Databáze lékárny

Implementační dokumentace projektu Kurz databázové systémy (IDS)

Autoři:
Jan Rybka (xrybka05)
Matěj Konopík (xkonop03)

Datum:
30. dubna 2022

Obsah

Úvod

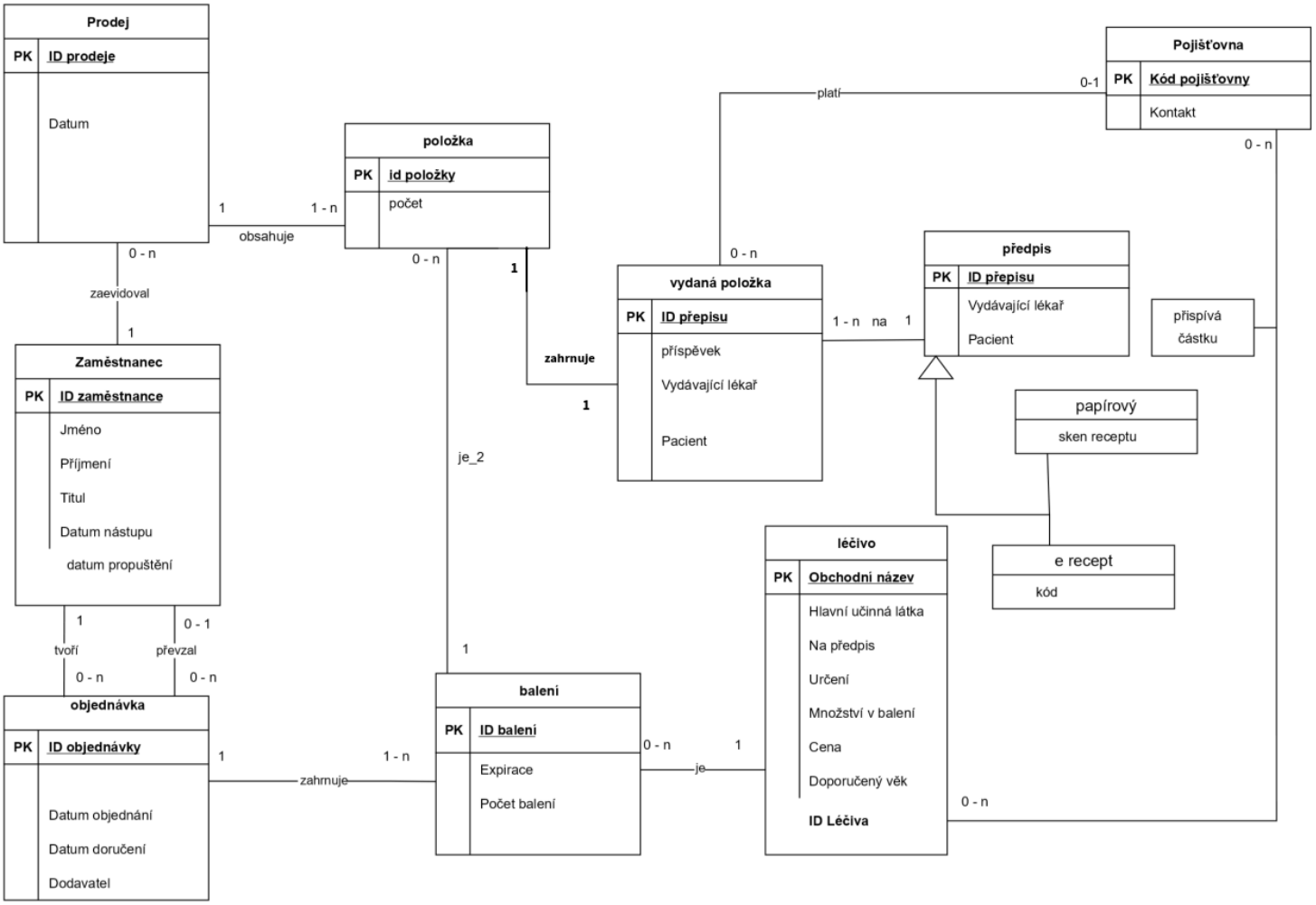
Naším cílem v tomto projektu bylo navrhnout a vytvořit databázi pro informační systém lékárny s využitím jazyka SQL a školního databázového serveru Oracle. Konkrétně tedy byla v souvislosti s užitým serverem uplatněna variace PL/SQL jazyka SQL. Práci jsme mezi sebe dělili intuitivně a podle individuálních možností. Pro sdílení kódu a verzování jsme využili Github.
Samotný proces práce byl dle zadání dělen na čtyři části: návrh, vytvoření tabulek a vzorových dat, dotazy select a naposledy pokročilé dotazy. Jednotlivé vývojové cykly jsou popsány níže v této dokumentaci.

