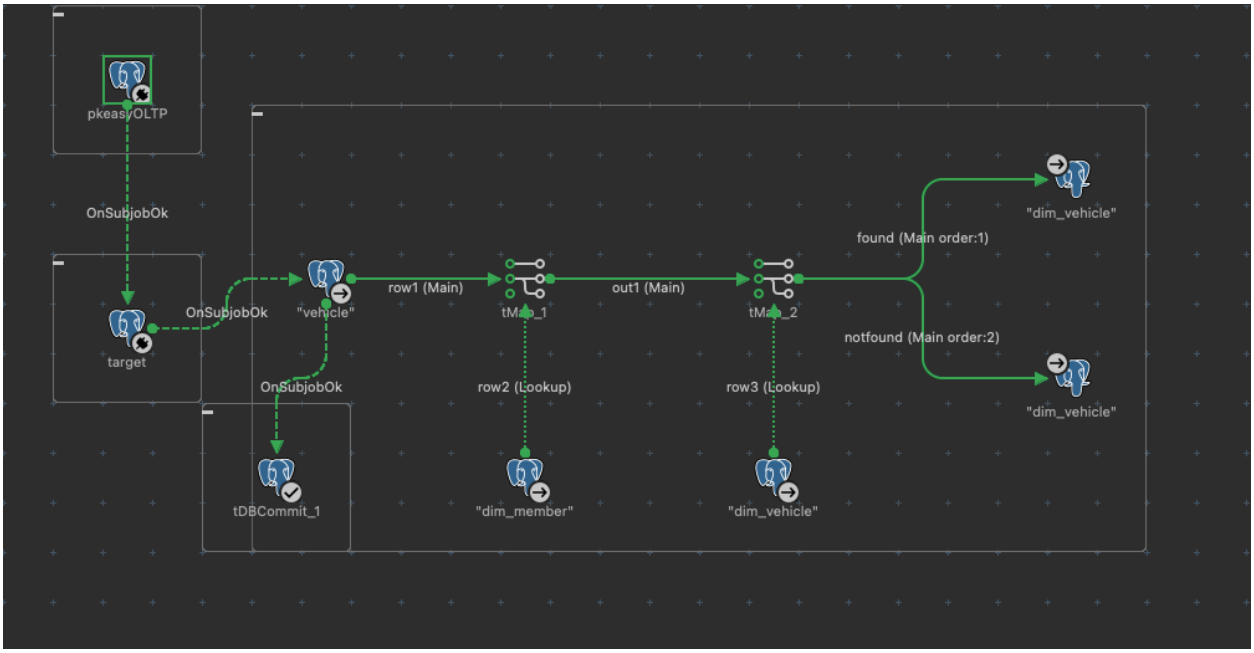
```
113  select * from DIM_Member;
114
```

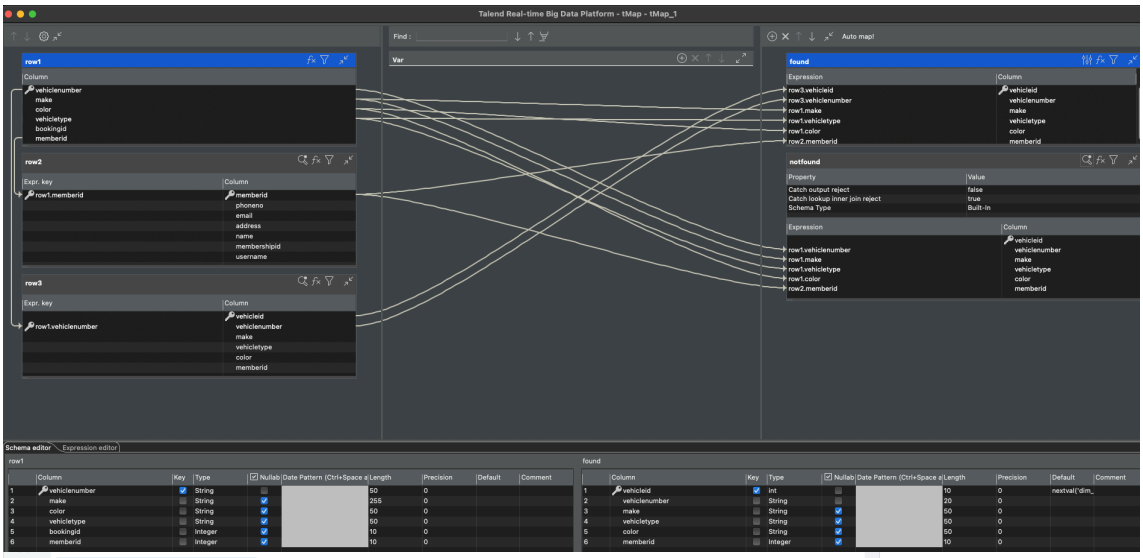
Data Output | Messages | Notifications

	memberid [PK] integer	phoneno character varying (15)	email character varying (100)	address character varying (255)	name character varying (100)	membershipid integer	username character varying (50)	oldaddress character varying (25
1	48	+1234567890430	member48@example.com	Address_48	Member_48	6	user48	Address_01
2	46	+1234567890843	member46@example.com	Address_46	Member_46	15	user46	[null]
3	47	+1234567890635	member47@example.com	Address_47	Member_47	50	user47	[null]
4	49	+1234567890149	member49@example.com	Address_49	Member_49	2	user49	[null]
5	27	+1234567890710	member27@example.com	Address_27	Member_27	4	user27	[null]
6	28	+1234567890712	member28@example.com	Address_28	Member_28	9	user28	[null]

8. DIM_Vehicle

DIM_Vehicle has a foreign key reference to DIM_Member, so it depends on DIM_Member.
Input Tables used were Vehicle, DIM_Member





