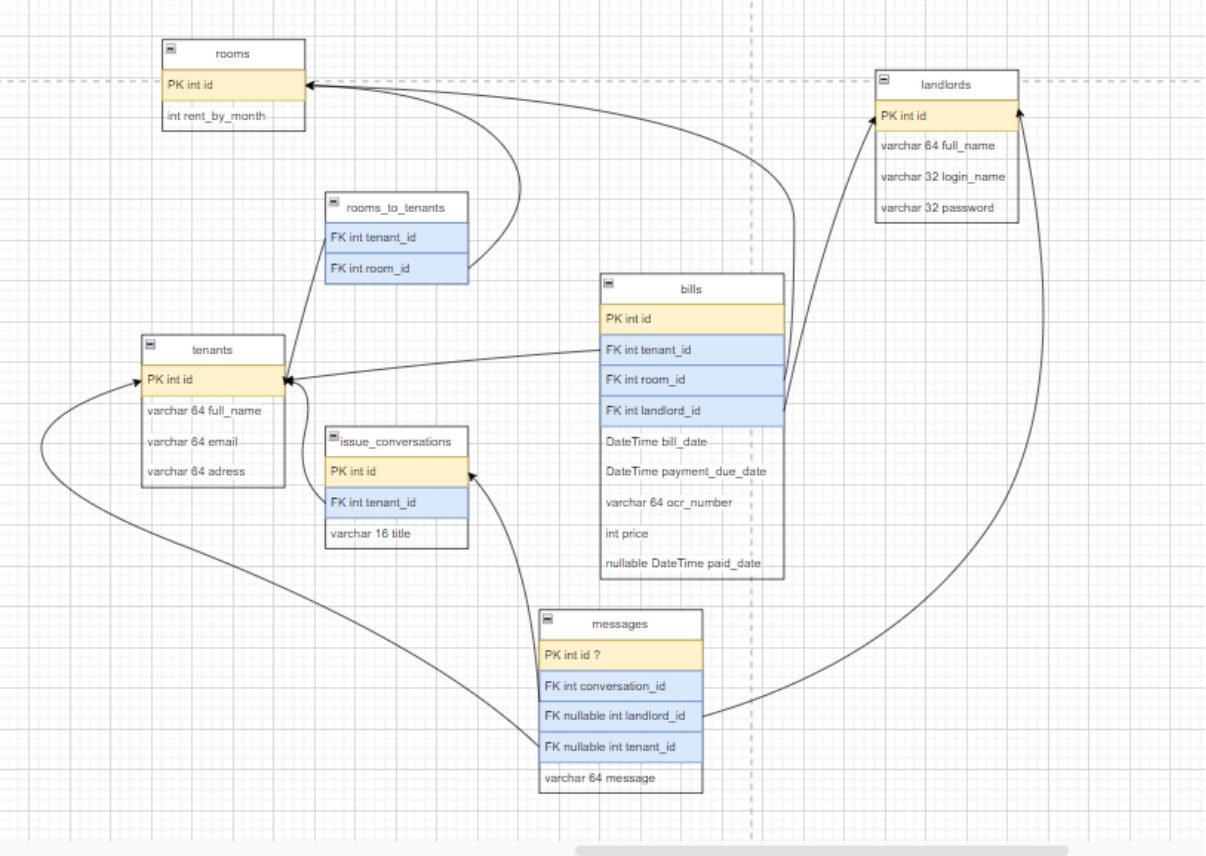
Rapport: RepOstök

Petrus Johansson

Relationsdiagram:



Denna databas är designad för att understödja ett hanteringssystem för hyrda replokaler. Tanken med systemet är att man skall kunna hantera lokaler, räkningar och meddelanden gällande önskemål eller fel i lokalerna.
Värt att notera i designen är att messages har en parent-tabell som håller koll på dess relationer uppåt i flödet. Bills har även en variabel vid namn ‘paid_date’ som förutom att visa om betalningen är sen, kan hålla koll på om räkningen är aktuell.

