VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projekt kurzu IDS 2019

Zadání č.44 z kurzu IUS

Upíří krevní banka

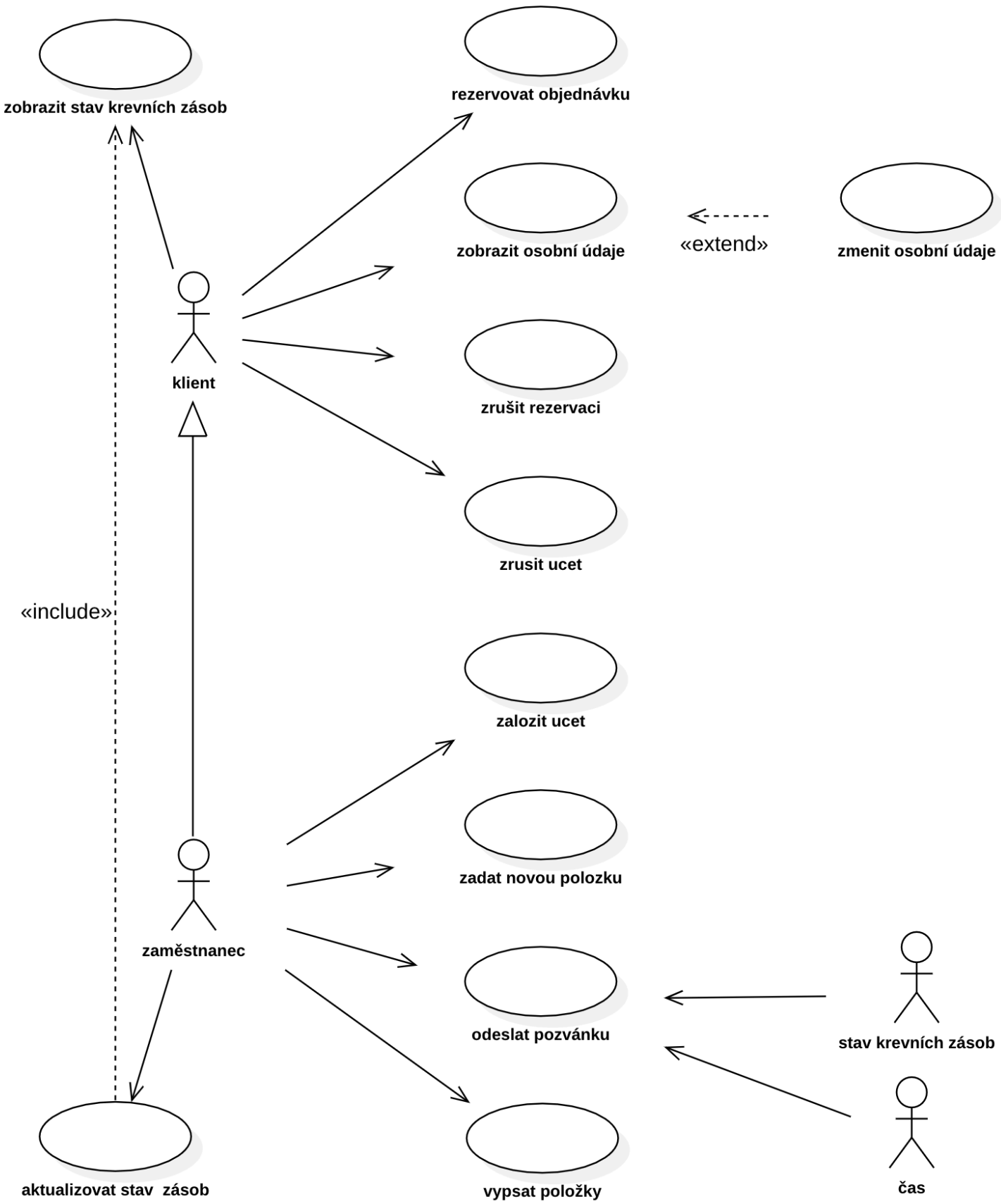
Libor ¨xdvora2t¨ Dvořáček
David ¨xvodak05¨ Vodák