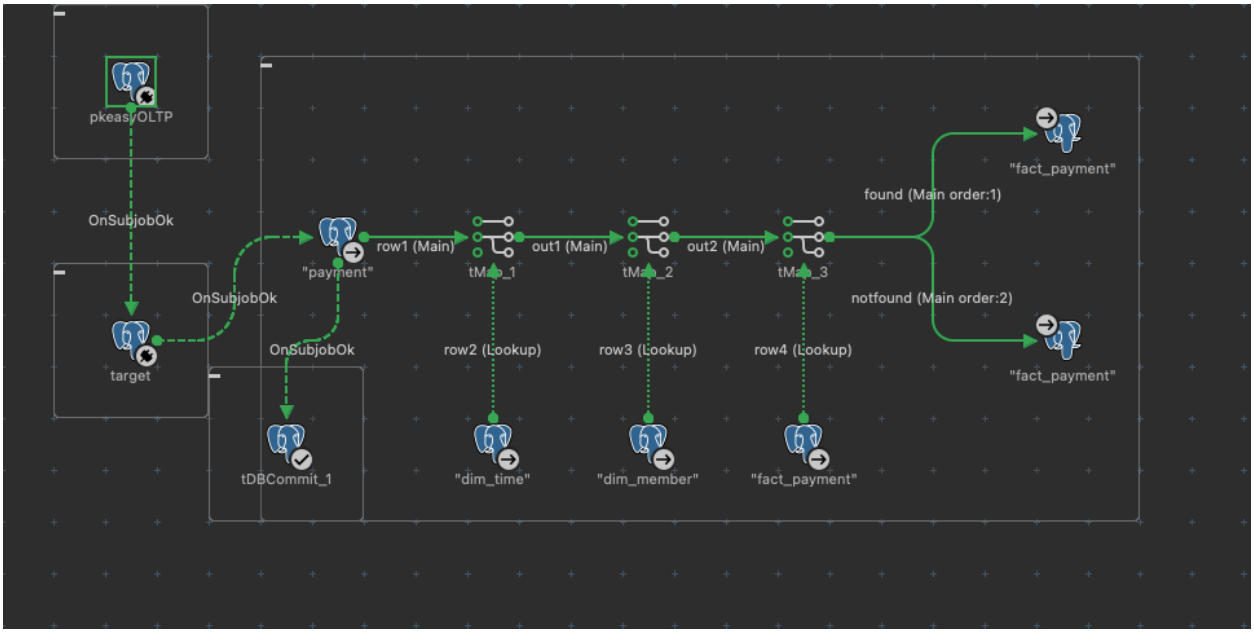

```
137  select * from DIM_Vehicle;
```

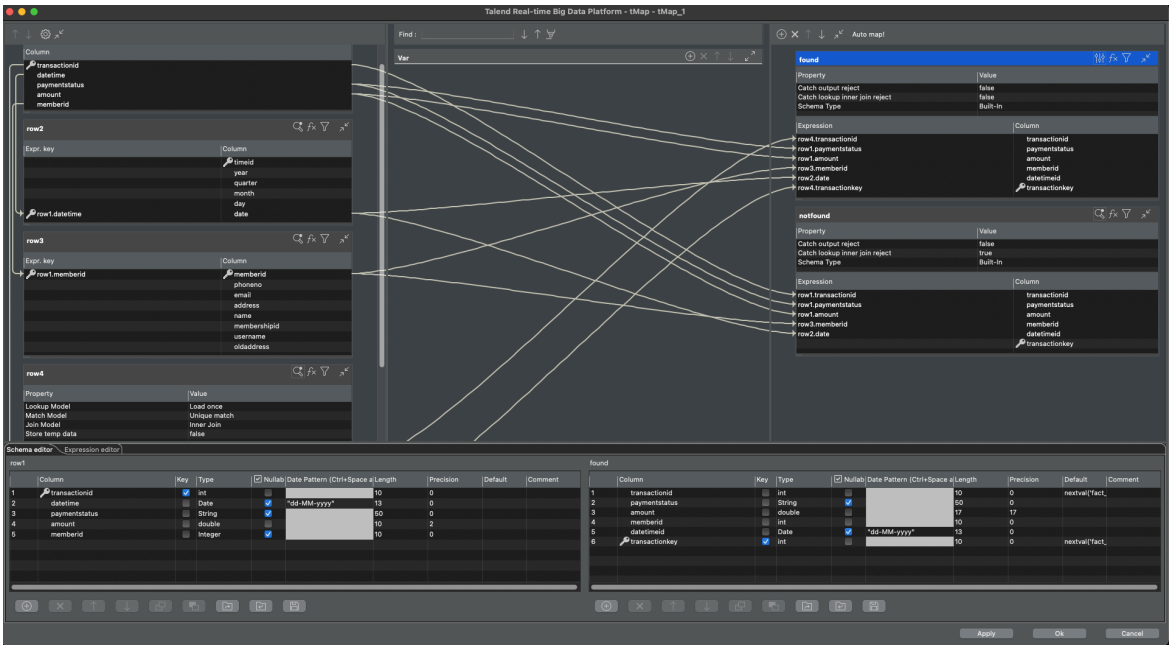
Data Output | Messages | Notifications

	vehicleid [PK] integer	vehiclenumber character varying (20)	make character varying (50)	vehicletype character varying (50)	color character varying (50)	memberid integer
1	1	VN_1	Toyota	SUV	Green	12
2	2	VN_2	Toyota	Sedan	Red	44
3	3	VN_3	Toyota	Truck	Red	32
4	4	VN_4	Honda	SUV	Green	21
5	5	VN_5	Ford	SUV	Green	10

9. FACT_Payment

FACT_Payment depends on DIM_Time (for DateTimeID) and DIM_Member. Input Tables used were Payment, DIM_Time, DIM_Member





