

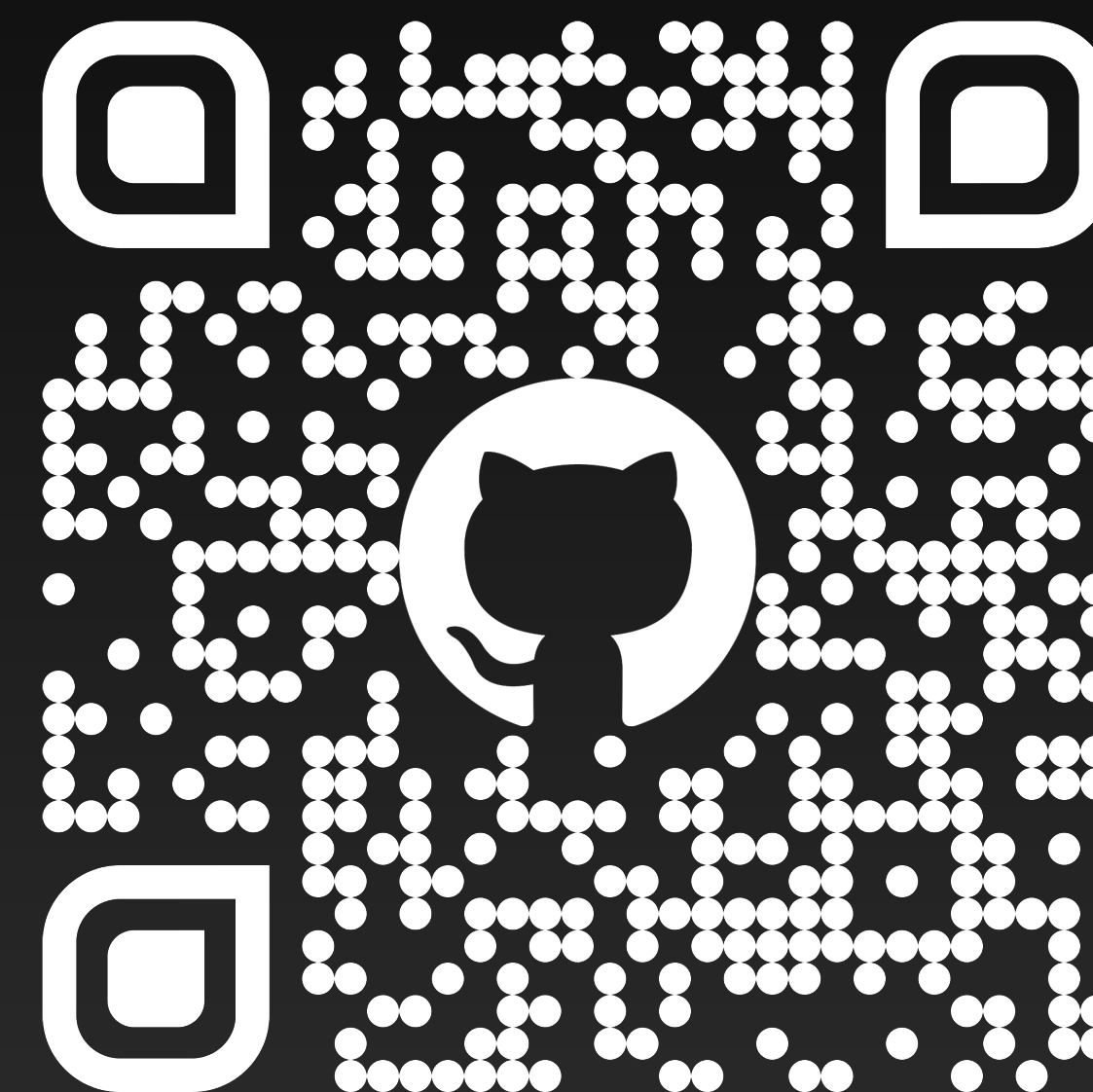
# EF 7: TPC

Новый способ хранения иерархий

Круглов Георгий

# О себе

- Учусь на третьем курсе университета ИТМО
- Опыт разработки на C# более двух лет
- Ведущий разработчик в Omnytech
- Middle Backend разработчик в InfoWise
- Преподаю ООП в университете ИТМО
- [github.com/ronimizy](https://github.com/ronimizy)



# NB

- Сторонник богатой модели данных
- Имею большой опыт в настройке модели EF
- Планирую перенос иерархии в pet-проекте на TPC

