IDS: Informační systém krevní banky

Popis implementace a použití pokročilých objektů schématu databáze

1 Triggery

1.1 Automatické generování hodnoty primárního klíče

Trigger **TRG_AUTO_ID_INC** automaticky generuje hodnotu primárního klíče tabulky *pobocka* v případě, že je hodnota primárního klíče při vkládání prázdná (tj. *NULL*). Výsledná hodnota klíče je generována ze sekvence *POBOCKA_SEQ*.

1.2 Aktualizace sloupce po vložení záznamu do tabulky

Tabulka *darce* obsahuje sloupec *posledni_odber*, který se musí vhodně aktualizovat po každém vložení nového záznamu do tabulky *davka*. Cílem triggeru **UPDATE_COLLECTION_DATE** je odstranění této závislosti a tím odstranění potenciálního zdroje chyb.
Jak již název napovídá, trigger nastaví datum a čas ve sloupci *posledni_odber* u patřičného dárce na aktuální, a to po každém vložení nového záznamu do tabulky *davka*.

2 Funkce a procedury

2.1 Výpočet věku dárce

Funkce **DONOR_AGE** slouží pro výpočet věku dárce z jeho data narození (protože ukládání věku do databáze je špatný nápad). Funkce má jediný argument a to datum narození. Výsledkem je věk dárce zaokrouhlený na jedno desetinné místo. V případě, že je výsledný věk menší jako 0, je vyvolána výjimka s číslem *-20001*.
Chování funkce je předvedeno hned v dalším bloku, kde je pro každého dárce z tabulky *darce* vypočítán věk a poté, společně se jménem a příjmením, vypsán na obrazovku. Mezi záznamy je přidán i jeden s neplatným datem narození, aby demonstroval vyvolání a odchycení výjimky.

2.2 Vypočítání data možného odběru

Další užitečnou funkcí je **NEXT_COLLECTION**, která z data posledního odběru dárce vypočítá datum následujícího možného odběru. Standardně se mezi odběry musí udělat minimálně tříměsíční pauza a takto je také tato funkce implementována. Výsledkem funkce je datum posledního odběru + tři měsíce v případně, že poslední odběr proběhl před méně než třemi měsíci. V opačném případě funkce vrací aktuální datum.
Jednoduchá ukázka funkce se nachází v následujícím bloku, kde je datum dalšího možného odběru vypočítáno pro všechny dárce z tabulky *darce*.

3 Index

K nalezení místa pro ideální umístění indexu je využit příkaz **EXPLAIN PLAN**. Ten nám pro zadaný dotaz vytvoří novou tabulku, ve které nalezneme postup vyhodnocení výrazu a získání výsledných dat.
Pro ukázku výstupu jsem vybral následující dotaz:

Výpis 1: Dotaz pro demonstraci optimalizace

```
SELECT d.jmeno, d.prijmeni, d.rodne_cislo, COUNT(o.id) AS pocet_odberu
FROM odber o
LEFT JOIN darce d
ON o.id_darce = d.id
WHERE d.id_pobocky = 1
GROUP BY d.jmeno, d.prijmeni, d.rodne_cislo
ORDER BY pocet_odberu DESC;
```

Zavoláním *EXPLAIN PLAN FOR* společně s výše zmíněným dotazem, dostaneme výslednou tabulku, která po vhodném zformátování vypadá následovně:

Výpis 2: EXPLAIN PLAN bez zavedení indexu

```
-----------------------------------------------------
| Id  | Operation                     | Name           |
-----------------------------------------------------
|   0 | SELECT STATEMENT              |                |
|   1 |  SORT ORDER BY                |                |
|   2 |   HASH GROUP BY               |                |
|   3 |    NESTED LOOPS               |                |
|   4 |     NESTED LOOPS              |                |
|   5 |      TABLE ACCESS FULL        | ODBER          |
|   6 |      INDEX UNIQUE SCAN        | SYS_C001740260 |
|   7 |     TABLE ACCESS BY INDEX ROWID| DARCE          |
-----------------------------------------------------
```

Ve výsledné tabulce lze zpozorovat několik zajímavých klíčkových slov. Prvním z nich je *NESTED LOOPS*. Tato funkce se využívá při spojování menších tabulek dohromady v rámci dotazu. Interní optimalizátor databázového systému si dané spojení rozdělí na dvě části (vnitřní a vnější tabulku) a pak se pro každý řádek vnější tabulky zpracují všechny řádky vnitřní tabulky.
Ve výpisu 2 je vnitřní tabulkou *darce* a vnější tabulkou *odber*. Vnější cyklus vybere všechny řádky z tabulky *odber* a pro každý získaný řádek vybere odpovídající řádek z tabulky *darce*. Dále se zde přistupuje jen k řádkům z tabulky *darce*, které mají určité ID.

Jednou z možností, jak tento dotaz optimalizovat, je zavedení indexu na sloupce, které se používají k vlastnímu spojování tabulek. V tomto případě se jako jeden z kandidátu jeví sloupec *id_pobocky* z tabulky *darce*. Po zavedení indexu je výstup EXPLAIN PLAN následující:

Výpis 3: EXPLAIN PLAN se zavedeným indexem

```
-------------------------------------------------------
| Id  | Operation                           | Name     |
-------------------------------------------------------
|   0 | SELECT STATEMENT                    |          |
|   1 |  SORT ORDER BY                      |          |
|   2 |   HASH GROUP BY                     |          |
|   3 |    HASH JOIN                        |          |
|   4 |     TABLE ACCESS FULL               | ODBER    |
|   5 |     TABLE ACCESS BY INDEX ROWID BATCHED| DARCE    |
|   6 |      INDEX RANGE SCAN               | TEST_IDX |
-------------------------------------------------------
```

Po zavedení indexu se dva vnořené cykly změnily na jeden *HASH JOIN*, který se využívá při spojování větších tabulek, kdy se pro menší tabulku udělá v paměti hashovací tabulka, která se potom využívá při prohledávání větší tabulky. V optimálním případě poté stačí pro výsledné data projít větší tabulku jen jednou. Také přístup k řádkům z tabulky *darce* se nyní provádí pomocí indexované hodnoty sloupce *id_pobocky*.