Database Foundations
ICCS225
Project Report: Garage Management System

Chawinthorn Kittivacharaphong

December 3, 2025

Contents

1 Introduction

This report outlines the development of a comprehensive Garage Management System designed to streamline operations for independent automotive service centers. The platform centralizes critical administrative functions, including mechanic scheduling, spare parts inventory tracking, equipment tracking, financial reporting, and customer vehicle history. The primary objective is to replace fragmented, manual processes—such as paper-based logs—with a unified digital solution to enhance operational efficiency and profitability.

2 Pain Points

Small garages face several problems that this project solves:

- **Budget Constraints:** Small garages cannot afford to hire professional programmers to build custom software.
- **Data Inconsistency:** Writing everything by hand is slow and tiring.
- **Prevent Data Loss:** Paper records get lost, damaged, or are hard to read. This system keeps data safe.

3 Solution Overview

The platform helps manage the entire garage through these features:

- **Work & Service:**
  - **Job Requests:** Create forms for new repair jobs.
  - **Service Management:** Track the status of vehicle repairs.
  - **Appointments:** Schedule dates and times for customers.
- **People Management:**
  - **Customers:** Keep a list of clients and their car history.
  - **Employees:** Manage staff details and shifts.
  - **Memberships:** Track loyal customers and their subscription status.
- **Shop & Finance:**
  - **Stock Tracker:** Count spare parts and equipment so you don't run out.
  - **Financial Reports:** See how much money is made from services and memberships.

4 Entity-Relationship Diagram

The Entity-Relationship Diagram (ERD) helps visualize the database structure for the Garage Management System. It shows how different entities like Customers, Employees, Vehicles, Services, Spare Parts, and Equipment relate to each other.

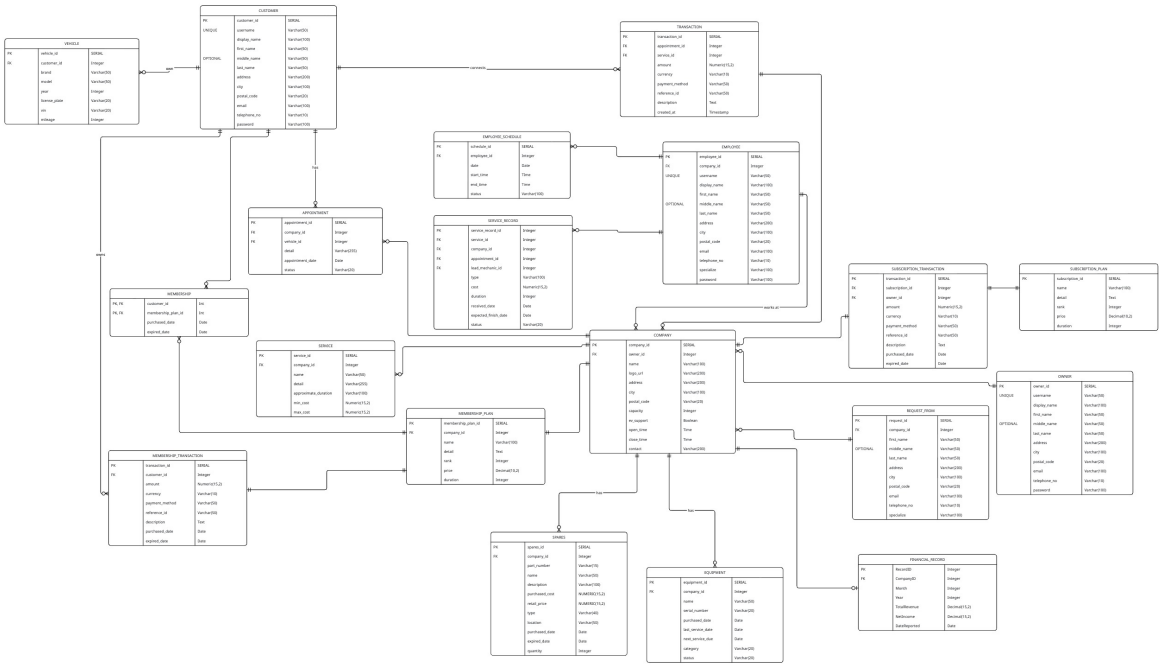
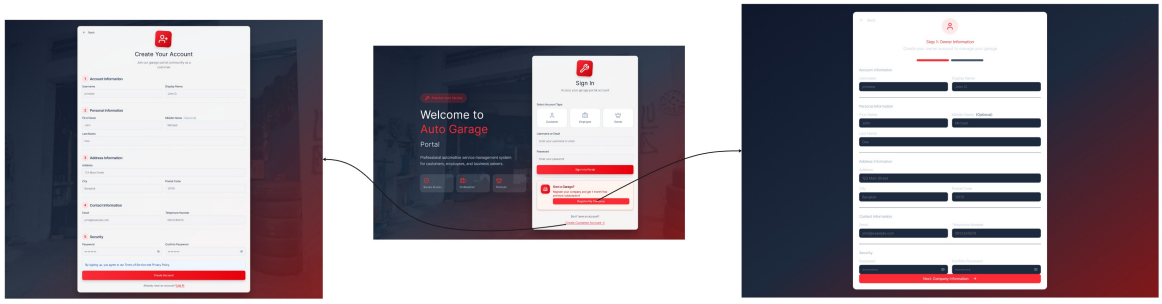


Figure 1: Entity-Relationship Diagram for Garage Management System

5 Applications Flow

5.1 Login & Sign up



Users can sign up as an owner, employee, or customer. After signing up, they can log in to access their respective dashboards. every user type has a dedicated authentication function in the database to verify their credentials during login.

5.1.1 Customer authenticate

```
create function authenticate_customer(p_identifier character varying,
    p_password character varying) returns integer
language plpgsql
as
$$
DECLARE
    v_user_id INT;
BEGIN
    SELECT id INTO v_user_id
    FROM "customer"
    WHERE (username = p_identifier OR email = p_identifier)
      -- Check if the input password matches the stored hash
      AND password = crypt(p_password, password);

    RETURN v_user_id; -- return null if user doesn't exist or input not
    match
END;
$$;

alter function authenticate_customer(varchar, varchar) owner to
    sc_db_project_user;
```

Explanation (authenticate_customer)
Purpose: Validates a customer attempting to log in using either their username or email plus a plaintext password. Returns the internal customer id if credentials match, otherwise NULL (no row selected).
Logic: Single SELECT on the `customer` table with an OR predicate for identifier matching. Uses PostgreSQL's `crypt` to hash the supplied password and compare with the stored salted hash.

5.1.2 Employee authenticate

```sql
create function authenticate_employee(p_identifier character varying,
    p_password character varying) returns integer
    language plpgsql
as
$$
DECLARE
    v_user_id INT;
BEGIN
    SELECT id INTO v_user_id
    FROM "employee"
    WHERE (username = p_identifier OR email = p_identifier)
      -- Check if the input password matches the stored hash
      AND password = crypt(p_password, password);

    RETURN v_user_id;
END;
$$;

alter function authenticate_employee(varchar, varchar) owner to
    sc_db_project_user;
```

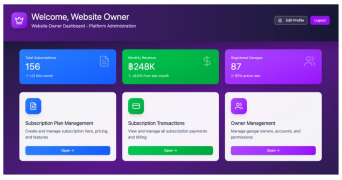
Explanation (authenticate_employee)
Purpose: Authenticates an employee for access to internal (non-owner) operational features.
Logic: Mirrors the customer approach; performs a lookup with OR on username/email and validates password using `crypt`.

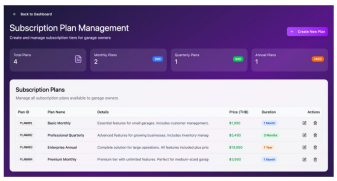
5.1.3 Owner authenticate

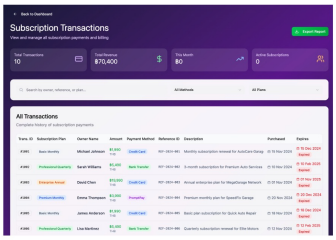
```
create function authenticate_owner(p_identifier character varying,
    p_password character varying) returns integer
    language plpgsql
as
$$
DECLARE
    v_user_id INT;
BEGIN
    SELECT id
    INTO v_user_id
    FROM "owner"
    WHERE (username = p_identifier OR email = p_identifier)
      -- Check if the input password matches the stored hash
      AND password = crypt(p_password, password);

    RETURN v_user_id;
END;
$$;

alter function authenticate_owner(varchar, varchar) owner to
    sc_db_project_user;
```

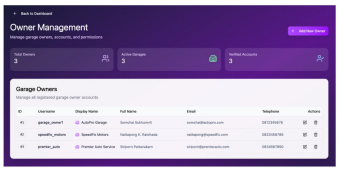
Explanation (authenticate_owner)
Purpose: Grants owners access to administrative dashboards (financials, employee management) after verifying credentials.
Logic: SELECT + password hash comparison via `crypt`.

5.2 Admin side





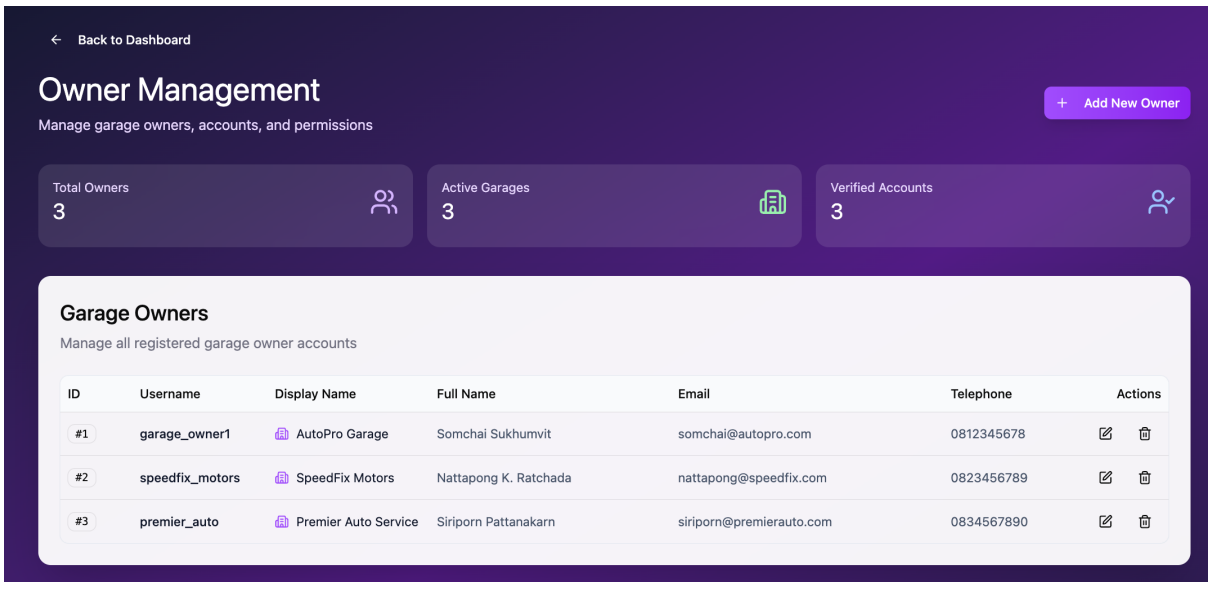




5.2.1 Owner management

```sql
create function show_owner()
    returns TABLE(id integer, username character varying, display_name character varying, full_name character varying, city character varying, email character varying, telephone_no character varying)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            o.id,
            o.username,
            o.display_name,
            (o.first_name || ' ' || COALESCE(o.middle_name || ' ', '') || o.last_name)::VARCHAR,
            o.city,
            o.email,
            o.telephone_no
        FROM
            owner o
        ORDER BY
            o.id;
END;
$$;

alter function show_owner() owner to sc_db_project_user;
```

Listing 1: show_owner.sql

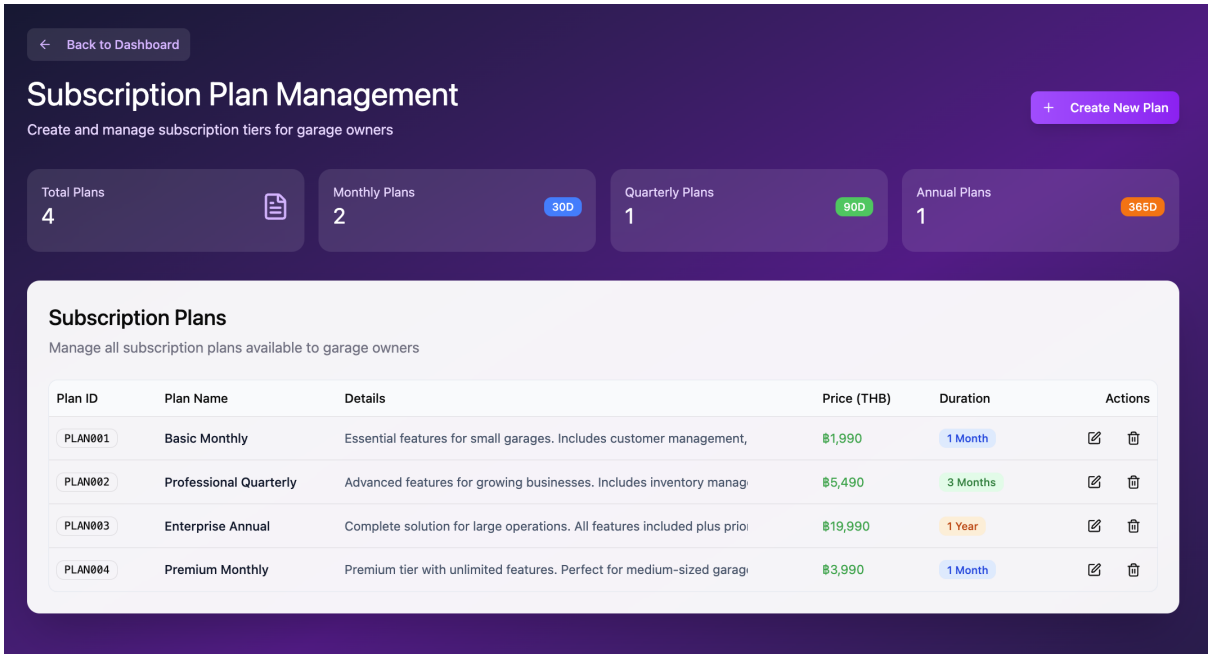


Explanation (show_owner.sql)
Purpose: Provides an overview of platform owners for administrative monitoring (support, billing, compliance).
Logic: Straightforward SELECT with ordered output by id for stable pagination/export. Middle name is conditionally appended using `COALESCE`.