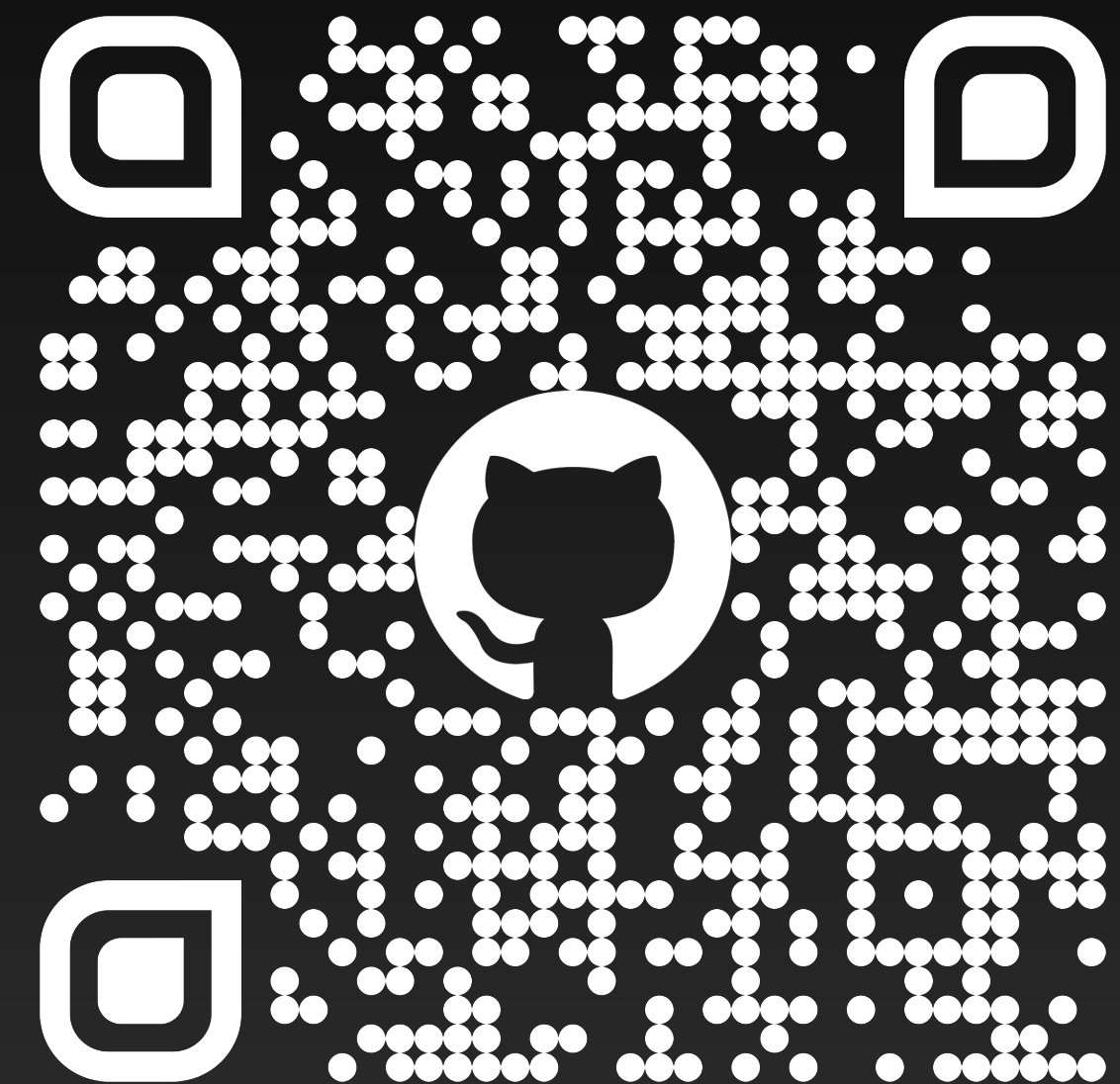
# Roadmap

- Разбор объектной модели для примеров
- Существующие реализации хранения иерархий
  - Table Per Hierarchy (TPH)
  - Table Per Type (TPT)
- Обзор Table Per Concrete type (TPC)
- Сравнение TPT и TPC
- Сравнение SQL, генерируемого TPH, TPT и TPC
- Бенчмарки TPH, TPT и TPC
- Миграции на TPC

# Ресурсы

[github.com/ronimizy/SpbDotNet.TPC](https://github.com/ronimizy/SpbDotNet.TPC)

- .Model
- .DataAccess
- .Application
- .Common
- .Tests
- .Benchmarks
- .Playground
- /Docker
- /Docs

