VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IDS PROJEKT

Ordinace praktického lékaře

Jakub Hlava (xhlava52)

Thi Bao Ngoc Vu (xvuthi00)

Výsledné řešení databáze

Výsledné řešení vesměs odpovídá původnímu návrhu z ER diagramu z 1. části projektu, který doznal několika málo rozšíření a úprav.

Součástí řešení jsou tabulky `Pacient`, `Navsteva`, `Pozvanka`, `Ockovani`, `PreventivniProhlidka`, `Vykon`, `ReceptNaLek`, `ZadankaNaVysetreni`, `PlanovanyVykon`, `VlastniPacient`, `AkutniPripad`, `Faktura`, `Pojistovna a Deti.`

Detailní popis tabulek

Název tabulky	Popis	Uchovávaná data
Pacient	Základní informace o registrovaných pacientech	Rodné číslo, jméno, příjmení, datum narození, telefon, email, ulice, město, psč, číslo pojišťovny
VlastniPacient	Pomocná tabulka pro identifikaci vlastních pacientů	Rodné číslo (spojení s Pacient)
AkutniPripad	Pomocná tabulka pro identifikaci akutních případů s jiným prakt. lékařem	Rodné číslo (spojení s Pacient), praktický lékař
Navsteva	Záznam o návštěvě pacienta u lékaře	Termín návštěvy, důvod k návštěvě, lékařská zpráva, rodné číslo (spojení s Pacient)
Pozvanka	Pomocná tabulka pro rezervace termínů budoucích návštěv	Termín návštěvy, rodné číslo (spojení s Pacient)
Ockovani	Záznamy o provedených očkováních	Typ očkování, datum expirace ochrany po očkování, rodné číslo (spojení s Pacient)
PreventivniProhlidka	Záznamy o preventivních prohlídkách	Expirace platnosti prohlídky, rodné číslo (spojení s Pacient)
Vykon	Záznamy o provedených výkonech během návštěv, vč. ceny pro pojišťovnu	Název výkonu, cena, číslo faktury (spojení s Faktura), termín návštěvy (spojení s Navsteva a tím i Pacient)
ReceptNaLek	Recepty na vydané léky, vč. ceny pro pojišťovnu	Název léku, množství léku, celková cena za recept, číslo faktury (spojení s Faktura), termín návštěvy (spojení s Navsteva a tím i Pacient)
ZadankaNaVysetreni	Žádanky na vyšetření u specialisty	Jméno lékaře, název ordinace, (obvykle zpětně) dodaná zpráva z vyšetření a termín návštěvy, při které bylo o vyšetření požádáno (spojení s Navsteva a Pacient)
PlanovanyVykon	Pomocná tabulka, umožňující k pozvánce doplnit i plán na provedení lékařského výkonu	Popis výkonu a termín návštěvy (spojení s Pozvanka)
Faktura	Záznamy o fakturách pro pojišťovny	Číslo faktury a číslo pojišťovny (spojeni s Pojistovna)
Pojistovna	Pomocná tabulka evidující existující pojišťovny a spojující číslo pojišťovny s jejím názvem	Číslo pojišťovny, název pojišťovny
Deti	Pomocná tabulka pro evidencí dětí a jejich zákonných zástupců (automaticky pomocí triggeru)	Rodné číslo, kontakt (jméno, příjmení, telefon) na zákonného zástupce

Primární klíče

Primární klíče tabulek jsme se snažili volit podle unikátních údajů v tabulce, tedy kde bylo k dispozici rodné číslo, tak jsme jej použili jako PK, u návštěv a pozvánek jsme se rozhodli použít jako primární klíč datum a čas návštěvy, protože v jedné ordinaci nemůže nastat situace, kdy by jeden lékař ošetřoval více pacientů zároveň a u tabulek, kde se na jedno rodné číslo nebo návštěvu může vázat více záznamů jsme se rozhodli přidat rostoucí posloupnost identifikačních čísel a tuto použít jako primární klíč.

Vzorová data

Tabulku jsme naplnili smyšlenými vzorovými daty v takovém množství, abychom mohli demonstrovat příkazy pro výběr dat z databáze v požadované míře a naplnili jsme všechny tabulky alespoň nějakým záznamem.

