# Permission System Migration Plan

## Overview

This document outlines the database schema and type definitions for the new permission system. The system uses a SQL Server database and will be accessed via C# APIs. The TypeScript types shown here represent the expected shape of the API responses.

## Database Connection

- \*\*Server\*\*: SQLDEVCFSS,1433

- \*\*Database\*\*: CBP\_Permissions\_Test

- \*\*Authentication\*\*: Windows Authentication

- \*\*Connection String\*\*: `Server=SQLDEVCFSS,1433;Database=CBP\_Permissions\_Test;Trusted\_Connection=True;`

## Core Concepts

- Permissions are represented directly by Roles

- Groups belong to specific Customers

- Users can belong to multiple Groups

- Groups can have multiple Roles

- All IDs are BIGINT in SQL and number in TypeScript

## Database Schema and Types

### Users and Authentication

```sql

-- SQL Schema

CREATE TABLE Users (

id BIGINT NOT NULL PRIMARY KEY,

tenant\_id INT NOT NULL,

is\_active BIT NOT NULL,

creation\_date DATETIME NOT NULL,

last\_login DATETIME NULL,

external\_id NVARCHAR(50) NULL,

customer\_id BIGINT NOT NULL,

first\_name NVARCHAR(50) NULL,

last\_name NVARCHAR(50) NULL,

department NVARCHAR(50) NULL,

is\_locked BIT NOT NULL,

password NVARCHAR(128) NULL,

CONSTRAINT FK\_\_Users\_\_customer\_\_\_52593CB8 FOREIGN KEY (customer\_id)

REFERENCES Customers(id)

);

CREATE INDEX IX\_Users\_CustomerId ON Users(customer\_id);

CREATE INDEX IX\_Users\_ExternalId ON Users(external\_id) WHERE external\_id IS NOT NULL;

```

```typescript

// Expected API Response Type

interface User {

id: number;

tenantId: number;

isActive: boolean;

creationDate: string; // ISO 8601 format

lastLogin?: string; // ISO 8601 format

externalId?: string;

customerId: number;

firstName?: string;

lastName?: string;

department?: string;

isLocked: boolean;

password?: string; // Only used for creation/updates, never returned in responses

}

```

### Customer Management

```sql

CREATE TABLE Customers (

id BIGINT NOT NULL PRIMARY KEY,

external\_id NVARCHAR(100) NULL,

name NVARCHAR(100) NOT NULL,

tenant\_id INT NOT NULL,

is\_active BIT NOT NULL,

created\_on DATETIME NOT NULL,

updated\_on DATETIME NULL,

contact\_name NVARCHAR(100) NULL,

contact\_email NVARCHAR(100) NULL,

contact\_phone NVARCHAR(20) NULL,

type NVARCHAR(50) NOT NULL,

status NVARCHAR(50) NOT NULL,

environment NVARCHAR(50) NOT NULL,

domain NVARCHAR(100) NULL,

sponsor\_id BIGINT NULL,

routing\_id NVARCHAR(50) NULL,

require\_2fa BIT NOT NULL DEFAULT 0,

logo\_url NVARCHAR(2048) NULL,

CONSTRAINT FK\_Customers\_Sponsor FOREIGN KEY (sponsor\_id)

REFERENCES Customers(id)

);

CREATE INDEX IX\_Customers\_ExternalId ON Customers(external\_id) WHERE external\_id IS NOT NULL;

CREATE INDEX IX\_Customers\_TenantId ON Customers(tenant\_id);

```

```typescript

interface Customer {

id: number;

externalId?: string;

name: string;

tenantId: number;

isActive: boolean;

createdOn: string; // ISO 8601 format

updatedOn?: string; // ISO 8601 format

type: string; // Enum in TypeScript

status: string; // Enum in TypeScript

environment: string; // Enum in TypeScript

domain?: string;

contactName?: string;

contactEmail?: string;

contactPhone?: string;

sponsorId?: string;

routingId?: string;

require2fa: boolean;

logoUrl?: string;

}

```

### Permission Management

```sql

-- Core Permission Tables

CREATE TABLE Groups (

id BIGINT NOT NULL PRIMARY KEY,

name NVARCHAR(100) NULL,

customer\_id BIGINT NOT NULL,

created\_at DATETIME NOT NULL,

updated\_at DATETIME NOT NULL,

CONSTRAINT FK\_\_Groups\_\_customer\_\_4E88ABD4 FOREIGN KEY (customer\_id)

REFERENCES Customers(id)

);

CREATE INDEX IX\_Groups\_CustomerId ON Groups(customer\_id);

CREATE TABLE Roles (

id BIGINT NOT NULL PRIMARY KEY,

name NVARCHAR(100) NOT NULL

);

CREATE UNIQUE INDEX IX\_Roles\_Name ON Roles(name);

-- Mapping Tables

CREATE TABLE GroupRoles (

group\_id BIGINT NOT NULL,

role\_id BIGINT NOT NULL,

PRIMARY KEY (group\_id, role\_id),

CONSTRAINT FK\_\_GroupRole\_\_group\_\_5BE2A6F2 FOREIGN KEY (group\_id)

REFERENCES Groups(id)

ON DELETE CASCADE,

CONSTRAINT FK\_\_GroupRole\_\_role\_\_\_5CD6CB2B FOREIGN KEY (role\_id)

REFERENCES Roles(id)

ON DELETE CASCADE

);

CREATE TABLE UserGroups (

user\_id BIGINT NOT NULL,

group\_id BIGINT NOT NULL,

PRIMARY KEY (user\_id, group\_id),

CONSTRAINT FK\_\_UserGroup\_\_user\_\_\_5535A963 FOREIGN KEY (user\_id)

REFERENCES Users(id)

ON DELETE CASCADE,

CONSTRAINT FK\_\_UserGroup\_\_group\_\_5629CD9C FOREIGN KEY (group\_id)

REFERENCES Groups(id)

ON DELETE CASCADE

);

