Databázové Systémy

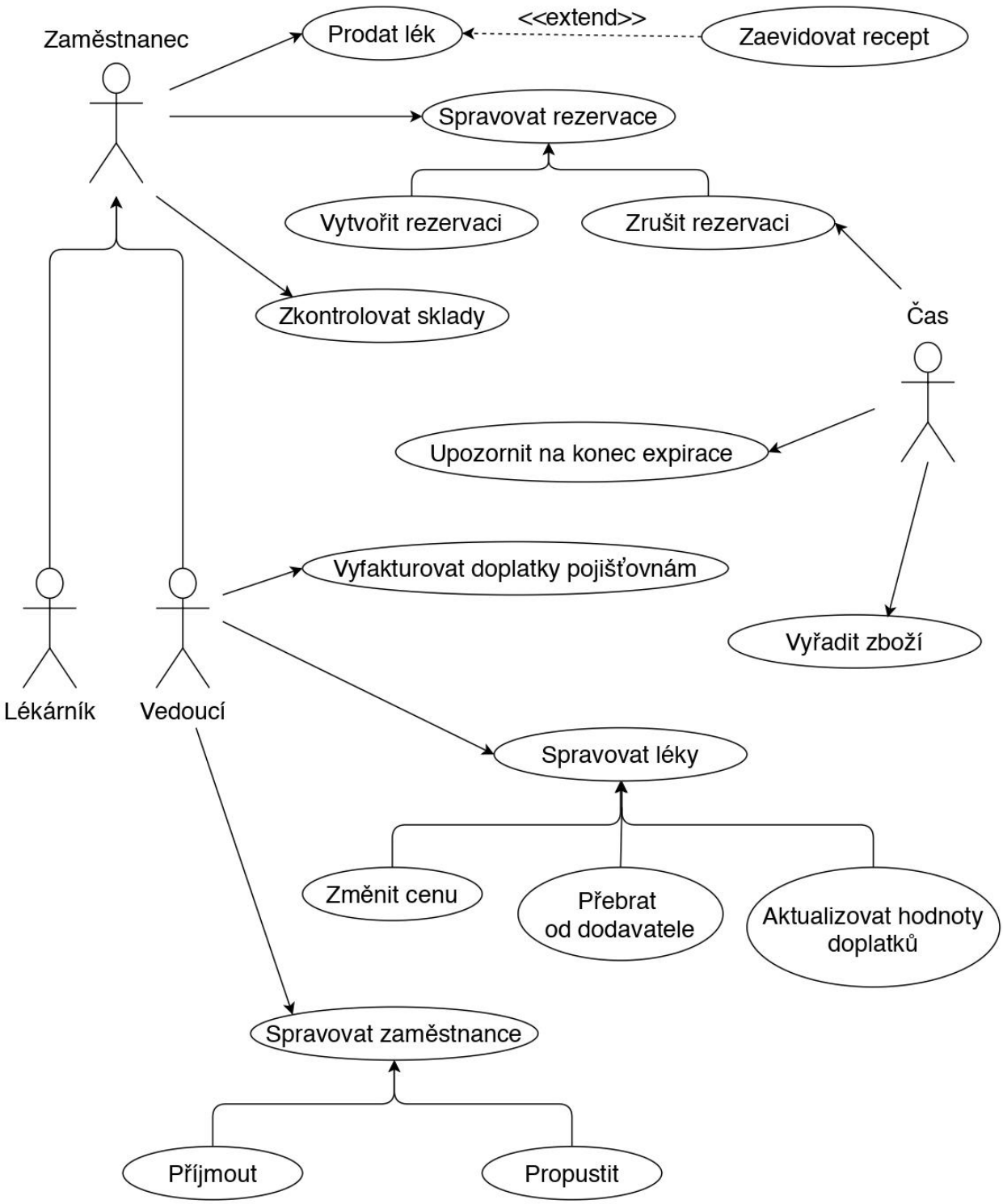
Lékárna

Michal Šedý (xsedym02)
Ondřej Pavlacký (xpavla15)