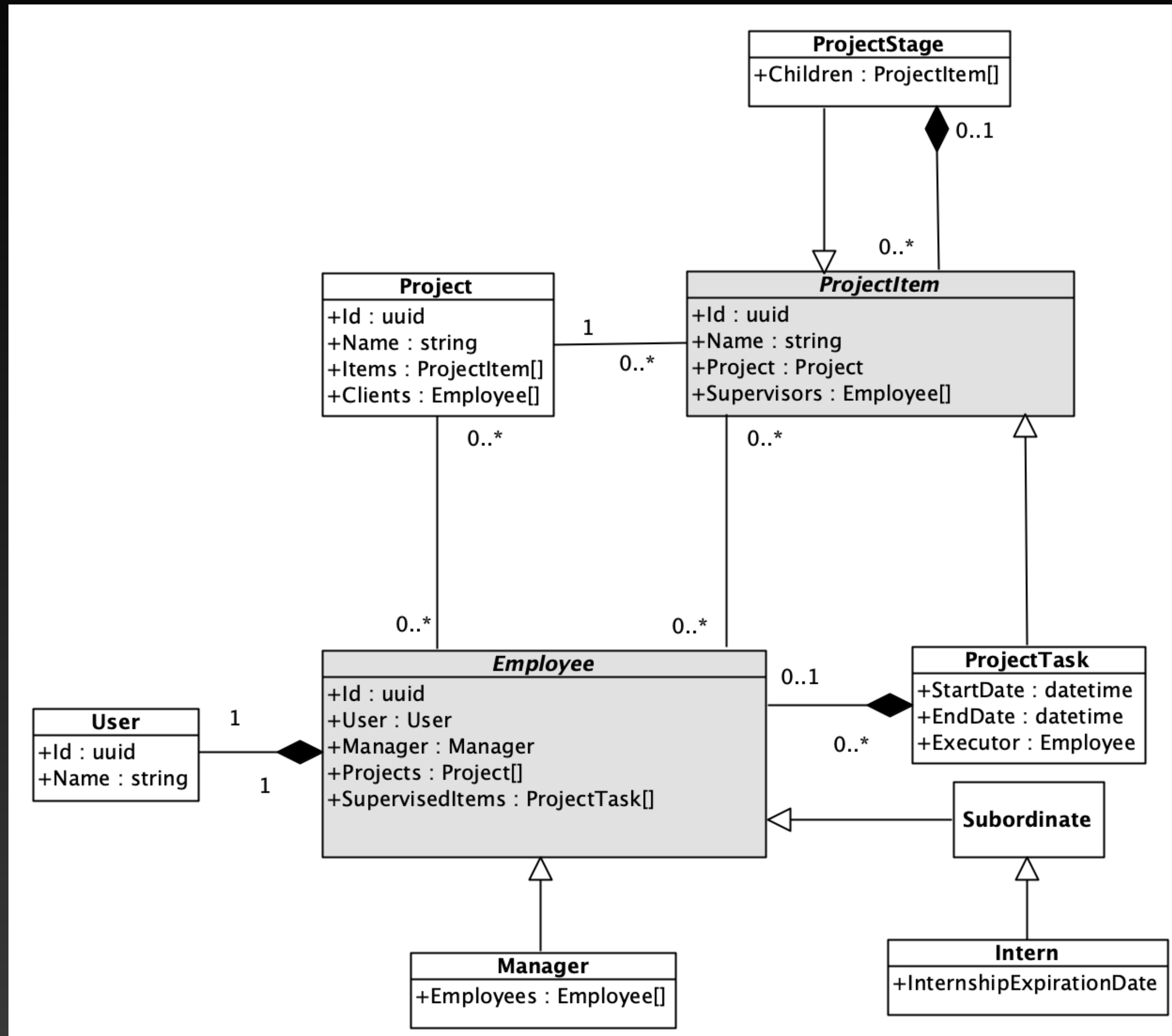


# Проблема

## Необходимость обеспечивать персистентность иерархий

- При моделировании ОО систем используется наследование
- Реляционные БД не поддерживают наследование
- ЕF предоставляет механизмы решения данной проблемы, но они имеют ряд недостатков

