Vysoké učení technické v Brně
Fakulta informačních technologií

Databázové systémy

Dokumentace k projektu - Veterinární klinika

24. srpna 2020

Martin Chládek (xchlad16)

Obsah

1 Zadání

Namodelujte jednoduchý informační systém veterinární kliniky, kde se střídá více lékařů veterinářů a sester. O sestrách a veterinářích informační systém uchovává osobní informace jako jméno, titul, adresa, číslo účtu, hodinová mzda, přičemž lékaři mají přístup k údajům sester, ale sestry k údajům lékařů nikoliv. Hlavním účelem informačního systému je správa informací o léčených zvířatech, jednotlivých léčbách a jejich majitelích. U každého zvířete, které podstoupilo alespoň jednu léčbu, je v systému vytvořen záznam (o léčbě i o majiteli zvířete). Majitele zvířete lze v systému vyhledat podle jména, příjmení či adresy. Jeden majitel může samozřejmě vlastnit více domácích mazlíčků, o kterých je vytvořen záznam s jejich jménem, o jaký druh zvířete se jedná (kočka, pes, křeček, atd.), datum narození, datum poslední prohlídky a informace o jednotlivých léčbách, které dané zvíře podstoupilo (diagnóza, datum zahájení léčby, stav, cena). Při léčbě mohou být naordinovány léky s určitým dávkováním a pro specifikovanou dobu podávání. U léků předpokládejte možnost dohledat typ léku, jeho účinnou látku, kontraindikace, pro jaký druh zvířete se hodí (kočka, pes, ...) a doporučené dávkování pro daný druh zvířete. Jeden typ léku se může hodit pro více druhů zvířat s různým dávkováním a pro léčbu různých nemocí (nemoc modelovat nemusíte, postačí pouze název). V případě podání léku přímo v ordinaci (např. injekce) je k dispozici informace, kdo daný lék podal, stejně tak u léčby pro konkrétní zvíře lze dohledat, který veterinář ji vypsal.

2 Generalizace/specializace

Náš návrh ER diagramu obsahoval vztah generalizace a zároveň i specializace. Tohoto vztahu jsme využili pro tabulku `Osoba` a tabulky `Veterinar`, `Sestra`, `Majitel`.

Pokud se podíváme ve směru generalizace, můžeme říci, že entitní množina `Osoba` je zobecněním množin `Veterinar`, `Sestra` a `Majitel`. V opačném směru můžeme říci, že tabulky `Veterinar`, `Sestra` a `Majitel` jsou speciálním případem Osoby.

Entitní množiny jsou v disjunktním vztahu, osoba tedy může být veterinářem, sestrou nebo majitelem, ale ne jejich kombinací. Příslušnost je v našem případě úplná, kdy musí každá entita generalizující entitní množiny spadat také do některé specializace. V našem řešení je tedy každá entitní množina implementována jako ojedinělá tabulka.

3 Triggery

Zadání vyžadovalo implementaci minimálně 2 triggerů.

3.1 Trigger `Osoba_generuj_PK` - generování hodnot primárního klíče

Prvním je trigger pro automatické doplnění primárního klíče v podobě jedinečného identifikačního kódu. Je využívána sekvence, jejíž implicitní počáteční hodnota je nastavena na 10000 a při každém použití je inkrementována o 1. Identifikační kódy, které nejsou automaticky dogenerovány mají kód v podobě desetimístného čísla. Je tedy zajištěno dost místa, aby v případě generování hodnot nedošlo ke kolizi. Příslušné operace triggeru jsou vykonány pokaždé, kdy je n-tice relace do tabulky přidána, nebo upravena. Spuštění je provedeno před vložením do tabulky. K doplnění primárního klíče dojde v případě, že daná hodnota není zadána, nedojde tedy k přepisu zadané hodnoty.

```
1  CREATE SEQUENCE Osoba_ID_sequence
2      START WITH 10000
3      INCREMENT BY 1;
4
5  CREATE OR REPLACE TRIGGER Osoba_generuj_PK
6      BEFORE INSERT OR UPDATE
7      ON Osoba
8      FOR EACH ROW
9      BEGIN
10         IF :NEW.id_osoby IS NULL
11         THEN
12         :NEW.id_osoby := Osoba_ID_sequence.nextval;
13         END IF;
14     END;
15 /
```