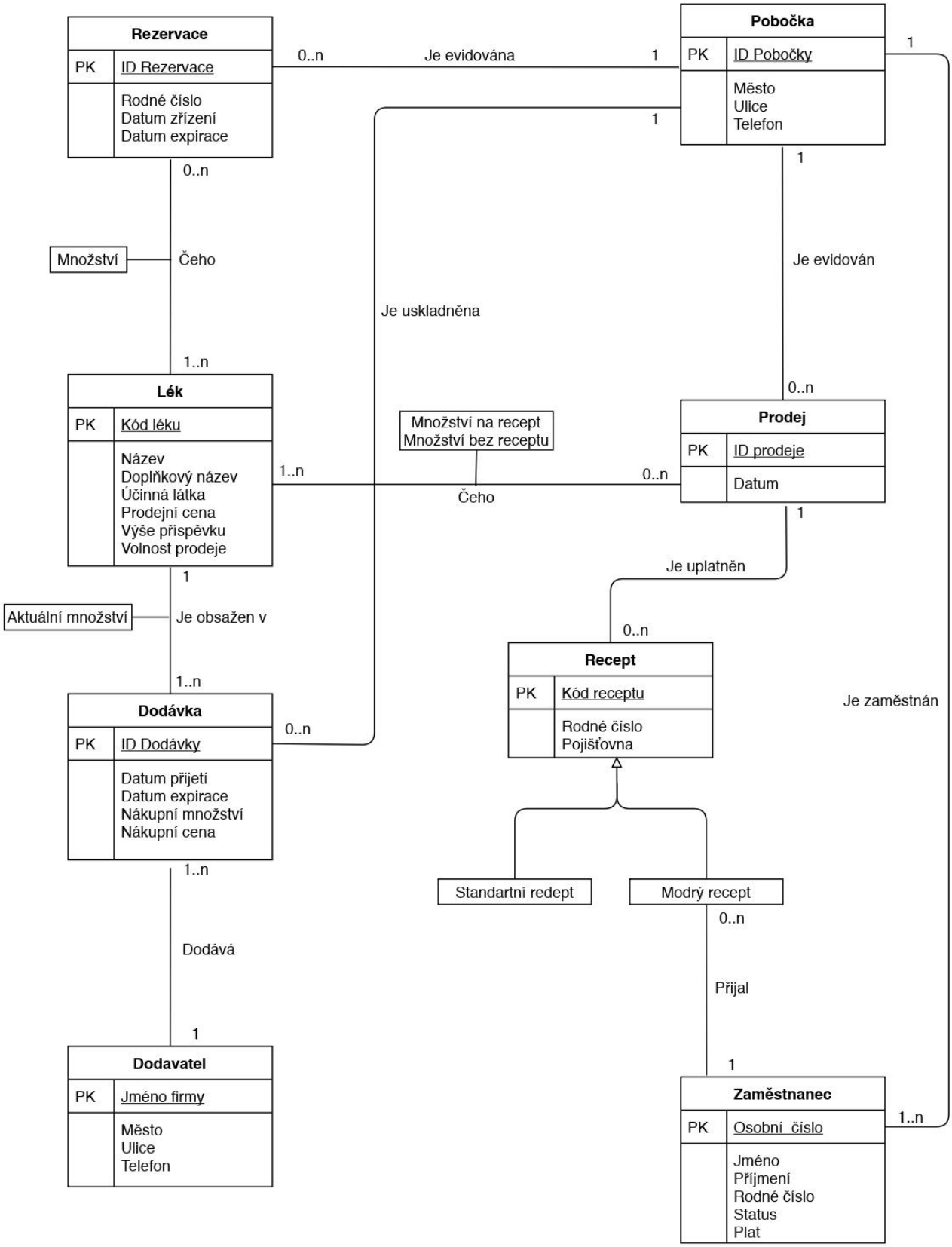
Zadání

Vaším úkolem je vývoj informačního systému lékárny. Lékárna vydává občanům léky jak na předpis, tak i bez předpisu. U léků vydávaných na předpis může část ceny hradit zdravotní pojišťovna. Některé léky se vydávají pouze na předpis. U léků na předpis musí systém evidovat informaci o rodném čísle zákazníka a pojišťovně, které bude částka za daný lék fakturována. Systém musí umožnit evidenci vydávaných léků, import aktuálních hodnot příspěvků na léky od zdravotních pojišťoven (může se čas od času měnit) a musí evidovat množství zboží na skladě. Léky jsou identifikovány číselným kódem či názvem. Firma má několik poboček, informační systém tedy musí být schopen zjistit množství zboží na skladě na zvolené pobočce, stejně tak se eviduje i prodané zboží vzhledem k dané pobočce. Požaduje-li zákazník zboží, které momentálně není na dané pobočce k dispozici, může si jej zarezervovat. Náš sortiment neobsahuje zboží, u kterého bychom si potřebovali pamatovat, kdo si jej koupil, u prodaného zboží si tedy nepamatujeme detaily o zákazníkovi, stejně tak v případě rezervací poslouží pouze jméno zákazníka. Informační systém bude obsluhován prozatím pouze zaměstnanci lékárny. Doplatky pojišťoven na jeden lék se různí, stejně tak jako cena léku prodaného bez předpisu. Evidujte i dodavatele lékárny, jakož i jejich ceny pro dané zboží. Objednávky u těchto dodavatelů již modelovat nemusíte. Předpokládejte, že není třeba evidovat zaměstnance, je irelevantní, kdo prodej zboží realizoval.

