VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IDS – Databázové systémy 2020

Projektová dokumentace
Zadání č. 26 Banka

30. dubna 2020

Michal Dostál (xdosta51), Radek Švec (xsvecr01)

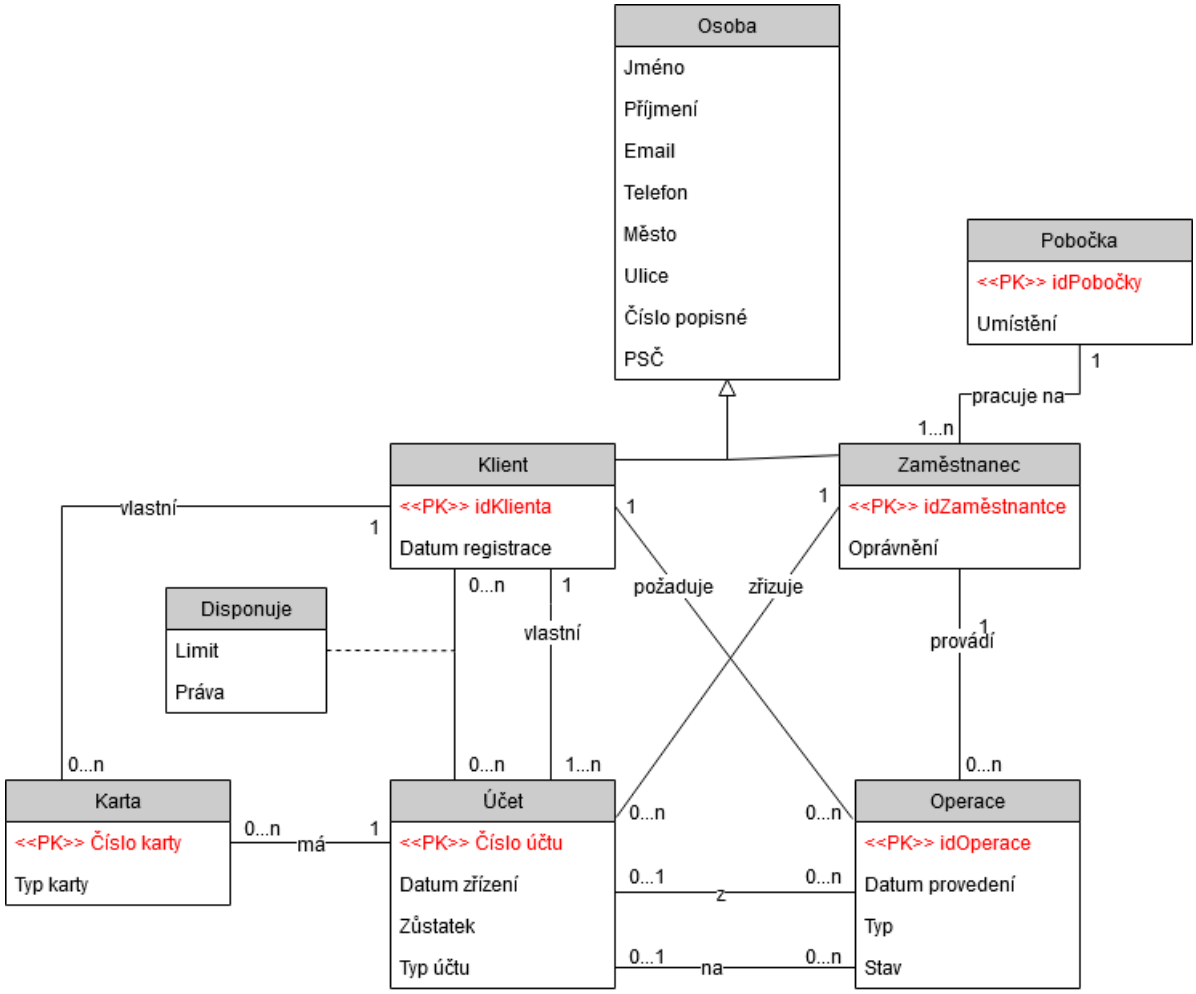
Obsah

1 Zadání

Navrhněte modul informačního systém banky pro správu účtů. Modul musí evidovat klienty, jejich účty a operace s nimi. Předpokládejte, že každý účet má jednoho vlastníka, ale s účtem může disponovat více osob, které určí vlastník. Operace zahrnují vklad na účet, výběr z účtu a převod na jiný účet (téže či jiné banky). Systém musí ukládat informaci o všech operacích s účtem (kdo zadal, kdy, jaká operace a částka, kdo provedl). Se