Dokumentace popisující finální schéma databáze

Egor Greb, Kirill Mikhailov

May 2, 2022

EXPLAIN PLAN FOR

```
EXPLAIN PLAN FOR
    SELECT COUNT(Surovina.cislo_suroviny), Surovina.nazev
    FROM Surovina, Obsahuje, Polozka
    WHERE Polozka.cislo_peciva = Obsahuje.cislo_peciva AND Obsahuje.cislo_suroviny = Surovina.cislo_suroviny
    GROUP BY Surovina.nazev, Surovina.cislo_suroviny
    ORDER BY COUNT(Surovina.cislo_suroviny) DESC;

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Tady vidíme použití *EXPLAIN PLAN* pro výpis plánu provedení databazového dotazu se spojením alespoň dvou tabulek, agregační funkcí a klauzulí GROUP BY. Ukazuje, ktera surovina je nejvic "spotrebovana" s ohledem na objednane polozky. Pro urychlení spracování využíme **indexy**.

```
CREATE INDEX i_surovina ON SUROVINA(CISLO_SUROVINY, NAZEV);
CREATE INDEX i_obsahuje ON OBSAHUJE(CISLO_SUROVINY, CISLO_PECIVA);
```

Mužeme jistě vidět, jaké režie byly před tím, než jsme využili indexy, abychom optimalizovali zpracování dotazů a jak se to zlepšilo po nim.

Bez indexů

```
Plan hash value: 1749747669

-------------------------------------------------------------------------------------
| Id  | Operation             | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |           |    21 |   567 |    10  (10)| 00:00:01 |
|   1 |  SORT ORDER BY        |           |    21 |   567 |    10  (10)| 00:00:01 |
|   2 |   HASH GROUP BY       |           |    21 |   567 |    10  (10)| 00:00:01 |
|*  3 |    HASH JOIN          |           |    21 |   567 |     9   (0)| 00:00:01 |
|*  4 |     HASH JOIN         |           |    21 |   189 |     6   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL| POLOZKA   |    11 |    33 |     3   (0)| 00:00:01 |
|   6 |      TABLE ACCESS FULL| OBSAHUJE  |    30 |   180 |     3   (0)| 00:00:01 |
|   7 |     VIEW              | VW_GBF_17 |    17 |   306 |     3   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL| SUROVINA  |    17 |   204 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

S indexy

```
Plan hash value: 1855576643

-------------------------------------------------------------------------------------------
| Id  | Operation             | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                |    21 |   567 |     6  (17)| 00:00:01 |
|   1 |  SORT ORDER BY        |                |    21 |   567 |     6  (17)| 00:00:01 |
|   2 |   HASH GROUP BY       |                |    21 |   567 |     6  (17)| 00:00:01 |
|*  3 |    HASH JOIN          |                |    21 |   567 |     5   (0)| 00:00:01 |
|*  4 |     HASH JOIN         |                |    21 |   189 |     4   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL| POLOZKA        |    11 |    33 |     3   (0)| 00:00:01 |
|   6 |      INDEX FULL SCAN  | OBSAHUJE_INDEX |    30 |   180 |     1   (0)| 00:00:01 |
|   7 |     VIEW              | VW_GBF_17      |    17 |   306 |     1   (0)| 00:00:01 |
|   8 |      INDEX FULL SCAN  | SUROVINA_INDEX |    17 |   204 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

Uložená procedura č.1

```
CREATE OR REPLACE PROCEDURE INCOME_FROM_TO (datum_od OBJEDNAVKA.datum%TYPE, datum_do OBJEDNAVKA.datum%TYPE) AS
    BEGIN
        DECLARE CURSOR income_cursor IS
            SELECT FAKTURA.celkova_castka, Objednavka.datum
            FROM FAKTURA, OBJEDNAVKA
            WHERE Faktura.cislo_faktury = Objednavka.cislo_faktury;
                celkem NUMBER;
                datum_objednani OBJEDNAVKA.datum%type;
                castka_objednavka FAKTURA.celkova_castka%type;
                BEGIN
                    celkem := 0;
                    open income_cursor;
                    LOOP
                        <<START>>
                        FETCH income_cursor INTO castka_objednavka, datum_objednani;
                        EXIT WHEN income_cursor%NOTFOUND;
                        IF datum_objednani < TO_DATE(datum_od, 'dd.mm.yyyy') THEN CONTINUE;
                        ELSIF datum_objednani > TO_DATE(datum_do, 'dd.mm.yyyy') THEN EXIT;
                        end if;
                        celkem := celkem + castka_objednavka;
                    end loop;
                    DBMS_OUTPUT.PUT_LINE( A: 'Zisk za dobu od ' || datum_od || ' do ' || datum_do || ' je ' || celkem);
                    close income_cursor;
                end;
    end;
```

Procedura, která počitá získ během určitou doby. Používáme tady proměnné **datum_od** a **datum_do**, odkazující na sloupce tabulky **Objednavka**. Vyskytuje tady taky kurzor **income_cursor** a ošetření výjimek. Data se čte až

Uložená procedura č.2

