Vysoké učení technické v Brně
Fakulta Informačních technologií

Databázové systémy
Zadání č. 50 - Klub anonymních alkoholiků

David Jahoda (xjahod05)
Jan Smejkal (xsmejk27)
28.4.2020

Obsah

Zadání

50. Klub Anonymních Alkoholiků

Navrhněte informační systém, který bude podporovat anonymní alkoholiky k organizaci sezení a evidenci vypitého alkoholu. Systém uchovává základní informace o alkoholicích, jako je jejich věk, pohlaví, patrony, kteří je podporují a se kterými se nepravidelně scházejí na různých místech a v různých datech a rovněž i informace o odbornících, kteří na ně lékařsky dohlíží. Odborníci musí mít patřičnou expertízu pro pečování o alkoholiky, a mít minimální lékařskou praxi, která je v systému evidována. Patronem však může být kdokoliv. Pravidelně se konají sezení, kterých se účastní až dvanáct alkoholiků a navíc mohou být přítomni jak patroni tak i odborníci a dohlížet nad diskuzí. U každého sezení nás zajímá datum, čas, a místo konání. Těchto míst je pouze několik oficiálních a dedikovaných. Každé sezení je vedeno jednou osobou. Neformální schůzky s patrony však mohou být organizovány v libovolné lokalitě. Alkoholici se musí alespoň třikrát ročně účastnit nějakého sezení, a v případě, že se více jak tři měsíce nedostaví na žádné sezení je jim systémem zaslána upomínka. U alkoholiků jsou pravidelně (i nepravidelně a nečekaně) prováděny kontroly odborníky, na kterých se měří míra alkoholu v jejich krvi. Tato míra vypitého alkoholu je pak evidována do systému, rovněž s původem a typem vypitého alkoholu. Alkoholici však mohou sami zaevidovat (ze špatného svědomí), že alkohol požili (tedy mimo prováděné kontroly) a tuto informaci rovněž přidat do systému.

Převod generalizace/specializace do schéma relační databáze

V projektu jsme využili tři generalizace/specializace (dále jako gen/spec).

První z gen/spec jsou tabulky `Patron` a `Odbornik`, které jsou specializací tabulky `Pracovnik`. Tuto gen/spec jsme vyřešili vytvořením tabulky pro každou z rolí, kde specializace `Patron` a `Odbornik` mají jako klíč referenci k primárnímu klíči `Pracovníka`. U `Odborníka` jsou navíc atributy `kvalifikace` a `delka_praxe`, u `Pracovníka` je zbytek atributů.

Druhou z gen/spec je Místo konání (`Misto_konani`), které má specializace `neoficiální` a `oficiální`. Tuto gen/spec jsme řešili atributem `oficialni` v tabulce `Misto_konani`, který vyjadřuje oficiálnost místa.

Třetí a poslední z gen/spec je `Sezení`, které má specializace podobně jako druhá gen/spec. Řešení je provedeno stejným stylem.

Triggery

max_12

První z triggerů `max_12` se využívá pro zajištění kardinality maxima 12 alkoholiků na jednom sezení. Spustí se při přidání záznamu v tabulce `Alkoholik_se_ucastni_sezeni`.

Tento trigger zjistí počet alkoholiků na sezení, kam chce přidávat. Pokud by přidání nového alkoholika překročilo povolené maximum 12, trigger zamezí provedení `INSERT`u.

Případ aktualizování dat tento trigger neošetřuje, protože se neočekává úprava dat v této tabulce. Případ špatně zadaného záznamu účasti alkoholika se vyřeší odstraněním a novým přidáním záznamu.

Při překročení kapacity je vyvolána programátorem definovaná výjimka `kapacita_chyba`, kde je následně vypsána chybová zpráva s návratovým kódem `-20005`.

neni_budouci

Druhý trigger `neni_budouci` slouží k ověření, že datum přidané kontroly není v budoucnosti, protože by nedávalo smysl přidávat kontrolu, která ještě neproběhla.

Trigger porovná datum nově přidávané položky s aktuálním datem na serveru. Pokud je vyšší, vyvolá aplikační chybu `datum_chyba`, která vypíše příslušnou chybovou zprávu s návratovým kódem aplikace `-20006` znemožní dokončení `INSERT`.

Procedury

Podil_Nalezu_Alkoholu

První procedura `Podil_Nalezu_Alkoholu` slouží ke zjištění procentuálního podílu kontrol s nálezem alkoholu v krvi ze všech provedených kontrol u alkoholika zadaného parametrem.