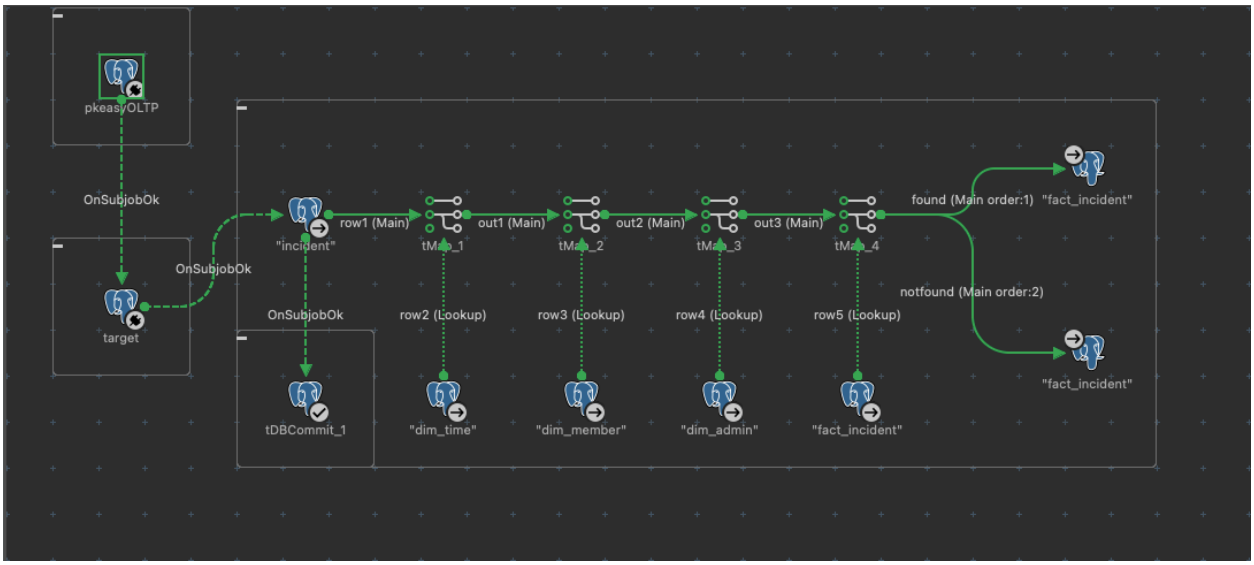
```
152  select * from FACT_Payment;
```

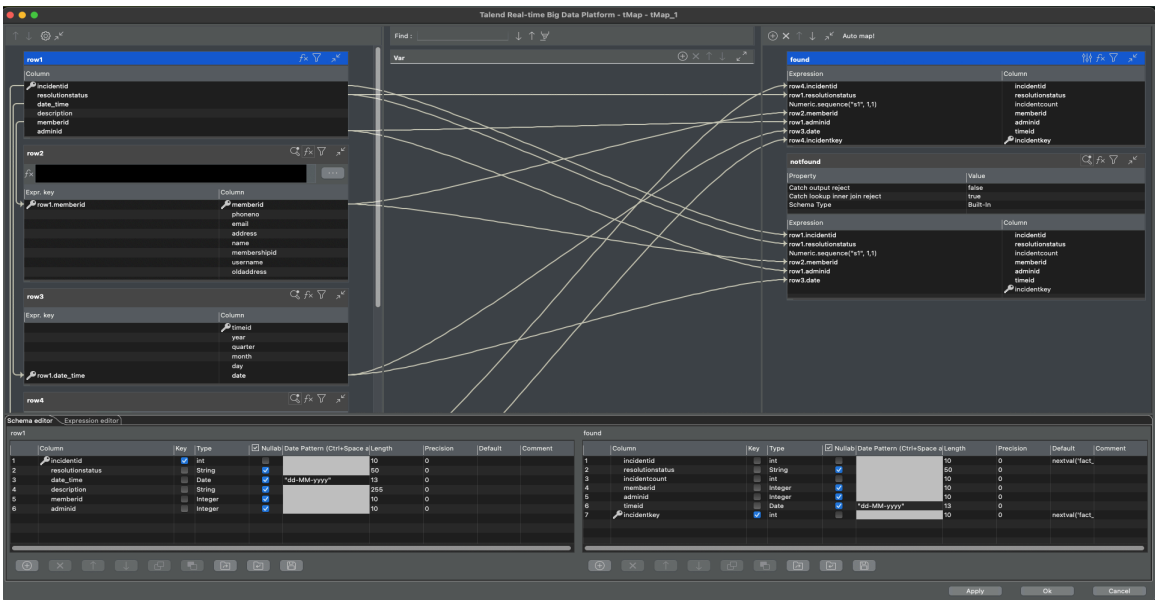
Data Output | Messages | Notifications

	transactionid [PK] integer	paymentstatus character varying (50)	amount double precision	memberid integer	datetimeid date	transactionkey integer
1	1	Completed	113.23	17	2024-10-21	1
2	2	Completed	138.03	25	2024-10-17	2
3	3	Completed	110.6	15	2024-10-19	3
4	4	Completed	128.51	20	2024-10-19	4

10. FACT_Incident

FACT_Incident references DIM_Time, DIM_Member, and DIM_Admin, so these tables were loaded first. Input Tables used were Incident, Inspect, DIM_Time, DIM_Member, DIM_Admin





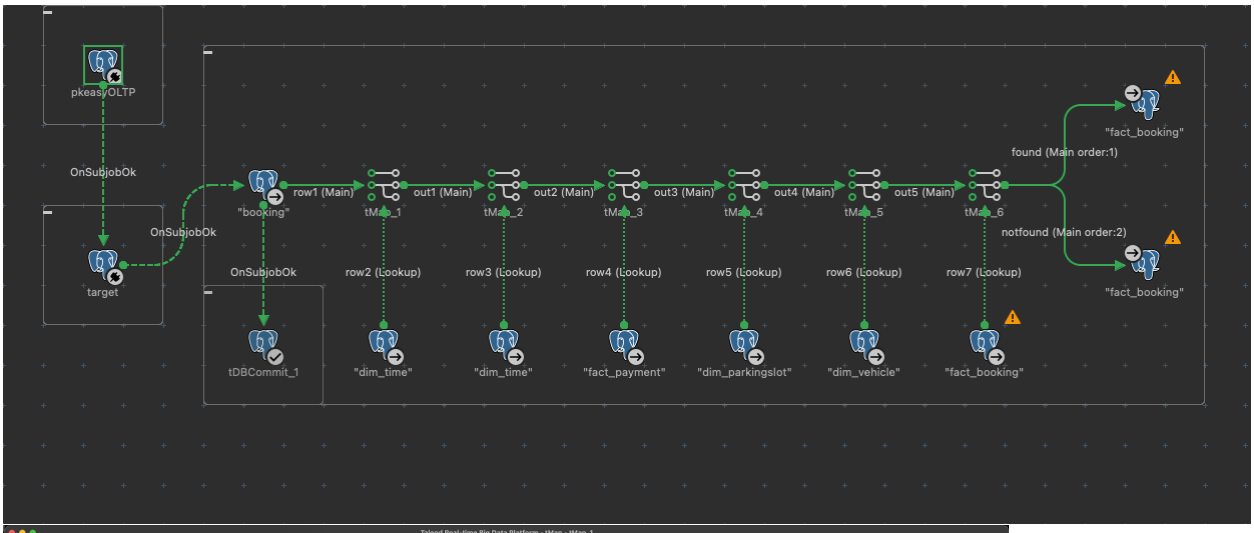
```
181  select * from FACT_Incident;
182
```