V Brně 24.dubna 2019

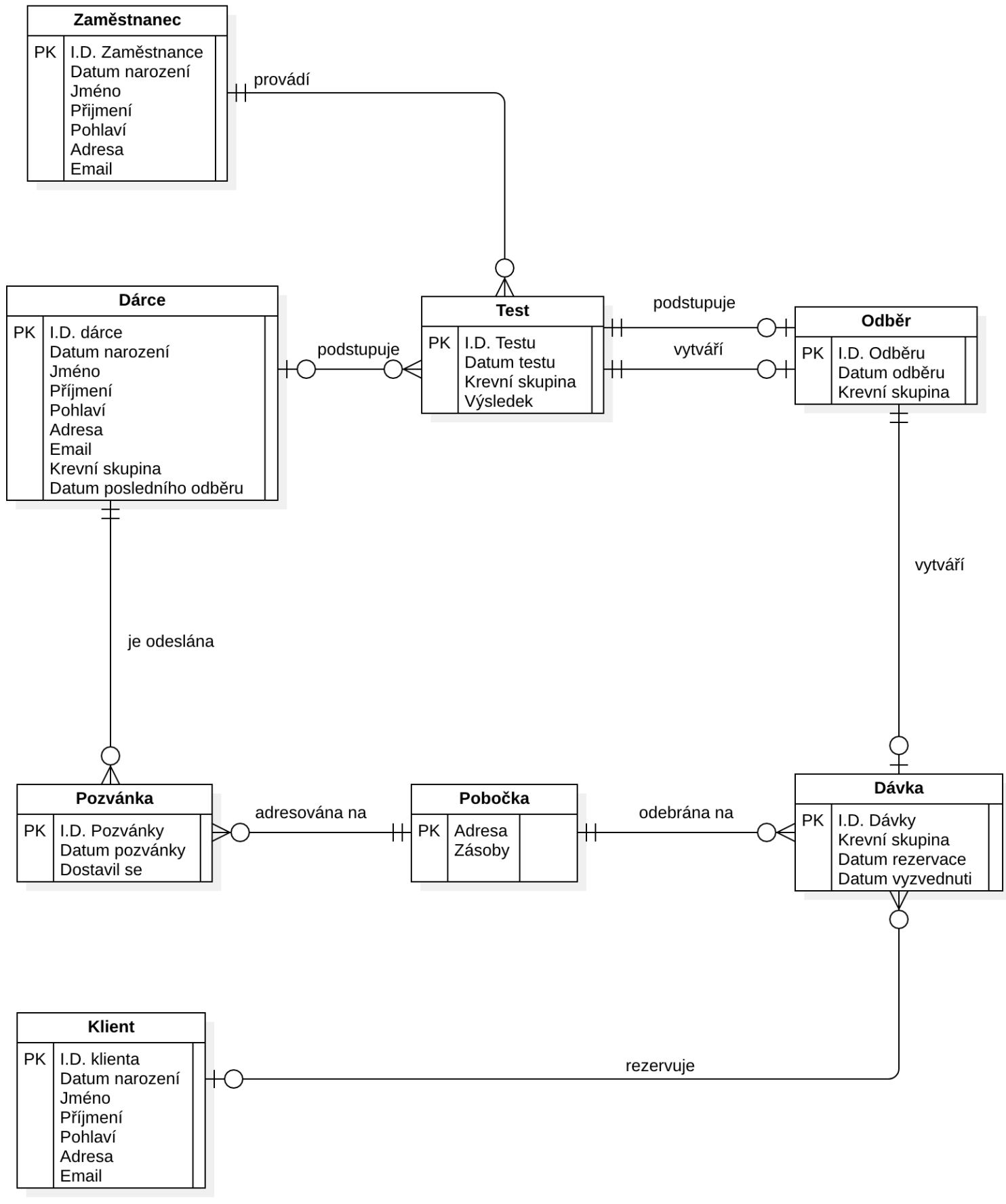
ZADÁNÍ

Navrhněte informační systém pro upíří krevní banku. Krevní banka si zve své dárce k dalšímu odběru podle stavu krevní banky a data posledního odběru daného dárce. Každý takový dárce obdrží pozvánku na konkrétní pobočku krevní banky. V IS jsou uloženy informace, zda dárce na pozvánku zareagoval - dostavil se k odběru či nikoliv. Před vlastním odběrem dárce podstoupí test, zda mu může být odebrána krev, pokud je test v pořádku je dárci odebrána krev (jedna dávka), která dále musí znovu podstoupit test, z něhož se vyhodnotí její vhodnost ke konzumaci. Vzniklou dávku si u krevní banky mohou rezervovat různí klienti, o nichž je v databázi uložen záznam. Rezervace se provádí pro zadané množství dávek u pobočky, na které je materiál uskladněn, na určité datum.

USE CASE DIAGRAM



ERD DIAGRAM


Zaměstnanec
PK
I.D. Zaměstnance
Datum narození
Jméno
Přijmení
Pohlaví
Adresa
Email
provádí
Dárce
PK
I.D. dárce
Datum narození
Jméno
Příjmení
Pohlaví
Adresa
Email
Krevní skupina
Datum posledního odběru
podstupuje
Test
PK
I.D. Testu
Datum testu
Krevní skupina
Výsledek
podstupuje
vytváří
Odběr
PK
I.D. Odběru
Datum odběru
Krevní skupina
vytváří
je odeslána
Pozvánka
PK
I.D. Pozvánky
Datum pozvánky
Dostavil se
adresována na
Pobočka
PK
Adresa
Zásoby
odebrána na
Dávka
PK
I.D. Dávky
Krevní skupina
Datum rezervace
Datum vyzvednuti
Klient
PK
I.D. klienta
Datum narození
Jméno
Příjmení
Pohlaví
Adresa
Email
rezervuje

IMPLEMENTACE

Nejdříve jsme vytvořili jednotlivé objekty databáze, definovali primární a cizí klíče a následně naplnili databází ukázkovými daty. Generalizaci jsme v našem případě vyřešili definicí jednotlivých tabulek podtypů, které obsahují společné atributy supertypu.

