Vysoké učení technické v Brně
Fakulta informačních technologií

Databázové systémy 2017-2018

Dokumentace

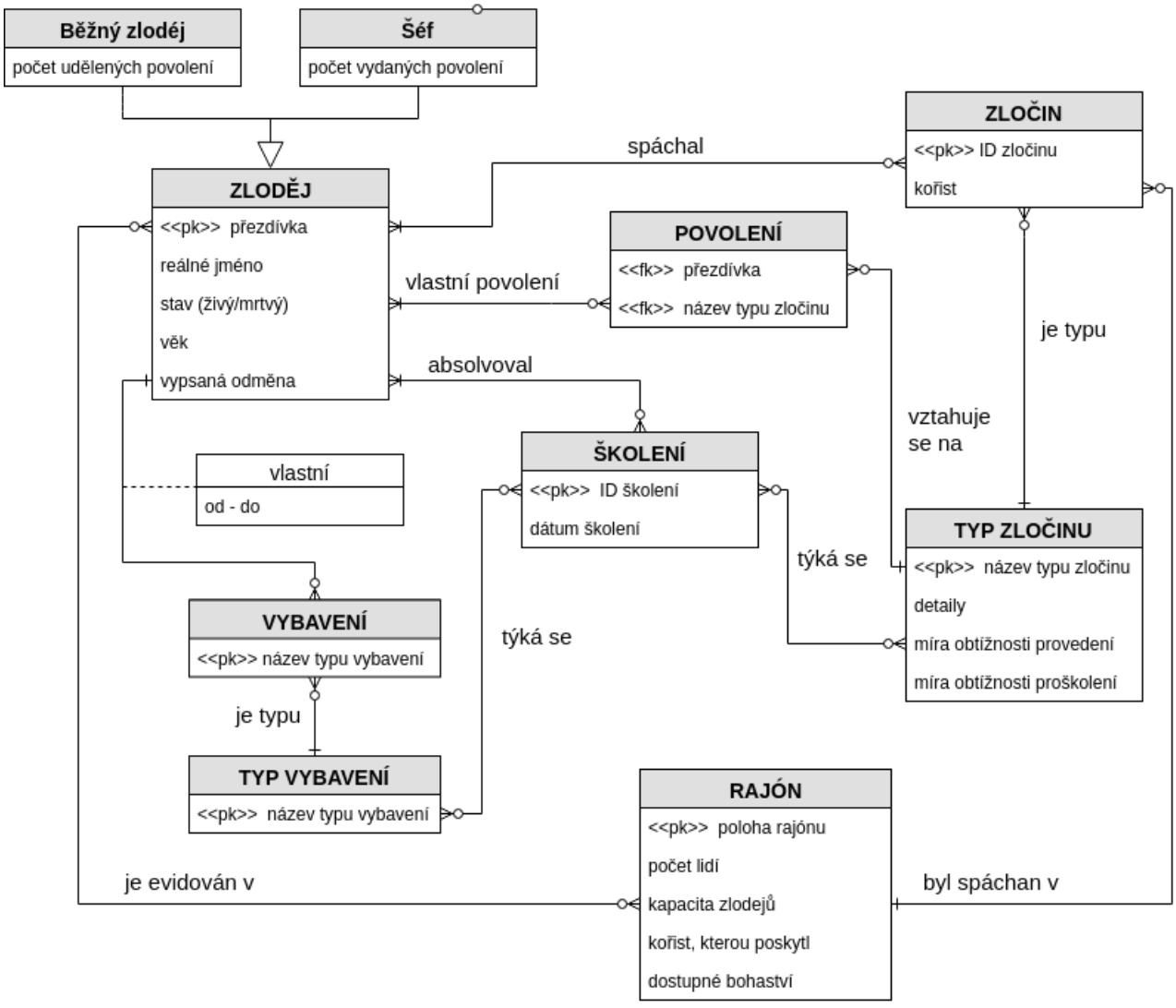
30. března 2018

Petr Marek (xmarek66)
Jakub Štefanišin (xstefa22)

1 Zadání

Cech zlodějů chce zefektivnit svoji práci a zadal výběrové řízení na vytvoření informačního systému pro evidence krádeží a loupeží. Zloději udávají své reálné jméno (někteří jsou však bezejmenní), úředně potvrzenou přezdívku (např. Vilda Dlouhoprsťák), věk, stav (mrtvý, živý), vypsanou odměnu, a navíc vlastní řadu vybavení, které jsou různých typů (zbraně, náčiní, pasti,.), pro které musí být důkladně proškolení. Školení se rovněž vztahuje na typy zločinů, přičemž u každého typu nás zajímají detaily, míry obtížnosti provedení/proškolení. Vybavení se může dědit a předávat dál (např. v případě smrti), přičemž evidujeme od kdy do kdy zloděj dané vybavení vlastnil. Z důvodu regulace zločinu ve městě se vydávají povolení (či poukázky) na provedení zločinu určitého typu. Tyto poukázky se uplatňují na konkrétní zločin, u kterého navíc evidujeme, ve kterém rajónu byl proveden a jakou kořist poskytl. Daný zločin pak mohlo provést více zločinců. Každý zloděj eviduje své rajóny, ve kterých se pohybuje, přičemž může mít více rajónů. Evidujte rovněž základní informace o rajónech, jako je pozice, počet lidí, kapacita zlodějů (tzn. kolik se jich tam uživí), celkově dostupné bohatství a pod. Systém pravidelně tiskne žebříček nejlepší zlodějů, podle míry provedených zločinů během měsíce, i podle absolutního počtu kořistí.

