Vysoké učení technické v Brně

Fakulta informačních technologií

Dokumentace projektu do předmětu Databázové systémy

Zadání číslo 48. - Kočičí informační systém

Petr Marek (xmarek69), Jan Fridrich (xfridr07)

29.04.2018

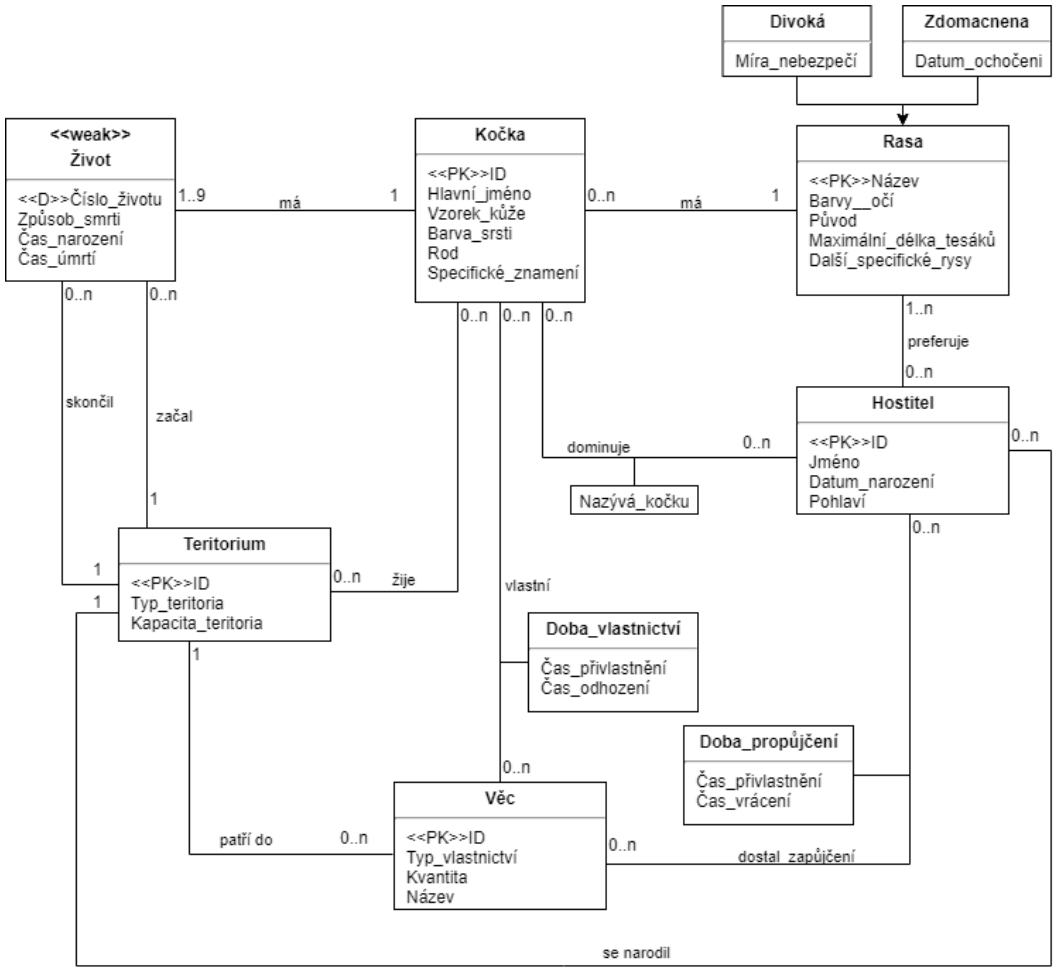
Obsah:

Zadání:

Zvolili jsme **48. Kočičí Informační Systém**:

> Kočky chtějí zefektivnit jejich dominanci lidského světa, a proto Vám zadaly vytvořit KIS (Kitty Information System). Tento systém uchovává informace o jednotlivých rasách koček, jejich specifické rysy, jako možné barvy očí, původ, maximální délku tesáků apod. a u konkrétních koček pak jejich hlavní jméno, vzorek kůže, barvu srsti apod. Každá kočka má právě devět životů, nicméně v systému vedeme pouze ty, které již proběhly a aktuálně probíhají, a vedeme u nich informaci o délce života, místo narození a případně (u minulých životů) o místě (v rámci, kterého teritoria) a způsobu smrti. Kočky jsou samozřejmě majetnické a chtějí si vést všechny teritoria (máme teritoria různých typů, jako např. jídelna, klub, .), ve kterých se kdy pohybovaly a které věci si přivlastnily a v kterém intervalu je vlastnily (kočky se lehce znudí a své věci prostě zahodí). Systém rovněž vede informace o jejich hostitelích, kteří jim slouží. Veďte u nich jejich základní informace (jméno, věk, pohlaví, místo bydlení, .), které rasy koček preferují a rovněž jméno, kterým kočku nazývali (např. Pan Tlapoň, Bublina, Gaston, .). Některé vlastnictví koček však mohou být propůjčována svým hostitelům. Současně veďte informaci (pokud je přítomna) o teritoriu v rámci kterého se vlastnictví nachází, typ vlastnictví (hračka, cokoliv,.) a jeho kvantitu. Jednotlivá teritoria však mají omezenou kapacitu na kočky a v případě překročení (doslova) se kočky přesídlí. Systém umožňuje kočkám zasílat pravidelné novinky o životech ostatních koček a nových dostupných hostitelích, ke kterým by se mohly přesídlit a věcech, které by mohly zabrat.

