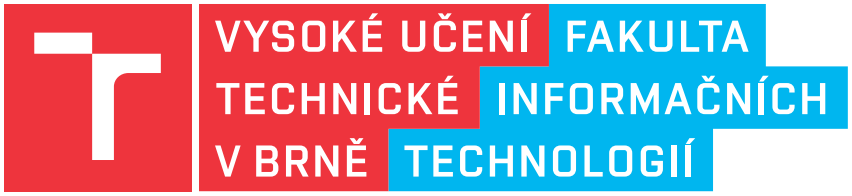

Projekt 5. časť
Projektová dokumentácia
Zoologická záhrada
Databázové systémy

27. dubna 2022

Samuel Dobroň xdobro23@stud.fit.vutbr.cz
Juraj Remeň xremen02@stud.fit.vutbr.cz

Obsah

1 Zadanie

Navrhněte informační systém pro zoologickou zahradu. V zoologické zahradě jsou živočichové umístěni do klecí, výběhů, či do klecí v pavilonech. Živočichové jsou děleni podle třídy, řádu, čeledě, rodu a druhu (např. lama alpaka je v třídě savců, řádu sudokopytníků, v čeledi velbloudovitých, v rodu lama a druhu alpaka). Pro zjednodušení předpokládejte striktně hierarchické dělení živočichů a to, že každý živočich je příslušníkem právě jednoho druhu. Jeden druh živočicha může být v několika výbězích či klecích, a naopak, v jednom výběhu či kleci může být více různých druhů. Systém musí být schopný vyhledávat živočichy podle jejich příslušností do jednotlivých kategorií. Pro každého živočicha je třeba uchovávat informace o datu narození (a případně úmrtí), jméno, historii výsledků měření (hmotnosti, rozměrů, ...), apod.

2 Implementácia

2.1 Úvod

Naša implementácia začala vytvorením tabuliek, ktoré odzrkadľovali návrh v podobe ER diagramu. Nasledovalo určenie primárnych kľúčov tabuliek, cudzích kľúčov a vytváranie tabuliek, ktoré zobrazovali vzťahy medzi entitami v ER diagrame. Po vytvorení tabuliek sme ich naplnili ukážkovými dátami.

Ďalšou fázou bolo vytvorenie SELECT dotazov, uvádzame len pár z nich, všetky ostatné sú okomentované a uvedené v kóde.

2.1.1 Spojenie 2 tabuliek príkazom SELECT

Vypíše meno, priezvisko, nazov pozicie a náplň práce zamestnanca:

```
SELECT meno, priezvisko, nazov, napln_prace FROM zamestnanec NATURAL JOIN
    pozicia WHERE pozicia.ID_pozicie = zamestnanec.pozicia;
```

2.1.2 Spojenie 3 tabuliek príkazom SELECT

Vypíše mená živočíchov, kde sú umiestnené a o aký typ umiestnenia ide:

```
SELECT meno, nazov, CASE WHEN interakcia is not null THEN 'Vybeh' ELSE
    'Pavilon' END AS Typ_umiestnenia FROM zivocich NATURAL JOIN
    bol_umiestneny NATURAL JOIN umiestnenie;
```

2.1.3 SELECT s použitím EXISTS

Vypíše ID, mená a priezviská zamestnancov, ktorí neošetrujú žiadne zviera:

```
SELECT ID_zamestnanca, meno, priezvisko FROM zamestnanec Z WHERE NOT
    EXISTS(SELECT * FROM osetruje WHERE osetruje.ID_zamestnanca =
    Z.ID_zamestnanca);
```

2.2 Generalizácia

Pri vytváraní vzťahu generalizácie sme použili jednu tabuľku nadtypu `UMIESTNENIE`, v ktorej podľa istého kritéria, v našom prípade ide o možnosť interakcie so zvieratami, rozlišujeme o aký typ umiestenia ide.

2.3 EXPLAIN PLAN a použitie INDEX

Úlohou `EXPLAIN PLAN` je zobrazenie postupnosti operácií za pomoci optimalizátorov.

