Databázové systémy
2019 / 2020

Antidopingová agentura

Dokumentace k 4. fázi projektu:

SQL skript pro vytvoření pokročilých objektů schématu databáze

Vypracovali: Jan Lorenc (xloren15)
Vojtěch Staněk (xstane45)

Datum: 21. 4. 2020

Schéma databáze

Tabulky ve výsledné databázi se téměř neliší od těch v navrženém ER diagramu z první fáze. Přidali jsme pouze spojovací tabulky pro N:N vztahy a generalizaci jsme řešili tak, že každý potomek dostal vlastní tabulku.

Hodnoty primárních klíčů všech záznamů jsou číselné hodnoty atributů Id opět dle ERD. Tyto PK se v případě, že je uživatel nevyplní, generují (inkrementací) automaticky pomocí příkazu `GENERATED BY DEFAULT AS IDENTITY`, s výjimkou tabulky Komisar, kde je tato funkcionalita obstarána triggerem (viz. dále) a tabulky Vzorek, která jako jediná nemá PK jako automaticky inkrementované a generované číslo, nýbrž jako hodnotu udávající číslo vzorku nutně zadanou uživatelem.

Dále jsme využili nemalé množství jednoduchých omezení, ve kterých jsme předurčili specifické hodnoty jednotlivým atributům, např.: pohlaví může nabývat pouze hodnot ‘M‘/‘Z‘, typ vzorku ‘Krev‘/‘Moc‘ a jiné.

Pokročilé objekty schématu databáze (4.úkol)

Triggery

Dle zadání jsme měli vytvořit trigger, který by automaticky generoval PK některé tabulky pomocí `SEQUENCE`, toto obstarává trigger SetKomisarId. Tu jsme narazili na problém, že pokud by uživatel PK zadal ručně, trigger by potom nepřiřadil hodnotu sám pomocí sekvence a tato sekvence by se stala zastaralou a s jistotou by hrozilo, že v budoucnu při autogenerování narazí na hodnotu specifikovanou uživatelem a generovala by se již existující hodnotu primárního klíče, což je nežádoucí. Vzhledem k tomu, že uvnitř těla triggeru nelze provést commit, hodnotu sekvence nemůžeme upravit příkazem `ALTER`. Problém jsme vyřešili dotazováním se, jestli nově vygenerovaná hodnota již neexistuje a pokud ano, tak pro každou nově inkrementovanou hodnotu znovu. Toto je bohužel drahé řešení v případě, že sekvence zaostává o mnoho hodnot. Zaručuje nám však identitu a zpětné doplnění přeskočených hodnot primárního klíče.

Druhý trigger UpozorniSportovce se stará o splnění jednoho případu užití systému, a to o upozornění sportovce v případě, že do systému byly zadány výsledky vzorku – kontroly. Vzhledem k tomu, že se nejedná o reálný systém a veškeré testování probíhá na školním serveru (na kterém pravděpodobně ani nemáme dostatečná oprávnění), je samotné poslání e-mailu v blokovém komentáři a neprovádí se. Jako důkaz, že se trigger spustil, necháváme do dbms outputu vypsat e-mail sportovce, kterému by bylo upozornění zasláno.

Procedury

První procedura PomerVysledkuSportovce přijímá Id sportovce a zjistí poměr mezi celkovým počtem pozitivních a negativních výsledků. Neprůkazné ignoruje. Výsledek vypíše do dbms outputu. Využívá všechny povinné požadavky. Kurzor pro iteraci kontrol, proměnnou ZaznamKontroly typu ROWTYPE, do které se ukládá řádek kurzoru a voláním výjimky ošetřuje neexistující záznamy kontrol sportovce.

Druhá procedura KomisarKontrolovalSportovce vypíše do dbms outputu sportovce, které daný komisař kontroloval. Komisař je dán vstupním parametrem Id komisaře. I tato procedura používá kurzor pro iterování přes všechny kontroly a sportovce. Následně na každý řádek výstupu vypíše jména kontrolovaných sportovců. Pokud komisař dle systému žádné sportovce nekontroloval, výstupem bude jen úvodní řádek.

Explain plan

Jedním z úkolů bylo provést optimalizaci dotazu použitím indexů a příkazu explain plan. Tento příkaz nám zobrazí, jak optimalizátor realizoval příkazy, jejich cenu a čas provedení. Na základě těchto údajů pak můžeme hledat vylepšení pro naše dotazy. Námi použitý příkaz

```
SELECT S.Jmeno, S.Prijmeni, Count(*) Pocet_Kontrol
FROM Sportovec S, Kontrola K
JOIN Vzorek HV ON K.HlavniVzorekId = HV.CiselnyKod
JOIN Vzorek KV ON K.KontrolniVzorekId = KV.CiselnyKod
WHERE S.Id = K.SportovecId AND HV.Vysledek != KV.Vysledek
GROUP BY S.Jmeno, S.Prijmeni;
```

vybere jména sportovců a počet kontrol, u kterých měly vzorky odlišné výsledky. Bez použití indexů vypadá náročnost dotazu následovně:

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                      | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |              |     1 |   131 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY                 |              |     1 |   131 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS                 |              |     1 |   131 |     3   (0)| 00:00:01 |
|   3 |    NESTED LOOPS                |              |     1 |   131 |     3   (0)| 00:00:01 |
|   4 |     NESTED LOOPS               |              |     1 |   106 |     3   (0)| 00:00:01 |
|   5 |      NESTED LOOPS              |              |     1 |    81 |     3   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL        | KONTROLA     |     1 |    39 |     3   (0)| 00:00:01 |
|   7 |       TABLE ACCESS BY INDEX ROWID| SPORTOVEC  |     1 |    42 |     0   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN       | PK_SPORTOVEC |     1 |       |     0   (0)| 00:00:01 |
|   9 |      TABLE ACCESS BY INDEX ROWID | VZOREK     |     1 |    25 |     0   (0)| 00:00:01 |
|* 10 |       INDEX UNIQUE SCAN        | PK_VZOREK    |     1 |       |     0   (0)| 00:00:01 |
|* 11 |     INDEX UNIQUE SCAN          | PK_VZOREK    |     1 |       |     0   (0)| 00:00:01 |
|* 12 |    TABLE ACCESS BY INDEX ROWID| VZOREK        |     1 |    25 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Z výpisu je vidět, že v rámci `SELECT`u dojde nejprve k provedení příkazu `GROUP BY` a poté k `JOIN`ům, které zabírají značnou část (Nested loops). Dále nám výpis říká, že k tabulce Kontrola je přistupováno plně, následně se přistoupí k tabulce Sportovec dle identifikátoru řádku v datovém souboru a vyhledají se odpovídající záznamy skenováním indexů PK, což je vidět ve sloupci Name, že se neprochází tabulka Sportovec, nýbrž PK_Sportovec. To je dáno tím, že tabulku spojujeme přes primární klíč, což nám zaručuje přístup pouze k jednomu řádku, proto také `INDEX UNIQUE SCAN`. Stejným způsobem se pak přistoupí k a vyhledají hodnoty z tabulky Vzorek pro hlavní i kontrolní.

Vzhledem k tomu, že ústřední tabulkou tohoto dotazu je Kontrola, která se spojuje jak s tabulkou Sportovec, tak se Vzorek, a vidíme, že se k ní přistupuje plně, byť nás zajímají jen hodnoty pro spojení s těmito tabulkami, rozhodli jsme se index využít na ni. Jako atributy pro indexování jsme zvolili HlavniVzorekId a KontrolniVzorekId, neboť na základě nich spojuje tabulku Vzorek, tedy se na základě nich vyhledává a proto jsou vhodnými kandidáty na optimalizaci indexem. Dále byl zvolen atribut SportovecId, neboť ten se zase využívá pro hledání v tabulce Sportovec. S využitím indexu

```
CREATE INDEX KontrolaIndex ON Kontrola(SportovecId, HlavniVzorekId,
KontrolniVzorekId);
```

výsledek explain plan vypadá následovně:

```
-----------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |     1 |   131 |     2  (50)| 00:00:01 |
|   1 |  HASH GROUP BY                |               |     1 |   131 |     2  (50)| 00:00:01 |
|   2 |   NESTED LOOPS                |               |     1 |   131 |     1   (0)| 00:00:01 |
|   3 |    NESTED LOOPS               |               |     1 |   131 |     1   (0)| 00:00:01 |
|   4 |     NESTED LOOPS              |               |     1 |   106 |     1   (0)| 00:00:01 |
|   5 |      NESTED LOOPS             |               |     1 |    81 |     1   (0)| 00:00:01 |
|   6 |       INDEX FULL SCAN         | KONTROLAINDEX |     1 |    39 |     1   (0)| 00:00:01 |
|   7 |       TABLE ACCESS BY INDEX ROWID| SPORTOVEC  |     1 |    42 |     0   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN      | PK_SPORTOVEC  |     1 |       |     0   (0)| 00:00:01 |
|   9 |      TABLE ACCESS BY INDEX ROWID | VZOREK     |     1 |    25 |     0   (0)| 00:00:01 |
|* 10 |       INDEX UNIQUE SCAN       | PK_VZOREK     |     1 |       |     0   (0)| 00:00:01 |
|* 11 |     INDEX UNIQUE SCAN         | PK_VZOREK     |     1 |       |     0   (0)| 00:00:01 |
|* 12 |    TABLE ACCESS BY INDEX ROWID| VZOREK        |     1 |    25 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Lze pozorovat, že pouze skenováním indexů místo průchodu celou tabulkou došlo k výraznému snížení cen. To, že se využilo indexů, si můžeme zkontrolovat na řádku 6. Je vidno, že na tabulku Kontrola již vůbec nebylo nahlédnuto. Na našich několika testovacích datech rozdíl v rychlosti sice necítíme, nicméně u většího množství by to bylo znát.

Přidáním dalšího indexu třeba pro sportovce nebo vzorky bychom si už nepomohli, neboť díky spojení s primárními klíči, jak bylo popsáno výše, dochází ke skenování indexů (jen primárních klíčů) a nedochází k průchodu celé tabulky.

Přístupová práva

K našemu systému by dle diagramu případů užití měli mít přístup dva typy uživatelů, a to komisař a laboratorní pracovník (když opomineme systémového správce). Rozhodli jsme se pro ukázku implementovat přístup laboratorního pracovníka.

Nejprve pomocí příkazu `CREATE` vytvoříme nového uživatele a následně mu přiřadíme všechna práva na tabulkách, ke kterým potřebuje mít přístup.

Materializovaný pohled

Slouží k rychlejšímu a jednoduššímu přístupu k datům v systému, a přesně za tímto účelem byl vytvořen. Jedná se o přehled, který by měl laboratorní pracovník mít. Kvůli tomu využíváme při vytváření mimo jiné `CACHE` a `REFRESH FAST ON COMMIT`.

Jsou sledovány změny na odpovídajících tabulkách: Vzorek, LaboratorniPracovnik a jejich propojovací tabulka LaboratorniPracovnik_Vzorek.

Jak vyplývá z výše uvedeného, při změně bázové tabulky (např. zadání výsledku vzorku) je potřeba použít příkaz `COMMIT`, aby se výsledek promítl i v materializovaném pohledu. V opačném případě se změny provedou, ale nebudou při používání materializovaného pohledu (dotazování na něj) pro uživatele (laboratorního pracovníka) viditelné.