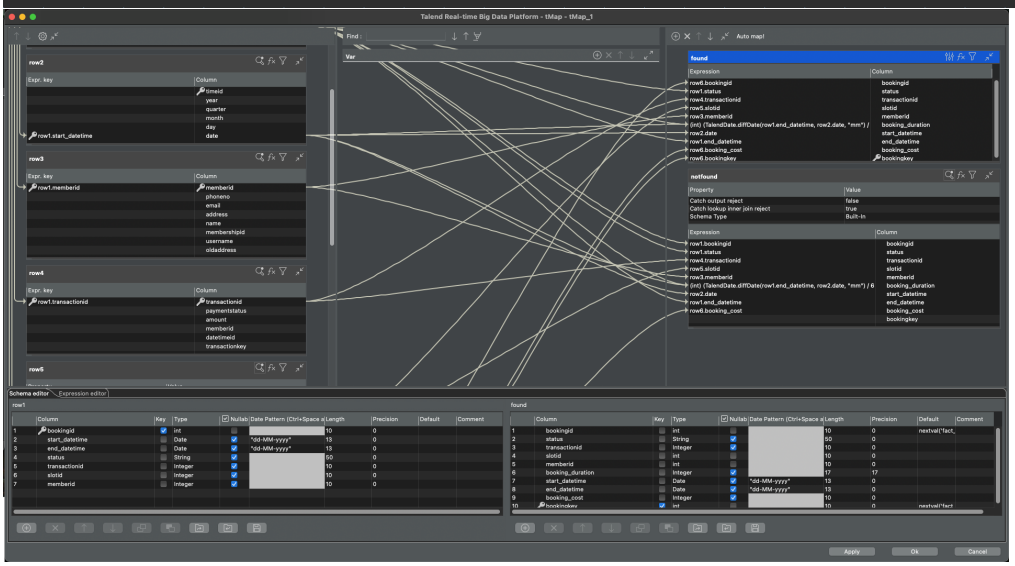
Data Output | Messages | Notifications

	incidentid [PK] integer	resolutionstatus character varying (50)	incidentcount integer	memberid integer	adminid integer	timeid date	incidentkey integer
1	1	Unresolved	1	50	[null]	2024-11-23	3
2	3	Unresolved	2	8	[null]	2024-11-18	4
3	5	Unresolved	3	47	[null]	2024-11-17	5
4	6	Resolved	4	26	[null]	2024-11-14	6

11. FACT_Booking

FACT_Booking has dependencies on DIM_Time, FACT_Payment, DIM_ParkingSlot, DIM_Vehicle, and DIM_Member. Input Tables used were Booking, DIM_Time, FACT_Payment, DIM_ParkingSlot, DIM_Vehicle, DIM_Member





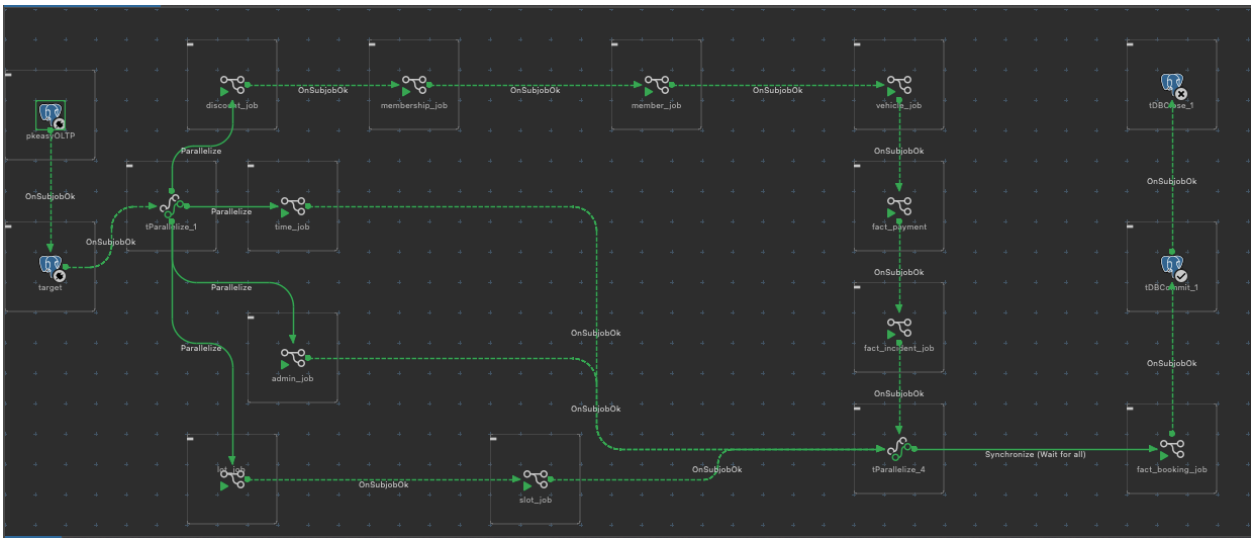
```
220  select * from FACT_Booking;
```

Data Output | Messages | Notifications

SQL

	bookingid [PK] integer	status character varying (50)	transactionid integer	slotid integer	memberid integer	booking_duration double precision	start_datetime date	end_datetime date	booking_cost integer	bookingkey integer
1	1	Confirmed	1	378	42	0	2024-10-23	2024-10-23	[null]	253
2	2	Cancelled	2	386	42	0	2024-10-01	2024-10-01	[null]	254
3	3	Pending	3	993	42	0	2024-10-29	2024-10-29	[null]	255
4	4	Pending	4	989	42	0	2024-11-02	2024-11-02	[null]	256
5	5	Pending	5	910	42	0	2024-08-07	2024-08-07	[null]	257

Control Flow implementation



13. Additional Features

1. Slowly Changing Dimensions (SCD): In this project, we have implemented a **Type 1 SCD** for the dimension tables. **Type 1 SCD** is a straightforward approach where changes in dimension data are directly overwritten, meaning that only the most recent value is stored in the data warehouse. With Type 1 SCD, there is no historical record of the previous values; the database always reflects the latest state.