3.2 Trigger `Sestra_zvedni_plat` - navýšení hodinové mzdy na základě vzdělání

Druhý trigger slouží pro zvýšení platu sestrám. Plat (v podobě hodinové mzdy) je navýšen o 15 % z aktuálního ohodnocení při dokončení bakalářského studijního programu a o dalších 10 % při dokončení magisterského programu. Jiné tituly jsme nebrali v potaz, protože jsme je pro tuto pracovní pozici nepovažovali za relevantní. Je předpokládáno, že při nástupu do funkce je v pracovní smlouvě už aktuální dosažené vzdělání zohledněno ve finanční odměně, trigger tedy reaguje pouze v případě, kdy je získaný titul aktualizován. Vypočítaná hodnota nové hodinové mzdy je zaokrouhlena pomocí funkce `ROUND` na celé číslo. Výsledek je zapisován do jiné tabulky, než která aktivaci triggeru způsobila. Pro aktualizaci hodnoty je tedy použit příkaz `UPDATE`.

```
CREATE OR REPLACE TRIGGER Sestra_zvedni_plat
     AFTER UPDATE ON Osoba
     FOR EACH ROW
     DECLARE
          procento_mzdy NUMBER;
          aktualni_mzda NUMBER;
          nova_mzda NUMBER;

     BEGIN
          procento_mzdy := 0;
          nova_mzda := 0;

          SELECT TO_NUMBER(hodinova_mzda) INTO aktualni_mzda FROM Sestra WHERE
     id_osoby = :NEW.id_osoby;

          IF ((:OLD.titul IS NULL) AND (:NEW.titul = 'Bc'))
          THEN
               procento_mzdy := (aktualni_mzda * 15) / 100;
          ELSIF ((:OLD.titul = 'Bc') AND ((:NEW.titul = 'Mgr') OR (:NEW.titul =
      'Ing')))
          THEN
               procento_mzdy := (aktualni_mzda * 10) / 100;
          END IF;

          nova_mzda := aktualni_mzda + procento_mzdy;
          nova_mzda := ROUND(nova_mzda, 0);
          UPDATE Sestra SET hodinova_mzda = CAST(nova_mzda AS CHAR(3)) WHERE
     id_osoby = :NEW.id_osoby;
     END;
/
```

4 Uložené procedury

Zadání specifikovalo implementovat minimálně 2 uložené procedury. My jsme vytvořili 3, jejich implementace jsou popsány níže.

4.1 Procedura `vypocitej_procento_leceb`

Tato procedura na vstupu očekává `id_osoby` a klade si za úkol vypočítat kolik procent z celkového počtu léčeb daná osoba léčila, přičemž je rozpoznáno, zda se jedná o veterináře, nebo sestru. Pokud nastane scénář, že zadané ID nepatří ani veterináři, ani sestře, je vyvolána výjimka `NO_DATA_FOUND`. Procedura využívá kurzor a několik deklarovaných proměnných, přičemž typ některých proměnných je pomocí `%TYPE` přejat z dané tabulky. Její jádro tvoří smyčka `LOOP` ve které se díky kurzoru prochází každý řádek tabulky a inkrementují se příslušné proměnné pro počítání léčeb. Cyklus se ukončí ve chvíli, kdy se v kurzoru už nevyskytují žádná data. Jsou vypočítána chtěná procenta a zaokrouhlena funkcí `ROUND` na 2 desetinná místa. Procedura disponuje ošetřením výjimek. V případě, že v databázi nejsou žádné dostupné záznamy o léčbách, dojde při počítání procent k vyvolání výjimky `ZERO_DIVIDE`, jestliže bylo zadáno špatné ID, je ošetřena výjimka `NO_DATA_FOUND`. Pokud k výjimce nedojde, je výsledek procedury vypsán na `DBMS` výstup.

Ukázkový výstup:

```
Veterinar Jan Novak provedl 39,13% z celkoveho poctu leceb.
Sestra Anna Slaba provedla 13,04% z celkoveho poctu leceb.
```

4.2 Procedura `vypocitej_mesicni_vyplatu`

Hlavní úlohou této procedury je spočítání měsíčního platu zaměstnance při odpracování standardních 8 hodin denně. Jako vstupní parametry jsou očekávány `id_osoby` a počet dní v měsíci, pro který je výplata počítána. Z celkového počtu dnů je spočítán počet pracovních dnů (víkendy jsou odečteny). Omezením této procedury je, že počítá s tím, že každý měsíc začíná pondělkem. Implicitně je předpokládáno, že zaměstnanec odpracoval celý měsíc bez jediného dne dovolené. Funkcionalita uvažuje, že kdyby si zaměstnanec vzal v měsíci dovolenou, bylo by to zohledněno v počtu dní ve vstupním parametru. Podobně jako procedura popsaná v předchozí kapitole, i tato disponuje ošetřením výjimek. Vypočítaná hodnota je vypsána na `DBMS` výstup.

Ukázkový výstup:

```
Mesicni vyplata pro tuto osobu je: 36432Kc.
```

4.3 Procedura `vypocitej_vekove_zastoupeni_zvirat`

Primárním úkolem procedury je klasifikovat zvířata do jednotlivých skupinek podle věku. Výstupem je poté počet zvířat v daném intervalu. Zvířata jsou podle stáří rozřazeny do následujících skupinek:

- 0–2 roky
- 2–4 roky
- 4–6 let
- 6–8 let
- 8–10 let
- 10 a více let

Procedura využívá kurzor a deklarované proměnné. Některé proměnné použitím atributu `%TYPE` přejímají datový typ z atributu relace příslušné tabulky. Srdce procedury tvoří cyklus `LOOP`, který končí v případě, že již byla všechna data z kurzoru testována. V tomto cyklu je vypočítáván věk zvířete. Pro jeho zjištění je využito datum narození zvířete a systémový čas `SYSDATE`. Užitím funkce `MONTHS_BETWEEN` s parametry data narození a systémového času je navrácen počet měsíců mezi těmito daty a ten je následně podělen 12. Hodnota z výpočtu je užitím funkce `FLOOR` upravena na celé číslo, které odpovídá právě stáří zvířete. Následně už jsou jen užitím `BETWEEN` klasifikována jednotlivá zvířata a inkrementována příslušná počítadla. V případě, že databáze neobsahuje žádná data o zvířatech, nebo všechna zvířata nemají uvedené datum narození, je vyvolána výjimka `NO_DATA_FOUND`. Výsledný přehled je vypsán na `DBMS` výstup.

Ukázkový výstup:

```
0 az 2 roky:   2
2 az 4 roky:   2
4 az 6 let:    3
6 az 8 let:    1
8 az 10 let:   3
10 a vice let: 4
```

5 Index a `EXPLAIN PLAN`

Explain plan je předveden pomocí příkazu `SELECT`. Nejdříve je spušten `PLAN` bez indexu a poté s vytvořeným indexem. `SELECT` byl předveden nad tabulkamy `Zvire` a `Lecba`, ze kterých získal druh zvířete a průměrnou cenu léčby.

5.1 Bez použití indexu

Mezi instrukcemi byl použit `NESTED LOOP` - to znamená, že se tabulky spojují tím způsobem, že pro každou položku z první tabulky se projde celá druhá tabulka. Tabulka se prochází pomocí `TABLE ACCESS FULL`, což znamená, že tabulka se prochází celá, ne podle indexů. Před použitím indexu vypadá tabulka následovně:

```
-----------------------------------------------------------------------------------------
| Id  | Operation                     | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |          |    23 |  1173 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY                |          |    23 |  1173 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS                |          |    23 |  1173 |     3   (0)| 00:00:01 |
|   3 |    NESTED LOOPS               |          |    23 |  1173 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL         | LECBA    |    23 |   598 |     3   (0)| 00:00:01 |
|*  5 |     INDEX UNIQUE SCAN         | PK_ZVIRE |     1 |       |     0   (0)| 00:00:01 |
```

5.2 S použitím indexu

Tabulka se prochází pomocí `INDEX FULL SCAN`, což znamená, že tabulka se prochází podle indexů. V příkazu se dále využívá `HASH JOIN`, který vytvoří hash tabulku, jejíž každý řádek odpovídá řádku původní tabulky. Díky tomu se porovnává pouze jedna hodnota a ne tolik hodnot, kolik je sloupců. Díky indexaci lze provádění těchto operací urychlit. Při naší snaze přístup zefektivnit jsme vytvořili následující 2 indexy:

```
1  CREATE INDEX indexZ ON Zvire(id_zvire, jmeno);
2  CREATE INDEX indexL ON Lecba(id_lecba, cena);
```

V příkazu `SELECT` jsou optimalizátoru indexy doporučeny a za využití „hintu“ `LEADING` upřednostněna tabulka `Lecba`, která je následně vázána na tabulku `Zvire`. Po této optimalizaci vypadá tabulka takto:

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                              | Name   | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                       |        |    23 |  1173 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY                         |        |    23 |  1173 |     4  (25)| 00:00:01 |
|*  2 |   HASH JOIN                            |        |    23 |  1173 |     3   (0)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID BATCHED | LECBA  |    23 |   598 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN                    | INDEXL |    23 |       |     1   (0)| 00:00:01 |
|   5 |    INDEX FULL SCAN                     | INDEXZ |    15 |   375 |     1   (0)| 00:00:01 |
```

Ve sloupci `Cost` můžeme vidět, že použití indexů vedlo ke snížení ceny přístupu. Optimalizaci lze provádět i dále a to zahrnutím více sloupců, se kterými v dotazu pracujeme.

6 Přístupová práva

Přidělení přístupovách práv má simulovat vyvtoření přístupu k určitým tabulkám pro uživatele, např. zaměstnance. Tento zaměstnanec tak nemá přístup ke všem tabulkám, ale pouze k těm, ke kterým ho potřebuje. Dále je přístup také přidán pro vytvořené procedury.

7 Materializovaný pohled

Materializovaný pohled je určen pro druhého člena týmu. Nejprve jsou vytvořeny materializované logy, které slouží pro zrychlení načtení celého pohledu. Poté je vytořen materializovaný pohled. Ten je poté předveden na ukázkovém příkazu `SELECT`, který vypíše vše co pohled obsahuje, poté vloží do tabulky nová data a znovu vše vypíše. Postup je opakován i pro demonstrační odstranění z tabulky.

V materializovaném pohledu jsou využity příkazy `CACHE`, který optimalizuje načítaná data, `BUILD IMMEDIATE`, který pohled hned po vytvoření naplní daty, `ENABLE QUERY REWRITE`, díky kterému bude pohled používán optimalizátorem.