5.2.2 Subscription plan management

```sql
create function show_subscription_plans()
    returns TABLE(plan_id integer, plan_name character varying, details
    text, price_thb numeric, duration_days integer)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            subscription_id,
            name,
            detail,
            price,
            duration
        FROM "subscription_plan"
        ORDER BY rank;
END;
$$;

alter function show_subscription_plans() owner to sc_db_project_user;
```

Listing 2: show_subscription_plans.sql

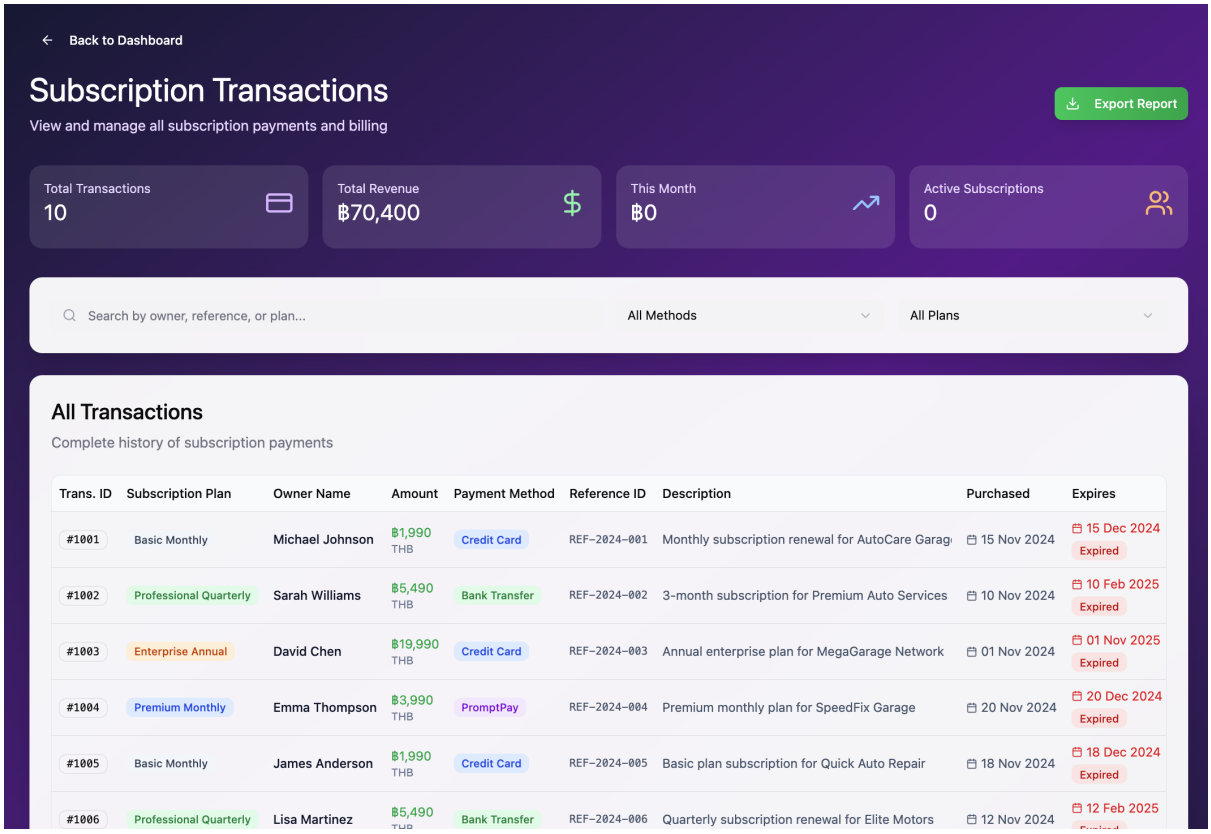


Explanation (show_subscription_plans.sql listing)
Purpose: Displays all subscription plans available for purchase by owners, supporting SaaS monetization tiers.
Logic: Simple ordered SELECT (by rank) enabling prioritized display of featured plans.

5.2.3 Subscription transaction management

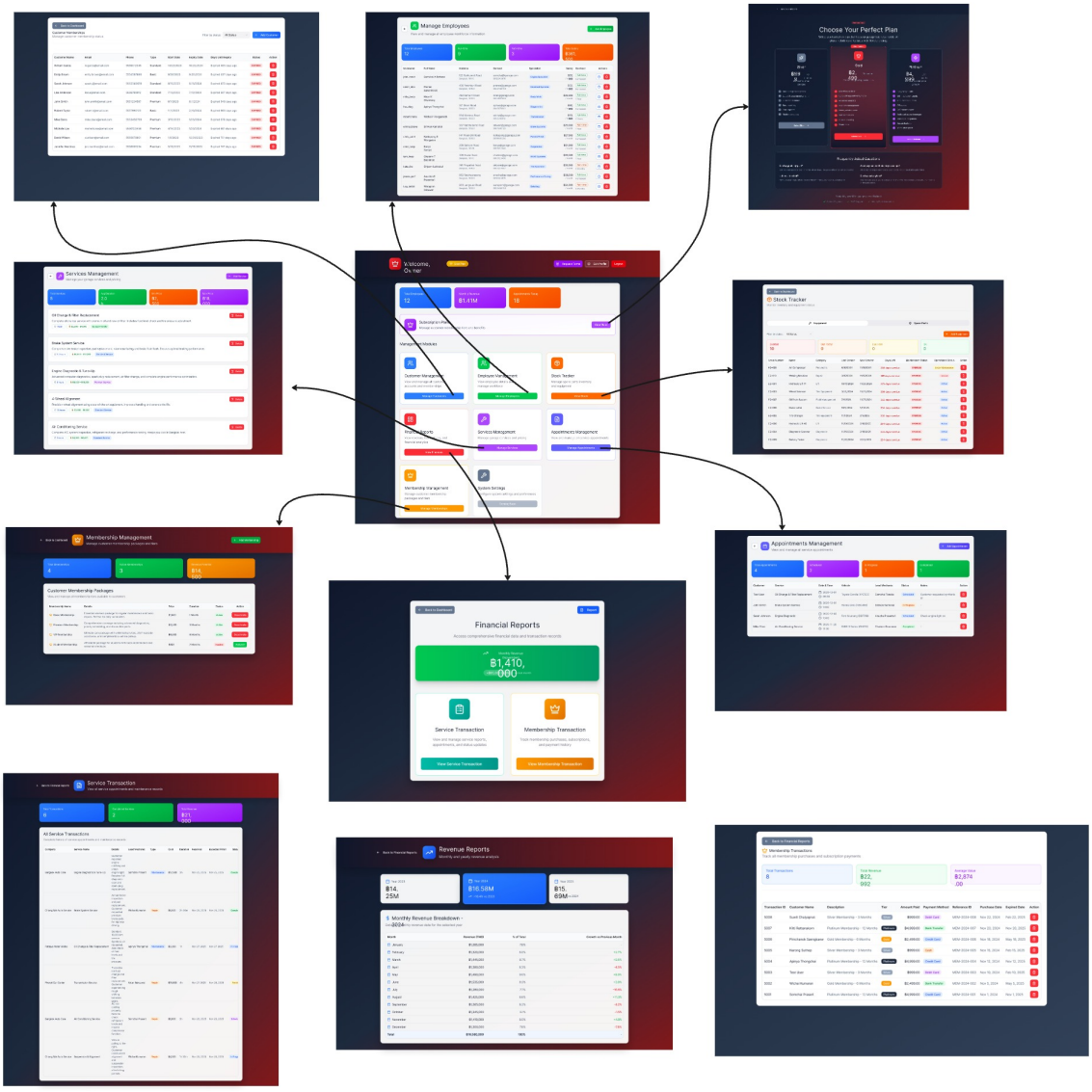
```sql
create function show_subscription_transaction()
    returns TABLE(trans_id integer, owner_name character varying,
    plan_name character varying, amount numeric, payment_method
    character varying, date_purchased date, date_expired date, status
    text)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            st.transaction_id,
            o.display_name::VARCHAR,
            sp.name::VARCHAR,
            st.amount,
            st.payment_method,
            st.purchased_date,
            st.expired_date,
            CASE
                WHEN st.expired_date < CURRENT_DATE THEN 'Expired'
                ELSE 'Active'
                END
        FROM
            subscription_transaction st
                JOIN
            subscription_plan sp ON st.subscription_id = sp.
    subscription_id
                JOIN
            owner o ON st.owner_id = o.id
        ORDER BY
            st.purchased_date DESC;
END;
$$;

alter function show_subscription_transaction() owner to
    sc_db_project_user;
```

Listing 3: show_subscription_transaction.sql



Explanation (show_subscription_transaction.sql)
Purpose: Central audit view of subscription purchases supporting revenue tracking, churn analysis, and compliance reviews.
Logic: Joins transaction with plan and owner; CASE expression derives status relative to current date.

5.3 Owner side

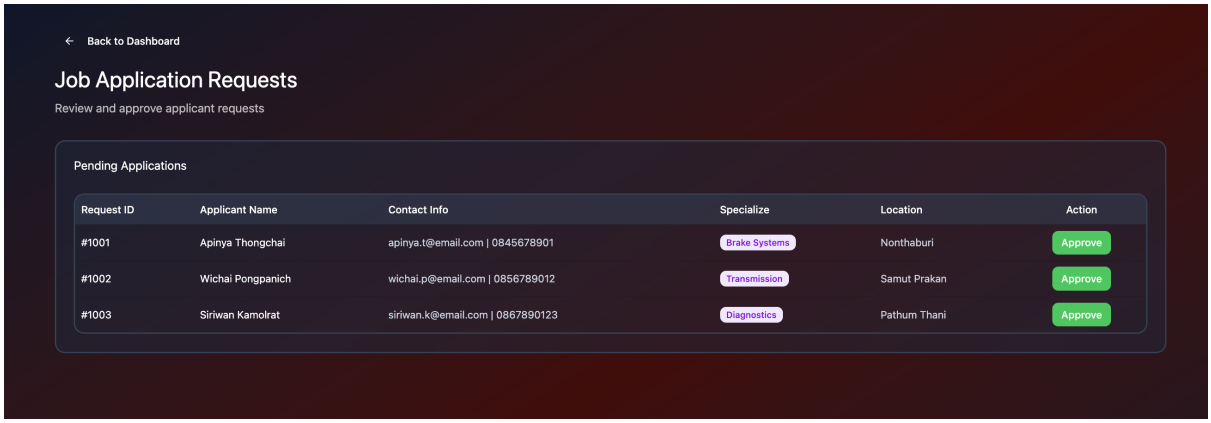


5.3.1 Approved requests and create employee

```
create function approve_request_and_create_employee(p_email character varying, p_salary numeric,
                                                    p_contract_type character varying,
                                                    p_contract_period character varying) returns integer
    language plpgsql
as
$$
DECLARE
r RECORD;
    v_new_username
VARCHAR;
    v_new_password
VARCHAR;
    v_employee_id
INTEGER;
BEGIN
SELECT *
INTO r
FROM "request_form"
WHERE request_form.email = p_email;

v_new_username
:= lower(r.first_name || '.' || r.last_name || r.request_id);
    -- Password: Default temporary password
    v_new_password
:= 'changeme1234';

    v_employee_id
:= add_employee(
            r.company_id,
            v_new_username,
            r.first_name,
            r.middle_name,
            r.last_name,
            r.address,
            r.city,
            r.postal_code,
            r.email,
            r.telephone_no,
            r.specialize,
            v_new_password,     -- Generated Password
            p_salary,
            CURRENT_DATE,
            p_contract_type,
            p_contract_period
    );

    -- delete the request
DELETE
FROM "request_form"
WHERE email = p_email;

RETURN v_employee_id;
END;
```

```
$$;

alter function approve_request_and_create_employee(varchar, numeric,
    varchar, varchar) owner to sc_db_project_user;
```

Listing 4: approve_request_and_create_employee.sql

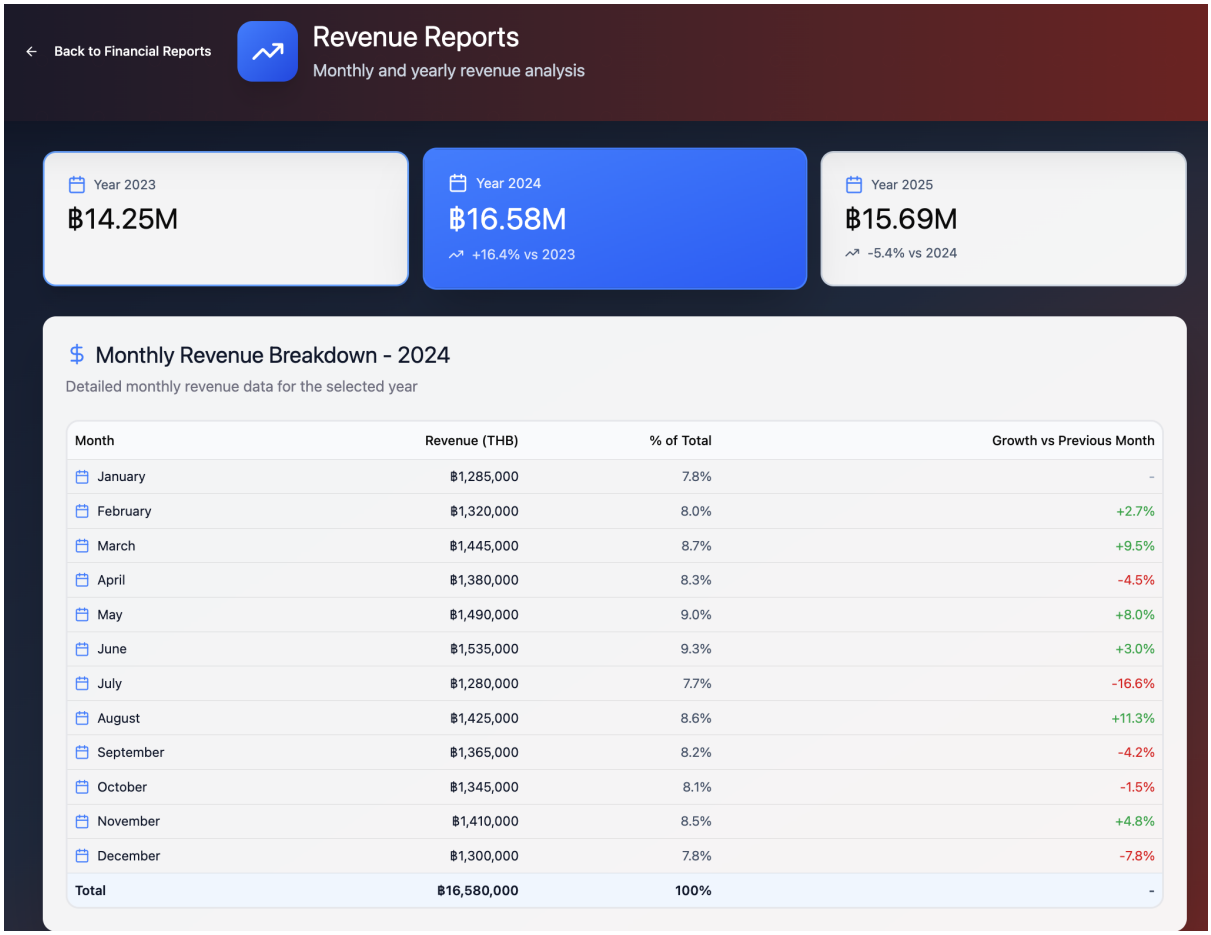


Explanation (approve_request_and_create_employee.sql)
Purpose: Automates hiring: transforms a pending applicant (request form) into an employee record, then removes the source request to prevent duplication.

5.3.2 yearly financial summary

```
create function get_yearly_financial_summary(p_company_id integer,
    p_year integer)
    returns TABLE
            (
                year                    integer,
                total_revenue_year      character varying,
                total_net_income_year character varying
            )
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT f.year,
               (COALESCE(SUM(total_revenue), 0.00)::VARCHAR || ' THB')
    ::VARCHAR,
               (COALESCE(SUM(net_income), 0.00)::VARCHAR || ' THB')::
    VARCHAR
        FROM "financial_record" AS f
        WHERE company_id = p_company_id
          AND f.year = p_year
        group by f.year;
END;
$$;

alter function get_yearly_financial_summary(integer, integer) owner to
    sc_db_project_user;
```

Listing 5: get_yearly_financial_summary.sql



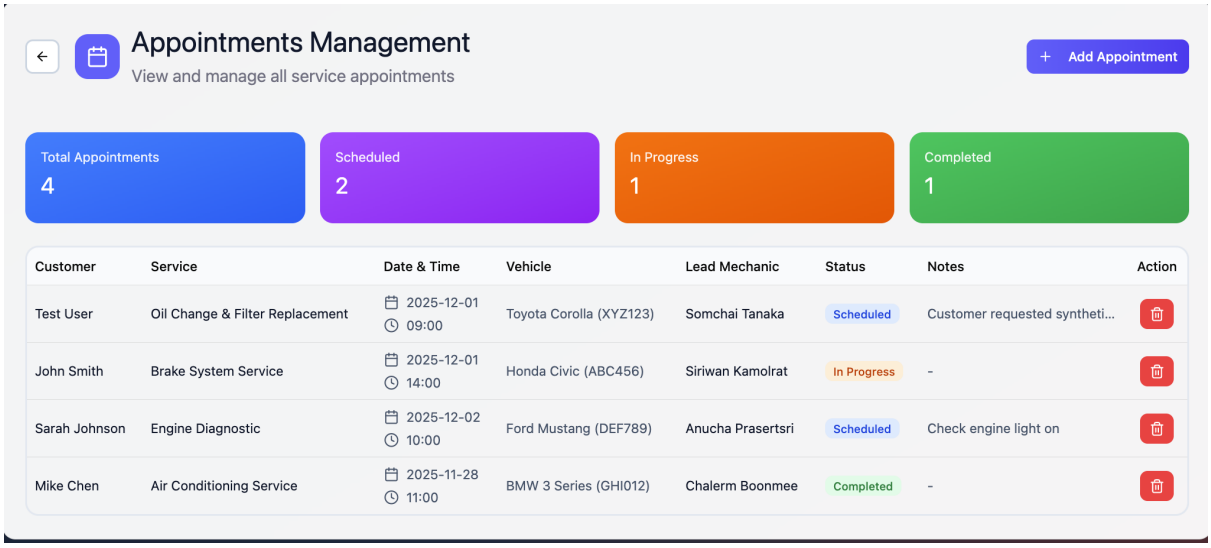
Month	Revenue (THB)	% of Total	Growth vs Previous Month
January	฿1,285,000	7.8%	-
February	฿1,320,000	8.0%	+2.7%
March	฿1,445,000	8.7%	+9.5%
April	฿1,380,000	8.3%	-4.5%
May	฿1,490,000	9.0%	+8.0%
June	฿1,535,000	9.3%	+3.0%
July	฿1,280,000	7.7%	-16.6%
August	฿1,425,000	8.6%	+11.3%
September	฿1,365,000	8.2%	-4.2%
October	฿1,345,000	8.1%	-1.5%
November	฿1,410,000	8.5%	+4.8%
December	฿1,300,000	7.8%	-7.8%
Total	**฿16,580,000**	**100%**	-

Explanation (get_yearly_financial_summary.sql)
Purpose: Produces aggregated yearly totals (revenue, net income) for dashboard totals and comparative analytics.
Logic: COALESCE + SUM across `financial_record` filtered by company and year; formatted with currency suffix for direct UI display.