systémem vždy přímo komunikuje pouze pracovník banky. Systém musí také mimo jiné poskytovat výpis z účtu, který se posílá vlastníkovi, tj. výpis všech operací s účtem za dané období.

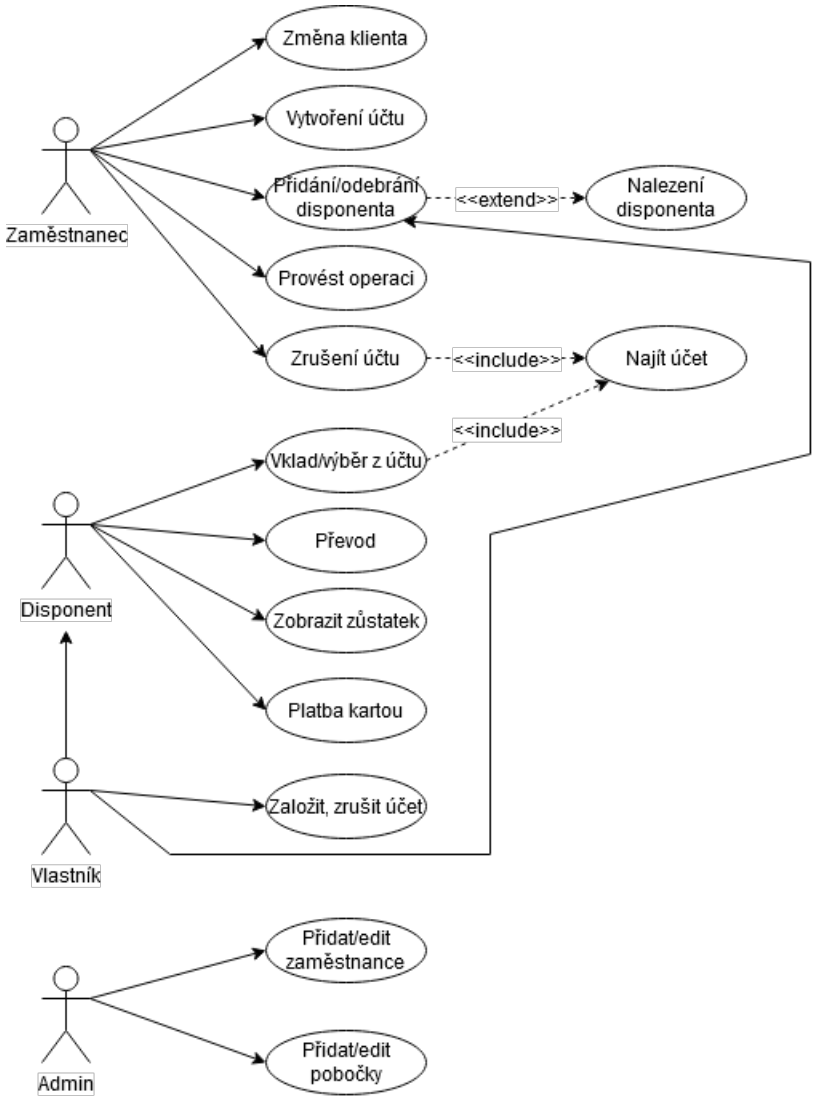
2 Diagramy

2.1 ER Diagram



Obrázek 1: Použitý ER Diagram

2.2 Diagram případu použití



Obrázek 2: Diagram případu použití

3 Popis řešení

V poslední části projektu jsme ve dvoučlenném týmu vytvářeli dvě databázové procedury, dva databázové triggery, z toho jeden pro automatické vytváření primárního klíče, materializovaný pohled, explain plan s využitím explicitně vytvořeného indexu a definici přístupových práv k databázovým objektům pro druhého člena týmu