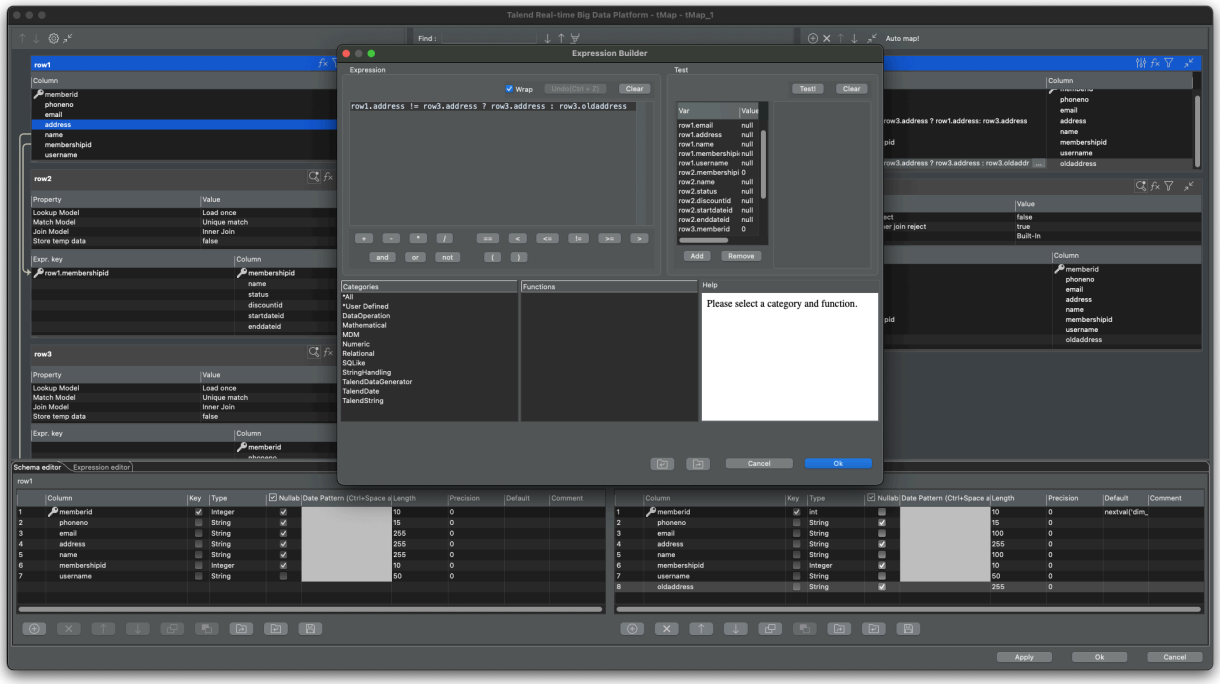
Type 1 SCD Implementation
For instance, if a product's UnitPrice or CategoryName changes, the ETL process will update the existing records in the DIM_Product or DIM_Category tables by simply overwriting the old values with the new ones. This approach ensures that our dimension data is current, without maintaining previous versions of the records. Implementing Type 1 SCD is suitable in cases where changes are not needed for historical analysis, or where only the latest information is relevant for decision-making.
Type 3 SCD Implementation

In a Type 3 Slowly Changing Dimension (SCD) implementation, changes in attributes are tracked by maintaining both the current value and the previous value of the attribute. For example, when a member's address changes, the ETL process will store the old address in a

separate column (previous_address) while updating the address field with the new value. This ensures that both the current and previous addresses are preserved, which is useful when historical comparisons are needed, but maintaining a full history of changes is not required.
In the case of the dim_member table:

- **Current Address**: The address column holds the latest address value.
- **Previous Address**: The previous_address column holds the old address value whenever the address is updated.

This approach is appropriate when tracking recent changes along with the immediate past value, allowing for some level of historical analysis without storing the entire history of changes.
By using Type 3 SCD, you can easily compare the current and previous values of an attribute like the address, but only a limited history is maintained, which reduces the data volume compared to other SCD types like Type 2.



```sql
CREATE TABLE DIM_Member (
    MemberID SERIAL PRIMARY KEY,
    PhoneNo VARCHAR(15),
    Email VARCHAR(100) NOT NULL UNIQUE,
    Address VARCHAR(255),
    Name VARCHAR(100) NOT NULL,
    MembershipID INT REFERENCES DIM_Membership(MembershipID),
    Username VARCHAR(50) NOT NULL UNIQUE
);

select * from DIM_Member;
```

Data Output | Messages | Notifications

	memberid [PK] integer	phoneno character varying (15)	email character varying (100)	address character varying (255)	name character varying (100)	membershipid integer	username character varying (50)	oldaddress character varying (25
1	48	+1234567890430	member48@example.com	Address_48	Member_48	6	user48	Address_01
2	46	+1234567890843	member46@example.com	Address_46	Member_46	15	user46	[null]
3	47	+1234567890635	member47@example.com	Address_47	Member_47	50	user47	[null]
4	49	+1234567890149	member49@example.com	Address_49	Member_49	2	user49	[null]
5	27	+1234567890710	member27@example.com	Address_27	Member_27	4	user27	[null]
6	28	+1234567890712	member28@example.com	Address_28	Member_28	9	user28	[null]