5.3.3 Appointment management

```sql
create function show_appointments(p_company_id integer)
    returns TABLE(appointment_id integer, customer_name text,
    vehicle_info text, date_str text, time_str text, service_detail
    character varying, status character varying)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            a.id AS appointment_id,
            (c.first_name || ' ' || c.last_name)::TEXT,
            (v.brand || ' ' || v.model)::TEXT,
            to_char(a.appointment_date, 'MM/DD/YYYY')::TEXT,
            to_char(a.appointment_date, 'HH12:MI AM')::TEXT,
            a.details,
            a.status
        FROM "appointment" a
                 JOIN "vehicle" v ON a.vehicle_id = v.id
                 JOIN "customer" c ON v.owner_id = c.id
        WHERE a.company_id = p_company_id

        ORDER BY
            CASE
                WHEN a.status = 'pending' THEN 1
                WHEN a.status = 'accepted' THEN 2
                ELSE 3
                END ,
            a.appointment_date;
END;
$$;

alter function show_appointments(integer) owner to sc_db_project_user;
```

Listing 6: show_appointments.sql



Explanation (show_appointments.sql)
Purpose: Retrieves enriched appointment listing for operational scheduling and status track-

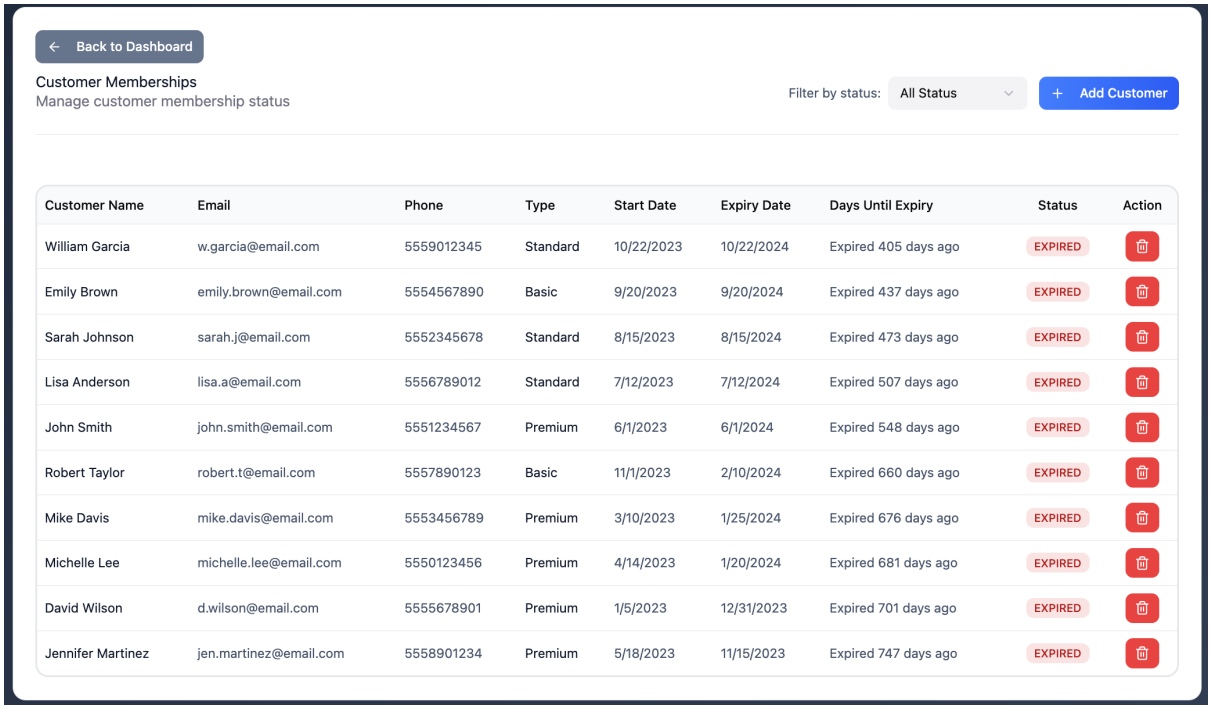
ing.

Logic: Multi-table join (appointment, vehicle, customer) plus CASE order block ensuring pending/accepted appear first.

5.3.4 Customer management

```sql
create function show_company_customers(p_company_id integer)
    returns TABLE(customer_name character varying, email character varying, phone character varying, membership_type character varying, start_date date, expiry_date date, days_until_expiry character varying, status character varying)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT (c.first_name || ' ' || c.last_name)::character varying AS customer_name,
               c.email,
               c.telephone_no,
               mp.name,
               m.purchased_date,
               m.expired_date,
               CASE
                   WHEN m.expired_date < CURRENT_DATE THEN
                       'Expired ' || (CURRENT_DATE - m.expired_date)::VARCHAR || ' days ago'
                   WHEN m.expired_date = CURRENT_DATE THEN
                       'Expires today'
                   ELSE
                       (m.expired_date - CURRENT_DATE)::VARCHAR || ' days left'
                   END::VARCHAR,
               CASE
                   WHEN m.expired_date < CURRENT_DATE THEN 'EXPIRED'
                   ELSE 'ACTIVE'
                   END::VARCHAR

        FROM "customer" c
                 JOIN "membership" m ON c.id = m.customer_id
                 JOIN "membership_plan" mp ON m.membership_plan_id = mp.membership_plan_id
        WHERE mp.company_id = p_company_id
        ORDER BY mp.rank DESC, m.expired_date DESC;
END;
$$;

alter function show_company_customers(integer) owner to sc_db_project_user;
```

Listing 7: show_company_customers.sql

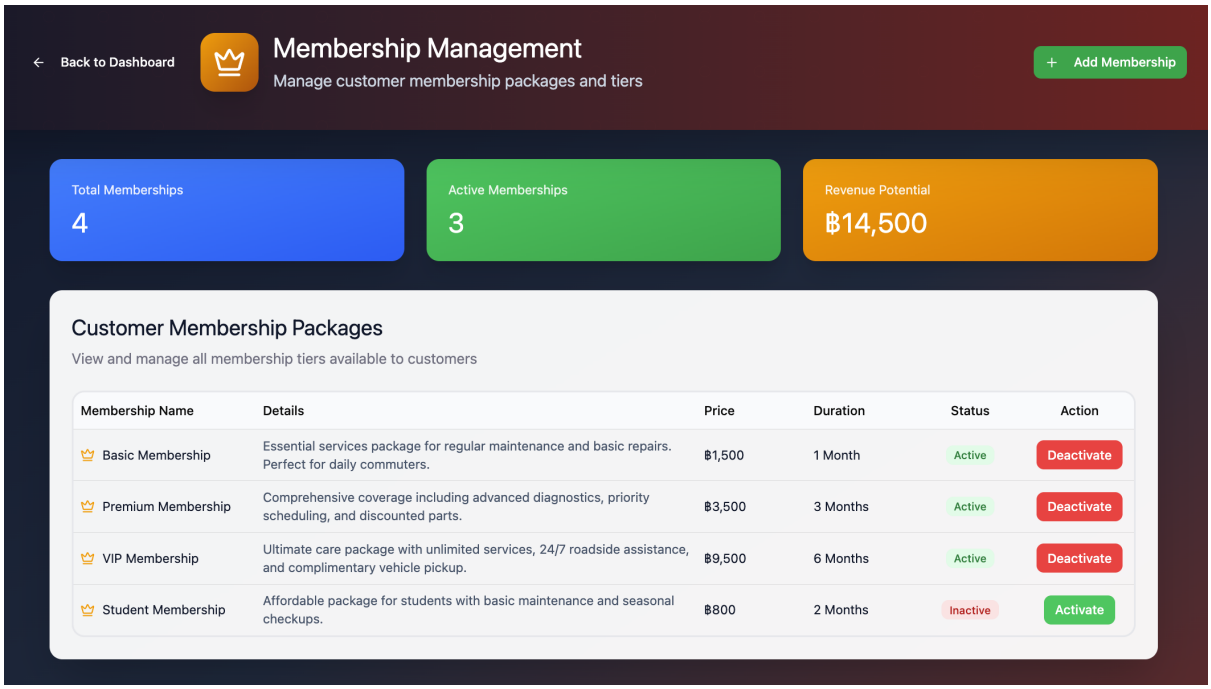


Explanation (show_company_customers.sql)
Purpose: Surfaces membership retention context (expiry countdown, active state) for owner renewal strategies.
Logic: Joins customer, membership, and membership plan; CASE generates human-readable expiry messages.

5.3.5 Membership plan management

```sql
create function show_company_membership_plans(p_company_id integer)
    returns TABLE(plan_id integer, plan_name character varying,
    plan_detail text, plan_rank integer, price_thb numeric,
    duration_days integer)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            membership_plan_id,
            name,
            detail,
            rank,
            price,
            duration
        FROM "membership_plan"
        WHERE company_id = p_company_id
        ORDER BY rank ;
END;
$$;

alter function show_company_membership_plans(integer) owner to
    sc_db_project_user;
```

Listing 8: show_company_membership_plans.sql

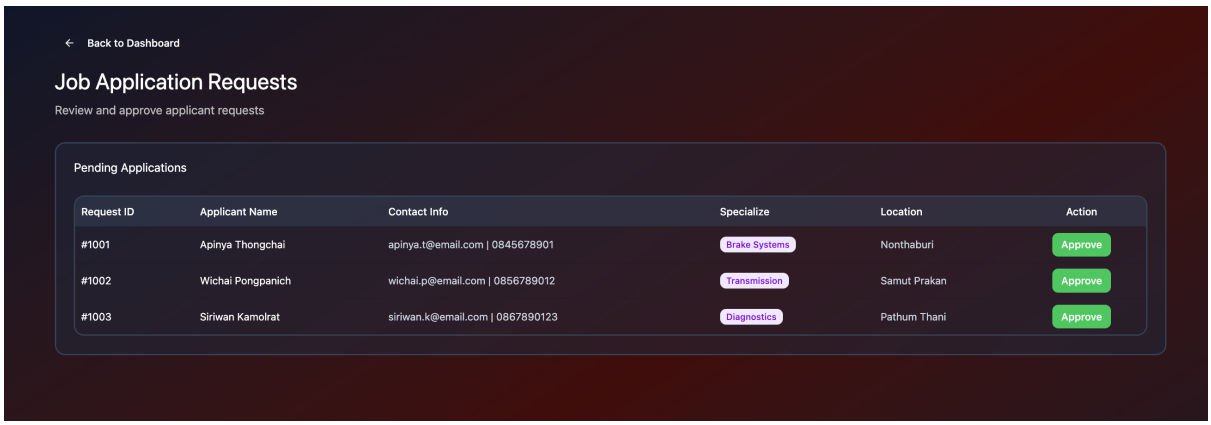


Explanation (show_company_membership_plans.sql)
Purpose: Displays company-specific loyalty offerings with rank-based ordering.

5.3.6 Employment request management

```sql
create function show_company_requests(p_company_id integer)
    returns TABLE(req_id integer, applicant_name text, contact_info
    text, specialize character varying, location text)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            r.request_id,
            (r.first_name || ' ' || r.last_name)::TEXT,
            (r.email || ' | ' || r.telephone_no)::TEXT,
            r.specialize,
            (r.city || ', ' || r.postal_code)::TEXT
        FROM "request_form" r
        WHERE r.company_id = p_company_id;
END;
$$;

alter function show_company_requests(integer) owner to
    sc_db_project_user;
```

Listing 9: show_company_requests.sql

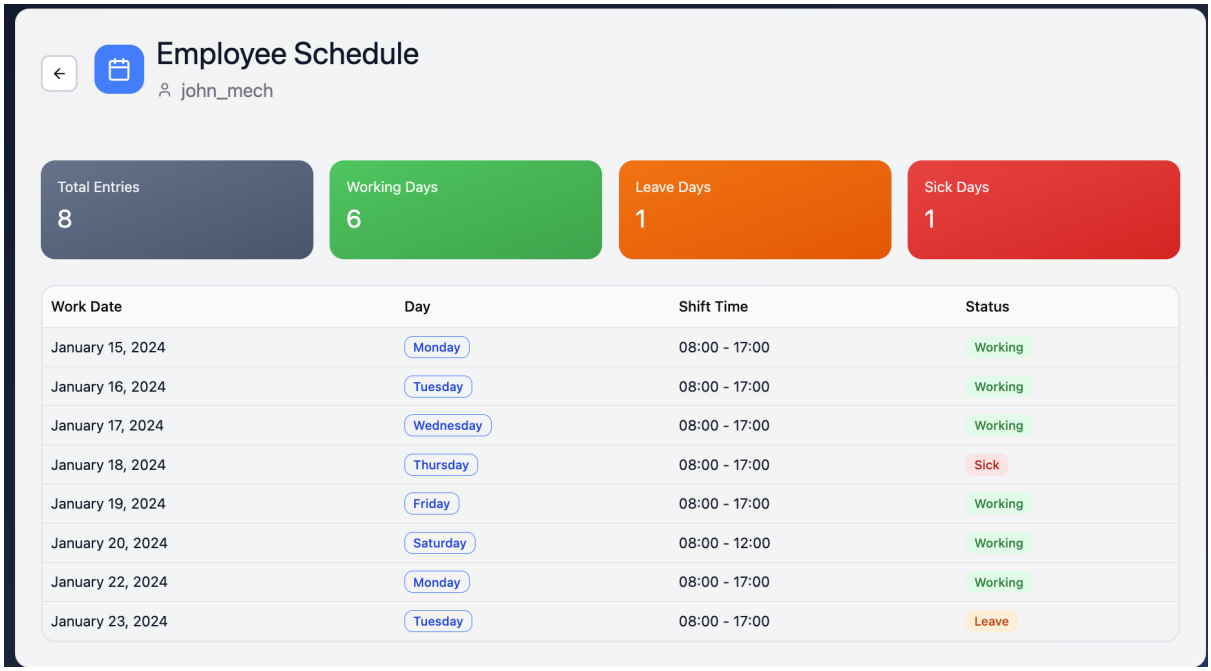


Explanation (show_company_requests.sql)
Purpose: Lists employment application requests.

