VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Projektová dokumentácia k predmetu IDS

Zadanie č. 33 - Lékárna

23. apríla 2020

Dávid Oravec (xorave05)
Dominik Boboš (xbobos00)

1 Zadanie

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.

2 Implementácia

2.1 Úvod

Naša implementácia začala vytvorením tabuliek, ktoré odzrkadlovali návrh v podobe ER diagramu. Nasledovalo určenie primárnych kľúčov tabuliek, cudzích kľúčov a vytváranie tabuliek, ktoré zobrazovali vzťahy medzi entitami v ER diagrame. Po vytvorení tabuliek sme ich naplnili ukážkovými dátami. Dáta o lekárňach a liekoch sme čerpali zo stránie lekární.

Ďalšou fázou bolo vytvorenie SELECT dotazov. Uvedieme len niekoľko príkladov, ostatné dotazy sú popísané v kóde.

2.1.1 Spojenie 2 tabuliek príkazom SELECT

```
SELECT *
   FROM LEKAREN NATURAL JOIN ZDRAVOTNA_POISTOVNA
   WHERE NAZOV = 'UNION';
```

Zobrazí všetky lekárne, ktoré majú zmluvu so zdravotnou poistovňou UNION.

```
SELECT DISTINCT ID_LIEK, VYSKA_DOPLATKU
   FROM ZDRAVOTNA_POISTOVNA NATURAL JOIN HRADI_DOPLATOK
   WHERE POISTOVNA = 1;
```

Zobrazí lieky a výšky doplatkov na liek, ktoré hradí poistovňa VŠZP.

2.1.2 Spojenie 3 tabuliek príkazom SELECT

```
SELECT DISTINCT ID_SKLAD, ID_LIEK, POCET_LIEKOV
   FROM SKLAD NATURAL JOIN POCET_LIEKOV_NA_SKLADE, NA_PREDPIS
   WHERE NA_PREDPIS.CISLO_LIEKU = POCET_LIEKOV_NA_SKLADE.ID_LIEK AND
      POCET_LIEKOV_NA_SKLADE.POCET_LIEKOV < 50 ;
```

Zobrazí sklady, ktoré majú počet na sklade nejakého lieku na predpis pod 50

2.2 Generalizácia

Pri vytváraní vzťahu generalizácie sme vytvorili tabuľku nadtypu + tabuľky pre podtypy s primárnym kľúčom nadtypu. Vznikli nám tabuľky `BEZ_PREDPISU` a `NA_PREDPIS`, ktoré dedia všetky atribúty z nadtypovej tabuľky `LIEK`. Vybrali sme si tento typ transformácie z toho dôvodu, že LIEK môže byť buď na predpis, bez predpisu alebo sa môžu tieto typy liekov prekrývať.

2.3 Triggery

V skripte sú implementované dva typy triggerov.
Prvý typ má za úlohu automaticky vygenerovať primárny kľuč `ID_POISTOVNE` v tabuľke `ZDRAVOTNA_POISTOVNA`.
Druhý typ zabezpečuje, že pri vložení položky do tabuľky `LIEK` sa vyvolá chyba, pokiaľ bol vložený liek, ktorý bol po dátume expirácie.

2.4 EXPLAIN PLAN a použitie INDEX

Úlohou `EXPLAIN PLAN` je zobrazenie postupnosti realizácie operácií optimalizátorom. Poskytuje aj informácie o výkonnostnej cene každej operácie a čas za ktorý sa vykonala.

2.4.1 Naša implementácia príkazu EXPLAIN PLAN

```
SELECT NAKUP.DATUM_PREDAJA, NAKUP.LEKAREN, SUM(LIEK.CENA) AS celkova_cena
FROM SUCAST_NAKUPU, NAKUP, NA_PREDPIS, LIEK
WHERE NA_PREDPIS.CISLO_LIEKU = SUCAST_NAKUPU.LIEK_ID AND
    NA_PREDPIS.CISLO_LIEKU = LIEK.CISLO_LIEKU
GROUP BY NAKUP.DATUM_PREDAJA, NAKUP.LEKAREN;
```

Úlohou tejto implementácie je vypísať všetky také nákupy, v ktorých sa nakúpili lieky na predpis, zobrazí dátum nákupu, v ktorej lekárni sa nákup uskutočnil a sumu všetkých zakúpených liekov na predpis. Takéto informácie sú vhodné napríklad pre zdravotnú poisťovňu.