2.3.1 Naša implementácia príkazu EXPLAIN PLAN

Na základe poskytnutých dát o cene a čase jednotlivých operácií sme si vybrali, tabuľkové hodnoty bude najlepšie optimalizovať.

```
   EXPLAIN PLAN FOR
SELECT Z.ID_zivocicha, Z.meno, COUNT(*) osetrovatelov
FROM zivocich Z
NATURAL JOIN
   typ_zivocicha T,
   vlastnost V,
   osetruje O
WHERE T.ID_triedy = 1
   AND V.ID_vlastnosti = 1
   AND V.ID_zivocicha = Z.ID_zivocicha
   AND Z.ID_zivocicha = O.ID_zivocicha
   AND hodnota > 100
GROUP BY Z.ID_zivocicha, meno
HAVING COUNT(O.ID_zamestnanca) < 3;
SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY());
```

Po zavolaní príkazu `EXPLAIN PLAN` a vypísaní na výstup, je možné vidieť tabuľku s postupnosťou vykonávaných operácií a taktiež ich výkonnostnú cenu.

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                |     2 |   510 |    11  (10)| 00:00:01 |
|*  1 |  FILTER                        |                |       |       |            |          |
|   2 |   HASH GROUP BY                |                |     2 |   510 |    11  (10)| 00:00:01 |
|*  3 |    HASH JOIN                   |                |     2 |   510 |    10   (0)| 00:00:01 |
|   4 |     NESTED LOOPS               |                |     1 |   242 |     7   (0)| 00:00:01 |
|   5 |      NESTED LOOPS              |                |     2 |   242 |     7   (0)| 00:00:01 |
|   6 |       NESTED LOOPS             |                |     2 |   432 |     5   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL       | VLASTNOST      |     2 |   312 |     3   (0)| 00:00:01 |
|   8 |        TABLE ACCESS BY INDEX ROWID| ZIVOCICH    |     1 |    60 |     1   (0)| 00:00:01 |
|*  9 |         INDEX UNIQUE SCAN      | SYS_C001554879 |     1 |       |     0   (0)| 00:00:01 |
|* 10 |       INDEX UNIQUE SCAN        | SYS_C001554911 |     1 |       |     0   (0)| 00:00:01 |
|* 11 |      TABLE ACCESS BY INDEX ROWID | TYP_ZIVOCICHA |     1 |    26 |     1   (0)| 00:00:01 |
|  12 |     TABLE ACCESS FULL          | OSETRUJE       |     9 |   117 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

Obrázek 1: EXPLAIN PLAN bez použitia indexov

2.3.2 INDEX

K urýchleniu prevedenia príkazu bol použitý príkaz `INDEX` nakoľko indexovanie môže byť užitočné v prípade častého vyhľadávania v určitej tabuľke. V našej implementácií často používame tabuľky `TYP_ZIVOCICHA`, `OSETRUJE`, `VLASTNOST` a `ZIVOCICH`. Preto sme implementovali príkazy `INDEX` nasledovne:

```
CREATE INDEX trieda_index ON typ_zivocicha(ID_triedy);
CREATE INDEX vlastnost_index ON vlastnost(ID_zivocicha, ID_vlastnosti,
    hodnota);
CREATE INDEX osetruje_index ON osetruje(ID_zivocicha);
```

Pri druhom spustení príkazov s EXPLAIN PLAN za použitia indexov je viditeľné, že operácie sa vykonali rýchlejšie s menšou výkonnostnou cenou. Taktiež miesto prechodu celou tabuľkou bolo zvolené skenovanie indexov. Táto zmena zabezpečila rýchlejšie/nenáročnejšie vykonanie väčšiny operácií v rámci príkazu.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		2	510	6 (17)	00:00:01
* 1	FILTER					
2	HASH GROUP BY		2	510	6 (17)	00:00:01
3	NESTED LOOPS		2	510	5 (0)	00:00:01
4	NESTED LOOPS		1	242	5 (0)	00:00:01
5	NESTED LOOPS		2	432	3 (0)	00:00:01
* 6	INDEX FULL SCAN	VLASTNOST_INDEX	2	312	1 (0)	00:00:01
7	TABLE ACCESS BY INDEX ROWID	ZIVOCICH	1	60	1 (0)	00:00:01
* 8	INDEX UNIQUE SCAN	SYS_C001554879	1		0 (0)	00:00:01
* 9	TABLE ACCESS BY INDEX ROWID	TYP_ZIVOCICHA	1	26	1 (0)	00:00:01
* 10	INDEX UNIQUE SCAN	SYS_C001554911	1		0 (0)	00:00:01
* 11	INDEX RANGE SCAN	OSETRUJE_INDEX	2	26	0 (0)	00:00:01