5.3.7 Employee schedule management

```sql
create function show_employee_schedule(p_employee_id integer)
    returns TABLE(work_date text,
                  day_name text,
                  shift_time text,
                  status character varying)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            to_char(s.date, 'DD/MM/YYYY')::TEXT,
            TRIM(to_char(s.date, 'Day'))::TEXT,
            (to_char(s.start_time, 'HH24:MI') || ' - ' || to_char(s.
    end_time, 'HH24:MI'))::TEXT,
            s.status
        FROM employee_schedule s
        WHERE s.employee_id = p_employee_id
        ORDER BY s.date;
END;
$$;

alter function show_employee_schedule(integer) owner to
    sc_db_project_user;
```

Listing 10: show_employee_schedule.sql

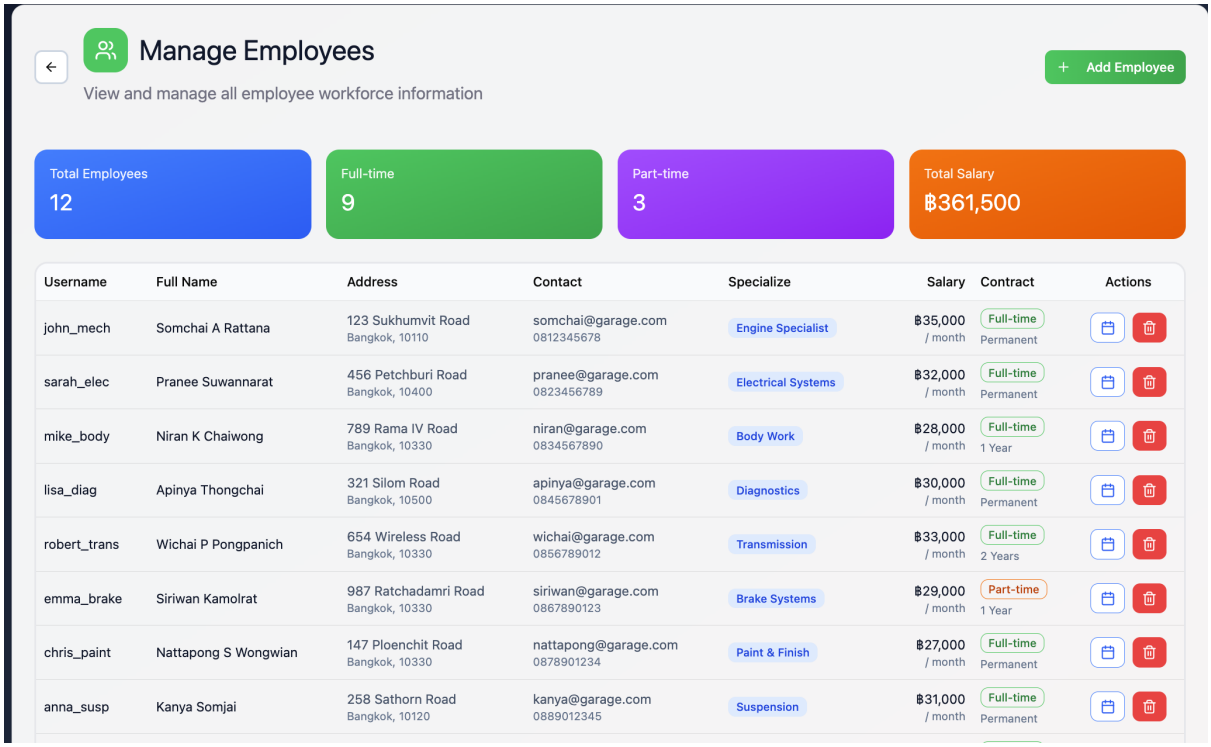


Explanation (show_employee_schedule.sql)
Purpose: Provides employee shift visibility—date, day name, time window, status.

5.3.8 Employee management

```sql
create function show_employee(p_company_id integer)
    returns TABLE(company_name character varying, username character
    varying, first_name character varying, middle_name character varying
    , last_name character varying, address character varying, city
    character varying, postal_code character varying, email character
    varying, telephone_no character varying, specialize character
    varying)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            (SELECT c.name FROM company AS c
             WHERE c.id = e.company_id),
            e.username,
            e.first_name,
            e.middle_name,
            e.last_name,
            e.address,
            e.city,
            e.postal_code,
            e.email,
            e.telephone_no,
            e.specialize
        FROM employee e
        WHERE e.company_id = p_company_id;
END;
$$;

alter function show_employee(integer) owner to sc_db_project_user;
```

Listing 11: show_employee.sql

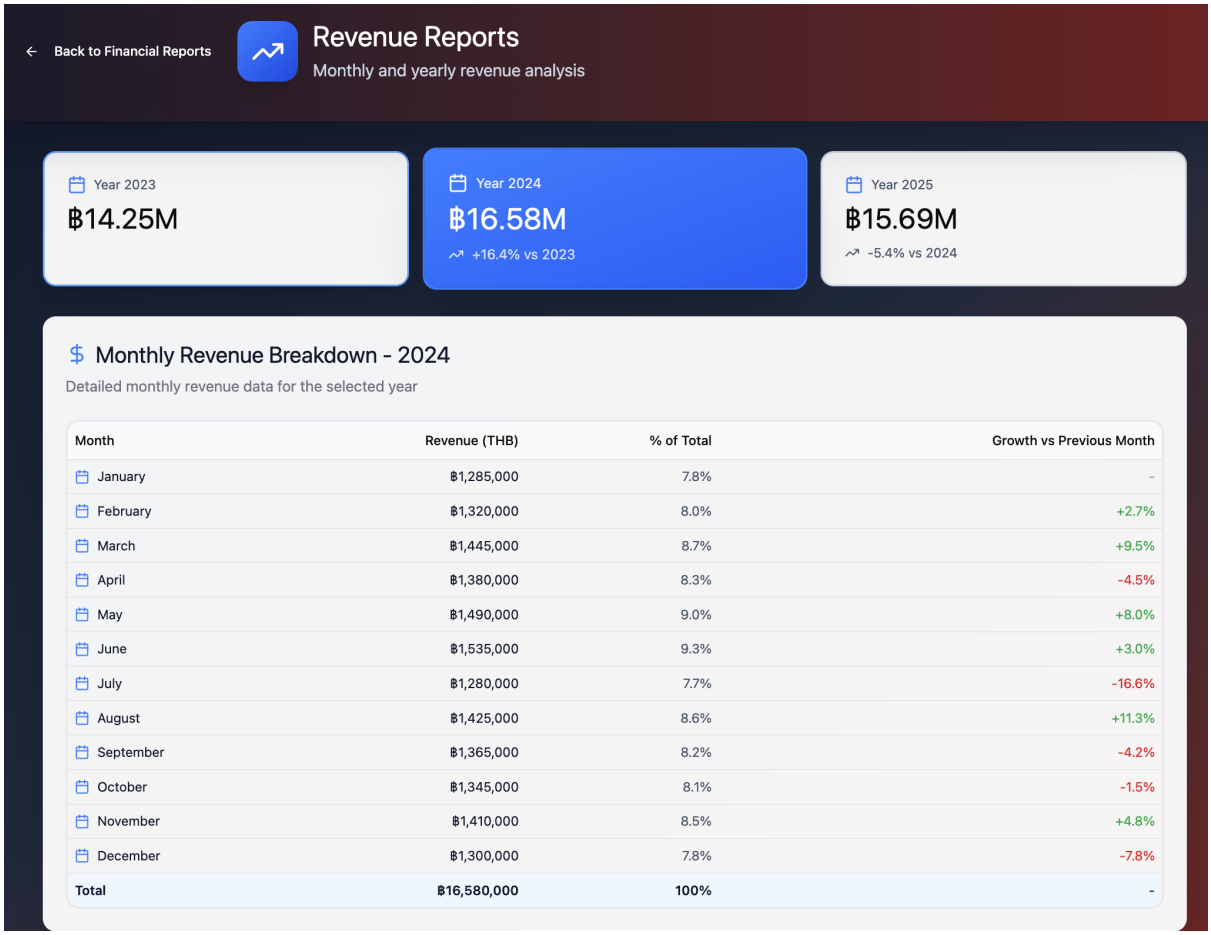


Explanation (show_employee.sql)
Purpose: Roster listing with personal + specialization data for management views.
Logic: Single table filtered by company id; potential to extend with role or permission columns.

5.3.9 Financial summary

```sql
create function show_financial_records(p_company_id integer)
    returns TABLE(record_id integer, company_name character varying,
    period character varying, total_revenue character varying,
    net_income character varying, date_reported date)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT fr.record_id,
               c.name::character varying,
               (fr.month || '/' || fr.year)::character varying,
               (fr.total_revenue::character varying || ' THB'),
               (fr.net_income::character varying || ' THB'),
               fr.date_reported
        FROM "financial_record" fr
                 JOIN "company" c ON fr.company_id = c.id
        WHERE fr.company_id = p_company_id
        ORDER BY fr.year DESC, fr.month DESC;
END;
$$;

alter function show_financial_records(integer) owner to
    sc_db_project_user;
```

Listing 12: show_financial_records.sql

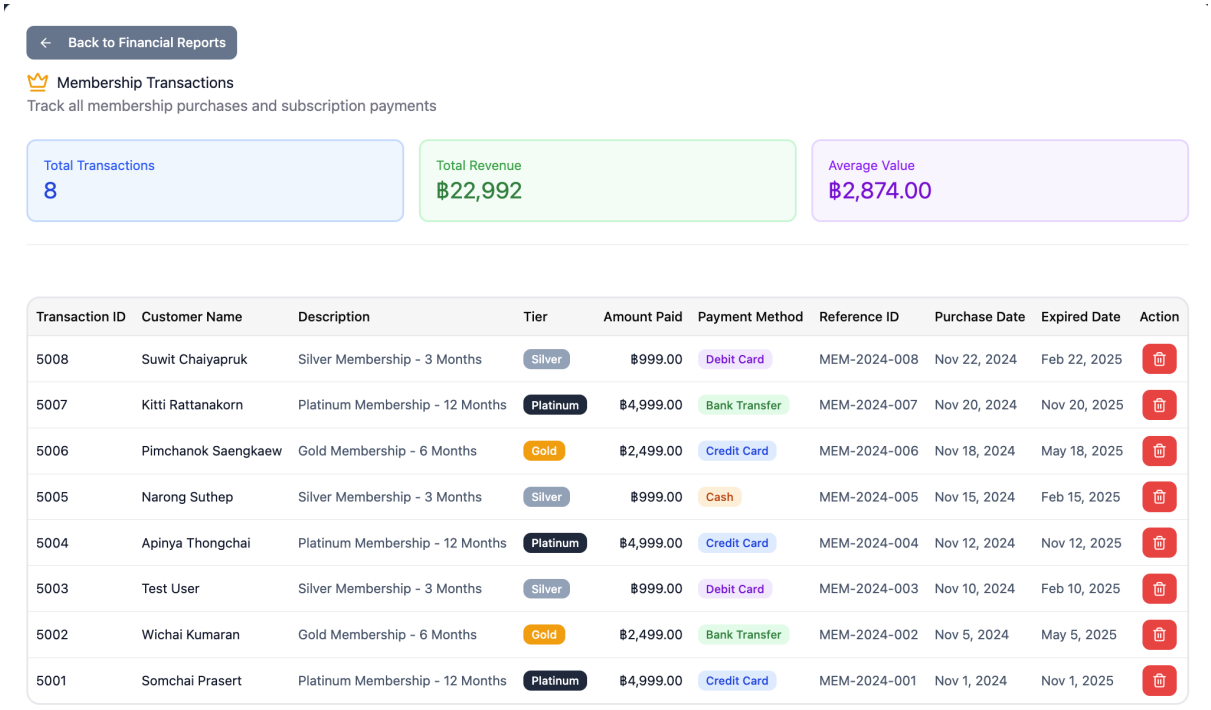


Explanation (show_financial_records.sql)
Purpose: Monthly ledger for trend analysis and gross/net performance tracking.
Logic: Join with company for naming; reverse chronological ordering surfaces recent performance first.

5.3.10 Membership transaction

```sql
create function show_membership_transactions(p_company_id integer)
    returns TABLE
            (
                transaction_id integer,
                customer_name  text,
                amount_paid    text,
                payment_method character varying,
                reference_id   character varying,
                purchased_date date,
                expired_date   date,
                description    text
            )
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT mt.id,
               (c.first_name || ' ' || c.last_name)::TEXT AS customer_name,
               (mt.amount::TEXT || ' ' || mt.currency),
               mt.payment_method,
               mt.reference_id,
               mt.purchased_date,
               mt.expired_date,
               mt.description

        FROM "membership_transaction" mt
                 JOIN "membership" m ON mt.member_id = m.id
                 JOIN "customer" c ON m.id = c.member_id
                 JOIN "membership_plan" mp ON m.membership_plan_id = mp.membership_plan_id

        WHERE mp.company_id = p_company_id
        ORDER BY mt.purchased_date DESC;
END;
$$;

alter function show_membership_transactions(integer) owner to sc_db_project_user;
```

Listing 13: show_membership_transactions.sql

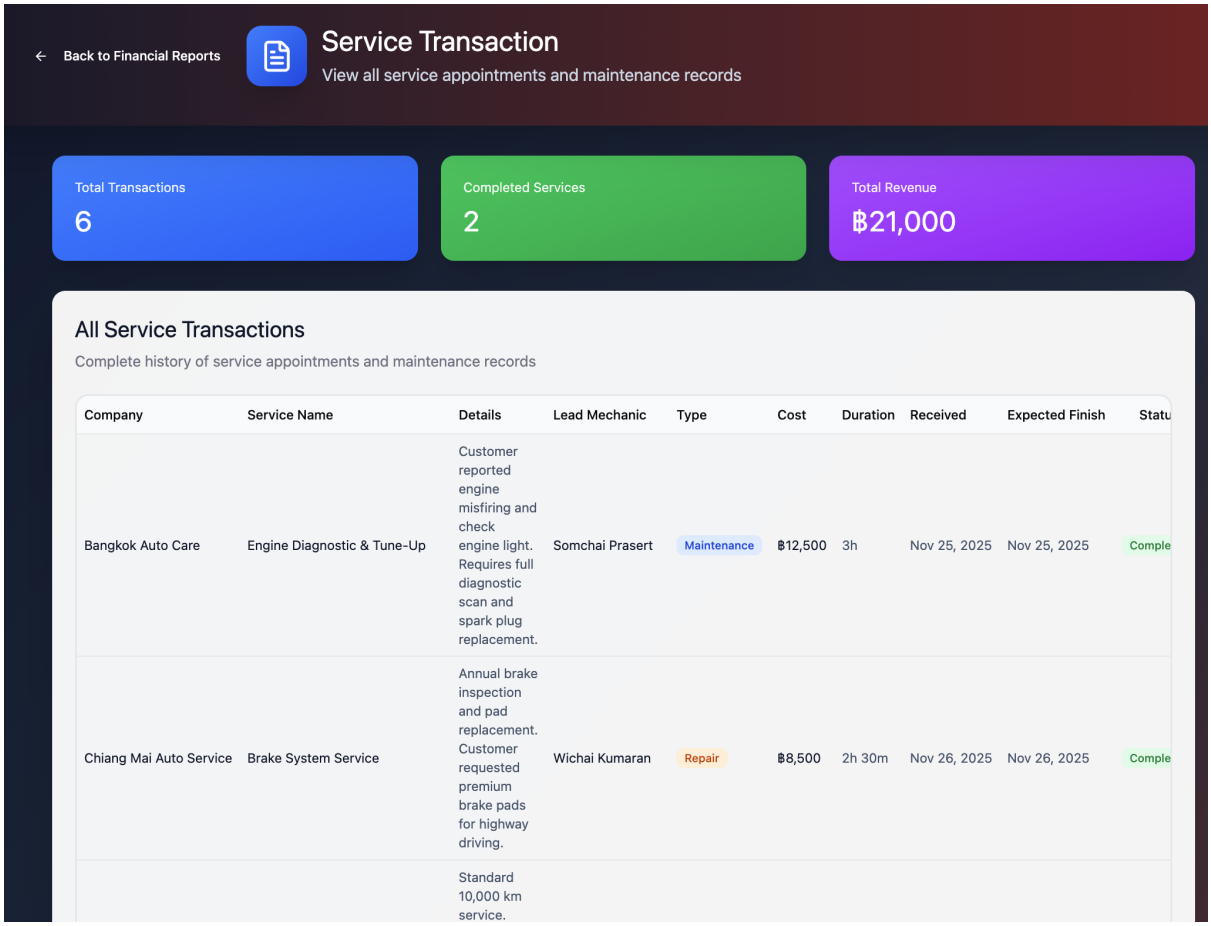


Explanation (show_membership_transactions.sql)
Purpose: Detailed purchase history for membership monetization and customer lifecycle tracking.
Logic: Joins membership transaction with membership, customer, and plan; denormalizes names and descriptions.