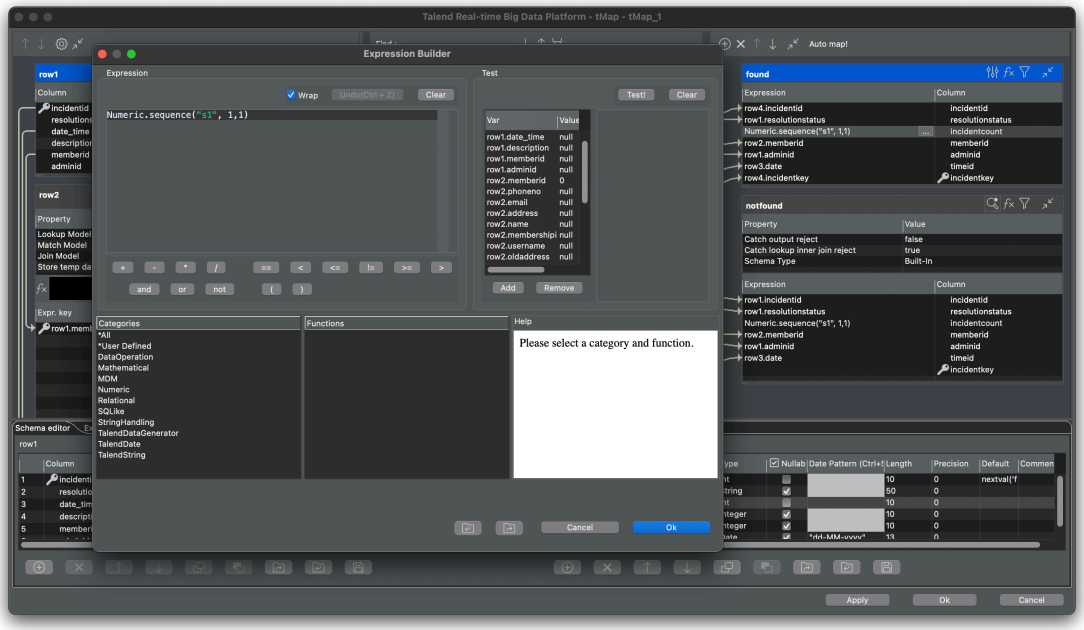
Total rows: 50 of 50 Query complete 00:00:00.075 Ln 115, Col 1

2. Calculated Measures:

FACT_Incident Measures

- **Total Incident Count**:
  - *Definition*: The total incident count is calculated as the sum of all incidents (IncidentCount column) in the FACT_Incident table.
  - *Purpose*: This measure gives a comprehensive view of the volume of incidents recorded, which helps in assessing the frequency and nature of issues reported in the system.