Výběr dat z databáze

Při tvorbě dotazů jsme se snažili o to, aby krom plnění požadavků zadání alespoň trochu praktické:

Spojení dvou tabulek jsme demonstrovali na dotazu, který zobrazí všechny návštěvy pacientů z Prostějova a na dotazu, který zobrazí pacienty a jejich očkování, pokud jim expirovala platnost.

```
SELECT DISTINCT P.jmeno, P.prijmeni, P.telefon, N.termin, N.duvod
FROM Pacient P, Navsteva N
WHERE N.rodneCislo = P.rodneCislo AND P.mesto = 'Prostějov';
```

```
SELECT DISTINCT P.jmeno, P.prijmeni, P.telefon, O.typ
FROM Pacient P, Ockovani O
WHERE P.rodneCislo=O.rodneCislo AND O.expirace < SYSDATE;
```

Spojení tří tabulek na zjištění celkové ceny výkonů a léků u jednotlivých faktur.

```
SELECT sv.cisloFaktury, COALESCE(v_ceny, 0)+COALESCE(rnl_ceny, 0) celkovaCena FROM
(SELECT SUM(V.cena) v_ceny, F.cisloFaktury
    FROM Faktura F, Vykon V
    WHERE V.cisloFaktury = F.cisloFaktury
    GROUP BY F.cisloFaktury) sv
LEFT JOIN
(SELECT SUM(RNL.celkovaCena) rnl_ceny, F.cisloFaktury
    FROM Faktura F, ReceptNaLek RNL
    WHERE F.cisloFaktury = RNL.cisloFaktury
    GROUP BY F.cisloFaktury) srnl ON sv.cisloFaktury = srnl.cisloFaktury
ORDER BY sv.cisloFaktury;
```

Klauzuli GROUP BY (krom dotazu výše) ještě na počtech pacientů z jednotlivých obcí a na cenách výkonů podle faktur.

```
SELECT COUNT(rodneCislo) pocet_pacientu, mesto
FROM Pacient
GROUP BY mesto
ORDER BY COUNT(rodneCislo) ASC;
```

```
SELECT F.cisloFaktury, SUM(V.cena) cena_vykonu FROM
Faktura F, Vykon V
WHERE F.cisloFaktury = V.cisloFaktury
GROUP BY F.cisloFaktury
ORDER BY F.cisloFaktury ASC;
```

A na závěr predikáty IN a EXISTS na získání kontaktů na všechny akutní případy (IN) a pacienty, kteří alespoň jednou navštívili ordinaci (EXISTS).

```
SELECT DISTINCT P.jmeno, P.prijmeni, P.rodneCislo
FROM Pacient P
WHERE EXISTS
    (SELECT N.termin FROM Navsteva N WHERE N.rodneCislo=P.rodneCislo);

SELECT DISTINCT P.jmeno, P.prijmeni, P.telefon, P.email, P.rodneCislo
FROM Pacient P
WHERE p.rodneCislo IN
      (SELECT rodneCislo FROM AkutniPripad);
```

Databázové triggery

Kvůli generování PK pomocí triggeru jsme přidali novou tabulku Deti. Primární klíč tabulky se generuje ze sekvence pocitadlo, začíná na hodnotě 1000 a postupně se inkrementuje o 1. Samotný trigger pridatDite se spustí po přidání nového pacienta do tabulky Pacient. Zkontroluje se věk přidaného a pokud je jeho věk nižší než 18 let, přidá se do tabulky Deti a bude mu přidělen vygenerovaný PK.

```
CREATE SEQUENCE pocitadlo
start with 1000
increment by 1;
CREATE TRIGGER pridatDite
  AFTER  INSERT ON Pacient
  FOR EACH ROW
BEGIN
    IF (TRUNC((SYSDATE - :NEW.datumNarozeni) / 365)) < 18 THEN
            INSERT INTO Deti(IDDITETE, rodneCislo ,  JMENOZASTUPCE, PRIJMENIZASTUPCE)
VALUES (pocitadlo.nextval , :NEW.rodneCislo, NULL, NULL);
    END IF;
END;
```