Znění zadání (č. 33)

Vaším úkolem je vývoj IS lékárny. Lékárna vydává občanům léky jak na předpis, tak za hotové. U léků vydávaných na předpis může část ceny hradit zdravotní pojišťovna. Některé léky se vydávají pouze na předpis. Systém musí umožnit evidenci vydávaných léků, import příspěvků na léky od zdravotních pojišťoven (může se čas od času měnit), musí poskytovat export výkazů pro zdravotní pojišťovny a musí mít vazbu na skladové zásoby (vidět, zda požadovaný lék je na skladě). Léky jsou identifikovány číselným kódem či názvem. [1]

Část 1: Use case a ER diagram

Obrázek 1: Use case diagram



Obrázek 2: ER diagram

Popis entit ER Diagramu

- Entitní množina léčivo reprezentuje rozdílné druhy léků. Její instancí by mohl například být lék paralen.
- Entitní množina balení reprezentuje skupiny balení léku na skladu či objednaná, prodaná nebo
- vyhozená. Příkladem by mohlo být deset balení paralenu, které projde 1. března.
- Entitní množina objednávka reprezentuje smluvenou dodávku léků od dodavatele.
- Entitní množina zaměstnanec reprezentuje zaměstnance.
- Entitní množina prodej reprezentuje jedno předání léků návštěvníkům lékárny. Instanci této množiny si lze představit jako účtenku.
- Entitní množina položka reprezentuje jednu položku nákupu návštěvníka lékárny. Na pomyslné účtence by tato entita položkou.
- Entitní množina vydaná položka položka reprezentuje jednu položku výdeje léků na předpis návštěvníkovi.
- Entitní množina předpis reprezentuje předpis, a to buď papírový nebo e-recept.
- Pojišťovna reprezentuje pojišťovnu.
- Vztahová množina platí reprezentuje, která pojišťovna přispěla na který recept.
- Vztahová množina přispívá reprezentuje jakou částku bude pojišťovna přispívat na výdej na předpis konkrétního druhu léku.
- Vztahová množina tvoří reprezentuje informaci, který zaměstnanec vytvořil, kterou objednávku.

Část 2: Převod diagramu na schéma tabulek a jejich definice

Tato část byla relativně jednoduchá, hlavně díky dobrému návrhu, jenž neobsahoval žádný vztah *m* ku *n*, který by vyžadoval tvorbu vazební tabulky. Všechny tabulky tedy byly převedeny dle diagramu se zavedením vhodných primárních a cizích klíčů. Pro generování číselných primárních klíčů jsme využili následující konstrukci jazyka SQL, která automaticky vytvoří nekolizních hodnoty:

```
<název_sloupce> INT GENERATED AS IDENTITY NOT NULL PRIMARY KEY
```

Specializaci entity receptu reprezentujeme vytvořením sloupce v tabulce přepis, ze níž je tato entita odvozena. Tento sloupec, typ, rozlišuje mezi druhem tohoto předpisu. Také byly přidány korespondující sloupce obou typů receptů, tyto mohou zůstat null, jedná-li se o druhý typ receptu. Tento koncept byl převzat z přednášek. Zároveň jsme implementovali i kontroly pro tyto hodnoty.