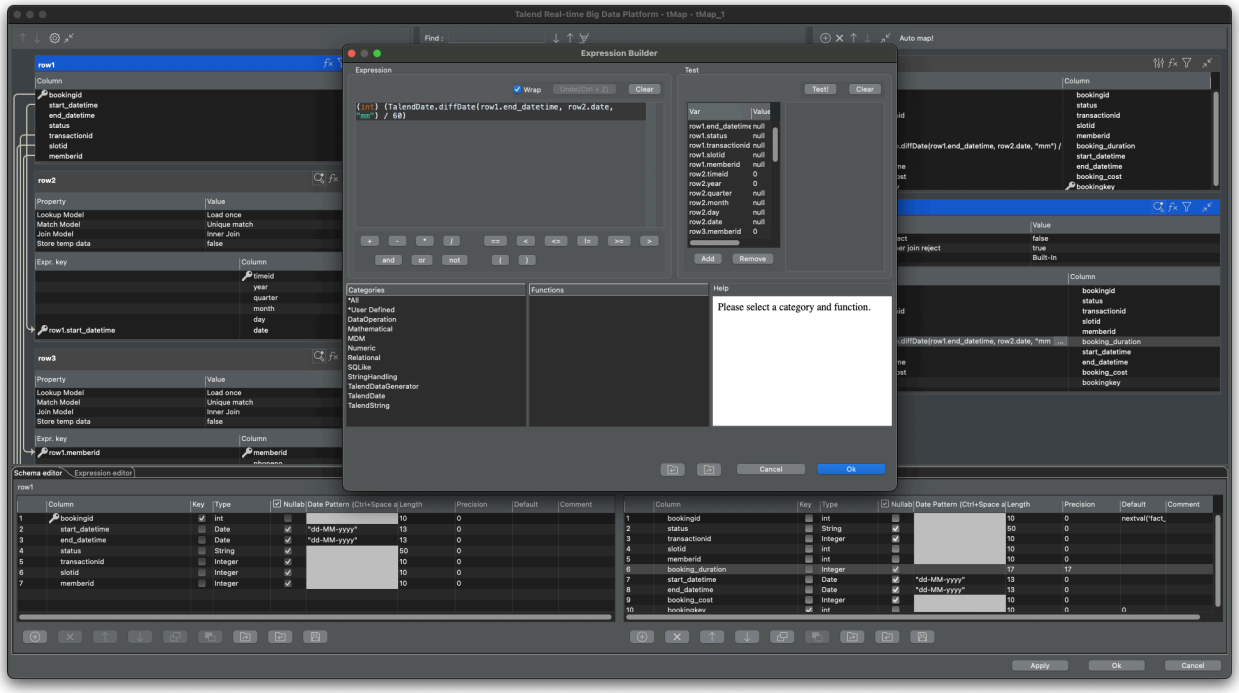
Implementation –

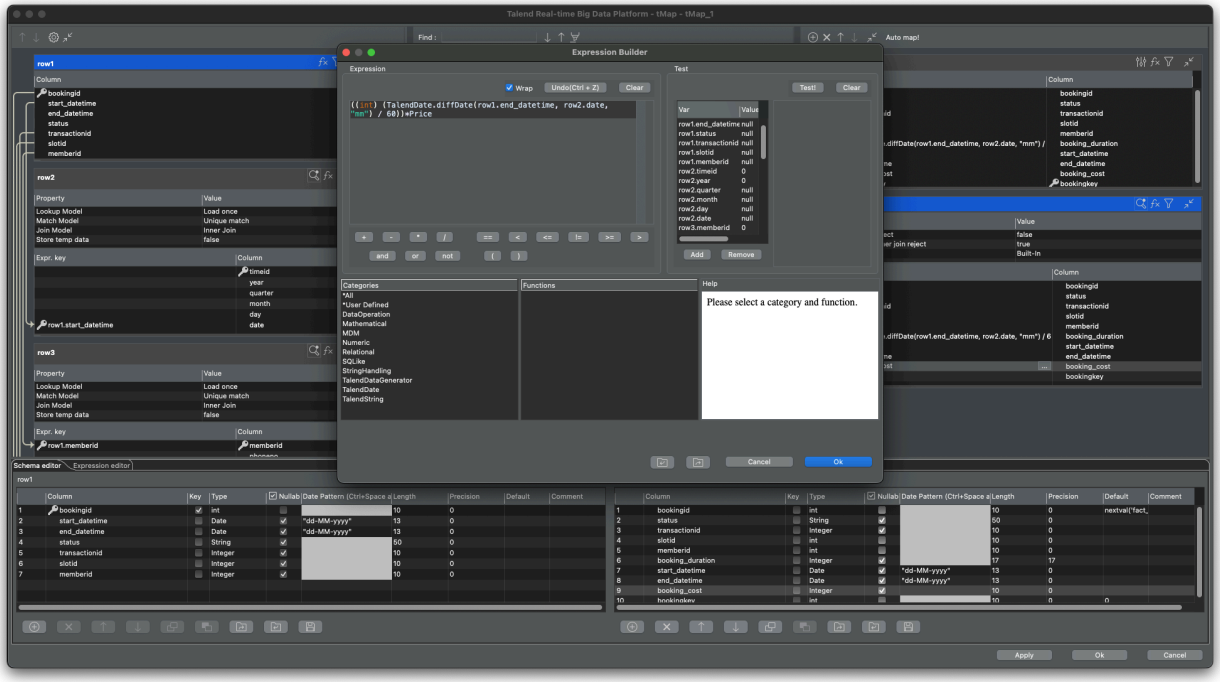


FACT_Booking Measures

- **Total Booking Cost**:
  - *Definition*: The total booking cost is calculated as the difference between End_DateTime and Start_DateTime multiplied by the price
  - *Purpose*: This measure tracks the total booking cost, providing an overview of the revenue generated through it
- **Total Booking Duration**:
  - *Definition*: The total booking duration is calculated as the sum of the difference between End_DateTime and Start_DateTime for all bookings in the FACT_Booking table.
  - *Purpose*: This measure calculates the cumulative duration of all bookings, allowing analysis of resource usage over time.

Implementation –





15. Transformations

1. In the ETL pipeline for the Parkeasy Management System, a key transformation step is datatype conversion when transferring data from the OLTP environment to the OLAP data warehouse. Datatype transformation ensures that the data types used in the OLAP schema are

optimized for analytical processing, which might differ from the data types used in the OLTP system designed for transactional operations.

Example of Datatype Transformation
Precision Adjustments for Numeric Data:

In the OLTP schema, certain fields such as Price in ParkingSlot and Amount in Payment are stored as DECIMAL data types, which are suitable for financial transactions with fixed precision.
In the OLAP schema, these fields are converted to FLOAT to accommodate analytical calculations, where a balance between performance and precision is considered. The FLOAT datatype is efficient for aggregations and other complex calculations performed during analysis.

2. String Length Adjustments:
The OLTP schema often uses broader character fields (e.g., VARCHAR(255)) for attributes like Address or Name to accommodate diverse user input.
In the OLAP schema, these fields may be adjusted to appropriate lengths (e.g., VARCHAR(100)) based on typical usage patterns in analytics, optimizing storage without compromising data integrity.

3. Lookup and Foreign Key Mapping
During the ETL process, foreign keys in fact tables (e.g., FACT_Payment, FACT_Incident, FACT_Booking) need to reference the surrogate keys in the respective dimension tables. Lookup transformations map OLTP primary keys (e.g., MemberID, SlotID) to their corresponding surrogate keys in the dimension tables. For example, MemberID in the FACT_Payment table must be mapped to MemberID in DIM_Member through a lookup transformation.
This ensures that fact tables correctly reference dimension tables using the new surrogate keys, aligning the OLAP schema with the star schema design.

4. Calculation of Derived and Aggregated Measures
Derived calculations are applied to generate measures in fact tables. For example:

- **Total Booking Amount** in FACT_Booking could be derived from booking duration (End_DateTime - Start_DateTime) multiplied by slot Price.
- **Total Booking Duration** in FACT_Booking is derived from End_DateTime - Start_DateTime to provide insights into total utilization time.

These derived measures allow the OLAP system to perform complex analytical queries without recalculating values every time, improving efficiency.

16. Dashboard Creation:

Dashboard 1: Parking Utilization & Operational Performance Report

The primary aim of this dashboard is to optimize the utilization of parking facilities by providing real-time and historical data insights. It focuses on the efficiency of space usage, administrative performance, and operational issues, which are critical for maintaining a high level of customer satisfaction and operational smoothness.

Key Performance Indicators (KPIs)

1. Occupancy Rate
   - Formula: Occupancy Rate = (Total Booked Slots / Total Available Slots) × 100
   - Purpose: This metric is crucial for measuring the efficiency of space utilization within the parking facilities. It helps determine how well parking resources are being maximized and provides a clear picture of demand versus capacity.
   - Display: The occupancy rate is prominently displayed as a large percentage value at the top of the dashboard, making it immediately visible upon viewing the dashboard to quickly assess utilization efficiency.

2. Total Revenue from Parking
   - Formula: Total Revenue = SUM(Booking Cost) (calculated from the FACT_Booking table)
   - Purpose: This figure represents the total financial earnings from all parking bookings. It's essential for assessing the profitability of the parking operations and supports financial planning and budgeting.
   - Display: Presented as a monetary value at the top of the dashboard, this KPI offers a quick snapshot of financial performance to stakeholders.

3. **Customer Satisfaction Score**:
   - Formula: Average of customer feedback scores.
   - Purpose: Measures customer satisfaction with the parking facility, influencing loyalty and repeat business.
   - Display: As a percentage alongside other top KPIs.