Djupdyk:

```
public List<SelectMapper> GetMessagesWithNames(int conversationId)
{
    var parameters = new { id = conversationId };

    string query = "SELECT m.id AS 'T1', m.message AS 'T2', t.full_name AS 'T3' FROM messages m" +
    " INNER JOIN tenants t ON t.id = m.tenant_id" +
    " WHERE m.conversation_id = @id;" +

    " SELECT m.id AS 'T1', m.message AS 'T2', l.full_name AS 'T3' FROM messages m" +
    " INNER JOIN landlords l ON l.id = m.landlord_id" +
    " WHERE m.conversation_id = @id;";

    using var connection = DBConnect();

    try
    {
        var multi = connection.QueryMultiple(query, parameters);
        var result = multi.Read<SelectMapper>().ToList();
        var result2 = multi.Read<SelectMapper>().ToList();

        result.AddRange(result2);
        result.Sort((x, y) => (int)x?.T1 - (int)y?.T1);

        return result;
    }
```

I exemplet ovan visas en metod som hämtar alla meddelanden som har en relation till en utvald konversation.

Eftersom att vi använder INNER JOIN i dessa queries för att hämta data som är utomstående för en singulär tabell så passar denna datan inte in på någon av databasens 'model' klasser i C#. Det är här det intressanta problemet kommer in.
Jag ville kunna mappa all joinad data till en generisk typ så att jag får en knytpunkt för all blandad data.

Först så prövade jag att använda en Dictionary för att mappa objekten, detta misslyckades då dictionaries endast kan ha unika keys.
Efter detta så undersökte jag om en lista med tuples skulle kunna funka, men kom snabbt fram till att dessa inte har stöd för Dappers mappning.
Slutligen så kom jag fram till lösningen att använda mig av en generisk 'model' klass med enbart properties av typen 'object' så att vilken data som helst skulle kunna mappas till denna. (I lösningen heter klassen SelectMapper).

Efter jag lyckats mappa meddelanden till SelectMapper så uppstod ännu ett intressant problem: Meddelanden var inte sorterade efter deras Id då multi-queryn först hämtade tenant datan och sedan landlord datan.
Sorteringen av datan löste jag dock relativt snabbt med List klassens inbyggda Sort metod. Jag använde mig av ett enkelt lambda uttryck som jämför Id't på listans medlemmar som illustrerat ovan. Eftersom att jag visste att jag inte skulle behöva sortera detta igen så använde jag mig av ett lambda uttryck istället för en dedikerad sorterar-klass.

Vidareutveckling:

Hade jag haft mer tid för detta projekt så hade jag huvudsakligen implementerat funktionalitet för bills. Detta visste jag från början att det skulle vara det svåraste momentet av implementation och utelämnade därför detta i mån av tid i implementationsprocessen. Hade jag implementerat det så hade jag skapat ett automatiskt system som läser av dagens datum och jämför detta mot ett satt datum för att utfärda räkningarna. Skulle dagens datum vara högre än det satta datumet så sätts en process igång där priserna för varje rum och hyresgäst räknas ut och räkningarna utfärdas till respektive hyresgäst genom databasen.
Hyresgästerna skulle i sin tur kunna "betala" och hyresvärdarna skulle kunna få en UI där man ser vem som har betalat och inte. Implementation av detta hade även använt datan för hyresgästernas personliga information. Data som i dagens läge är överflödigt.

På längre sikt hade jag velat implementera ett beställningssystem där hyresvärdarna kan beställa varor för att underhålla deras lokaler. Saker som lampor, brytare etc.
I databasen hade detta sett ut ungefär som LekOStök med tabeller för order och produkter. Detta hade varit väldigt lätt att utveckla inom databasen då de utökade tabellerna endast skulle ha en relation till 'landlords som' i sin tur endast skulle kräva minimal modifikation.