Další dva triggery slouží pro ověřování konzistence dat – formátu telefonního čísla a případně i zda je datum narození platné (tedy z minulosti), jeden v tabulce Pacient a druhý v tabulce Deti (ten slouží primárně pro demonstraci chování procedury viz níže).

```
-- Ověření formátu telefonu a data narození jako příklad využití triggeru pro zajištění
integrity dat
CREATE OR REPLACE TRIGGER overitUdaje
    BEFORE INSERT ON Pacient
    FOR EACH ROW
DECLARE
    chybny_telefon EXCEPTION;
    PRAGMA exception_init ( chybny_telefon, -20000 );
    chybne_datum_narozeni EXCEPTION;
    PRAGMA exception_init ( chybne_datum_narozeni, -20001 );
BEGIN
    IF NOT REGEXP_LIKE(:NEW.telefon, '^\+[0-9]{1,3}[0-9]{9}$') THEN
        raise_application_error(-20000, 'Neplatné telefonní číslo (formát
+(predvolba)xxxxxxxxx)');
    end if;
    IF (SYSDATE < :NEW.datumNarozeni) THEN
        raise_application_error(-20001, 'Datum narození musí být v minulosti!');
    end if;
END;
```

```
-- Pomocný trigger pro demonstraci exception handling v proceduře níže
CREATE OR REPLACE TRIGGER overitTelefonDite
    BEFORE INSERT OR UPDATE ON Deti
    FOR EACH ROW
declare
    chybny_telefon EXCEPTION;
    PRAGMA exception_init ( chybny_telefon, -20000 );
begin
    IF NOT REGEXP_LIKE(:NEW.telefonZastupce, '^\+[0-9]{1,3}[0-9]{9}$') THEN
        raise_application_error(-20000, 'Neplatné telefonní číslo (formát
+(predvolba)xxxxxxxxx)');
    end if;
end;
```

Procedury

Pro demonstrací využití kurzoru a proměnných s typem odkazujícím se na sloupec tabulky jsme vytvořili proceduru kontakty, která na terminál klienta vypíše seznam pacientů narozených v určitém roce (např. pro objednání pacientů na pravidelné preventivní prohlídky)

```
CREATE OR REPLACE PROCEDURE kontakty(rok in INT)
    IS
    CURSOR cur_pacienti IS
        SELECT jmeno, prijmeni, datumNarozeni, telefon, email, rodneCislo
        FROM Pacient
        WHERE datumNarozeni BETWEEN TO_DATE(TO_CHAR(rok), 'YYYY') AND
TO_DATE(TO_CHAR(rok+1), 'YYYY');
    jmeno Pacient.jmeno%type;
    prijmeni Pacient.prijmeni%type;
    telefon Pacient.telefon%type;
    email Pacient.email%type;
    datumNarozeni Pacient.datumNarozeni%type;
    rodneCislo Pacient.rodneCislo%type;
BEGIN
    OPEN cur_pacienti;

    LOOP
        FETCH cur_pacienti INTO jmeno, prijmeni, datumNarozeni, telefon, email, rodneCislo;
        EXIT WHEN cur_pacienti%NOTFOUND;
        dbms_output.PUT_LINE('Pacient: ' || jmeno || ' ' || prijmeni || ' (' || rodneCislo
|| ') nar. ' || datumNarozeni || ': ' || telefon || ', ' || email);
    end loop;
        dbms_output.PUT_LINE('Celkem pacientů narozených v roce '|| rok || ': ' ||
cur_pacienti%ROWCOUNT);

    CLOSE cur_pacienti;
    return;
end;
```

Ošetření výjimek jsme demonstrovali na proceduře, sloužící pro doplnění informací o zákonném zástupci k záznamu dítěte

```
CREATE OR REPLACE PROCEDURE
nastavRodice(rcDite in varchar, jmenorodice in varchar, prijmenirodice in varchar,
telefonrodice in varchar) IS
    chybny_telefon EXCEPTION;
    PRAGMA exception_init ( chybny_telefon, -20000 );
    BEGIN
        UPDATE Deti SET jmenoZastupce=jmenorodice, prijmeniZastupce=prijmenirodice,
telefonZastupce=telefonrodice WHERE rodneCislo=rcDite;
    EXCEPTION
        WHEN chybny_telefon THEN
            dbms_output.PUT_LINE('Zadaný telefon má nesprávný formát, zkuste to prosím
znovu');
    end;
```

Index a EXPLAIN PLAN

Rozhodli jsme se demonstrovat urychlení dotazu, EXPLAIN PLAN a vytvoření indexu demonstrovat na jednom případě dohromady.