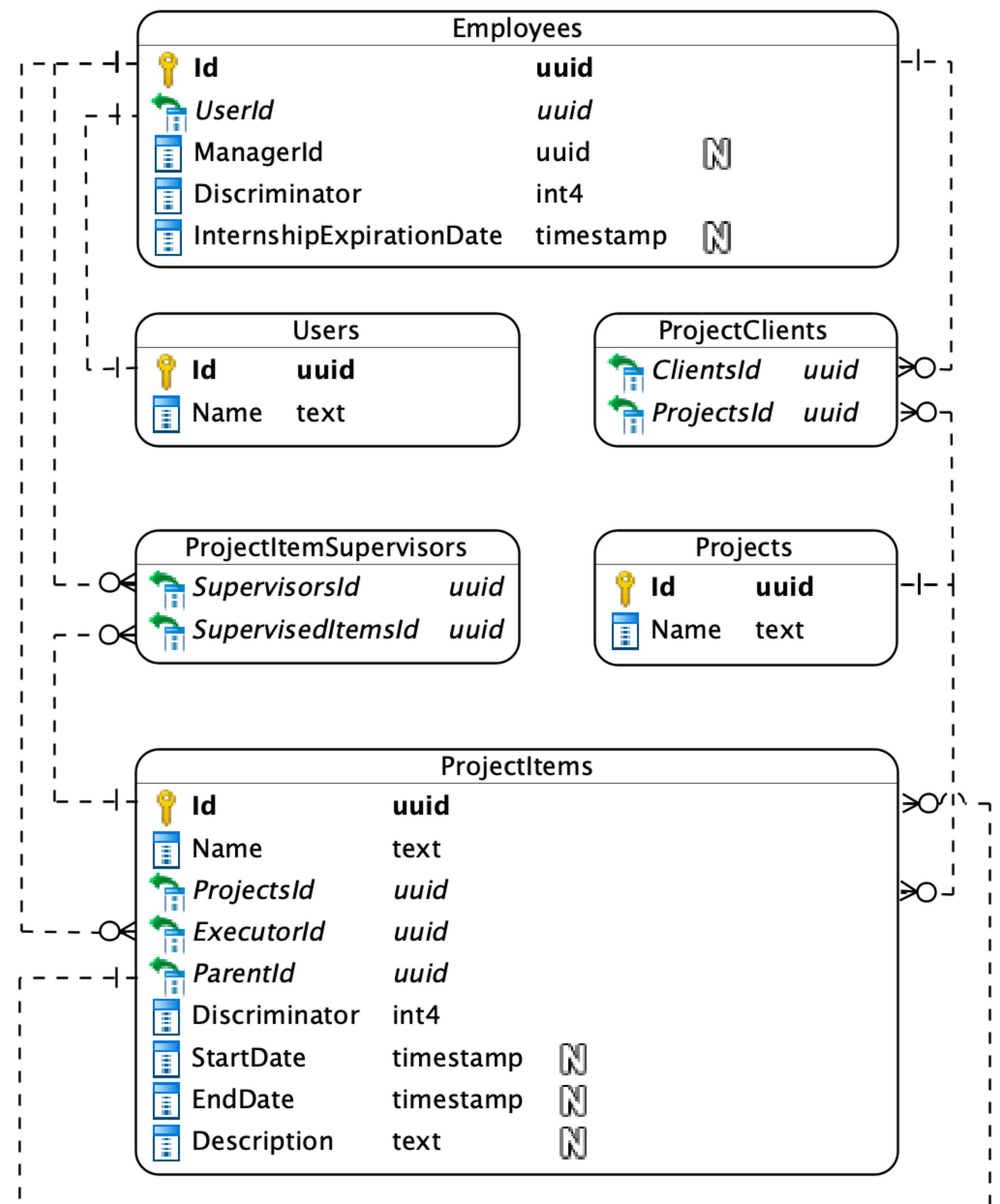
# Описание объектной модели



# TPH

## Table Per Hierarchy

- Используется в EF по-умолчанию
- Для хранения иерархии используется одна таблица
- Таблица будет содержать колонки для всех свойств всех сущностей иерархии
- Таблица будет содержать дополнительную колонку – дискриминатор  
Дискриминатор – значение, определяющее тип кортежа








# ТРН

## Пример данных










### Employees

 Id	 UserId	 ManagerId	 Discriminator	 InternshipExpiration
27723876...	fa68f44a-e...	<null>	3	<null>
fb7f9e45...	34b49962-7...	<null>	3	<null>
076dad7b...	d0926277-5...	fb7f9e45-b046...	1	2023-07-03 07:48:04.6420...
62c2bdbb...	d8ff847f-1...	27723876-cdea...	1	2023-09-13 18:37:20.1948...
38faeb0a...	5b2cb788-d...	27723876-cdea...	3	<null>
2629eddc...	fdf68b1e-7...	27723876-cdea...	2	<null>
aa94bbf8...	068d22f9-6...	fb7f9e45-b046...	2	<null>
43c1f266...	2e1ee7e7-2...	38faeb0a-aa64...	1	2023-03-16 09:02:15.3370...
c06c90dd...	04681946-d...	38faeb0a-aa64...	1	2023-10-15 06:21:53.6160...
f14a2ff3...	520016bc-9...	38faeb0a-aa64...	2	<null>

# ТРН

## Пример данных

### ProjectItems

 Id ▾	 Name ▾	 ProjectId ▾	 ParentId ▾	 Discriminator ▾	 StartDate ▾	 EndDate ▾	 ExecutorId ▾	 Description ▾
1c3351a...	Awesome ...	f80cd4e2-3f0...	<null>	2	<null>	<null>	<null>	<null>
610bdd5...	Gorgeous...	f80cd4e2-3f0...	<null>	2	<null>	<null>	<null>	<null>
6d303fa...	Handcraf...	f80cd4e2-3f0...	<null>	2	<null>	<null>	<null>	<null>
ce4ebd3...	Gorgeous...	f80cd4e2-3f0...	<null>	2	<null>	<null>	<null>	<null>
e18c5c7...	Handmade...	f80cd4e2-3f0...	<null>	2	<null>	<null>	<null>	<null>
52238fa...	Licensed...	f80cd4e2-3f0...	1c3351a1-3b1...	2	<null>	<null>	<null>	<null>
825217b...	Small Co...	f80cd4e2-3f0...	ce4ebd38-d3e...	2	<null>	<null>	<null>	<null>
8a64c1d...	Rustic P...	f80cd4e2-3f0...	e18c5c78-ebb...	2	<null>	<null>	<null>	<null>
e599c87...	Ergonomi...	f80cd4e2-3f0...	610bdd51-dc5...	2	<null>	<null>	<null>	<null>
ea85ca8...	Handmade...	f80cd4e2-3f0...	6d303faa-f50...	2	<null>	<null>	<null>	<null>
27d9496...	Incredib...	f80cd4e2-3f0...	1c3351a1-3b1...	1	2022-06-05 20...	2023-06-17...	<null>	Saepe ex quos ...
4a989cb...	Practica...	f80cd4e2-3f0...	610bdd51-dc5...	1	2022-07-08 23...	2023-07-14...	<null>	Saepe exercita...
91e0fb3...	Refined ...	f80cd4e2-3f0...	6d303faa-f50...	1	2022-09-18 11...	2023-02-08...	<null>	Dolorem ut et ...
9d0b68d...	Ergonomi...	f80cd4e2-3f0...	ce4ebd38-d3e...	1	2022-10-11 05...	2023-08-25...	<null>	Itaque nisi be...
b5529e0...	Tasty Me...	f80cd4e2-3f0...	e18c5c78-ebb...	1	2022-09-02 09...	2023-12-06...	<null>	Totam consequa...
7a4151f...	Awesome ...	f80cd4e2-3f0...	825217b5-2e1...	1	2022-03-17 07...	2023-08-09...	<null>	Qui voluptas r...
8ac74a4...	Handcraf...	f80cd4e2-3f0...	ea85ca88-c51...	1	2022-03-11 11...	2023-12-04...	<null>	Vel architecto...
aa1b952...	Tasty Co...	f80cd4e2-3f0...	52238fad-b8f...	1	2022-03-24 19...	2023-05-05...	<null>	Quis reprehend...
ae2a83f...	Tasty Co...	f80cd4e2-3f0...	e599c87d-e2c...	1	2022-12-29 16...	2023-12-19...	<null>	Modi ea dolore...
d898d87...	Licensed...	f80cd4e2-3f0...	8a64c1d6-438...	1	2022-04-22 19...	2023-09-24...	<null>	Vitae fugit mo...