5.3.11 Service Record

```sql
create function show_service_records(p_company_id integer)
    returns TABLE
            (
                company_name          character varying,
                service_name          character varying,
                appointment_detail    character varying,
                lead_mechanic_name    character varying,
                record_type           character varying,
                cost                  integer,
                duration_minutes      integer,
                received_date         date,
                expected_finish_date  date,
                status                character varying
            )
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT c.name::VARCHAR                                    AS company_name,
               s.name::VARCHAR                                    AS service_name,
               COALESCE(a.details, 'N/A')::VARCHAR                AS appointment_detail,
               (e.first_name || ' ' || e.last_name)::VARCHAR AS lead_mechanic_name,

               sr.type,
               sr.cost,
               sr.duration,
               sr.received_date,
               sr.expected_finish_date,
               sr.status

        FROM "service_record" sr
                 JOIN "service" s ON sr.service_id = s.id
                 JOIN "company" c ON s.company_id = c.id
                 JOIN "appointment" a ON sr.appointment_id = a.id
                 LEFT JOIN "employee" e ON sr.lead_mechanic_id = e.id

        WHERE s.company_id = p_company_id
        ORDER BY sr.received_date DESC;
END;
$$;

alter function show_service_records(integer) owner to sc_db_project_user;
```

Listing 14: show_service_records.sql

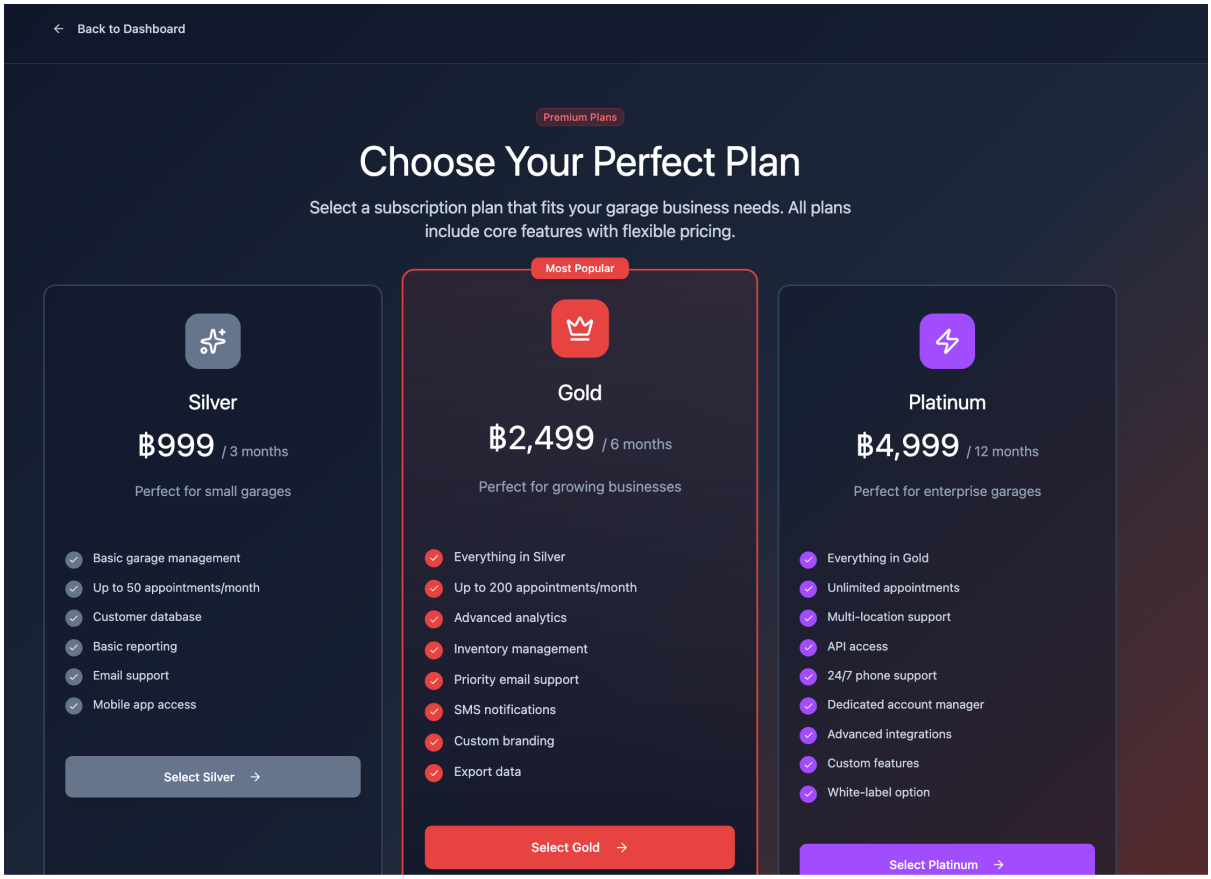


Explanation (show_service_records.sql)
Purpose: Operational and historical view of service execution, including lead mechanic and timing.
Logic: Multi-table joins; COALESCE safeguards missing appointment details.

5.3.12 Subscription plan

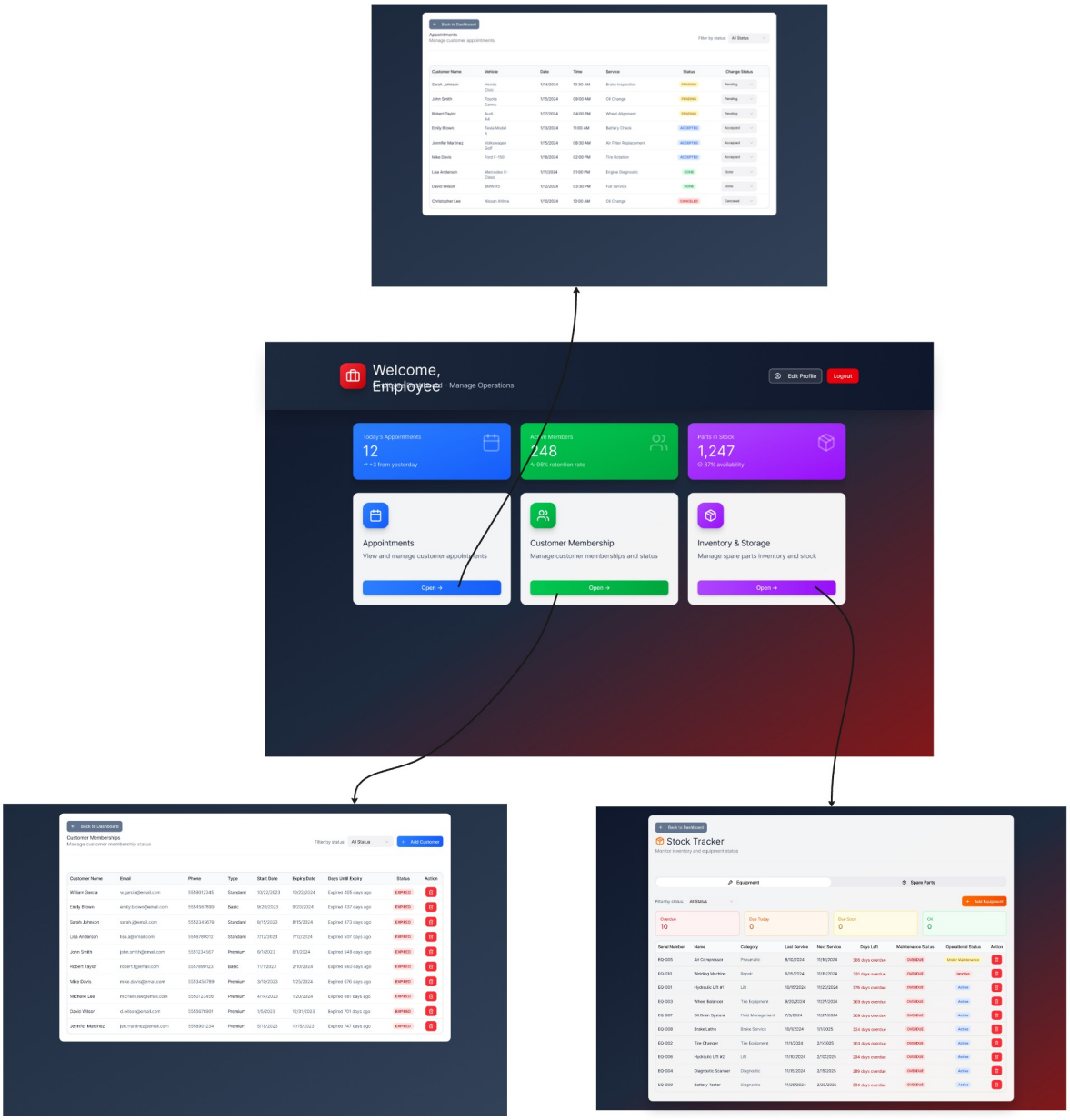
```sql
create function show_subscription_plans()
    returns TABLE(plan_id integer, plan_name character varying, details
    text, price_thb numeric, duration_days integer)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            subscription_id,
            name,
            detail,
            price,
            duration
        FROM "subscription_plan"
        ORDER BY rank;
END;
$$;

alter function show_subscription_plans() owner to sc_db_project_user;
```

Listing 15: show_subscription_plans.sql



Explanation (show_subscription_plans.sql owner view)
Purpose: Owner-facing list of available subscription options with pricing and duration.

5.4 Employee side



They shared some features with the owner.

eg. show_appointments.sql, show_employee_schedule.sql, show_service_records.sql

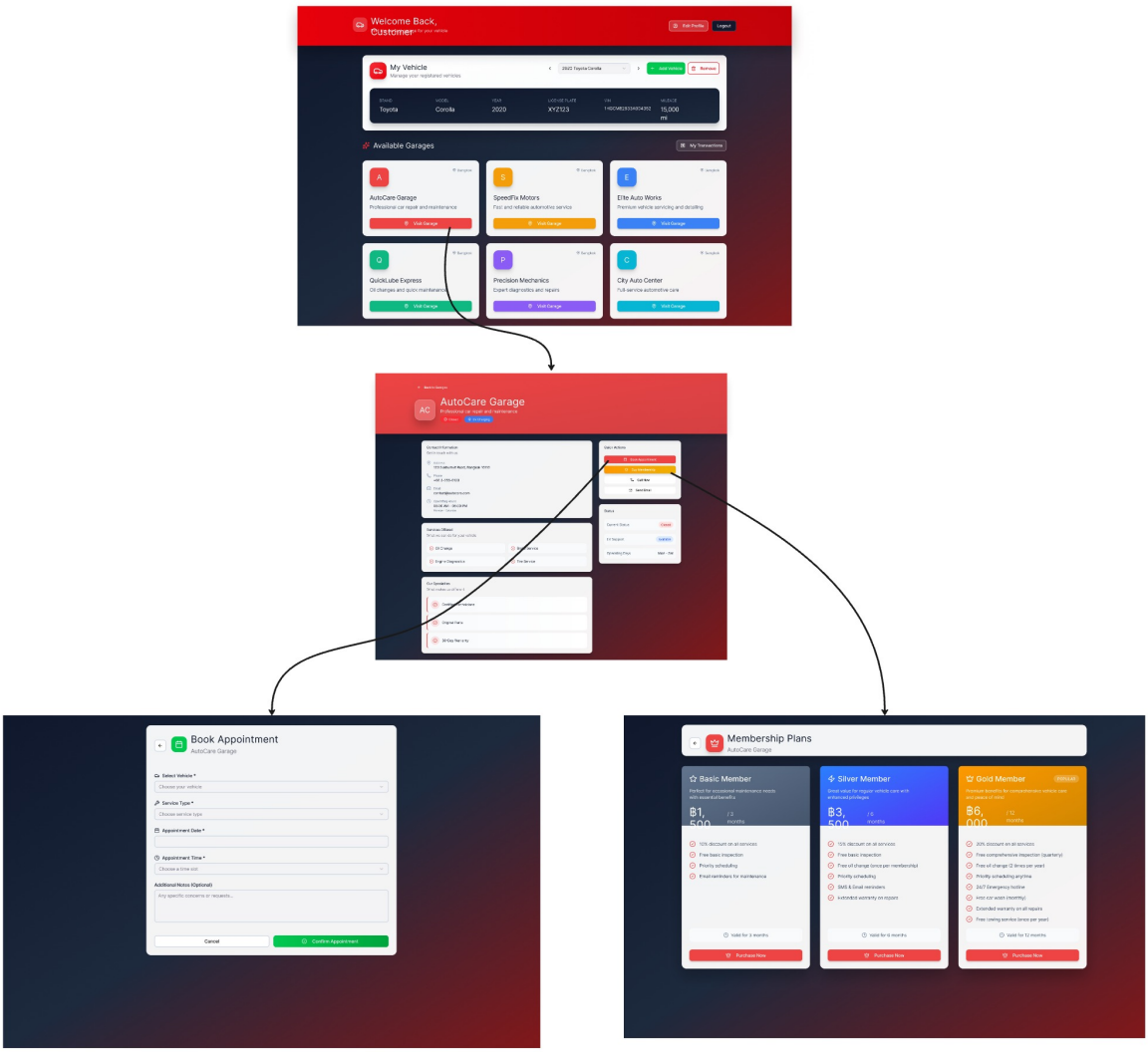
5.4.1 Change appointment status

```sql
create function update_appointment_status(p_appointment_id integer, p_new_status character varying) returns integer
    language plpgsql
as
$$
DECLARE
    v_rows_updated INTEGER;
BEGIN
    UPDATE "appointment"
    SET status = p_new_status
    WHERE id = p_appointment_id;
    GET DIAGNOSTICS v_rows_updated = ROW_COUNT;
    IF v_rows_updated > 0 THEN
        RETURN 1;
    ELSE
        RETURN 0;
    END IF;
END;
$$;

alter function update_appointment_status(integer, varchar) owner to sc_db_project_user;
```

Listing 16: update_appointment_status.sql

Explanation (update_appointment_status.sql)
Purpose: Performs controlled state mutation on a single appointment row.
Logic: UPDATE followed by row count diagnostic; returns 1 (success) or 0 (no row changed).

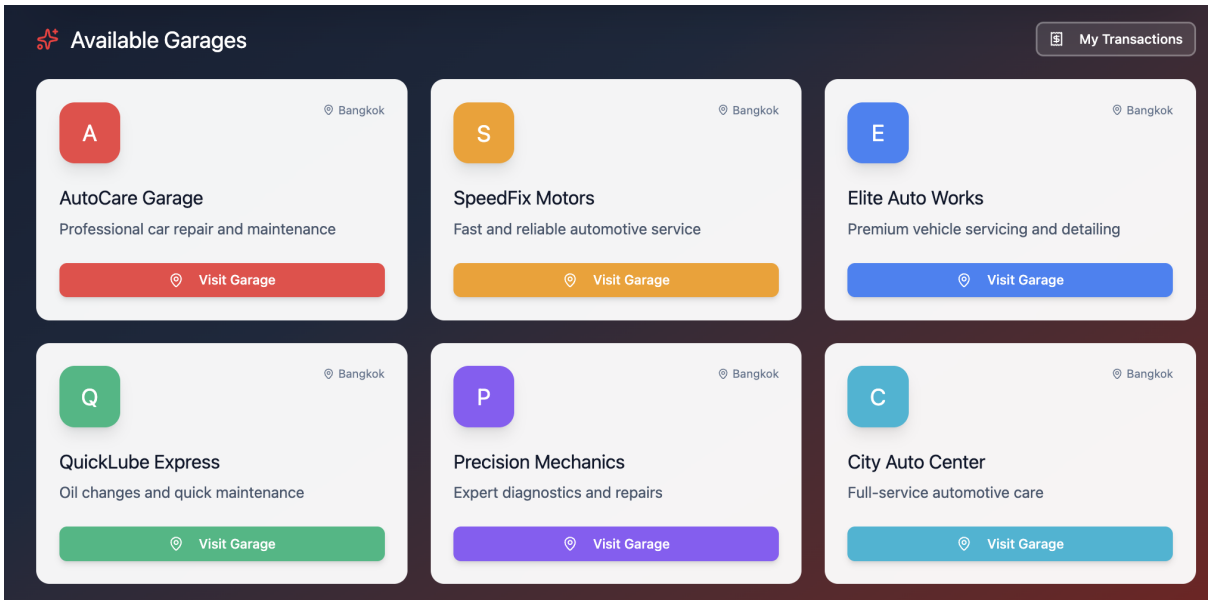
5.5 Customer side



5.5.1 Companys menu

```sql
create function show_company()
    returns TABLE(owner_full_name character varying, company_name
    character varying, logo_url character varying, address character
    varying, city character varying, postal_code character varying,
    capacity integer, open_time time without time zone, close_time time
    without time zone, social_media_contact character varying,
    ev_support boolean)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            (e.first_name || ' ' || e.last_name)::VARCHAR AS
    owner_full_name,
            c.name,
            c.logo_url,
            c.address,
            c.city,
            c.postal_code,
            c.capacity,
            c.open_time,
            c.close_time,
            c.social_media_contact,
            c.ev_support

        FROM company c
                 JOIN employee e ON c.employee_id = e.id;
END;
$$;

alter function show_company() owner to sc_db_project_user;
```

Listing 17: show_company.sql



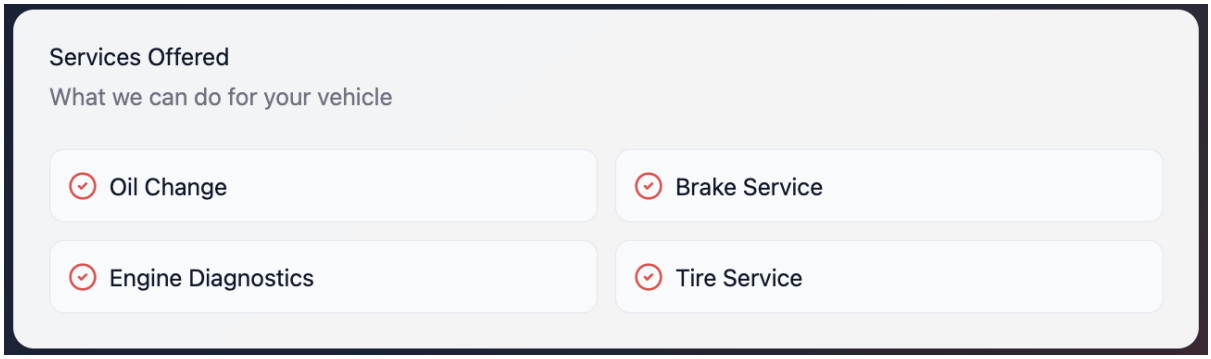
Explanation (show_company.sql)

Purpose: Customer discovery of available garages with key operational attributes.
Logic: Join company with an employee (likely owner mapping) to present ownership context.