Po zavolaní príkazu EXPLAIN PLAN a vypísaní na výstup, je možné vidieť tabuľku s postupnosťou vykonávaných operácií a taktiež ich výkonnostnú cenu.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		10	740	9 (12)	00:00:01
1	HASH GROUP BY		10	740	9 (12)	00:00:01
* 2	HASH JOIN		10	740	8 (0)	00:00:01
3	MERGE JOIN CARTESIAN		6	366	5 (0)	00:00:01
4	NESTED LOOPS		2	78	1 (0)	00:00:01
5	NESTED LOOPS		2	78	1 (0)	00:00:01
6	INDEX FULL SCAN	PK_NA_PREDPIS	2	26	1 (0)	00:00:01
* 7	INDEX UNIQUE SCAN	PK_LIEK	1		0 (0)	00:00:01
8	TABLE ACCESS BY INDEX ROWID	LIEK	1	26	0 (0)	00:00:01
9	BUFFER SORT		3	66	5 (0)	00:00:01
10	TABLE ACCESS FULL	NAKUP	3	66	2 (0)	00:00:01
11	TABLE ACCESS FULL	SUCAST_NAKUPU	5	65	3 (0)	00:00:01

Obr. 1: EXPLAIN PLAN bez použitia indexov

2.4.2 INDEX

K urýchleniu prevedenia príkazu bol použitý príkaz INDEX nakoľko indexovanie môže byť užitočné v prípade častého vyhľadávania v určitej tabuľke. V našej implementácií často používame tabuľky `LIEK` a `SUCAST_NAKUPU`. Preto sme implementovali príkazy INDEX nasledovne:

```
CREATE INDEX sucast_index ON SUCAST_NAKUPU (ID_TABLE, NAKUP_ID, LIEK_ID);
CREATE INDEX nakup_index ON NAKUP (CISLO_POKLADNICNEHO_BLOKU, DATUM_PREDAJA,
    LEKAREN);
```

```
-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |    10 |   740 |     5  (20)| 00:00:01 |
|   1 |  HASH GROUP BY                |               |    10 |   740 |     5  (20)| 00:00:01 |
|   2 |   MERGE JOIN CARTESIAN        |               |    10 |   740 |     4   (0)| 00:00:01 |
|   3 |    NESTED LOOPS               |               |     3 |   156 |     1   (0)| 00:00:01 |
|   4 |     NESTED LOOPS              |               |     3 |   156 |     1   (0)| 00:00:01 |
|   5 |      NESTED LOOPS             |               |     3 |    78 |     1   (0)| 00:00:01 |
|   6 |       INDEX FULL SCAN         | SUCAST_INDEX  |     5 |    65 |     1   (0)| 00:00:01 |
|*  7 |       INDEX UNIQUE SCAN       | PK_NA_PREDPIS |     1 |    13 |     0   (0)| 00:00:01 |
|*  8 |      INDEX UNIQUE SCAN        | PK_LIEK       |     1 |       |     0   (0)| 00:00:01 |
|   9 |     TABLE ACCESS BY INDEX ROWID| LIEK          |     1 |    26 |     0   (0)| 00:00:01 |
|  10 |    BUFFER SORT                |               |     3 |    66 |     5  (20)| 00:00:01 |
|  11 |     INDEX FAST FULL SCAN      | NAKUP_INDEX   |     3 |    66 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Obr. 2: EXPLAIN PLAN s použitím indexov

Pri druhom spustení príkazov s EXPLAIN PLAN za použitia indexov je viditeľné, že operácie sa vykonali rýchlejšie s menšou výkonnostnou cenou. Taktiež miesto prechodu celou tabuľkou bolo zvolené skenovanie indexov. Táto zmena zabezpečila rýchlejšie/nenáročnejšie vykonanie väčšiny operácií v rámci príkazu.

2.5 Procedúry

Vytvorili sme dve procedúry:
`percenta_lieku_z_kapacity`