2 Finální ER diagram



3 Popis řešení

3.1 Check

Pomocí funkcí Check kontrolujeme, zda hodnoty vkládané do některých z tabulek odpovídají určitým pravidlům. V našem případe kontrolujeme, zda hodnoty primárního klíče pro zločin a školení jsou větší než 0.

3.2 Triggery

V našem řešení jsme implementovali dva triggery. První inkrementuje primární klíč (ID_zločin) v tabulce zločin za pomoci sekvence, která si uchovává informaci o posledním přiřazeném čísle a která začíná na čísle 1.

Druhý trigger zvyšuje počítadlo udělených povolení u běžného zlodeje za každé nové povolení, které mu je uděleno.

3.3 Procedury

První ze dvou implementovaných procedur **detail_zlocin** nám po zadání ID zločinu ověří, zda zločin se zadaným ID existuje, a pokud ano, vypíše o něm všechny detaily, které jsou v tabulce evidovány.

Druhá procedura s názvem **pocty_povoleni** slouží pro výpis běžných zlodějů, kteří mají stejný, větší nebo menší počet udělených povolení, než zadaný zloděj.

3.4 EXPLAIN PLAN

Pro demonstraci příkazu EXPLAIN PLAN jsme využili příkazu SELECT v kombinaci s agregační funkcí SUM, spojení dvou tabulek s klauzulí WHERE a použití klauzule GROUP BY. Právě na dotazu s klauzulí WHERE je nejlépe pozorovatelné urychlení databáze použitím indexů. Vytvoří se totiž index pouze s daty, které jí odpovídají. Tím se výrazně sníží počet čtení disku při prohledávání dat a provedení dotazu se urychlí.

3.4.1 Bez použití indexu

Nejprve ukázka EXPLAIN PLAN bez použití indexu: Na této ukázce je dobře vidět, že při provedení

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
Plan hash value: 3867896995

-------------------------------------------------------------------------------------------
| Id  | Operation              | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |               |     4 |   192 |     7  (29)| 00:00:01 |
|   1 |  SORT ORDER BY         |               |     4 |   192 |     7  (29)| 00:00:01 |
|   2 |   HASH GROUP BY        |               |     4 |   192 |     7  (29)| 00:00:01 |
|*  3 |    HASH JOIN OUTER     |               |     4 |   192 |     5   (0)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL  | ZLOCIN        |     3 |    78 |     3   (0)| 00:00:01 |
|   5 |     INDEX FAST FULL SCAN| SYS_C00123333 |     5 |   110 |     2   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("A"."ID_ZLOCINU"(+)="B"."ID_ZLOCINU")
   4 - filter("B"."KORIST">1000)
```

tohoto dotazu se nejprve získají sloupce první tabulky (SELECT). Poté se z druhé tabulky vyberou potřebné sloupce a navíc vyfiltrují řádky pomocí klauzule WHERE. Tato tabulka se nakonec spojí s první tabulkou. Všimněte si čtvrtého řádku tabulky - TABLE ACCESS FULL. Znamená, že tabulka ZLOCIN se pro nalezeni řádků, kde platí klauzule WHERE, prohledává úplně celá. Nicméně Oracle zde využil HASH JOIN a HASH GROUP BY, které využívají hashovací tabulky a významě přispějí k urychlení databáze.

3.4.2 S použitím indexu

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
Plan hash value: 3291745247

-------------------------------------------------------------------------------------------------------
| Id  | Operation                            | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                     |               |     4 |   192 |     6  (34)| 00:00:01 |
|   1 |  SORT ORDER BY                       |               |     4 |   192 |     6  (34)| 00:00:01 |
|   2 |   HASH GROUP BY                      |               |     4 |   192 |     6  (34)| 00:00:01 |
|*  3 |    HASH JOIN OUTER                   |               |     4 |   192 |     4   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED| ZLOCIN        |     3 |    78 |     2   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                | KORIST_INDEX  |     3 |       |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
------------------------------------------------------------------------------------------------------------
|   6 |     INDEX FAST FULL SCAN             | SYS_C00123333 |     5 |   110 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - access("A"."ID_ZLOCINU"(+)="B"."ID_ZLOCINU")
   5 - access("B"."KORIST">1000)
```

Zde je u operací patrný rozdíl na čtvrtém řádku. TABLE ACCESS BY INDEX ROWID BATCHED znamená, že tabulka ZLOCIN se již neprohledává celá, ale podle indexu, který jsme vytvořili. Ovšem platíme za to cenou prohledávání indexu - INDEX RANGE SCAN, kde vyhledáváme řádek, kde platí klauzule WHERE. Navíc využití procesoru je při použití indexu vyšší. Velký posun k lepšímu můžeme pozorovat na počtu čtení z disku. U každé operace kromě INDEX FAST FULL SCAN je o jeden nižší, než v případě nepoužití indexu.

4 Materializovaný pohled

Pro jednoho člena týmu byl vytvořen materializovaný pohled, který zobrazuje pouze počet školení v daný den. V praxi by mohlo jít například o organizátora školení. Tomuto členovi byla také udělena práva na všechny tabulky databáze.

Pro optimalizaci v materializovaném pohledu byl využit log. Ten vytvoří zvláštní tabulku, kam se ukládají informace o změnách pohledu, které se poté využijí pro zobrazení aktualizovaného pohledu, namísto jeho znovuvytvoření. Této optimalizace se využije použitím možnosti REFRESH FAST. Ta zajistí inkrementální změny pohledu na základě změn dat. Pro další možné urychlení byly využity možnosti CACHE a ENABLE QUERY REWRITE.