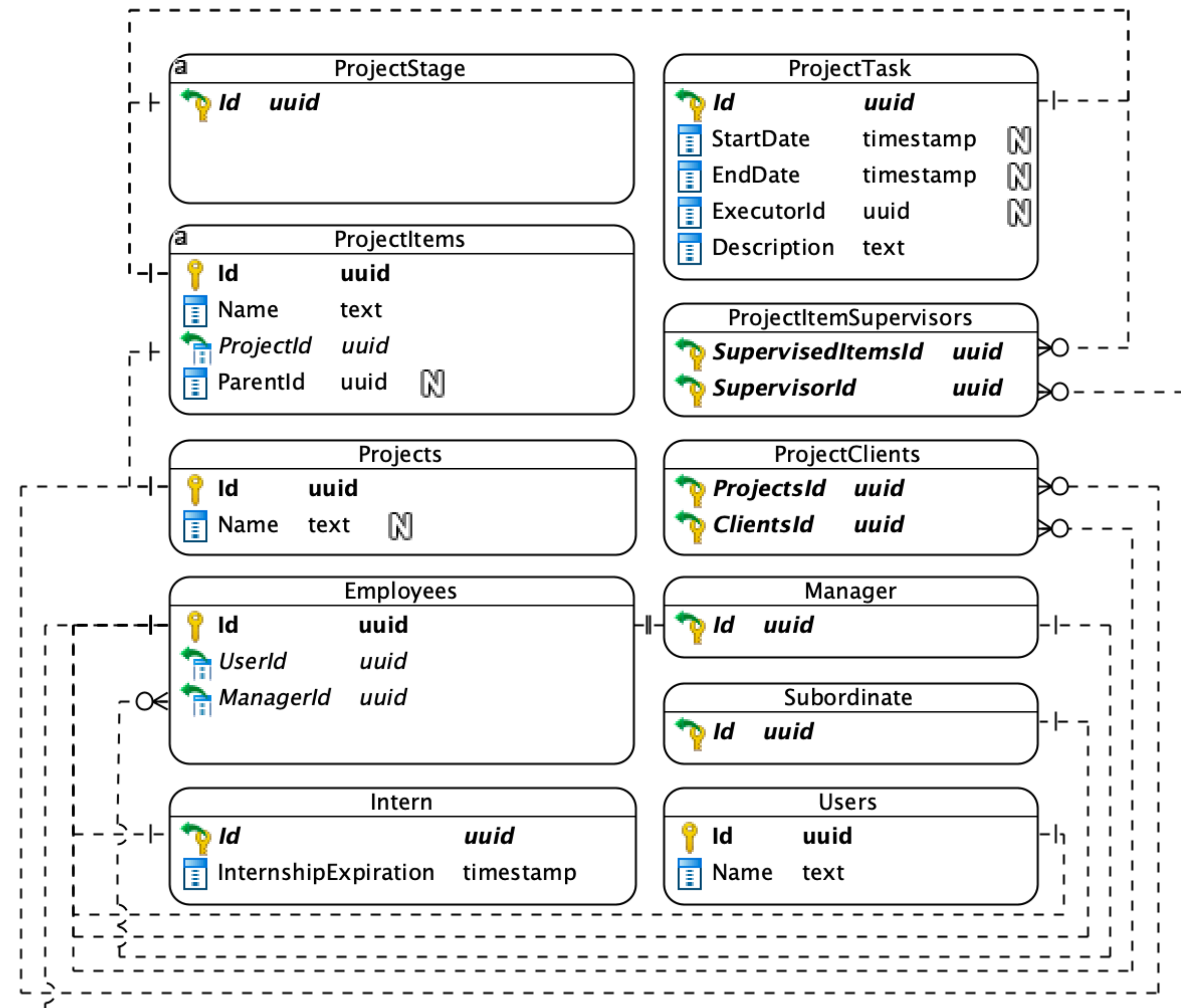
# ТРН

## Недостатки

- Нарушение третьей нормальной формы  
Наличие данных в столбцах зависит от неключевого атрибута (дискриминатора)
- Разреженность хранимых данных

**TPT**

- Для **каждого** типа иерархии создаётся своя таблица
- Данные одного объекта, могут храниться в нескольких таблицах
- Данные нормализованы
- Нет разреженных таблиц





# ТРТ

## Пример данных (Employee)

### Subordinate

Id
076dad7b-...
62c2bdbb-...
2629eddc-...
aa94bbf8-...
43c1f266-...
c06c90dd-...
f14a2ff3-...