TRIGGERY

Všechny primární klíče generujeme pomocí souboru příkazů `GENERATED BY DEFAULT ON NULL AS IDENTITY`, avšak pro splnění zadání je v definici tabulky `Fix` výraz `ON NULL` vynechán a generování primárního klíče zajištěno sekvencí a požadovaným triggerem.

Druhý trigger slouží pro kontrolu věku dárce před jeho přidáním do tabulky. Ve skutečnosti ho tvoří triggery dva. První vkládá záznam do pole v případě, že není splněna věková podmínka a druhý trigger toto pole prochází a záznamy maže. Garance správného pořadí triggerů je dána jejich konstrukcí kdy pouze první trigger prochází všechny řádky tabulky.

PROCEDURY

Procedura `bad_blood()` je při zavolání zodpovědná za průchod tabulky `Test` a vymazání z databáze všech dárců, kteří třemi nebo více testy neprošli.

Procedura `warn_donors()` má varovat dárce krve před nebezpečným upírem, který se pohybuje ve dnech mezi 12.01.2019 a 20.01.2019 u jedné z poboček (Coronary St.). Procedura k tomu používá kurzor `invs`, do kterého ukládá tabulku `Invitation` a proměnnou `l_inv`, do které ukládá jeden řádek tohoto kuzoru. V cyklu se potom prochází všechy řádky kurzoru a hledají se dárci kteří by mohli být potenciálně v ohrožení. Pro ty se pak vytvoří "varovný email".

EXPLAIN PLAN A INDEX

Při optimalizaci vyhledávání pomocí indexu se nám při tak skromném naplnění našich tabulek nedařilo dosáhnout lepšího výsledku, protože SŘBD zřejmě nepokládal za nutné index používat. Toto jsem vyřešili explicitním nařízením systému `INDEX` použít pomocí index hint.

Před použitím indexu

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
Plan hash value: 3065434750
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |    13 |   468 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY                      |              |    13 |   468 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS                      |              |    13 |   468 |     3   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                     |              |    13 |   468 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL               | FIX          |    13 |   169 |     3   (0)| 00:00:01 |
|*  5 |     INDEX UNIQUE SCAN               | PK_CLIENT    |     1 |       |     0   (0)| 00:00:01 |
|   6 |    TABLE ACCESS BY INDEX ROWID      | CLIENT       |     1 |    23 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

Po použití indexu

```
PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
Plan hash value: 1997630806
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |    13 |   468 |     6  (17)| 00:00:01 |
|   1 |  HASH GROUP BY                      |              |    13 |   468 |     6  (17)| 00:00:01 |
|*  2 |   HASH JOIN                         |              |    13 |   468 |     5   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL                | FIX          |    13 |   169 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS BY INDEX ROWID BATCHED| CLIENT       |    20 |   460 |     2   (0)| 00:00:01 |
|   5 |     INDEX FULL SCAN                 | CLIENT_SNAME |    20 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

Z výše uvedených příkladů je patrné, že jsme se vyhnuli prohledávání tabulek ve vnořených cyklech a použili párování záznamů v tabulkách pomocí hash klíčů. Tímto se nám podařilo snížit zátěž procesoru na úkor více přístupů do paměti. Toto řešení je v případě kdy testujeme rovnost výhodnější.

Podrobnější výpis:

`SELECT` příkaz

`HASH GROUP BY` - shlukování podle hashovacího klíče

`NESTED LOOPS` - pro každý řádek první tabulky se prochází tabulka druhá a kontroluje podmínku

`TABLE ACCESS FULL` - prochází se celá tabulka (bez indexu)

`INDEX UNIQUE SCAN` - přístup k tabulce přes b-strom

`TABLE ACCESS BY INDEX ROWID` - přístup přes konkrétní řádek

S indexem:

`HASH JOIN` - spojení tabulek za pomocí hashovacích klíčů

`TABLE ACCESS BY INDEX ROWID BATCHED` - načtení několika `ROWID`s najednou

`INDEX FULL SCAN` - pro výpis se použil index

MATERIALIZOVANÝ POHLED

Před vytvořením materializovaného pohledu pro druhého člena týmu jsme vytvořili `LOG`, abychom mohli použít rýchlé obnovení na místo kompletního. Jako dálši atributy pohledu jsme zvolili použití `CACHE` při čtení bloků, `BUILD IMEDIATE` pro naplnění pohledu hned po jeho vytvoření a `ENABLE QUERY REWRITE` pro použití pohledu pro optimalizaci stejného dotazu.

ZÁVĚR

Při práci na projektu jsme čerpali infromace převážně ze slidů kurzu IDS a oficiálních dokumentů Oracle. Pro připojení ke školnímu databázovému serveru jsem použili aplikaci `SQL Developer` a image bežící z docker containeru `ORACLE Instant Client 12c`.