Část 3: Dotazy select

Třetí fáze opět proběhla relativně bez problémů, bylo jen potřeba vytvořit smysluplné selecty v kontextu naší databáza a zároveň dodržet podmínky ze zadání. Konečné selecty slouží následujícím účelům:

1. Vyhodnocení tržby za určitý den, včetně složek - přímý zisk z prodeje a dotace pojišťoven.
2. Zobrazení léčiv, jež byly objednány z velkoobchodu ale dosud nebyly doručeny
3. Které pojišťovny přispívají na určité léčivo a kolik
4. Jaká je průměrná částka příspěvku na dané léčivo
5. Jaká volná dostupná léčiva se prodaly během daného dne a kolik balení to bylo
6. Vyhodnocení statistické souvislosti mezi typem receptu (volný prodej/na recept) s příspěvky pojišťoven
7. Odkud bylo objednáno dané balení léčiva pro případ reklamace

Konkrétní plnění zadaných podmínek pro tyto dotazy je uvedeno přímo v kódu.

Část 4: Pokročilé objekty databáze

Triggery

První trigger se týká zadané podmínky, tedy automatické generování hodnoty primárního klíče, zde pro tabulku `PRODEJ`. Vytvořili jsme sekvenci pro možnost využití metody nextval, které samotnou hodnotu generuje. Samotný trigger je nastavený na spuštění před vkládáním do této tabulky, kde je primární klíč vkládaného řádku NULL. V takovém případě je tomuto řádku přiřazena generovaná hodnota.
Druhý trigger je lehce složitější a slouží pro kontrolu příspěvku pojišťovny, který nesmí být z logiky věci vyšší než cena dotovaného léčiva. Využívá se vazební tabulky `PŘISPÍVÁ` a proměnné `limit`. Do této proměnné je selectem nahrána cena léčiva a ta je poté porovnána s hodnotou příspěvku pojišťovny. V případě že by příspěvek byl vyšší je provolána aplikační chyba s kódem -20201 a je vypsáno chybové hlášení. Kód chyby je zvolen v povoleném uživatelském intervalu -20000 až -20999. Dále je implementováno odchytávání výjimek `NO_DATA_FOUND` a `OTHER`. První se zavolá pokud by došlo k chybě při selectu, jenž naplňuje proměnnou limit, druhá v případě jiné chyby.

`NO_DATA_FOUND` by při konzistenci dat v databázi nemělo být nikdy provoláno, a může zde právě na nekonzistenci dat poukazovat.

Procedury

První procedura využívá kurzor k nalezení všech prodaných balení léků, která jsou po datu spotřeby. Následně vypíše za pomoci `DBMS_OUTPUT.PUT_LINE` identifikátory všech položek, jejichž balení jsou po datu spotřeby. V rámci tohoto zpracovávání dat je využita proměnná s datovým typem odkazujícím na sloupec tabulky.
Druhá procedura prezentuje vlastnosti vyjímek v jazyce PL/SQL.

Index a explain plan

Explain plan a optimalizační indexace úzce souvisí, uvádíme je tedy v jedné kapitole. Pro optimalizaci jsme vybrali dotaz select na výpočet statistik prodejů během určitého dne. Optimalizace spočívá v indexaci sloupce `DATUM`, který je použit pro vyhledávání tohoto dne. Celková výpočetní cena se tak sníží o z původních 42 bodů na 37. Níže jsou snímky před a po přidání indexu:

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                       |     1 |   640 |     6   (0)| 00:00:01 |
|   1 |  SORT GROUP BY NOSORT          |                       |     1 |   640 |     6   (0)| 00:00:01 |
|   2 |   NESTED LOOPS                 |                       |     1 |   640 |     6   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                |                       |     1 |   640 |     6   (0)| 00:00:01 |
|   4 |     NESTED LOOPS               |                       |     1 |   618 |     5   (0)| 00:00:01 |
|   5 |      NESTED LOOPS              |                       |     1 |   348 |     4   (0)| 00:00:01 |
|   6 |       NESTED LOOPS             |                       |     1 |    78 |     3   (0)| 00:00:01 |
|   7 |        TABLE ACCESS FULL       | POLOZKA               |     1 |    52 |     3   (0)| 00:00:01 |
|   8 |        TABLE ACCESS BY INDEX ROWID| VYDANA_POLOZKA        |     1 |    26 |     0   (0)| 00:00:01 |
|*  9 |         INDEX UNIQUE SCAN      | VYDANA_POLOZKA_POINTER|     1 |       |     0   (0)| 00:00:01 |
|  10 |       TABLE ACCESS BY INDEX ROWID | BALENI                |     1 |   270 |     1   (0)| 00:00:01 |
|* 11 |        INDEX UNIQUE SCAN       | SYS_C002035635        |     1 |       |     0   (0)| 00:00:01 |
|  12 |      TABLE ACCESS BY INDEX ROWID  | LECIVO                |     1 |   270 |     1   (0)| 00:00:01 |
|* 13 |       INDEX UNIQUE SCAN        | SYS_C002035645        |     1 |       |     0   (0)| 00:00:01 |
|* 14 |     INDEX UNIQUE SCAN          | SYS_C002035613        |     1 |       |     0   (0)| 00:00:01 |
|* 15 |    TABLE ACCESS BY INDEX ROWID | PRODEJ                |     1 |    22 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name                  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                       |     1 |   652 |     5   (0)| 00:00:01 |
|   1 |  SORT GROUP BY NOSORT          |                       |     1 |   652 |     5   (0)| 00:00:01 |
|   2 |   NESTED LOOPS                 |                       |     1 |   652 |     5   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                |                       |     1 |   652 |     5   (0)| 00:00:01 |
|   4 |     NESTED LOOPS               |                       |     1 |   382 |     4   (0)| 00:00:01 |
|   5 |      NESTED LOOPS              |                       |     1 |   112 |     3   (0)| 00:00:01 |
|   6 |       MERGE JOIN CARTESIAN     |                       |     1 |    60 |     2   (0)| 00:00:01 |
|   7 |        TABLE ACCESS BY INDEX ROWID | VYDANA_POLOZKA        |     1 |    26 |     1   (0)| 00:00:01 |
|   8 |         INDEX FULL SCAN        | VYDANA_POLOZKA_POINTER|     1 |       |     1   (0)| 00:00:01 |
|   9 |        BUFFER SORT             |                       |     1 |    34 |     1   (0)| 00:00:01 |
|  10 |         VIEW                   | VW_GBF_6              |     1 |    34 |     1   (0)| 00:00:01 |
|  11 |          TABLE ACCESS BY INDEX ROWID| PRODEJ                |     1 |    22 |     1   (0)| 00:00:01 |
|* 12 |           INDEX UNIQUE SCAN    | INDEX_DATUM_SORT      |     1 |       |     0   (0)| 00:00:01 |
|* 13 |       TABLE ACCESS BY INDEX ROWID | POLOZKA               |     1 |    52 |     1   (0)| 00:00:01 |
|* 14 |        INDEX UNIQUE SCAN       | SYS_C002035626        |     1 |       |     0   (0)| 00:00:01 |
|  15 |      TABLE ACCESS BY INDEX ROWID  | BALENI                |     1 |   270 |     1   (0)| 00:00:01 |
|* 16 |       INDEX UNIQUE SCAN        | SYS_C002035635        |     1 |       |     0   (0)| 00:00:01 |
|* 17 |     INDEX UNIQUE SCAN          | SYS_C002035645        |     1 |       |     0   (0)| 00:00:01 |
|  18 |    TABLE ACCESS BY INDEX ROWID | LECIVO                |     1 |   270 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

Přístupová práva

Dalším zadáním bylo přidat přístupová práva na databázové schéma prvního studenta druhému studentovi. Toto se realizovalo jednoduše pomocí následující klauzule aplikovatelné na různé objekty databáze:

```
GRANT ALL ON <objekt> TO <user>;
```

Jelikož jsme v tomto případě oba rovnocenní vývojáři/administrátoři tohoto systému, dává smysl přidat tomu druhému všechna práva. Pokud bychom ale chtěli přidat uživatel, jenž by mohl například pouze číst z tabulek, stačilo by klauzuli změnit následujícím způsobem:

```
GRANT SELECT ON <tabulka> TO <user>;
```

V reálném prostředí lékárny by se toto dalo využít na tvorbu přístupu pro lékárníka, který by mohl z některých tabulek data pouze číst. Možností je ovšem více.