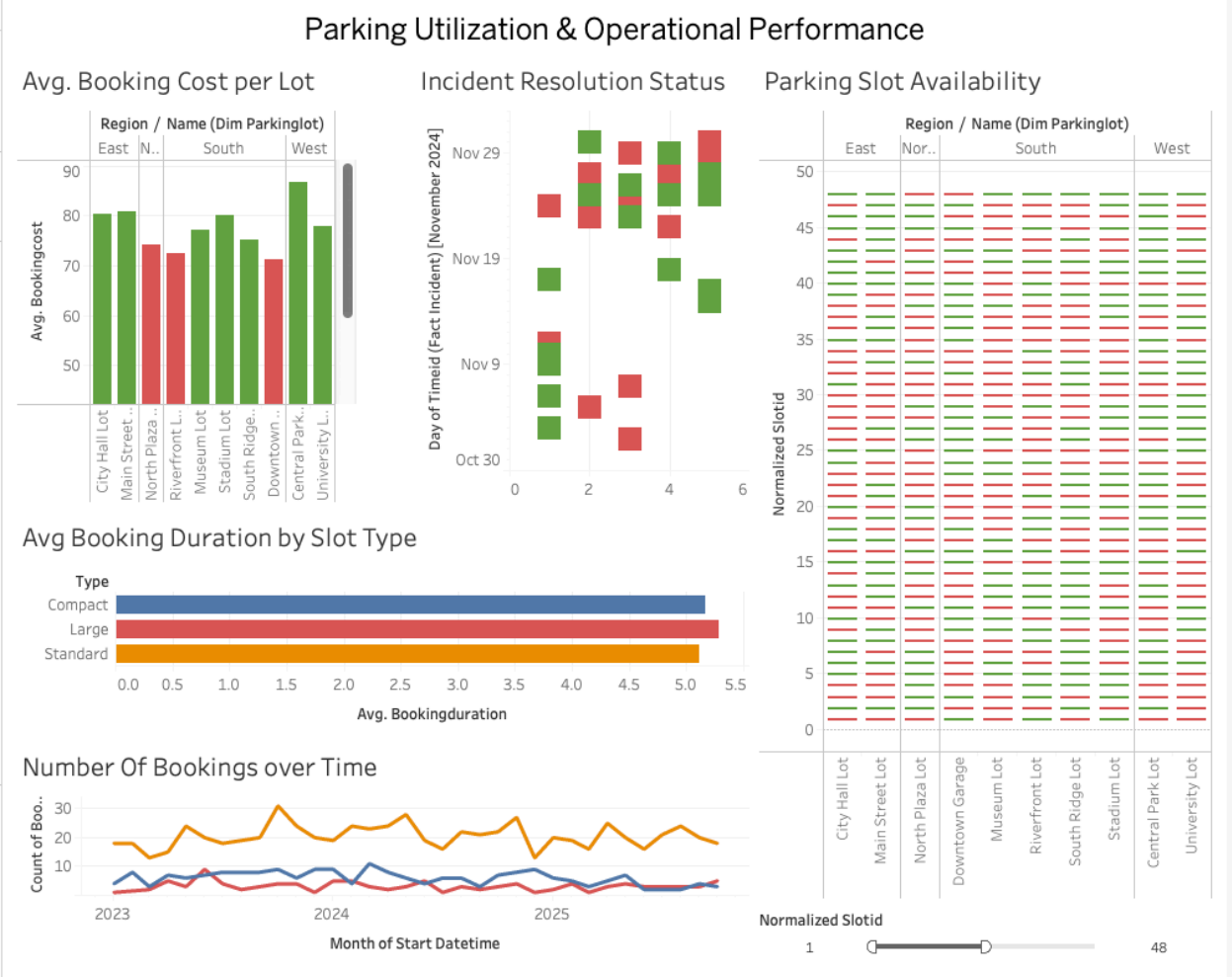
4. Peak Usage Times:
   - Formula: Identify hours with maximum slot occupancy.
   - Purpose: Helps in identifying the busiest times at the parking facilities, crucial for staffing and operational adjustments.
   - Display: Displayed as a timeline with peak times highlighted.

5. Maintenance and Repair Incidents:
   - Formula: Count of reported and resolved maintenance issues.
   - Purpose: Ensures efficient upkeep of the parking lot, enhancing customer experience and safety.
   - Display: As a component of the incident status heatmap, with specific icons for maintenance-related issues.

6. Revenue per Available Slot (RevPAS):
   - Formula: Total Revenue / Number of Available Slots.
   - Purpose: This metric assesses revenue efficiency relative to the available parking inventory.
   - Display: Presented alongside total revenue to provide insight into how effectively each slot is being monetized.

Visualizations

1. Average Booking Cost per Lot (Bar Chart)
   - Purpose: This chart helps identify which parking lots are generating the most revenue per booking, informing decisions about pricing and marketing strategies for different locations.
   - Key Metric: Revenue per Lot.
   - Details: The chart provides a breakdown by lot, allowing managers to easily compare performance across different sites.
2. Parking Availability by Slot (Heatmap)
   - Purpose: This visualization is key for operational planning, showing the availability of slots throughout the day and identifying times of high or low demand.
   - Key Metric: Slot Utilization.
   - Details: Color-coded for clarity, the heatmap offers a time-based view that helps in adjusting staffing and security measures according to expected occupancy.
3. Number of Bookings Over Time (Line Chart)
   - Purpose: Tracking booking trends over time enables the prediction of future demand and assists in strategic planning for peak periods.
   - Key Metric: Bookings Trend Over Time.
   - Details: The line chart can be filtered by booking status—completed, pending, canceled—to provide deeper insights into customer behavior and operational issues.
4. Incident Status Heatmap of Admins
   - Purpose: This heatmap assesses the performance of staff in handling incidents and the administrative load, crucial for managing human resources and improving customer service.
   - Key Metric: Incident Resolution Rate.
   - Details: It displays the rate of resolved incidents by admin, highlighting areas where training or additional resources may be needed.
5. Average Booking Duration by Slot Type (Bar Chart)
   - Purpose: Understanding the duration for which different types of parking slots are booked helps in optimizing the allocation and pricing of these resources.
   - Key Metric: Average Booking Duration by Slot Type.
   - Details: This chart differentiates between short-term, long-term, and specialty parking slots, offering insights into usage patterns that could influence promotional efforts or infrastructural adjustments.



Analysis:
The analysis focuses on several key areas to enhance the management of parking facilities:

1. **Average Booking Cost per Lot**: This bar chart shows the average cost of bookings across different parking lots categorized by region (East, North, South, West). The green bars indicate a higher cost, suggesting either premium pricing or higher demand in these areas. Analyzing these variations can help adjust pricing strategies to maximize revenue.
2. **Incident Resolution Status**: The heatmap indicates the frequency and resolution of incidents over different dates. Green squares show days with a high number of successfully resolved incidents, while red indicates unresolved incidents. This visualization is essential for monitoring the effectiveness of operational procedures and staff performance.
3. **Parking Slot Availability**: The detailed grid layout displays the availability of parking slots in various lots, with green indicating available slots and red showing occupied ones. This real-time data is crucial for customers seeking parking and helps management optimize space utilization.
4. **Average Booking Duration by Slot Type**: The bar chart categorizes parking slots by type (Compact, Large, Standard) and shows the average duration of bookings. This

metric helps in understanding how different slot types are used, aiding in effective resource allocation and pricing adjustments based on usage patterns.

5. **Number of Bookings over Time**: This line chart tracks the total number of bookings over several years (2023-2025), providing insights into trends and seasonal variations in parking demand. It is valuable for forecasting and planning purposes.