```
CREATE OR REPLACE PROCEDURE INGREDIENT_CONSUMPTION(surovina_ID Surovina.cislo_suroviny%TYPE, datum_od OBJEDNAVKA.datum%TYPE) AS
    BEGIN
        DECLARE CURSOR obsahuje_cursor is
            SELECT Obsahuje.cislo_suroviny, Obsahuje.gramaz
            FROM Polozka, Faktura, Objednavka, Obsahuje
            WHERE Faktura.cislo_objednavky = Objednavka.objednavkaID and Polozka.cislo_faktury = Faktura.cislo_faktury and Objednavka.datum > datum_od;
            celkem_suroviny NUMBER;
            surovina_ID_fetched Surovina.cislo_suroviny%TYPE;
            gramaz Obsahuje.gramaz%TYPE;
            BEGIN
                celkem_suroviny := 0;
                gramaz := 0;
                OPEN obsahuje_cursor;
                LOOP
                    FETCH obsahuje_cursor INTO surovina_ID_fetched, gramaz;
                    EXIT WHEN obsahuje_cursor%NOTFOUND;
                    IF NOT surovina_ID_fetched = surovina_ID THEN CONTINUE;
                    ELSE celkem_suroviny := celkem_suroviny + gramaz;
                    END IF;
                END LOOP;
                DBMS_OUTPUT.PUT_LINE( A: 'Od '|| datum_od || ' bylo spotrebeno ' || celkem_suroviny || ' gramu suroviny c.'|| surovina_ID);
                CLOSE obsahuje_cursor;
            END;
    END;
```

Další procedura zjišťuje jaké mnozství dane suroviny bylo spotřebovano na objednavky od urcitého data. Použili jsme tady taky cursory, proměnné s datovými typy a ošetření výjimek. Data se čtou z čtyřech tabulek.

Materializované pohledy

```
CREATE MATERIALIZED VIEW LOG ON Zamestnanec WITH PRIMARY KEY, ROWID;
CREATE MATERIALIZED VIEW LOG ON Rozvoz WITH PRIMARY KEY, ROWID;
```

Vytvořili jsme pohled na tabulky **Zamestnanec** a **Rozvoz**, ale ty které skutečně uložený ve fyzické pamětí.

```
CREATE MATERIALIZED VIEW PECIVO_NEJVIC_DRUHU_SUROVIN
NOLOGGING
CACHE
BUILD IMMEDIATE
AS SELECT Obsahuje.cislo_peciva as pecivo_ID, COUNT(*)
    FROM Obsahuje JOIN Pecivo on Obsahuje.cislo_peciva = Pecivo.cislo_peciva
    GROUP BY Obsahuje.cislo_peciva;
```

Na tomhle obrazku jde o pohled, který zahrnuje kolik druhu surovin je protreba na kazdy druh peciva. Rozhodli jsme využit materializovaný pohled tady z důvodu zvýšení efektivnosti, resp. omezený přístup k datům.

Přístupová práva

```
GRANT ALL ON FAKTURA TO XGREBE02;
GRANT ALL ON OBJEDNAVKA TO XGREBE02;
GRANT ALL ON OBSAHUJE TO XGREBE02;
GRANT ALL ON PECIVO TO XGREBE02;
GRANT ALL ON POLOZKA TO XGREBE02;
GRANT ALL ON ROZVOZ TO XGREBE02;
GRANT ALL ON SUROVINA TO XGREBE02;
GRANT ALL ON VYZVEDNUTI TO XGREBE02;
GRANT ALL ON ZAKAZNIK TO XGREBE02;
GRANT ALL ON ZAMESTNANEC TO XGREBE02;

GRANT ALL ON PECIVO_NEJVIC_DRUHU_SUROVIN TO XGREBE02;
GRANT EXECUTE ON INGREDIENT_CONSUMPTION TO XGREBE02;
GRANT EXECUTE ON INCOME_FROM_TO TO XGREBE02;
```

Tady vidíme definici přístupových práv k databázovým objektům pro jedného z členů týmu.