### Manager

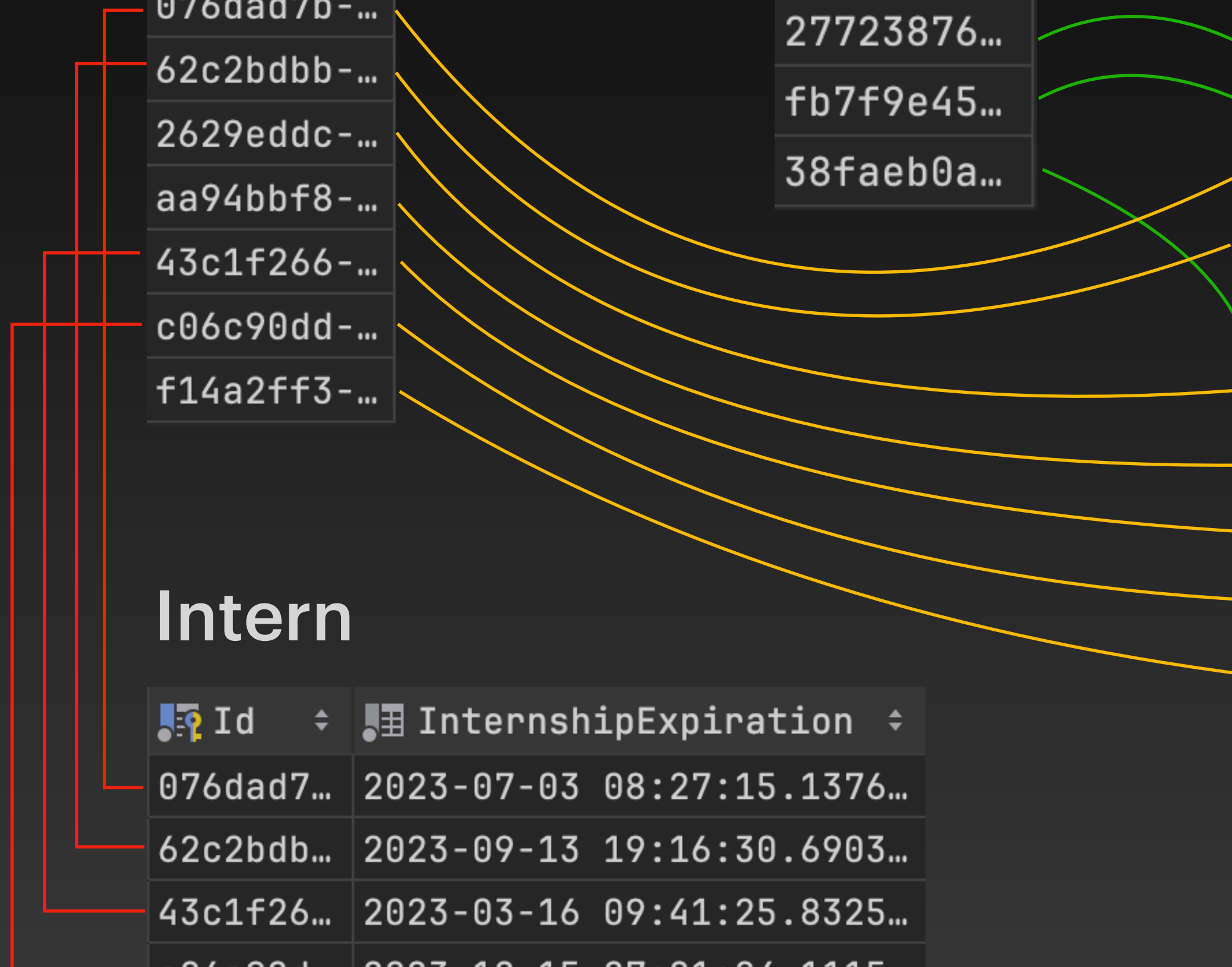
Id
27723876...
fb7f9e45...
38faeb0a...

### Employee

Id	UserId	ManagerId
277238...	fa68f44a-...	<null>
fb7f9e...	34b49962-...	<null>
076dad...	d0926277-...	fb7f9e45-b04...
62c2bd...	d8ff847f-...	27723876-cde...
38faeb...	5b2cb788-...	27723876-cde...
2629ed...	fdf68b1e-...	27723876-cde...
aa94bb...	068d22f9-...	fb7f9e45-b04...
43c1f2...	2e1eefe7-...	38faeb0a-aa6...
c06c90...	04681946-...	38faeb0a-aa6...
f14a2f...	520016bc-...	38faeb0a-aa6...

### Intern

Id	InternshipExpiration
076dad7...	2023-07-03 08:27:15.1376...
62c2bdb...	2023-09-13 19:16:30.6903...
43c1f26...	2023-03-16 09:41:25.8325...
c06c90d...	2023-10-15 07:01:04.1115...



# ТРТ

## Недостатки

- Производительность
- Добавление объекта приводит к нескольким операциям вставки

```
INSERT INTO "Employees" ("Id", "UserId")  
VALUES (Guid_1, Guid_2);
```

```
INSERT INTO "Subordinate" ("Id", "ManagerId")  
VALUES (Guid_1, NULL);
```