Nejdříve zjistí, jestli alkoholik má nějaké záznamy o provedené kontrole. Pokud nemá, ukončí provádění procedury s chybou `neexistuji_zaznamy` a kódem `-20002`. Pokud nějaké záznamy má, spustí podproceduru `Podil_Nalezu_Alkoholu_Vypocet`, kde za využití kurzoru `kontrola` vypočítáme podíl pozitivních a negativních nálezů alkoholu.

Obsazenost_sezeni

Druhá procedura `Obsazenost_sezeni` zjistí a vypíše obsazenost na sezeních vedených pracovníkem zadaným v argumentu.

Stejně jako u první procedury i zde je využíván kurzor, pro procházení jednotlivých záznamů získaných příkazem SELECT, u kterého jsou záznamy seskupeny po jednotlivých sezeních (na základě atributu `sezeniID`).

Procedura využívá pomocné proměnné a to, `akt_sezeni` kam si uloží ID aktuálně zpracovávaného sezení a proměnnou `pocitadlo`, do které postupným procházením zaznamenává počet alkoholiků na sezení. Jakmile kurzor narazí na jiné sezení než to, které je uloženo v `akt_sezeni`, vypíše zprávu o sezení, které bylo právě zpracováváno do pomocných proměnných. Po tomto výpisu se pomocné proměnné vynulují a uloží se do nich informace o novém sezení z kurzoru.

Pro případ, že se sezení neúčastnil žádný alkoholik, bylo použito spojení tabulek pomocí `FULL OUTER JOIN` namísto `FULL INNER JOIN`, který pro tento případ nevracel záznam s `alkoholikID=null`. Procedura v tomto případě neinkrementuje proměnnou `pocitadlo` a při dalším průchodu již hodnota `akt_sezeni` a kurzoru nesouhlasí a tudíž se vypíše zpráva i pro prázdné sezení.

Explain Plan

Pro ukázku využití jsme použili tento příkaz:

```
SELECT Kontrola.alkoholikID, MAX(Kontrola.promile), Poziti.typ
 FROM Kontrola JOIN Poziti ON Kontrola.pozitiID = Poziti.pozitiID
 GROUP BY Kontrola.alkoholikID, Kontrola.promile, Poziti.typ
 ORDER BY Kontrola.promile DESC;
```

První provedení příkazu (bez optimalizací) bylo neefektivní:

- 2x bylo použito prohledávání spojených tabulek vnořenými cykly => plýtvání časem, cena celkem 12
- vygenerován pohled (`VIEW`, cena 4)
- 1x `GROUP BY` děláno hashem (cena 4) následováno 1x `TABLE ACCESS FULL` (plným průchodem tabulky `Kontrola` kvůli spojení tabulek a výběru informací pro návrat `SELECT`u, cena 3)
- následně byl pro spojení tabulek použit `INDEX UNIQUE SCAN` pro tabulku `Poziti`
- a nakonec je pro návrat příkazu `SELECT` přistupováno do tabulky `Poziti` do atributu `typ` (`TABLE ACCESS BY INDEX ROWID`, cena 1)

Níže se nachází výstup prvního (neoptimalizovaného) EXPLAIN PLANu:

```
| Id  | Operation                     | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |            |     5 |   655 |     7  (29)| 00:00:01 |
|   1 |  SORT GROUP BY                |            |     5 |   655 |     7  (29)| 00:00:01 |
|   2 |   NESTED LOOPS                |            |     5 |   655 |     6  (17)| 00:00:01 |
|   3 |    NESTED LOOPS               |            |     5 |   655 |     6  (17)| 00:00:01 |
|   4 |     VIEW                      | VW_GBC_5   |     5 |   260 |     4  (25)| 00:00:01 |
|   5 |      HASH GROUP BY            |            |     5 |   195 |     4  (25)| 00:00:01 |
|   6 |       TABLE ACCESS FULL       | KONTROLA   |     5 |   195 |     3   (0)| 00:00:01 |
|*  7 |     INDEX UNIQUE SCAN         | PK_POZITI  |     1 |       |     0   (0)| 00:00:01 |
|   8 |    TABLE ACCESS BY INDEX ROWID| POZITI     |     1 |    79 |     1   (0)| 00:00:01 |
```

Jako řešení tohoto velkého množství přístupů jsme zavedli indexy pro tabulku `Kontrola`, stejně tak pro tabulku `Poziti`, které se nazývají `poziti_typ` a `kontrola_index`.