Materializovaný pohled

Materializovaný pohled z tabulek prvního člena týmu (xkonop03) pro druhého (xrybka05) byl poslední úlohou této sekce. Rozhodli jsme se pro vytvoření materializovaného pohledu s automatickou aktualizací, přesněji `REFRESH FAST ON COMMIT`. Tato možnost způsobí, že se lokálně ukládaná tabulka materializovaného pohledu obnoví pokaždé, kdy se spustí klauzule `COMMIT`. Pro správnou funkčnost je také třeba definovat log, jenž musí obsahovat primární klíč tabulky (popřípadě tabulek) a `ROWID`. Ten u nás nabývá následující podoby:

```
CREATE MATERIALIZED VIEW LOG ON XKONOP03.BALENI WITH PRIMARY KEY, ROWID;
```

Samotná tabulka pohledu je poté generována jednoduchým selectem, jenž z tabulky `BALENI` zobrazuje informace o skladových zásobách.
Dále ve skriptu demonstrujeme obecné vlastnosti materializovaného pohledu. Jedná se o samostatnou tabulku, která je uložena stranou od databázových tabulek, z nichž čerpá, a slouží pro urychlení vyhledávání a ke snížení čtení přímo z databáze v případech, kdy nepotřebujeme detailně znát všechny aktuální prvky tabulky. Pokud chceme, aby se tento pohled aktualizoval, je třeba to provést ručně, popřípadě nastavit automatickou časovanou aktualizaci. Pokud tedy přidáme do zdrojové tabulky databáze řádek a zobrazíme tuto tabulku pomocí přímého selectu a poté selectu z materializovaného pohledu, zjistíme,
že materializovaný pohled nebyl aktualizován a tento vložený řádek v něm narozdíl databázové tabulky chybí. Po manuálním spuštění klauzule `COMMIT` se spustí aktualizace změn z logu a materializovaný pohled poté při selectu již nová data zobrazí. Demonstrace je ilustrována ve snímcích z běhu skriptu na další straně.

```
606  --výběr z material view po přidání (není aktualizovaný)
607  SELECT Obchodni_nazev, Pocet_baleni
608  FROM materialized_view_baleni;
```

1 row

	OBCHODNI_NAZEV	POCET_BALENI
1	Osino	5

Obrázek 3: select z materializovaného pohledu po změně tabulky

```
610  --pro porovnání výběr přímo z tabulky, jenž aktualizovana je
611  SELECT Obchodni_nazev, Pocet_baleni
612  FROM BALENI;
```

2 rows

	OBCHODNI_NAZEV	POCET_BALENI
1	Osino	5
2	Ospen	20

Obrázek 4: select z tabulky databáze po její změně

```
615  --commit dle nastavení mat. pohledu aktualizuje data
616  COMMIT;
617  -- nyní jsou v lokálním mat. pohledu jiz data aktualizovane
618  SELECT Obchodni_nazev, Pocet_baleni
619  FROM materialized_view_baleni;
```

2 rows

	OBCHODNI_NAZEV	POCET_BALENI
1	Osino	5
2	Ospen	20

Obrázek 5: select z materializovaného pohledu po aktualizace skrze

Závěr

Celý vývoj probíhal bez výrazných obtíží a z praktického hlediska nám osvětlil nejen práci s jazykem SQL a samotným databázovým systémem, ale také korektního návrhu.

Jediná věc, jež byla při vývoji nepříjemná, byly časté chybové hlášky ohledně překročení limitu sessions, které se nedaly korektně odstranit v Oracle Developer rozšíření pro Visual Studio Code. Pomohl přechod na IDE DataGrip, kde vše fungovalo perfektně.