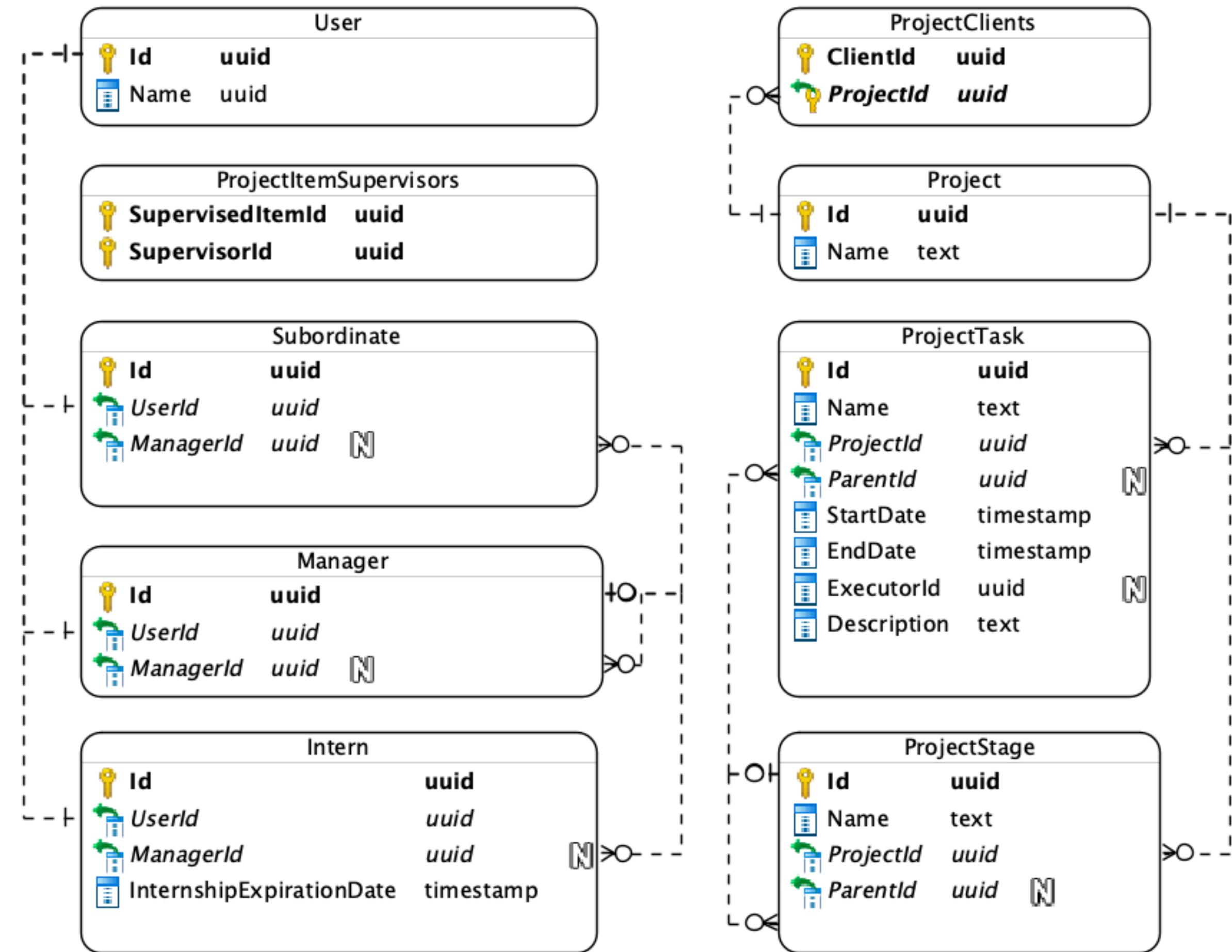
- Получение объектов приводит к операциям JOIN

```
SELECT e."Id",  
       e."UserId",  
       m."SuperiorId",  
       s."ManagerId",  
       i."InternshipExpiration",  
       CASE  
         WHEN i."Id" IS NOT NULL THEN 'Intern'  
         WHEN s."Id" IS NOT NULL THEN 'Subordinate'  
         WHEN m."Id" IS NOT NULL THEN 'Manager'  
         END AS "Discriminator"  
  
FROM "Employees" AS e  
     LEFT JOIN "Manager" AS m ON e."Id" = m."Id"  
     LEFT JOIN "Subordinate" AS s ON e."Id" = s."Id"  
     LEFT JOIN "Intern" AS i ON e."Id" = i."Id"
```

# TPC

## Table Per Concrete type

- Таблицы создаются только для **конкретных** типов
- Данные одного объекта хранятся в одной таблице
- Данные нормализованы
- Нет разреженных таблиц



# TPT vs TPC

- В обеих стратегиях, данные денормализуются в памяти при запросах
- Между таблицами TPC нет пересечений по первичным ключам
- TPC конкатенирует данные через `UNION ALL` вместо метчинга через `JOIN`
- Данные одного объекта вставляются одним запросом при использовании TPC



# SQL TPH vs TPT vs TPC

## SELECT

### TPH

```
SELECT e."Id",
       e."Discriminator",
       e."UserId",
       e."SuperiorId",
       e."ManagerId",
       e."InternshipExpiration"
FROM "Employees" AS e
```

### TPT

```
SELECT e."Id",
       e."UserId",
       m."SuperiorId",
       s."ManagerId",
       i."InternshipExpiration",
       CASE
         WHEN i."Id" IS NOT NULL THEN 'Intern'
         WHEN s."Id" IS NOT NULL THEN 'Subordinate'
         WHEN m."Id" IS NOT NULL THEN 'Manager'
         END AS "Discriminator"
FROM "Employees" AS e
     LEFT JOIN "Manager" AS m ON e."Id" = m."Id"
     LEFT JOIN "Subordinate" AS s ON e."Id" = s."Id"
     LEFT JOIN "Intern" AS i ON e."Id" = i."Id"
```

### TPC

```
SELECT m."Id",
       m."UserId",
       m."SuperiorId",
       NULL::uuid AS "ManagerId",
       NULL::timestamp AS "InternshipExpiration",
       'Manager' AS "Discriminator"
FROM "Manager" AS m
```

```
UNION ALL
SELECT s."Id",
       s."UserId",
       NULL AS "SuperiorId",
       s."ManagerId",
       NULL AS "InternshipExpiration",
       'Subordinate' AS "Discriminator"
FROM "Subordinate" AS s
```

```
UNION ALL
SELECT i."Id",
       i."UserId",
       NULL AS "SuperiorId",
       i."ManagerId",
       i."InternshipExpiration",
       'Intern' AS "Discriminator"
FROM "Intern" AS i
```