Výsledný EXPLAIN PLAN vypadá mnohem příznivěji:

```
| Id  | Operation           | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                |     5 |   655 |     3  (34)| 00:00:01 |
|   1 |  SORT GROUP BY      |                |     5 |   655 |     3  (34)| 00:00:01 |
|*  2 |   HASH JOIN         |                |     5 |   655 |     2   (0)| 00:00:01 |
|   3 |    VIEW             | VW_GBF_6       |     4 |   316 |     1   (0)| 00:00:01 |
|   4 |     INDEX FULL SCAN | POZITI_TYP     |     4 |   316 |     1   (0)| 00:00:01 |
|   5 |    VIEW             | VW_GBC_5       |     5 |   260 |     1   (0)| 00:00:01 |
|   6 |     HASH GROUP BY   |                |     5 |   195 |     1   (0)| 00:00:01 |
|   7 |      INDEX FULL SCAN| KONTROLA_INDEX |     5 |   195 |     1   (0)| 00:00:01 |
```

Zbyly 2x `INDEX FULL SCAN`y (díky využití dříve zmíněných indexů, cena celkem 2), `HASH JOIN`, `HASH GROUP BY` a `SORT GROUP BY` s celkovou cenou 13 oproti původní ceně 38.

Přístupová práva

Byla přidělena veškerá práva ke všem tabulkám, se kterými v databázi pracujeme, stejně tak byla přidělena práva spouštění všech procedur.

Materializovaný pohled

Pro demonstraci využití pohledů jsme využili dotaz ke zjištění adresy konání sezení. Tento dotaz je realizován spojením dvou tabulek, a to pomocí INNER JOIN na základě shody klíčů `mistokonaniID`.

Vypadá takto:

```
SELECT sezeniID, Sezeni.mistokonaniID, ulice, cislo_popisne, mesto, psc,
    Sezeni.rowid as sezeni_rowid, Misto_konani.rowid as
misto_konanani_rowid
    FROM Sezeni INNER JOIN Misto_konani ON Sezeni.mistokonaniID =
Misto_konani.mistokonaniID;
```

Samotný pohled jsme implementovali takto:

Pro materializovaný pohled jsme nejdříve vytvořili 2 logy pro tabulky `Misto_sezeni` a `Misto_Konani`. Těmto tabulkám se konkrétněji budeme věnovat později.

Následně jsme vytvořili materializovaný pohled `pohled_sezeni_adresa`. U tohoto pohledu jsme se rozhodli nastavit tyto vlastnosti:

- CACHE pro povolení optimalizace čtení z našich pohledů
- BUILD IMMEDIATE pro naplnění pohledu ihned po vytvoření
- REFRESH FAST ON COMMIT pro akutalizaci dat v pohledu po změně zdrojových záznamů, z kterých vychází pohled. Tato aktualizace se provede při COMMITu.
- ENABLE_QUERY_REWRITE pro povolení využívání pohledu optimalizátorem

Na našem příkladu při využití EXPLAIN PLAN jsme bez použití materializovaného pohledu dosáhli tohoto výsledku s celkovou cenou 18:

```
| Id  | Operation          | Name         | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT   |              |     6 |   942 |     6   (0)|
|*  1 |  HASH JOIN         |              |     6 |   942 |     6   (0)|
|   2 |   TABLE ACCESS FULL| MISTO_KONANI |     3 |   357 |     3   (0)|
|   3 |   TABLE ACCESS FULL| SEZENI       |     6 |   228 |     3   (0)|
```

Po povolení využívání pohledů k optimalizaci dotazů jsme dosáhli výsledku níže s celkovou cenou 6:

```
| Id  | Operation                    | Name                 | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------------
-------
|   0 | SELECT STATEMENT             |                      |     6 |   324 |     3   (0)|
|   1 |  MAT_VIEW REWRITE ACCESS FULL| SEZENI_KOMPLET_ADRESA |     6 |   324 |     3   (0)|
```

Jak jste mohli vidět, neoptimalizované řešení použilo 2x plný přístup do tabulek oproti optimalizovanému řešení, kde byl pouze využit 1x plný přístup do materializovaného pohledu.

Závěr

V projektu jsme si vyzkoušeli práci s databázemi. Kromě prezentací jsme využili zejména manuálových stránek Oracle, případně různých fór.

Největší překvapení pro nás bylo, že komentář za příkazem

```
EXECUTE Obsazenost_sezeni(2); --komentar co zpusobuje chybu
```

skončil neočekávanou chybou, zatímco bez komentáře proběhl tentýž příkaz nad stejnými daty úspěšně.

V aplikaci jsme nad rámec zadání zavedli tvorbu primárních klíčů pomocí sekvencí, které začínají od 1 a zvyšují se vždy o hodnotu 1. Tyto sekvence jsme pomocí omezení `DEFAULT` použili pro generování primárních klíčů jednotlivých tabulek.

Níže můžete vidět ukázku kódu generování sekvencí pro primární klíče v tabulce `Pracovnik`:

```
    pracovnikID integer DEFAULT Pracovnik_seq.NEXTVAL,
```