Use Case diagram

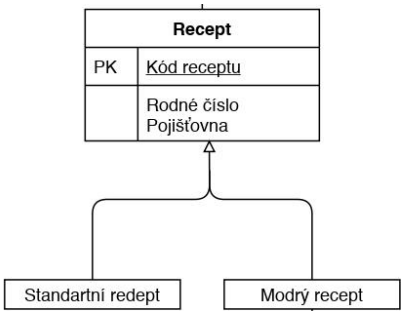


Entity-Relationship diagram



Generalizace/Specializace

V řešení jsme využili vztah generalizace/specializace k vytvoření dvou typů receptu se stejnými parametry, ale jinými vlastnostmi. U modrého receptu si na rozdíl od standartního potřebujeme uchovat také informaci o pracovníkovi, který na daný recept vydal léky.



Protože informace o receptu jsou pro pobočku stejně hodnotné a informace o pracovníkovi, který recept přijal se využije jen zřídka, byl pro uchování obou receptů zvolen pouze jeden řádek tabulky, se sloupcem pracovníka, který vydal na recept zboží. V případě standartního receptu, a nepotřebnosti si pamatovat prodejce bude pole obsahovat hodnotu NULL.

Trigger

Ve skriptu se nacházejí celkem tři triggery, z toho dva pro generování hodnot. Všechny se spouštějí před vložením hodnoty do tabulky (`BEFORE INSERT OR UPDATE`).

Trigger využívající sekvenci slouží k přidělení osobního čísla zaměstnanci v případě, že mu při nahrávání do tabulky nebude žádné přiděleno. Zaměstnanci s vygenerovaným číslem by měli mít hodnoty těchto osobních čísel vzestupné podle pořadí vkládání.

Další trigger přiděluje jednotlivým zaměstnancům platové ohodnocení v závislosti na jejich pozici v lékárně. Obyčejný řadový lékárník dostane přiděleno 27 tisíc a vedoucí provozovny 39 tisíc, pokud není plat stanoven explicitně.

Poslední trigger koriguje ceny léků a příspěvků pojišťoven. Zajišťuje, aby nebyl na lék větší příspěvek, než je jeho prodejní cena. Pokud takový případ nastane, pak automaticky zvýší cenu léku na hodnotu příspěvku. Zákazník bude mít stejně lék stoprocentně hrazen pojišťovnou a lékárna si více vydělá.

Procedury

K manipulaci s daty byly vytvořeny tři procedury. Dvě z nich obsahují výjimky (`EXCEPTION`), proměnné s datovým typem odkazujícím na sloupec tabulky (`table_name.column_name%TYPE`) a `CURSOR`.

První procedura `povyseni(prjm IN VARCHAR)` slouží k povýšení zaměstnance s příjmením definovaným proměnnou `prjm` ze stavu lékárníka do stavu vedoucího a s tím související i zvýšení platu o sedm tisíc. Pokud již je zaměstnanec vedoucím, pak dostane dva tisíce přidáno. Procedura si vytvoří dvě pomocné proměnné `stat` k určení pozice zaměstnance v lékárně a `plt` uchovávající jeho dosavadní platové ohodnocení. Pokud je zaměstnanec vedoucím, aktualizuje se v tabulce zaměstnanců na řádku odpovídajícímu příjmení zaměstnance sloupec platu, a to navýšením o sedm tisíc a také se aktualizuje status (pozice) na vedoucího. V opačném případě se aktualizuje pouze plat navýšením o dva tisíce.

Druhá procedura `propustit(os_c IN INT)` provede propuštění zaměstnance specifikovaného osobním číslem (`os_c`), používá se osobní číslo a ne příjmení, aby nemohlo dojít k záměně. Pokud je propouštěným řádový lékárník, pouze odstraní z tabulky zaměstnanců. Pokud je ale propouštěným vedoucí, pak je potřeba najít za něj náhradu. Jeho funkci převezme služebně nejstarší zaměstnanec (s nejnižším osobním číslem) na pobočce. Takovýto zaměstnanec bude do funkce vedoucího povýšen již dříve uvedenou procedurou `povyseni`. Pokud po propuštění, již na pobočce není žádný zaměstnanec bude na konzoli klienta vypsáno varování o nemožnosti nahradit propouštěného vedoucího. V případě propouštění vedoucího se využije kurzor, ze kterého se v cyklu čerpají data o zaměstnancích ovlivněné pobočky a hledá se zaměstnanec s nejnižším osobním číslem. Pokud nebude nalezen žádný náhradník, vyvolá následující příkaz `SELECT` výjimku `NO_DATA_FOUND`, která detekuje prázdnost pobočky.