5.5.2 Service menu

```sql
create function show_company_services(p_company_id integer)
    returns TABLE(service_name character varying, service_detail
    character varying, duration_hour numeric, minimum_cost numeric,
    maximum_cost numeric)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            name,
            detail,
            approximate_duration,
            min_cost,
            max_cost
        FROM "service"
        WHERE company_id = p_company_id;
END;
$$;

alter function show_company_services(integer) owner to
    sc_db_project_user;
```

Listing 18: show_company_services.sql

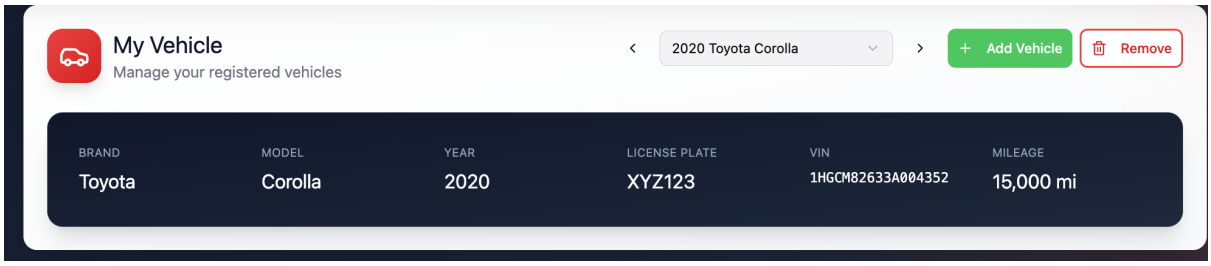


Explanation (show_company_services.sql)
Purpose: Displays service menu with estimated time and price band for booking decisions.

5.5.3 Vehicle management

```sql
create function show_vehicle_by_owner_id(p_owner_id integer)
    returns TABLE(p_brand character varying, p_model character varying,
    p_year integer, p_license_plate character varying, p_vin character
   varying, p_mileage integer)
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT
            v.brand,
            v.model,
            v.year,
            v.license_plate,
            v.vin,
            v.mileage
        FROM "vehicle" v
        WHERE v.owner_id = p_owner_id;
END;
$$;

alter function show_vehicle_by_owner_id(integer) owner to
   sc_db_project_user;
```

Listing 19: show_vehicle_by_owner_id.sql

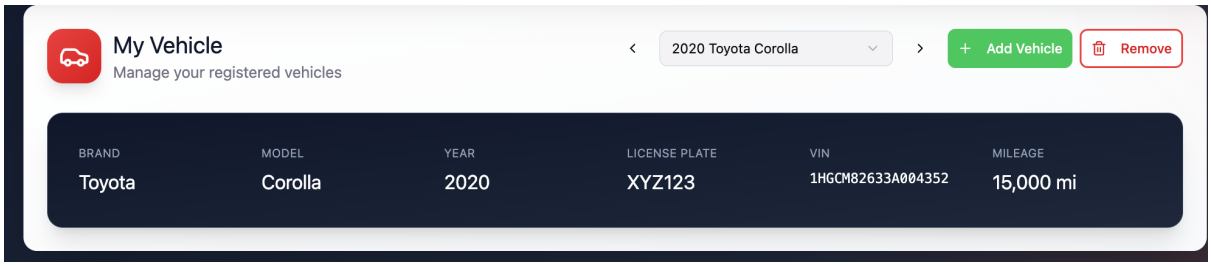


Explanation (show_vehicle_by_owner_id.sql)
Purpose: Enables customers to select from their registered vehicles during appointment creation.

5.5.4 Remove vehicle

```sql
create function remove_vehicle(p_vin character varying) returns boolean
    language plpgsql
as
$$
DECLARE
    v_deleted_count INT;
BEGIN
    DELETE FROM "vehicle"
    WHERE vin = p_vin;

    -- Check if a row was actually deleted
    GET DIAGNOSTICS v_deleted_count = ROW_COUNT;

    -- Return TRUE if deleted, FALSE if not found or mismatch
    RETURN v_deleted_count > 0;
END;
$$;

alter function remove_vehicle(varchar) owner to sc_db_project_user;
```

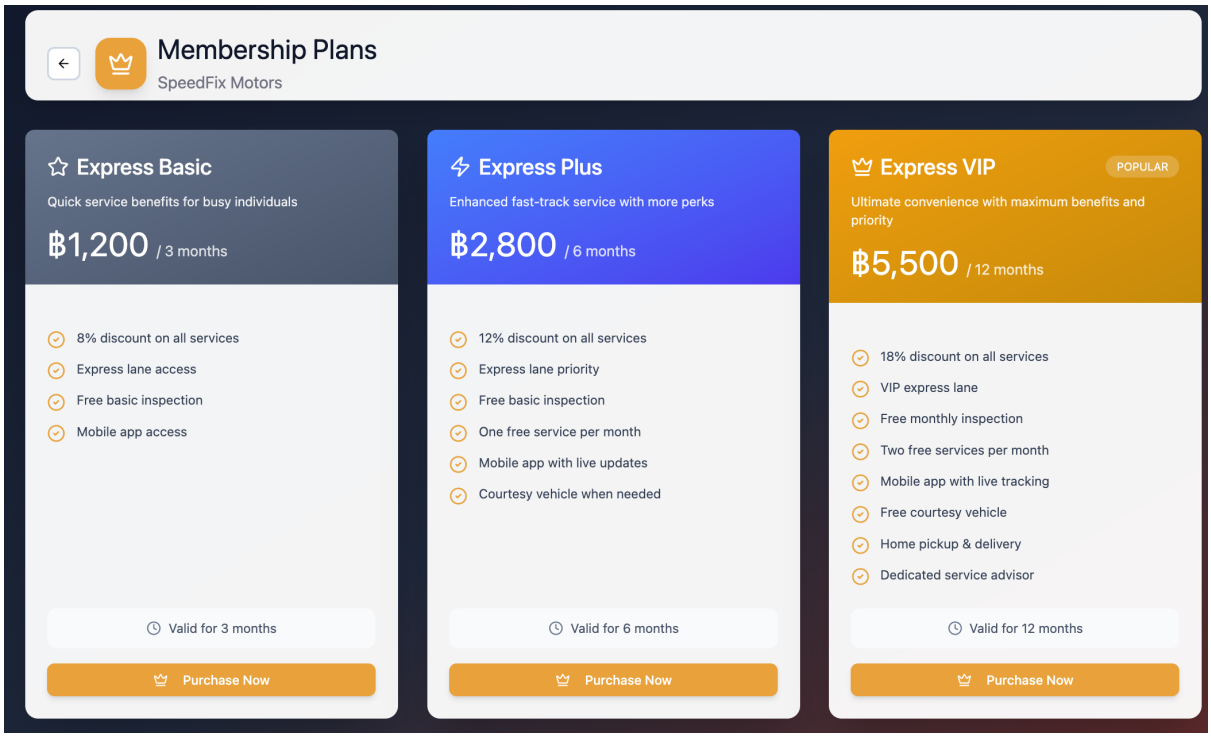
Listing 20: remove_vehicle.sql



Explanation (remove_vehicle.sql)
Purpose: Deletes a vehicle no longer owned or mistakenly registered.
Logic: DELETE by VIN, returns TRUE if any row removed.

```sql
create function show_membership_plans_for_customers(p_company_id integer)
    returns TABLE
            (
                plan_name      character varying,
                plan_detail    text,
                plan_rank      integer,
                price_thb      numeric,
                duration_days integer
            )
    language plpgsql
as
$$
BEGIN
    RETURN QUERY
        SELECT name,
               detail,
               rank,
               price,
               duration
        FROM "membership_plan"
        WHERE company_id = p_company_id
          AND is_active = TRUE -- Only show active plans
        ORDER BY rank;
END;
$$;

alter function show_membership_plans_for_customers(integer) owner to sc_db_project_user;
```

Listing 21: show_membership_plans_for_customers.sql

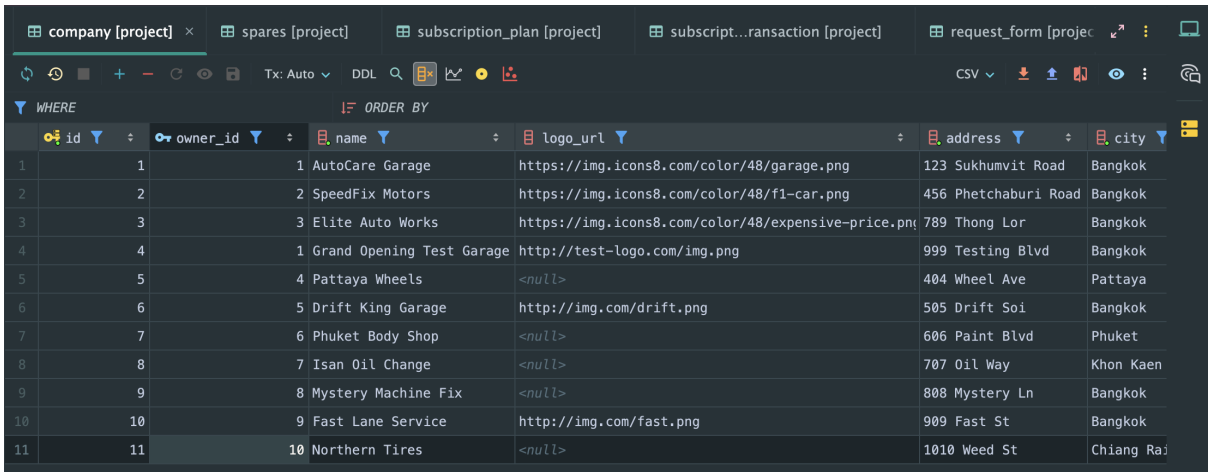


Explanation (show_membership_plans_for_customers.sql)

Purpose: Customer-facing list of purchasable membership plans filtered to active ones for a company.
Logic: Ordered by rank to highlight premium tiers; exposes duration for expiry calculation post-purchase.

6 Table details

6.1 Company table


company [project] | spares [project] | subscription_plan [project] | subscript...ransaction [project] | request_form [projec

Tx: Auto | DDL | CSV

WHERE | ORDER BY

	id	owner_id	name	logo_url	address	city
1	1	1	AutoCare Garage	https://img.icons8.com/color/48/garage.png	123 Sukhumvit Road	Bangkok
2	2	2	SpeedFix Motors	https://img.icons8.com/color/48/f1-car.png	456 Phetchaburi Road	Bangkok
3	3	3	Elite Auto Works	https://img.icons8.com/color/48/expensive-price.png	789 Thong Lor	Bangkok
4	4	1	Grand Opening Test Garage	http://test-logo.com/img.png	999 Testing Blvd	Bangkok
5	5	4	Pattaya Wheels	<null>	404 Wheel Ave	Pattaya
6	6	5	Drift King Garage	http://img.com/drift.png	505 Drift Soi	Bangkok
7	7	6	Phuket Body Shop	<null>	606 Paint Blvd	Phuket
8	8	7	Isan Oil Change	<null>	707 Oil Way	Khon Kaen
9	9	8	Mystery Machine Fix	<null>	808 Mystery Ln	Bangkok
10	10	9	Fast Lane Service	http://img.com/fast.png	909 Fast St	Bangkok
11	11	10	Northern Tires	<null>	1010 Weed St	Chiang Ra

6.1.1 Purpose

Stores all companies (garages) using the platform.

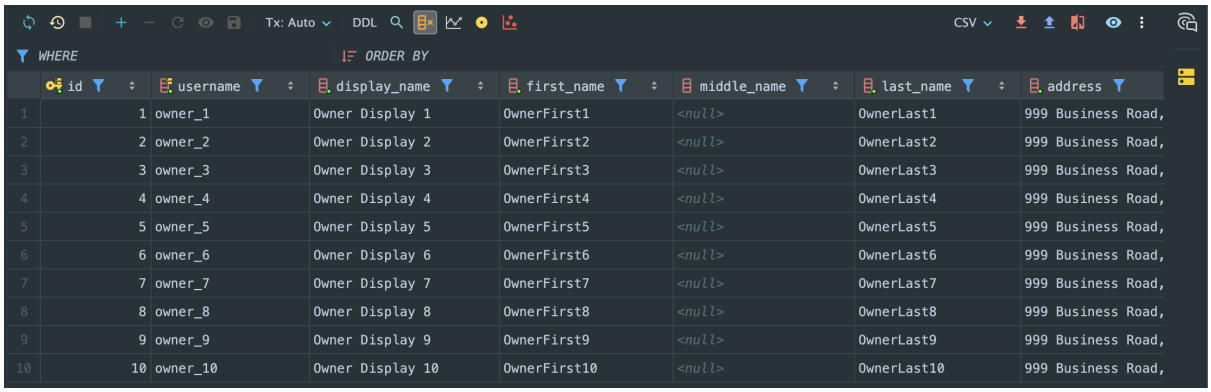
6.1.2 Important Columns

- `id`: Surrogate identifier.
- `owner_id`: Owning account reference.
- `name`: Company/garage name.
- `logo_url`: Company logo URL.
- `address`: Physical address.
- `city`: City location.
- `postal_code`: Postal code.
- `capacity`: Service bay/workload capacity.
- `open_time`: Operating hours.
- `close_time`: Operating hours.
- `social_media_contact`: Public handle/contact.
- `ev_support`: EV servicing capability flag.

6.1.3 Keys & Constraints

- PK: `id`
- FK: `owner_id` → `owner.id`

6.2 Owner table



WHERE ORDER BY

	id	username	display_name	first_name	middle_name	last_name	address
1	1	owner_1	Owner Display 1	OwnerFirst1	<null>	OwnerLast1	999 Business Road,
2	2	owner_2	Owner Display 2	OwnerFirst2	<null>	OwnerLast2	999 Business Road,
3	3	owner_3	Owner Display 3	OwnerFirst3	<null>	OwnerLast3	999 Business Road,
4	4	owner_4	Owner Display 4	OwnerFirst4	<null>	OwnerLast4	999 Business Road,
5	5	owner_5	Owner Display 5	OwnerFirst5	<null>	OwnerLast5	999 Business Road,
6	6	owner_6	Owner Display 6	OwnerFirst6	<null>	OwnerLast6	999 Business Road,
7	7	owner_7	Owner Display 7	OwnerFirst7	<null>	OwnerLast7	999 Business Road,
8	8	owner_8	Owner Display 8	OwnerFirst8	<null>	OwnerLast8	999 Business Road,
9	9	owner_9	Owner Display 9	OwnerFirst9	<null>	OwnerLast9	999 Business Road,
10	10	owner_10	Owner Display 10	OwnerFirst10	<null>	OwnerLast10	999 Business Road,

6.2.1 Purpose

Stores account information for company owners who administer garages.