```

```typescript

interface Group {

id: number;

name?: string;

customerId: number;

createdAt: string; // ISO 8601 format

updatedAt: string; // ISO 8601 format

}

interface Role {

id: number;

name: string;

}

interface GroupRole {

groupId: number;

roleId: number;

}

interface UserGroup {

userId: number;

groupId: number;

}

```

## Required API Endpoints

### Customer Management

```

GET /api/v1/customers # List customers with pagination and filtering

GET /api/v1/customers/{id} # Get customer details

POST /api/v1/customers # Create customer

PUT /api/v1/customers/{id} # Update customer

DELETE /api/v1/customers/{id} # Delete customer (should check for users first)

```

### User Management

```

GET /api/v1/users # List users with pagination and filtering

GET /api/v1/users/{id} # Get user details

POST /api/v1/users # Create user

PUT /api/v1/users/{id} # Update user

DELETE /api/v1/users/{id} # Delete user

GET /api/v1/users/{id}/groups # Get user's groups

```

### Group Management

```

GET /api/v1/groups # List groups with pagination and filtering

GET /api/v1/groups/{id} # Get group details

POST /api/v1/groups # Create group

PUT /api/v1/groups/{id} # Update group

DELETE /api/v1/groups/{id} # Delete group

GET /api/v1/groups/{id}/users # Get group's users

GET /api/v1/groups/{id}/roles # Get group's roles

```

### Role Management

```

GET /api/v1/roles # List all roles

GET /api/v1/roles/{id} # Get role details

POST /api/v1/roles # Create role

PUT /api/v1/roles/{id} # Update role

DELETE /api/v1/roles/{id} # Delete role (should check for usage first)

```

### Group-Role Assignment

```

POST /api/v1/groups/{id}/roles # Assign roles to group

DELETE /api/v1/groups/{id}/roles # Remove roles from group

```

### User-Group Assignment

```

POST /api/v1/users/{id}/groups # Assign user to groups

DELETE /api/v1/users/{id}/groups # Remove user from groups

```

### Common Query Parameters

- \*\*Pagination\*\*: `page`, `limit`

- \*\*Filtering\*\*:

- Users: `customerId`, `isActive`, `isLocked`

- Groups: `customerId`

- Customers: `tenantId`, `isActive`, `type`, `status`, `environment`

- \*\*Search\*\*: `q` (searches name fields)

### Common Response Format

```typescript

interface PaginatedResponse<T> {

items: T[];

total: number;

page: number;

limit: number;

}

interface ErrorResponse {

error: string;

details?: string;

code: string;

}

```

### Authentication Requirements

- All endpoints require authentication

- APIs should validate that users can only access resources within their customer scope

- Customer-specific endpoints should validate customer access

- Role management endpoints should be restricted to system administrators

### Validation Rules

1. \*\*Users\*\*

- Cannot create users without a valid customer

- External ID must be unique if provided

- Cannot modify tenant\_id after creation

2. \*\*Customers\*\*

- Name must be unique within a tenant

- Cannot modify tenant\_id after creation

- Type, status, and environment must be valid enum values

3. \*\*Groups\*\*

- Name must be unique within a customer

- Must belong to a valid customer

- Cannot change customer\_id after creation

4. \*\*Roles\*\*

- Name must be unique across system

- Cannot delete roles that are in use

## Key Requirements for C# API

1. \*\*Date Handling\*\*

- All dates should be returned in ISO 8601 format

- UTC should be used for all datetime operations

2. \*\*ID Types\*\*

- All IDs are BIGINT in SQL

- Should be serialized as numbers in JSON

- No string IDs

3. \*\*Nullable Fields\*\*

- Follow SQL nullability in API responses

- Optional fields in TypeScript (marked with ?) must be nullable in C#

4. \*\*Case Convention\*\*

- Use PascalCase for C# models

- API responses should use camelCase for JSON properties

- Axios in the TypeScript client will handle the case conversion

5. \*\*Foreign Keys\*\*

- Maintain referential integrity in the database

- Add appropriate foreign key constraints

- Cascade deletes should be carefully considered for each relationship

## Next Steps for API Development

1. Create C# model classes matching these types

2. Implement CRUD endpoints for each entity

3. Implement group management endpoints

4. Add user-group assignment endpoints

5. Create permission validation middleware