Poslední procedura `najdi_lek(jmeno IN VARCHAR)` vyhledá na všech pobočkách lék daného jména a vypíše jeho počet na konkrétní pobočce. K prohledávání dodávek, které obsahují informace o pobočce, na které je dodávka léku uskladněna a skladové množství, je využit kurzor `dodavka_na_pobocce`, ze kterého jsou v cyklu postupně získávána a prohledávána data o dotyčném léku.

Explain plán

Explain plán umožňuje zobrazit postup zpracování dotazu, a v závislosti na těchto informacích upravit způsob požadavku, nebo přidat indexaci tak, aby se jeho provádění optimalizovalo.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		17	1496	7 (15)	00:00:01
1	HASH GROUP BY		17	1496	7 (15)	00:00:01
*2	HASH JOIN		17	1496	6 (0)	00:00:01
*3	TABLE ACCESS FULL	lek	6	318	3 (0)	00:00:01
4	TABLE ACCESS FULL	dodavka	26	910	3 (0)	00:00:01

Obrázek 1: EXPLAIN PLAN před vytvořením indexu.

Před přidáním indexu sloupci tabulky, ovlivněného příkazem `WHERE`, byl pořízen výsledek explain plánu. Celkové časy provádění jednotlivých operací jsou zanedbatelné, protože byly použity tabulky s relativně malým množstvím dat.

V sekci operací můžeme vidět jaké kroky byly prováděny k zodpovězení dotazu na tabulky. `SELECT STATEMENT` informuje, že byl vykonán dotaz `SELECT`. `HASH GROUP BY` shluknul položky podle hashovacího klíče. `HASH JOIN` pomocí hashovacích klíčů propojuje prvky dvou tabulek. `TABEL ACCESS FULL` projde celou tabulku, jak řádky, tak sloupce.

Protože zkoumaný dotaz spojoval dvě tabulky `lek` a `dodavka`, a snažil se odpovědět na otázku: Kolik volně prodejných léků se stejnou účinnou látkou se nachází, na jaké pobočce? Bylo rozhodnuto přidat index vlastnosti léku, podle které se léky filtrovaly. Tou byla možnost volného prodeje.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		17	1496	6 (17)	00:00:01
1	HASH GROUP BY		17	1496	6 (17)	00:00:01
* 2	HASH JOIN		17	1496	5 (0)	00:00:01
3	TABLE ACCESS BY INDEX ROWID BATCHED	lek	6	318	2 (0)	00:00:01
* 4	INDEX RANGE SCAN	volnost	6		1 (0)	00:00:01
5	TABLE ACCESS FULL	dodavka	26	910	3 (0)	00:00:01

Obrázek 2: EXPLAIN PLAN po přidání INDEXU.

Oproti předcházejícímu výsledku explain plánu, se zde vyskytly dvě nové operace. `TABLE ACCESS BY INDEX ROWID BATCHED` přistoupí přímo k reprezentaci daného řádku za pomocí indexu v batched modu, který byl přidán do Oracle 12c a optimalizuje indexování. `INDEX RANGE SCAN` získá hodnoty z daného intervalu v příslušném sloupci (`volnost` – možnost volného prodeje).

Druhý indexem pozměněný explain plán obsahuje levnější operace a celkové v součtu se jedná také o trochu nižší cenu výsledného dotazu. `INDEF RANGE SCAN` umožňuje přímo vypustit ty léky, které jsou na předpis (ptáme se na léky bez předpisu).

Přidělení práv

Druhému členovi týmu (xpavla15) byla pomocí příkazu *`GRAND ALL`* přidělena veškerá práva ke všem tabulkám, sekvencím a jedné proceduře (`najdi_lek`).

Materializovaný pohled

Materializované pohledy umožňují optimalizaci přístupu k datům, protože data jsou fyzicky uložena v databázi. Tato data je možné aktualizovat. Pro materializovaný pohled byl zvolen příkaz `REFRESH ON COMMIT`, který pro aktualizaci v tabulkách a provedení `COMMIT` aktualizuje i pohled. Pokud by `COMMIT` nebyl proveden, v materializovaném pohledu by zůstala staré neaktuální informace (viz. ukázka ve skriptu).