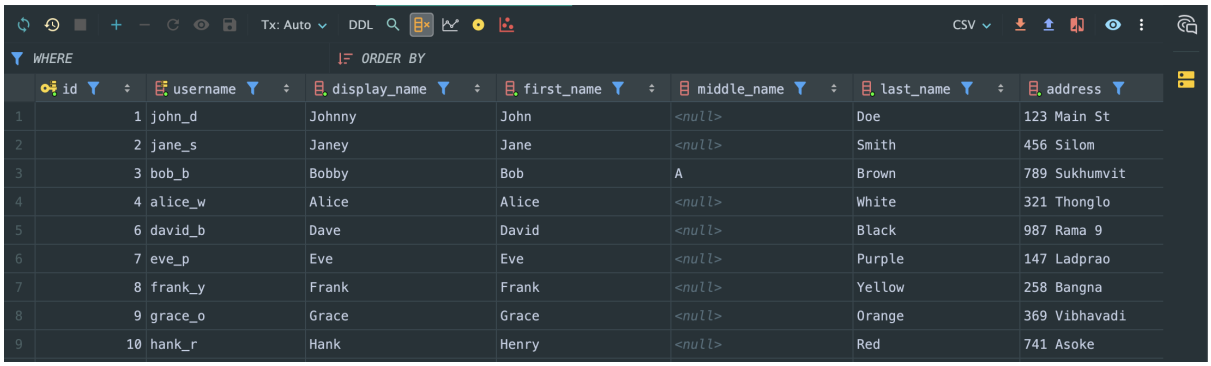
6.2.2 Important Columns

- `id`: Surrogate identifier.
- `username` (unique): Username for login.
- `email` (unique): Email address for contact.
- `display_name`: Name shown publicly.
- `first_name`: Owner's first name.
- `middle_name`: Owner's middle name.
- `last_name`: Owner's last name.
- `address`: Owner's address.
- `city`: Owner's city.
- `postal_code`: Owner's postal code.
- `telephone_no`: Owner's telephone number.
- `password`: Hashed password.

6.2.3 Keys & Constraints

- PK: `id`
- Unique: `username`, `email`

6.3 Customer table



WHERE ORDER BY

	id	username	display_name	first_name	middle_name	last_name	address
1	1	john_d	Johnny	John	<null>	Doe	123 Main St
2	2	jane_s	Janey	Jane	<null>	Smith	456 Silom
3	3	bob_b	Bobby	Bob	A	Brown	789 Sukhumvit
4	4	alice_w	Alice	Alice	<null>	White	321 Thonglo
5	6	david_b	Dave	David	<null>	Black	987 Rama 9
6	7	eve_p	Eve	Eve	<null>	Purple	147 Ladprao
7	8	frank_y	Frank	Frank	<null>	Yellow	258 Bangna
8	9	grace_o	Grace	Grace	<null>	Orange	369 Vibhavadi
9	10	hank_r	Hank	Henry	<null>	Red	741 Asoke

6.3.1 Purpose

End-user accounts (vehicle owners) who book services.

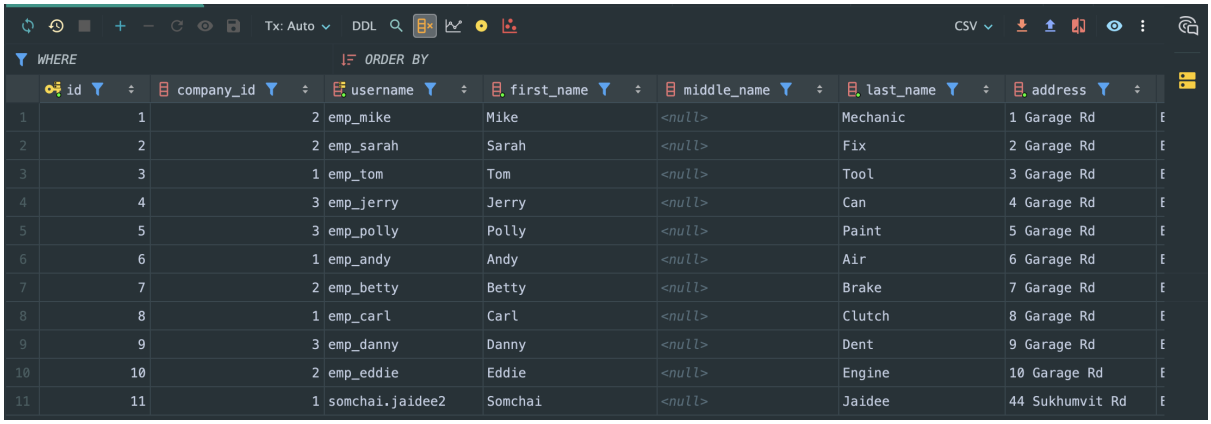
6.3.2 Important Columns

- `id`: Surrogate identifier.
- `username` (unique): Username for login.
- `email` (unique): Email address for contact.
- `display_name`: Name shown publicly.
- `first_name`: Customer's first name.
- `middle_name`: Customer's middle name.
- `last_name`: Customer's last name.
- `address`: Customer's address.
- `city`: Customer's city.
- `postal_code`: Customer's postal code.
- `telephone_no`: Customer's telephone number.
- `password`: Hashed password.

6.3.3 Keys & Constraints

- PK: `id`
- Unique: `username`, `email`

6.4 Employee table



WHERE ORDER BY

	id	company_id	username	first_name	middle_name	last_name	address
1	1	2	emp_mike	Mike	<null>	Mechanic	1 Garage Rd
2	2	2	emp_sarah	Sarah	<null>	Fix	2 Garage Rd
3	3	1	emp_tom	Tom	<null>	Tool	3 Garage Rd
4	4	3	emp_jerry	Jerry	<null>	Can	4 Garage Rd
5	5	3	emp_polly	Polly	<null>	Paint	5 Garage Rd
6	6	1	emp_andy	Andy	<null>	Air	6 Garage Rd
7	7	2	emp_betty	Betty	<null>	Brake	7 Garage Rd
8	8	1	emp_carl	Carl	<null>	Clutch	8 Garage Rd
9	9	3	emp_danny	Danny	<null>	Dent	9 Garage Rd
10	10	2	emp_eddie	Eddie	<null>	Engine	10 Garage Rd
11	11	1	somchai.jaidee2	Somchai	<null>	Jaidee	44 Sukhumvit Rd

6.4.1 Purpose

Company staff records with identity and employment details.

6.4.2 Important Columns

- `id`: Surrogate identifier.
- `company_id`: Employer company reference.
- `username` (unique): Username for login.
- `email` (unique): Email address for contact.
- `first_name`: Employee's first name.
- `middle_name`: Employee's middle name.
- `last_name`: Employee's last name.
- `address`: Employee's address.
- `city`: Employee's city.
- `postal_code`: Employee's postal code.
- `telephone_no`: Employee's telephone number.
- `specialize`: Employee's specialization.
- `password`: Hashed password.
- `salary` (default 15000.00): Monthly salary.
- `hire_date` (default CURRENT_DATE): Employment start date.
- `contract_type` (default 'Full-Time'): Employment contract type.
- `contract_period`: Duration of the contract.

6.4.3 Keys & Constraints

- PK: `id`
- FK: `company_id` → `company.id`
- Unique: `username`, `email`

6.5 Employee Schedule table

WHERE ORDER BY

	schedule_id	employee_id	date	start_time	end_time	status
1	1	1	2025-11-27	09:00:00	18:00:00	Working
2	2	1	2025-11-28	09:00:00	18:00:00	Working
3	3	1	2025-11-29	09:00:00	18:00:00	Working
4	4	1	2025-11-30	09:00:00	18:00:00	Working
5	5	1	2025-12-01	09:00:00	18:00:00	Working
6	6	1	2025-12-02	09:00:00	18:00:00	Working
7	7	1	2025-12-03	09:00:00	18:00:00	Working
8	8	2	2025-11-27	09:00:00	18:00:00	Working
9	9	2	2025-11-28	09:00:00	18:00:00	Working
10	10	2	2025-11-29	09:00:00	18:00:00	Working

6.5.1 Purpose

Work schedules and shift assignments for employees.

6.5.2 Important Columns

- `schedule_id`: Surrogate identifier.
- `employee_id`: Employee reference.
- `date`: Scheduled date.
- `start_time`: Shift start time.
- `end_time`: Shift end time.
- `status`: Schedule status (e.g., 'scheduled', 'completed', 'canceled').

6.5.3 Keys & Constraints

- PK: `schedule_id`
- FK: `employee_id` → `employee.id`

6.6 Equipment table

	id	company_id	name	serial_number	purchase_cost	last_service_date
1	1	1	Two-Post Hydraulic Lift	LFT-2023-001	120000.00	2024-01-15
2	2	1	Industrial Air Compressor	AIR-COMP-55	35000.00	2024-06-01
3	3	1	OBD-II Advanced Scanner	OBD-SCAN-99	15000.00	2023-11-20
4	4	2	3D Wheel Alignment System	WHEEL-3D-X	250000.00	2024-03-10
5	5	2	Automatic Tire Changer	TIRE-CH-200	85000.00	2024-05-20
6	6	2	Digital Wheel Balancer	BAL-DIGI-50	60000.00	2024-02-15
7	7	3	Dust-Free Paint Booth	PNT-BOOTH-01	500000.00	2024-01-01
8	8	3	Dual Action Polisher	POL-DA-PRO	12000.00	2024-07-01
9	9	3	ECU Remapping Tool	ECU-MAP-V3	45000.00	2024-04-10
10	10	1	Hydraulic Lift	LFT-001	50000.00	2023-01-15

6.6.1 Purpose

Durable equipment/assets tracked per company.

6.6.2 Important Columns

- `id`: Surrogate identifier.
- `company_id`: Company reference.
- `name`: Equipment name.
- `serial_number`: Unique serial number.
- `purchase_cost`: Cost at purchase.
- `last_service_date`: Date of last service.
- `category`: Equipment category.
- `quantity` (default 1): Number of units.
- `purchased_date` (default CURRENT_DATE): Date purchased.
- `next_service_date`: Scheduled next service.
- `operational_status` (default 'active'): Current status.

6.6.3 Keys & Constraints

- PK: `id`
- FK: `company_id` → `company.id` (ON DELETE CASCADE)

6.7 Vehicle table

	id	owner_id	brand	model	year	license_plate	vin	mileage
1	1	1	Toyota	Camry	2020	1ABC1234	VIN1234567890001	50
2	2	2	Honda	Civic	2019	2DEF5678	VIN1234567890002	60
3	3	3	Mazda	3	2021	3GHI9012	VIN1234567890003	30
4	4	4	Nissan	Almera	2018	4JKL3456	VIN1234567890004	80
5	5	5	Ford	Ranger	2022	5MNO7890	VIN1234567890005	20
6	6	6	Isuzu	D-Max	2020	6PQR1234	VIN1234567890006	55
7	7	7	BMW	320d	2021	7STU5678	VIN1234567890007	15
8	8	8	Benz	C200	2019	8VWX9012	VIN1234567890008	40
9	9	9	Suzuki	Swift	2017	9YZA3456	VIN1234567890009	90
10	10	10	MG	ZS	2022	1BCD7890	VIN1234567890010	10

6.7.1 Purpose

Vehicles registered by customers; used for appointments and services.

6.7.2 Important Columns

- `id`: Surrogate identifier.
- `owner_id`: Vehicle owner reference.
- `brand`: Vehicle brand/make.
- `model`: Vehicle model.
- `year`: Manufacturing year.
- `mileage`: Current mileage.
- `license_plate`: Unique license plate number.
- `vin`: Unique vehicle identification number.

6.7.3 Keys & Constraints

- PK: `id`
- FK: `owner_id` → `customer.id` (ON DELETE SET NULL)
- Unique: `license_plate`, `vin`

6.8 Service table

	id	company_id	name	detail	approximate_duration	min_cost	max_cost
1	1	1	Oil Change	Synthetic Oil	45	1200.00	18(
2	2	1	Brake Check	Check Pads	60	500.00	8(
3	3	2	Wheel Alignment	Laser Align	60	800.00	15(
4	4	3	Ceramic Coating	3 Layers	360	8000.00	150(
5	5	1	Standard Oil C…	Replace oil and f…	45	800.00	15(
6	6	1	Brake Inspectior	Check pads and ro…	30	300.00	5(
7	7	2	EV Battery Hea…	Scan battery cells	60	1500.00	20(
8	8	3	Tire Rotation	Rotate 4 wheels	40	400.00	6(
9	9	4	Full Body Polish	Wash, clay bar, p…	240	2500.00	40(
10	10	5	ECU Remap	Stage 1 Tuning	120	9000.00	150(

6.8.1 Purpose

Service catalog entries offered by a company.

6.8.2 Important Columns

- `id`: Surrogate identifier.
- `company_id`: Offering company reference.
- `name`: Service name.
- `detail`: Service description.
- `approximate_duration`: Estimated time to complete.
- `min_cost` (numeric(10,2)): Minimum service cost.
- `max_cost` (numeric(15,2)): Maximum service cost.

6.8.3 Keys & Constraints

- PK: `id`
- FK: `company_id` → `company.id` (ON DELETE SET NULL)

6.9 appointment

	id	vehicle_id	details	appointment_date	status	company_id
1	4	7	Regular Service Checkup #4	2025-11-02 06:39:11.441300	accepted	
2	5	3	Regular Service Checkup #5	2025-11-09 16:56:13.691379	accepted	
3	6	4	Regular Service Checkup #6	2025-11-21 12:02:14.575321	accepted	
4	7	7	Regular Service Checkup #7	2025-11-07 17:43:37.007861	accepted	
5	8	6	Regular Service Checkup #8	2025-11-09 17:29:04.575759	accepted	
6	9	6	Regular Service Checkup #9	2025-11-21 09:15:23.366415	accepted	
7	10	7	Regular Service Checkup #10	2025-11-08 20:33:36.411441	accepted	
8	12	1	Regular Service Checkup #12	2025-11-25 01:44:17.334287	accepted	
9	13	5	Regular Service Checkup #13	2025-11-07 15:22:44.131702	accepted	
10	14	1	Regular Service Checkup #14	2025-11-21 00:24:52.743512	accepted	

6.9.1 Purpose

Booked service appointments for specific vehicles and companies.

6.9.2 Important Columns

- `id`: Surrogate identifier.
- `vehicle_id`: Vehicle reference.
- `company_id`: Company reference.
- `details`: Appointment details/notes.
- `appointment_date` (timestamp): Scheduled date and time.
- `status` (default 'pending'): Appointment status.

6.9.3 Keys & Constraints

- PK: `id`
- FK: `vehicle_id` → `vehicle.id`
- FK: `company_id` → `company.id`

6.10 Service Record table

	service_id	appointment_id	lead_mechanic_id	type	cost	duration	received_da
1	2	1	8	Standard Repair	500	60	2025-11-18
2	1	2	6	Standard Repair	1200	45	2025-11-21
3	3	3	1	Standard Repair	800	60	2025-11-01
4	1	4	6	Standard Repair	1200	45	2025-11-02
5	1	5	3	Standard Repair	1200	45	2025-11-09
6	3	6	7	Standard Repair	800	60	2025-11-21
7	2	7	3	Standard Repair	500	60	2025-11-07
8	3	8	7	Standard Repair	800	60	2025-11-09
9	3	9	1	Standard Repair	800	60	2025-11-21
10	3	10	10	Standard Repair	800	60	2025-11-08

6.10.1 Purpose

Historical records of performed services.

6.10.2 Important Columns

- `service_id`: Service reference.
- `appointment_id`: Appointment reference.
- `lead_mechanic_id`: Responsible mechanic reference.
- `type`: Service type/category.
- `cost`: Final service cost.
- `duration`: Actual time taken.
- `received_date`: Date service was received.
- `expected_finish_date`: Projected completion date.
- `status` (default 'in progress'): Current service status.

6.10.3 Keys & Constraints

- PK: composite (`service_id`, `appointment_id`)
- FK: `service_id` → `service.id`
- FK: `appointment_id` → `appointment.id`
- FK: `lead_mechanic_id` → `employee.id`

6.11 Spares table

	id	company_id	part_number	name	description	purchase_cost	retail_
1	2	1	FIL-001	Oil Filter	Standard	100.00	
2	3	2	TIRE-MIC	Michelin Pilot	245/45/R18	4000.00	
3	1	1	OIL-001	Synthetic Oil 5W-30	4L Bottle	800.00	
4	4	1	FIL-OIL-001	Oil Filter	Standard Oil Filter	100.00	
5	5	1	OIL-SYN-5W30	Synthetic Oil	5W30 Full Synthetic 4L	800.00	
6	6	2	BAT-12V-45	Car Battery	12V 45Ah Maintenance F…	1500.00	
7	7	2	EV-CHG-CBL	Type 2 Cable	EV Charging Cable	3000.00	
8	8	3	TIR-265-65	Offroad Tire	265/65 R17 All Terrain	3500.00	
9	9	4	BRK-PAD-F	Brake Pads Front	Ceramic Brake Pads	800.00	
10	10	5	TURBO-KIT	Turbocharger	Stage 1 Turbo Kit	15000.00	

6.11.1 Purpose

Spare parts inventory and stock details per company.