`volne_miesto_v_sklade`

Procedúra `percenta_lieku_z_kapacity(sklady_id_arg)` počíta koľko percent liekov je na sklade s ID v parametri *sklad_id_arg* z celkového počtu liekov nachádzajúcich sa vo všetkých skladoch. Procedúra teda prechádza všetkými skladmi v databáze a postupne počíta celkový počet liekov a počet ukladá do premennej *pocet_vsetkych_liekov* ak sa spracováva sklad s primárným kľúčom zhodným s *sklad_id_arg*, vtedy sa zároveň počet liekov ukladá do premennej *lieky_sklad*. Následne sa spočítajú percentá a daný výsledok sa s presnosťou na dve desatinné miesta vypíše na DBMS výstup. V prípade, že sklady sú prázdne, teda počet liekov je rovný 0, vyvolá sa výnimka.

Procedúra `volne_miesto_v_sklade` postupne zisťuje dostupnú kapacitu vo všetkých skladoch. Procedúra neprijíma nijaké parametre. Prehľadáva jednotlivé sklady a do premennej *dostupna_kapacita* zapisuje dostupnú kapacitu. Túto informáciu pre každý sklad potom vypíše na DBMS výstup. V prípade, že by nejaký sklad bol plne obsadený, teda má dostupnú kapacitu 0, procedúra sa skončí s chybou.

2.6 Materializovaný pohľad

Je to objekt, ktorý poskytuje dáta v rovnakej podobe ako tabuľka. Narozdiel od tabuľky, ktorá obsahuje priamo dáta, pohľad obsahuje len predpis akým spôsobom dáta získať z iných tabuliek alebo pohľadov. Materializovaný pohľad sa niekam ukladá a preto je dotaz rychlejší, čo neplatí o nematerializovanom pohľade.

Náš materializovaný pohľad vytvára tabuľku `mesta_lekarni`, ktorá vytvára dáta okamžite a obnovuje ich pri commite. Následne je povolený prepis obsahu pohľadu. Pri demonštrácii je najprv vypísaný pôvodný pohľad, ktorý zobrazuje mestá, v ktorých sa lekárne nachádzajú a ich počet v danom meste. Potom vložíme nové dáta pomocou INSERT a vykonáme commit. Pri následujúcom výpise pomocou príkazu SELECT v pohľade uvidíme aktualizované dáta.

2.7 Práva

Pomocou príkazu GRANT na tabuľky, potom následuje zoznam práv, ktoré povoľujeme (napr. INSERT, DELETE,...), ten sme zvolili ako ALL (všetky práva). Následujú názvy tabuliek, na ktoré chceme udeliť práva (opäť všetky tabuľky) a na koniec treba uviesť, komu sa tieto práva prideľujú, čo v našom prípade bol kolega *xbobos00*.

3 Záver

Pri vypracovaní projektu sme nemali žiadne závažné problémy v komunikácii, keďže už sme zohratá dvojica a prácu sme si rozdelili rovnomerne. Projekt sme mali vypracovaný relatívne včas, aby sme sa vyhli zbytočným stresom pred deadline a prípadným preťaženiam školského servera, ako to býva zvykom pár dní pred odovzdávaním projektov. Vývoj projektu prebiehal na školskom serveri v prostredí od JetBrains - Data Grip.

Jedinou nevýhodou respektíve „prekážkou“ bola súčasná situácia vo svete a obmedzenie kontaktnej formy štúdia. Nemohli sme niektoré nejasnosti konzultovať s pedagógmi a to spravilo prácu na projekte značne ťažšou.