3.1 Popis vytvořených triggerů

V námi vytvořeném řešení jsou implementovány dva triggery, a to:

- Trigger pro automatické generování primárního klíče (podle zadání). - `klient_increment`
- Trigger pro kontrolu zůstatku a provedení operace. - `kontrola_zustatku`

Trigger - `klient_increment`

Trigger pro automatické generování jsme implementovali pro tabulku `klient`, kde automaticky generuje a inkrementuje idenfitikační číslo (PK) - `klient_id`

Trigger - `kontrola_zůstatku`

Druhý námi implementovaný trigger se stará o kontrolu dostatku financí, při provádění operace převodu z účtu, pokud tato kontrola proběhla v pořádku trigger provede odečtení financí z odchozího účtu a přičte tuto částku na účet cílový. Pokud kontrola neproběhla v pořádku je vyvolána chyba pomocí `raise_applicaton_error()`.

3.2 Vytvořené procedury

Implementované procedury

- Procedura pro připočtení úroku. - `pripocti_urok()`
- Procedura pro zobrazení útraty z účtu. - `zobraz_utratu()`

Procedura - `pripocti_urok()`

Tato procedura má povinný argument `procento`, který udává úrokovou sazbu. Po zavolání této procedury se pomocí kurzoru prochází přes jednotlivé účty a pokud je typ účtu spořící přičte se vypočítaný úrok odpovídající zůstatku a úrokové sazbě. Pro každý účet se vypisuje zůstatek na účtu před a po zúročení. Na konci této procedury se vypíše celková částka peněz k zúročení a celkvoý úrok.

Procedura - `zobraz_utratu()`

V této proceduře jsme uvažovali možnost používání účtu více klienty, a proto je tato procedura primárně určena pro účty s vícero disponentama. Procedura má povinný argument `cislo_uctu`, který údavá, o kterém účtu chceme dané informace zjistit, pracuje se se dvěma kurzory, jeden pro procházení klientů a druhý pro procházení operací. Procedura nejprve projde cyklem operace a sečte celkovou útratu na daném účtu, poté tuto částku vypíše. Přes další cyklus prochází jednotlivé klienty a pro každého klienta je vnořen další cyklus, který prochází všechny operace a pokud se jedná o klienta z nadřazeného cyklu a číslo účtu u operace odpovídá číslu z argumentu, tak se do proměnné celkové útraty klienta přičte částka operace. Celková útrata jednotlivých klientů je poté vypsaná v penězích a procentuálně z celkové útraty na účtu.

Pokud z účtu nejsou utraceny žádné peníze nastane vyjímka dělení nulou a vypíše se chybová hláška.

3.3 Explain Plan

3.3.1 Popis plánu

Pro plány je použit dotaz se spojením dvou tabulek, agregační funkcí a klauzulí GROUP BY. Tento dotaz provádí výpis účtů, které mají více než jednu kartu a jejich počet.

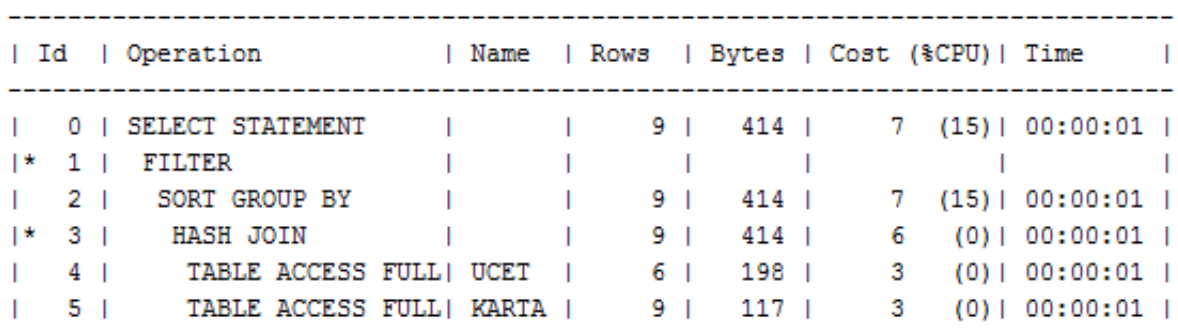
3.3.2 Původní plán

Původní plán je horší z důvodu toho, že tabulka s kartama není nijak seřazená. Toto jsme se rozhodli optimalizovat seřazením karet podle vlastnícího účtu.