6.11.2 Important Columns

- `id`: Surrogate identifier.
- `company_id`: Owning company reference.
- `part_number`: Unique part number.
- `name`: Spare part name.
- `description`: Spare part description.
- `purchase_cost`: Cost at purchase.
- `retail_price`: Selling price.
- `quantity` (default 1): Stock quantity.
- `expired_date`: Expiration date.
- `reorder_level`: Minimum stock before reorder.
- `location`: Storage location.
- `purchased_date` (default CURRENT_DATE): Date purchased.

6.11.3 Keys & Constraints

- PK: `id`
- FK: `company_id` → `company.id` (ON DELETE CASCADE)

6.12 Membership Plan table

	membership_plan_id	company_id	name	detail	rank	price	duration
1	1	1	Bronze Care	5% Labor Discount	1	499.00	
2	2	1	Gold Premium	Priority Queue + 10% Discount	2	2500.00	
3	3	2	Street Racer	Discount on Tires & Oil	1	1200.00	
4	4	2	Track Pro	Free Alignment + 15% Parts	2	3500.00	
5	5	3	Elite Access	Valet Service & Free Coffee	1	2000.00	
6	6	3	Black Card	All Access + Concierge	2	50000.00	
7	27	1	Bronze Care	5% Labor Discount	1	500.00	
8	28	1	Silver Care	10% Labor Disc	2	1500.00	
9	29	2	Street Racer	Tire Discount	1	800.00	
10	30	2	Track Pro	Alignment Free	2	2500.00	

6.12.1 Purpose

Tiered membership plan definitions for companies.

6.12.2 Important Columns

- `membership_plan_id`: Surrogate identifier.
- `company_id`: Owning company reference.
- `name`: Plan name.
- `detail`: Plan description.
- `rank`: Plan tier/rank.
- `price`: Plan price.
- `duration`: Validity duration in days.
- `is_active`: Active status flag.

6.12.3 Keys & Constraints

- PK: `membership_plan_id`
- FK: `company_id` → `company.id` (ON DELETE CASCADE)

6.13 Membership table

	customer_id	membership_plan_id	purchased_date	expired_date
1	1	4	2025-11-29	2026-11-29
2	3	3	2025-11-29	2026-05-28
3	5	5	2025-11-29	2025-12-29
4	6	3	2025-11-29	2026-05-28
5	7	2	2025-11-29	2026-11-29
6	8	3	2025-11-29	2026-05-28
7	9	5	2025-11-29	2025-12-29
8	10	1	2025-11-29	2025-12-29
9	12	5	2025-11-29	2025-12-29
10	13	5	2025-11-29	2025-12-29

6.13.1 Purpose

Active membership instances linking customers to plans.

6.13.2 Important Columns

- `customer_id`: Customer reference.
- `membership_plan_id`: Membership plan reference.
- `purchased_date` (default CURRENT_DATE): Date of purchase.
- `expired_date`: Expiration date

6.13.3 Keys & Constraints

- PK: composite (`customer_id`, `membership_plan_id`)
- FK: `customer_id` → `customer.id`
- FK: `membership_plan_id` → `membership_plan.membership_plan_id`

6.14 Membership Transaction table

	id	customer_id	amount	currency	payment_method	reference_id	description
1	1	1	500.00	THB	Cash	REC-001	Monthly Fee Payme
2	3	3	2500.00	THB	Cash	REC-001	Monthly Fee Payme
3	4	1	3500.00	THB	Credit Card	REF-TEST-1	New Member Signup
4	5	3	1200.00	THB	Credit Card	REF-TEST-3	New Member Signup
5	6	5	2000.00	THB	Credit Card	REF-TEST-5	New Member Signup
6	7	6	1200.00	THB	Credit Card	REF-TEST-6	New Member Signup
7	8	7	2500.00	THB	Credit Card	REF-TEST-7	New Member Signup
8	9	8	1200.00	THB	Credit Card	REF-TEST-8	New Member Signup
9	10	9	2000.00	THB	Credit Card	REF-TEST-9	New Member Signup
10	11	10	499.00	THB	Credit Card	REF-TEST-10	New Member Signup

6.14.1 Purpose

Financial entries for customer membership purchases and renewals.

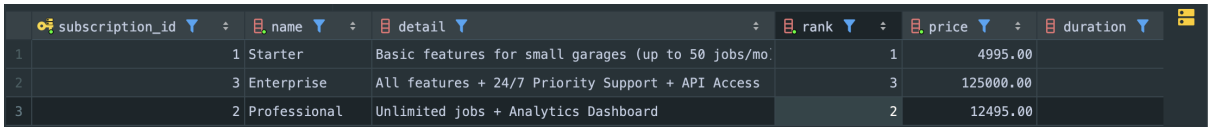
6.14.2 Important Columns

- `id`: Surrogate identifier.
- `customer_id`: Customer reference.
- `amount`: Transaction amount.
- `currency` (default 'THB'): Transaction currency.
- `payment_method`: Payment method used.
- `reference_id`: External payment reference.
- `description`: Transaction description.
- `purchased_date` (default): Date of purchase.
- `expired_date`: Membership expiration date.

6.14.3 Keys & Constraints

- PK: `id`
- FK: `customer_id` → `customer.id` (ON DELETE CASCADE)

6.15 Subscription Plan table



	subscription_id	name	detail	rank	price	duration
1	1	Starter	Basic features for small garages (up to 50 jobs/mo	1	4995.00	
2	3	Enterprise	All features + 24/7 Priority Support + API Access	3	125000.00	
3	2	Professional	Unlimited jobs + Analytics Dashboard	2	12495.00	

6.15.1 Purpose

Platform-level SaaS subscription plans for companies.

6.15.2 Important Columns

- `subscription_id`: Surrogate identifier.
- `name`: Plan name.
- `detail`: Plan description.
- `rank`: Plan tier/rank.
- `price`: Plan price.
- `duration`: Validity duration in days (default 30 days).

- `subscription_id`, `name`, `detail`, `rank`, `price`, `duration` (default 30 days).

6.15.3 Keys & Constraints

- PK: `subscription_id`

6.16 Subscription Transaction table

	transaction_id	subscription_id	owner_id	amount	currency	payment_method	re
1	1	3	1	25000.00	THB	Bank Transfer	REF-A
2	2	2	2	2499.00	THB	PromptPay	REF-A
3	3	3	3	25000.00	THB	Credit Card	REF-A
4	4	2	4	2499.00	THB	Credit Card	REF-A
5	5	1	5	999.00	THB	PromptPay	REF-A
6	6	1	6	999.00	THB	PromptPay	REF-A
7	7	2	7	2499.00	THB	Bank Transfer	REF-A
8	8	1	8	999.00	THB	PromptPay	REF-A
9	9	3	9	25000.00	THB	Credit Card	REF-A
10	10	1	10	999.00	THB	Bank Transfer	REF-A

6.16.1 Purpose

Payments for company subscriptions to platform plans.

6.16.2 Important Columns

- `transaction_id`: Surrogate identifier.
- `subscription_id`: Subscription plan reference.
- `owner_id`: Paying owner reference.
- `amount`: Transaction amount.
- `currency` (default 'THB'): Transaction currency.
- `payment_method`: Payment method used.
- `reference_id`: External payment reference.
- `description`: Transaction description.
- `purchased_date` (default): Date of purchase.
- `expired_date`: Subscription expiration date.

6.16.3 Keys & Constraints

- PK: `transaction_id`
- FK: `subscription_id` → `subscription_plan.subscription_id`
- FK: `owner_id` → `owner.id`

6.17 Transaction table

WHERE ORDER BY

	transaction_id	appointment_id	service_id	amount	currency	payment_method	r
1	1	1	1	1038.31	THB	Credit Card	TXN-
2	2	2	1	1060.83	THB	Credit Card	TXN-
3	3	3	3	2030.51	THB	Credit Card	TXN-
4	4	4	1	1348.34	THB	Credit Card	TXN-
5	5	5	1	837.74	THB	Credit Card	TXN-
6	6	6	3	1586.65	THB	Credit Card	TXN-
7	7	7	1	1298.84	THB	Credit Card	TXN-
8	8	8	3	1539.98	THB	Credit Card	TXN-
9	9	9	3	2419.22	THB	Credit Card	TXN-
10	10	10	3	798.91	THB	Credit Card	TXN-

6.17.1 Purpose

General financial transactions for services & appointments.

6.17.2 Important Columns

- `transaction_id`: Surrogate identifier.
- `appointment_id`: Related appointment reference.
- `service_id`: Related service reference.
- `amount`: Transaction amount.
- `currency` (default 'THB'): Transaction currency.
- `payment_method`: Payment method used.
- `reference_id`: External payment reference.
- `description`: Transaction description.
- `created_at` (default CURRENT_TIMESTAMP): Timestamp of transaction creation.

6.17.3 Keys & Constraints

- PK: `transaction_id`
- FK: `appointment_id` → `appointment.id` (ON DELETE SET NULL)
- FK: `service_id` → `service.id` (ON DELETE SET NULL)

6.18 Request Form table

	request_id	company_id	first_name	middle_name	last_name	address	city
1	3	1	David	Alan	Wilson	12 Silom Soi 4	Bangkok
2	4	1	Somsak	<null>	Rakthai	88 Ladprao Rd	Bangkok
3	5	1	Mary	<null>	Johnson	55 Sathorn Tai	Bangkok
4	6	2	Kenji	<null>	Tanaka	99 Thong Lor	Bangkok
5	7	2	Wichai	<null>	Srisuk	101 Phetchaburi Rd	Bangkok
6	8	2	Sarah	Jane	Connor	23 Asoke Montri	Bangkok
7	9	3	Nopadol	<null>	Kaewmanee	77 Rama 9	Bangkok
8	10	3	James	<null>	Bond	007 Wireless Rd	Bangkok
9	11	3	Ploy	<null>	Pailin	456 Ratchada	Bangkok

6.18.1 Purpose

Incoming requests/applications associated with a company.

6.18.2 Important Columns

- `request_id`: Surrogate identifier.
- `company_id`: Company reference.
- `first_name`: Requester's first name.
- `middle_name`: Requester's middle name.
- `last_name`: Requester's last name.
- `address`: Requester's address.
- `city`: Requester's city.
- `postal_code`: Requester's postal code.
- `email`: Requester's email address.
- `telephone_no`: Requester's telephone number.
- `specialize`: Requester's specialization or area of interest.

6.18.3 Keys & Constraints

- PK: `request_id`
- FK: `company_id` → `company.id` (ON DELETE CASCADE)

7 Database Design Quality, Security, and efficiency

7.1 Normalization Design

The schema follows proper normalization to reduce redundancy and anomalies while keeping queries practical.

- **1NF:** All columns store atomic values (e.g., single email/telephone per user row; no repeating groups).
- **2NF:** Non-key attributes depend on the whole key in composite PK tables (e.g., `service_record` depends on both `service_id` and `appointment_id`).
- **3NF:** Non-key attributes do not transitively depend on keys (e.g., company city/postal code are attributes of the company, not derived from other non-keys).
- **Reference Integrity:** Foreign keys enforce valid relationships (e.g., vehicle `owner_id` → customer, membership → plan/customer).
- **Controlled Denormalization (where justified):** Read-heavy listing views include joined names and labels at query-time rather than stored duplicates, preserving normalization in base tables.

7.2 Security Mindset & Implementation

Security is treated as a first-class design constraint across data, access, and operations.

- **Least Privilege:** Separate roles for application, admin, and CI with minimal permissions; no superuser credentials in app runtime.
- **Password Storage:** Use `crypt` (bcrypt) with per-user salt; never store plaintext or reversible encryption.
- **Authentication:** Dedicated functions for login that return minimal user info on success; avoid exposing sensitive fields.
- **Input Validation:** Constrain data via types, CHECKs, and FK references; reject invalid states early.
- **Auditability:** Transaction and membership history retained to support investigations and dispute resolution.

7.3 Dummy Database

Improvement proposal: introduce a dedicated dummy (non-production) database with least-privilege roles so day-to-day workers and app processes cannot perform any DDL (DROP/ALTER/CREATE). This avoids accidental destructive changes while enabling safe development/testing using synthetic data. The dummy environment mirrors the schema but contains only synthetic or anonymized records.

- **Purpose:** Enable safe iteration, demos, and automated tests without risking real user or financial data.
- **Data Strategy:** Populate with generated data that respects constraints (FKs, uniques) and realistic distributions; never reuse production credentials or PII.
- **Isolation:** Host on a separate instance/DB name with distinct users and roles; restrict network access to development IPs only.
- **DDL Restrictions:** Revoke CREATE/ALTER/DROP on schemas/tables for worker roles; grant only necessary DML (SELECT/INSERT/UPDATE/DELETE). This ensures

workers cannot drop any tables.

- **Role Design:** Use separate roles, e.g., read-only (`app_ro`), read-write (`app_rw`) without DDL, and admin (`app_admin`) restricted to maintainers.

Example (PostgreSQL least-privilege setup)

```
-- Lock down default schema DDL for everyone
REVOKE CREATE ON SCHEMA public FROM PUBLIC;

-- Roles without DDL
CREATE ROLE app_ro  NOINHERIT NOCREATEDB NOCREATEROLE NOSUPERUSER;
CREATE ROLE app_rw  NOINHERIT NOCREATEDB NOCREATEROLE NOSUPERUSER;

-- Optional admin role for maintainers only
CREATE ROLE app_admin NOINHERIT NOCREATEDB NOCREATEROLE NOSUPERUSER;

-- Grant minimal privileges
GRANT USAGE ON SCHEMA public TO app_ro, app_rw;
GRANT SELECT ON ALL TABLES IN SCHEMA public TO app_ro;
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO
    app_rw;

-- Ensure future tables are also protected
ALTER DEFAULT PRIVILEGES IN SCHEMA public
    GRANT SELECT ON TABLES TO app_ro;
ALTER DEFAULT PRIVILEGES IN SCHEMA public
    GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO app_rw;

-- Do NOT grant CREATE/ALTER/DROP to app_ro/app_rw (prevents dropping
    tables)
```

7.4 Query Efficiency Implementation

Queries are structured for clarity and performance with appropriate indexing and access patterns.

- **Primary/Foreign Keys:** Implicit btree indexes on PKs and recommended indexes on high-selectivity FKs (e.g., `appointment.company_id`, `membership.membership_plan_id`).
- **Lookup Indexes:** Create btree indexes on frequent filters and joins (e.g., `vehicle.owner_id`, `employee.company_id`, `service.company_id`).
- **Search Fields:** Index unique identifiers used for authentication and CRUD (`username`, `email`, `vin`, `license_plate`).
- **Time-Ordered Access:** For reporting, order by date/time with supporting indexes (e.g., `financial_record.record_date`, `membership_transaction.purchased_date`).
- **Selective Projections:** Views/functions project only needed columns to reduce I/O and transfer.

8 Financial Considerations

The total operational cost for deploying this comprehensive Garage Management Platform is projected at approximately 4,999 THB per month.

This all-inclusive subscription fee grants full access to the platform's suite of enterprise modules, including Customer Relationship Management (CRM), Inventory and Stock Tracking, and Financial Reporting.

Beyond the software licensing, this cost covers all essential infrastructure requirements, such as secure cloud database hosting, daily data backups, and server maintenance.

9 Database access

- Hostname: dpg-d4hvij0gjchc73di0oqg-a
- Port: 5432
- Database: sc_db_project
- Username: sc_db_project_user
- Password: jOtTexUwkUPGFP3E3rvevTmixbtYvliZ
- Internal Database URL:

  postgresql://sc_db_project_user:jOtTexUwkUPGFP3E3rvevTmixbtYvliZ@dpg-d4hvij0gjchc73di0oqg-a/sc_db_project
- External Database URL:

  postgresql://sc_db_project_user:jOtTexUwkUPGFP3E3rvevTmixbtYvliZ@dpg-d4hvij0gjchc73di0oqg-a.singapore-postgres.render.com/sc_db_project
- PSQL Command:

  PGPASSWORD=jOtTexUwkUPGFP3E3rvevTmixbtYvliZ psql -h dpg-d4hvij0gjchc73di0oqg-a.singapore-postgres.render.com -U sc_db_project_user sc_db_project

10 Github Repository

You can access the repository at the following URL: https://github.com/Tawancs/cs-db-project