# SQL TPH vs TPT vs TPC

## INSERT

### TPH

```
INSERT INTO "Employees" ("Id", "Discriminator", "ManagerId", "UserId")  
VALUES (Guid_1, 2, NULL, Guid_2);
```

### TPT

```
INSERT INTO "Employees" ("Id", "UserId")  
VALUES (Guid_1, Guid_2);
```

```
INSERT INTO "Subordinate" ("Id", "ManagerId")  
VALUES (Guid_1, NULL);
```

### TPC

```
INSERT INTO "Subordinate" ("Id", "ManagerId", "UserId")  
VALUES (Guid_1, NULL, Guid_2);
```

# TPH vs TPT vs TPC

## Сравнение производительности

- CPU: Xeon E5-2667 v2 (3.5GHz)
- RAM: 32 GB DDR3 1866 MHz
- Storage: Samsung 860 EVO 1TB
- Данные распределены равномерно между типами иерархий
- PostgreSQL 14.2 (default)

# TPH vs TPT vs TPC

## SELECT full hierarchy (Employees)

Size	Strategy	Mean
1000	Tpc	12.221 ms
1000	Tph	10.994 ms
1000	Tpt	17.827 ms
10000	Tpc	125.475 ms
10000	Tph	111.297 ms
10000	Tpt	214.081 ms
100000	Tpc	1,510.143 ms
100000	Tph	1,453.715 ms
100000	Tpt	2,552.307 ms

# TPH vs TPT vs TPC

## SELECT hierarchy slice (Subordinates)

Size	Strategy	Mean
1000	Tpc	5.345 ms
1000	Tph	4.828 ms
1000	Tpt	9.975 ms
10000	Tpc	41.257 ms
10000	Tph	41.019 ms
10000	Tpt	98.131 ms
100000	Tpc	529.300 ms
100000	Tph	489.487 ms
100000	Tpt	1,085.802 ms

# TPH vs TPT vs TPC

## SELECT single hierarchy type (Interns)

Size	Strategy	Mean
1000	Tpc	2.823 ms
1000	Tph	2.888 ms
1000	Tpt	5.617 ms
10000	Tpc	19.336 ms
10000	Tph	20.893 ms
10000	Tpt	45.978 ms
100000	Tpc	255.946 ms
100000	Tph	276.843 ms
100000	Tpt	538.376 ms

# Highly sparse tables

## Model

```
public class DisplayEmployeeUniform
    : EmployeeUniform
{
    int TuxedoColorArgb
    string TuxedoName
    int TuxedoMostPopularSize
    int DisplayShirtColorArgb
    string DisplayShirtName
    int DisplayShirtMostPopularSize
    int DisplayShoeColorArgb
    string DisplayShoeName
    int DisplayShoeMostPopularSize
    int DisplayTieColorArgb
    string DisplayTieName
    int DisplayTieMostPopularSize
    int DisplayBeltColorArgb
    string DisplayBeltName
    int DisplayBeltMostPopularSize
}
```

```
public class OfficialEmployeeUniform
    : EmployeeUniform
{
    int JacketColorArgb
    string JacketName
    int JacketMostPopularSize
    int OfficialPantsColorArgb
    string OfficialPantsName
    int OfficialPantsMostPopularSize
    int OfficialShoesColorArgb
    string OfficialShoesName
    int OfficialShoesMostPopularSize
    int OfficialShirtColorArgb
    string OfficialShirtName
    int OfficialShirtMostPopularSize
    int OfficialTieColorArgb
    string OfficialTieName
    int OfficialTieMostPopularSize
}
```

```
public class CasualEmployeeUniform
    : EmployeeUniform
{
    int ShirtColorArgb
    string ShirtName
    int ShirtMostPopularSize
    int JeansColorArgb
    string JeansName
    int JeansMostPopularSize
    int ShoesColorArgb
    string ShoesName
    int ShoesMostPopularSize
    bool HoodieAllowed
}
```

# Highly sparse tables

## SELECT single hierarchy type

### PostgreSQL

Size	Strategy	Mean
1000	Tpc	1.627 ms
1000	Tph	1.740 ms
1000	Tpt	2.196 ms
10000	Tpc	9.326 ms
10000	Tph	10.009 ms
10000	Tpt	13.197 ms

### SQL Server

Size	Strategy	Mean
1000	Tpc	3.102 ms
1000	Tph	5.364 ms
1000	Tpt	5.896 ms
10000	Tpc	19.785 ms
10000	Tph	26.423 ms
10000	Tpt	24.268 ms



# Migrations

- Автоматические миграции с сохранением данных не реализованы
- Реализация данного функционала не планируется  
[github.com/dotnet/efcore/issues/30083](https://github.com/dotnet/efcore/issues/30083)
- Миграции генерируют структуру
- Доработка миграция – достаточно тривиальная SQL вставка

# Выводы

- TPT всегда будет проигрывать в производительности
- TRH и TPC схожи по производительности
  - При фильтрации по типу в TRH execution plan оптимальнее, так как это простой **WHERE**
  - TPC наиболее оптимален на выборке из одного терминального типа
- Схожесть производительности TRH и TPC можно объяснить оверхедом на разных уровнях

# Выводы

## ТРС

- Плюсы
  - Нормализация данных
  - Отсутствие необходимости в специальных оптимизациях хранения со стороны БД
  - Высокая производительность при запросах по конкретному типу
  - Высокая производительность при вставках
- Минусы
  - Оверхед на денормализацию данных

**Спасибо за внимание**