3.3.3 Optimalizovaný plán

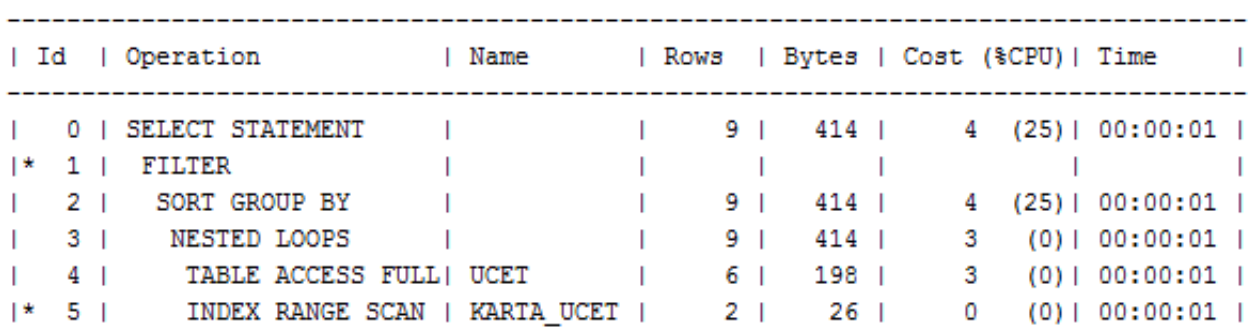
Poté, co jsme použili explicitní index pro tabulku `karta`, aby se řazení provádělo podle vlastnícího účtu byly výsledky plánu skoro dvakrát lepší.

3.3.4 Porovnání obou plánů



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		9	414	7 (15)	00:00:01
* 1	FILTER					
2	SORT GROUP BY		9	414	7 (15)	00:00:01
* 3	HASH JOIN		9	414	6 (0)	00:00:01
4	TABLE ACCESS FULL	UCET	6	198	3 (0)	00:00:01
5	TABLE ACCESS FULL	KARTA	9	117	3 (0)	00:00:01

Obrázek 3: Tabulka výsledků původního plánu



Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		9	414	4 (25)	00:00:01
* 1	FILTER					
2	SORT GROUP BY		9	414	4 (25)	00:00:01
3	NESTED LOOPS		9	414	3 (0)	00:00:01
4	TABLE ACCESS FULL	UCET	6	198	3 (0)	00:00:01
* 5	INDEX RANGE SCAN	KARTA_UCET	2	26	0 (0)	00:00:01

Obrázek 4: Tabulka výsledků optimalizovaného plánu

Z uvedených tabulek lze zjistit, že optimalizovanou metodou byl spotřebován přibližně poloviční čas procesoru, než tomu bylo v původním případě.

3.4 Definice přístupových práv

Ve skriptu jsou přístupové práva pro druhého členu uděleny na všechny tabulky, procedury a materialozvané pohledy. Toto udělení bylo potřebné provést, jelikož materializovaný pohled má patřit druhému členovi z týmu.

3.5 Materializovaný pohled

Materializovaný pohled obsahuje souhrn informací o jednotlivých účtech a počet z nich provedených operací. Tento pohled načítá data z tabulek patřících druhému členovi týmu. Pro demonstraci jsme v našem skriptu vytvořili dva dotazy na výpis všech dat z toho pohledu. První výpis se provede před úpravou dat v tabulkách, poté jsou tři řádky z tabulky smazány, pohled je zavolán znovu pro ukázku, že se žadné z dat v něm nezměnily, to je způsobeno tím, že materializovaný pohled vytváří otisk původních tabulek.