Pro demonstraci jsme zvolili dotaz, zjišťující počty návštěv jednotlivých pacientů.

```
SELECT DISTINCT P.jmeno, P.prijmeni, P.rodneCislo, COUNT(N.termin)
FROM Pacient P LEFT JOIN Navsteva N
    on P.rodneCislo = N.rodneCislo
GROUP BY P.jmeno, P.prijmeni, P.rodneCislo;
```

Bez optimalizací byl výsledek EXPLAIN PLAN následující:

```
----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |     6 |   282 |     7  (29)| 00:00:01 |
|   1 |  HASH GROUP BY                |                |     6 |   282 |     7  (29)| 00:00:01 |
|   2 |   MERGE JOIN OUTER            |                |     6 |   282 |     6  (17)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID| PACIENT        |     6 |   168 |     2   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN           | SYS_C002080516 |     6 |       |     1   (0)| 00:00:01 |
|*  5 |    SORT JOIN                  |                |     4 |    76 |     4  (25)| 00:00:01 |
|   6 |     TABLE ACCESS FULL         | NAVSTEVA       |     4 |    76 |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------------------
```

První optimalizací bylo zavedení indexu na pole pro klauzuli GROUP BY, tedy:

```
CREATE INDEX jm ON Pacient (rodneCislo, jmeno, prijmeni);
```

Po vytvoření tohoto indexu se vykonávání dotazu zrychlilo následovně:

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          |     4 |   188 |     5  (20)| 00:00:01 |
|   1 |  HASH GROUP BY      |          |     4 |   188 |     5  (20)| 00:00:01 |
|*  2 |   HASH JOIN OUTER   |          |     6 |   282 |     4   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN  | JM       |     6 |   168 |     1   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| NAVSTEVA |     4 |    76 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Návrhem na další optimalizaci by mohlo být zavedení indexu na rodných číslech v tabulce Navsteva:

```
CREATE INDEX rc ON Navsteva (rodneCislo);
```

Ten dále snižuje odhadovanou cenu operací následovně:

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                | Name             | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                  |     4 |   188 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY           |                  |     4 |   188 |     4  (25)| 00:00:01 |
|*  2 |   HASH JOIN OUTER        |                  |     6 |   282 |     3   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN       | JM               |     6 |   168 |     1   (0)| 00:00:01 |
|   4 |    VIEW                  | index$_join$_002 |     4 |    76 |     2   (0)| 00:00:01 |
|*  5 |     HASH JOIN            |                  |       |       |            |          |
|   6 |      INDEX FAST FULL SCAN| RC               |     4 |    76 |     1   (0)| 00:00:01 |
|   7 |      INDEX FAST FULL SCAN| SYS_C002080522   |     4 |    76 |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Poslední možnost optimalizace již ale neodpovídá poslednímu příkazu, proto ji přidáváme „navíc“, ale pro určité účely by bylo možné uvažovat vyřazení pacientů bez návštěv ze statistiky výměnou LEFT JOIN spojení za INNER JOIN a v kombinaci s indexem rc následovně zásadním způsobem urychlit dotaz:

```
-------------------------------------------------------------------------------
| Id  | Operation            | Name | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |      |     4 |   156 |     1   (0)| 00:00:01 |
|   1 |  SORT GROUP BY NOSORT|      |     4 |   156 |     1   (0)| 00:00:01 |
|   2 |   NESTED LOOPS       |      |     4 |   156 |     1   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN   | JM   |     6 |   168 |     1   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN  | RC   |     1 |    11 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Materializovaný pohled

Materializovaný pohled jsme vytvořili Z uživatele XVUTHI00 nad dotazem nad tabulkami v DB uživatele XHLAVA52 pro výpočet „nákladů“ pojišťoven na jednotlivé pacienty.

```
CREATE MATERIALIZED VIEW ceny_pacientu
    NOLOGGING
    CACHE
    BUILD IMMEDIATE
    AS SELECT P.jmeno, P.prijmeni, P.rodneCislo, cv.celkemv vykonyCelkem, cl.celkeml
lekyCelkem, COALESCE(cv.celkemv,0) + COALESCE(cl.celkeml,0) celkem
        FROM XHLAVA52.PACIENT P JOIN (SELECT P.rodneCislo rc, SUM(V.cena) celkemv FROM
XHLAVA52.PACIENT P, XHLAVA52.VYKON V WHERE termin IN (SELECT termin from XHLAVA52.NAVSTEVA
WHERE rodneCislo=P.rodneCislo) GROUP BY P.rodneCislo) cv on P.rodneCislo=cv.rc LEFT JOIN
         (SELECT P.rodneCislo rc, SUM(RNL.celkovaCena) celkeml FROM XHLAVA52.PACIENT P,
XHLAVA52.RECEPTNALEK RNL WHERE termin IN (SELECT termin from XHLAVA52.NAVSTEVA WHERE
rodneCislo=P.rodneCislo) GROUP BY P.rodneCislo) cl ON P.rodneCislo=cl.rc;
```