ER Diagram (liší se od prvního odevzdání):



Převod ER diagramu do SQL scriptu

Nejdříve jsme si vytvořili tabulky KOCKA, RASA, HOSTITEL, ZIVOT, VEC a TERITORIUM, které jsme naplnili potřebnými položkami podle zadání a aby nám vyhovovala práce s nimi. Jako další jsme vytvořili tabulky pro generalizaci DIVOKA a ZDOMACNENA, které jsme propojili s rasou, tak že jsme jim dali jako cizí klič *nazev_rasy* používaný jako primární klíč v tabulce RASA. A nakonec jsme vytvořili tabulky ZIJE, VLASTNÍ, DOMINUJE, PREFERUJE, ZAPUJCENO, které slouží jako spojení výše vyjmenovaných tabulek pomocí vztahů n ku n.

TRIGGERY:

Implementovali jsme dva typy triggerů.

První typ triggeru nastavuje ID u tabulek KOCKA, HOSTITEL, ID a VEC. Trigger se spustí vždy při přidání nové položky do tabulky.

Druhý typ triggeru kontroluje počet životů v tabulce ZIVOT. Kontroluje zda jsou životy nastavovány od 1 a také kontroluje jestli maximální hodnota číslo životu nepřesáhlo číslo 9, protože pak by došlo k porušení zadání

Procedury:

V našem řešení máme proceduru na vypsaní všech míst kde se narodili kočky. A proceduru na vypsání všech věcí hostitele při zadaném ID kočky.

EXPLAIN PLAN:

Úlohou EXPLAIN PLAN je zobrazení postupnosti realizace operaci optimalizátorem Oracle pro vybraný SQL příkaz a poskytnutí informací o výkonnostní ceně pro každou operaci a čas na jejich vykonání.

Pro příklad je použit příkaz na zobrazení počtu koček podle jejich barvy očí:

```
SELECT R.Barvy_oci, COUNT(*) AS Pocet_kocek
FROM Rasa R, Kocka A
WHERE R.Nazev_rasy = A.Nazev_rasy
GROUP BY R.Barvy_oci;
```

Ukázka výstup:

```
BARVY_OCI                    POCET_KOCEK
---------------------------- -----------
zelena                                 3
modra                                  2
```

A po zavolání EXPLAIN PLAN pro tento příkaz dostaneme tuto tabulku, kde můžeme vidět postupné provádění operací SELECT, GROUP BY, JOIN a následný přístup k tabulkám

```
-----------------------------------------------------------------------------
| Id  | Operation           | Name  | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |       |     5 |   255 |     7  (15)| 00:00:01 |
|   1 |  HASH GROUP BY      |       |     5 |   255 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN         |       |     5 |   255 |     6   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| RASA  |     3 |   102 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| KOCKA |     5 |    85 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

INDEX:

Použití indexování může být výhodné v případě častého přístupu k určité tabulce. Někdy to ovšem může vést k opaku, a to k zpomalení při nevyužívání indexovaných tabulek.

Pokud se vrátíme k našemu příkladu často přistupujeme k tabulkám RASA a KOČKA, z tohoto důvodu jsme použili index:

```
CREATE INDEX index_rasa ON Rasa(Barvy_oci,Nazev_rasy);
```

A následný výstup při použití opět EXPLAN PLAN:

```
-----------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     5 |   255 |     5  (20)| 00:00:01 |
|   1 |  HASH GROUP BY      |             |     5 |   255 |     5  (20)| 00:00:01 |
|*  2 |   HASH JOIN         |             |     5 |   255 |     4   (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN  | INDEX_RASA  |     3 |   102 |     1   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| KOCKA       |     5 |    85 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------
```

Z tohoto můžeme zjistit, že došlo ke zrychlení téměř u všech operací (zde hovoříme o zrychlení tak málem, že to můžeme pozorovat pouze na využití procesoru)

Materializovaná pohled:

Jde o fyzickou tabulku definovanou příkazem SELECT, což zvyšuje rychlost práce s databází. Samozřejmě se toto vyplatí jenom u SELECTu, který je často používán. Při práci v tomto pohledu se sledují změny v této tabulce a jejich provedení se uskuteční až po narazení na příkaz COMMIT, do té doby zůstává tabulka beze změny.

V našem scriptu se nachází materializovaný pohled jako propojení tabulek KOCKA a RASA, při přidávání nové kočky. A práce s příkazem COMMIT je ukázána, tak že i po projetí přes příkaz INSERT se hodnoty v tabulce nezměnily, ale po provedení příkazu COMMIT se všechny změny uložili do tabulky.