Obrázek 2: EXPLAIN PLAN s použitím indexov

2.4 Procedúry

V skripte sú implementované dve procedúry:
`kompletny_prehlad()`
`obsadenost(potrebna_plocha_param, typ_umiestnenia)`

Procedúra `kompletny_prehlad()` vypíše aktuálny počet živých alebo mŕtvych zvierat v databáze, priemerný počet zvierat v jednotlivom umietnení a priemerný počet zvierat na ošetrovateľa. Procedúra nepríjma žiadne parametre. Vypočíta potrebné premenné pomocou agregačnej funkcie `COUNT` a pomocou nich prepočíta priemery a sumy. Na DBMS výstup neskôr vypíše tieto údaje. V prípade, že by hodnota počtu umiestnení, alebo počtu ošetrovateľov mala byť rovná 0, tak sa vyvolá výnimka.

Procedúra `obsadenost(potrebna_plocha_param, typ_umiestnenia)` vypíše percentuálnu obsadenosť typu umiestnenia podľa zadanej priemernej potrebnej životnej plochy a typu umiestnenia. Táto informácia je potrebná v prípade obstarania nových zvierat. Procedúra príjma 2 parametre od užívateľa. V prípade chybného vstupu sa vyvolá výnimka. Procedúra prechádza všetkými umiesteniami v databáze a na základe užívateľom zadaného typu vypočíta celkovú využiteľnú plochu daného typu umiestnenia a počet zvierat umiestnených v danom type. Na DBMS výstup neskôr vypíše celkovú percentuálnu obsadenosť typu umiestnenia. V prípade, že by sa celková plocha umiestnenia rovnala 0, tak opäť výnimku.

2.5 Triggre

V skripte sú implementované dva triggre:
"pochovaj_zviera"
"zahashuj_heslo"

Trigger "pochovaj_zviera" má za úlohu nastaviť dobu **do** v bol_umiestneny pri zadaní dátumu úmrtia zvieraťa do databázy a odstrániť jeho ošetrovateľa v osetruje. Zároveň má za úlohu aj kontrolu validity dátumu úmrtia, v prípade, že je dátum úmrtia starší ako dátum narodenia sa vyvolá výnimka.

Trigger "zahashuj_heslo" má za úlohu zahashovať zadané **heslo** v zamestanec pomocou funckie DMBS_OBFUSCATION_TOOLKIT() a nami zadaného **saltu**.

2.6 Materializovaný pohľad

Je to objekt, ktorý poskytuje dáta v rovnakej podobe ako tabuľka. Narozdiel od tabuľky, ktorá obsahuje priamo dáta, pohľad obsahuje len predpis akým spôsobom dáta získať z iných tabuliek alebo pohľadov. Materializovaný pohľad sa niekam ukladá a preto je dotaz rychlejší, čo neplatí o nematerializovanom pohľade.

Náš materializovaný pohľad vytvára tabuľku pocet_zivocichov_v_umiestneniach, ktorá vytvára dáta, konkrétne počet živočíchov v jednotlivých pavilónoch a výbehoch.

2.7 Práva

Pomocou príkazu GRANT na tabuľky nasledovaný zoznamom práv, ktoré chceme udeliť, zvolili sme príkaz ALL (všetky práva), nasledovaný názvom tabuliek, ku ktorým chceme pristupovať a udeliť práva (všetky tabuľky a materializované pohľady) a nakoniec treba uviesť, komu sa tieto práva udeľujú, čo v našom prípade bol kolega *xremen02*.

3 Záver

Pri vypracovaní projektu sme nemali žiadne závažné problémy v komunikácii, keďže sme spolubývajúci a vždy sme sa dohodli na kontrétnom dni a hodine kedy a o koľkej budeme na projekte pracovať. Na komunikáciu sme používali Discord a osobné stretnutia. Na verzovanie projektu sme využili systém Github pre ľahšiu orientáciu. Všetky projekty sme vypracovávali v predstihu, aby sme sa vyhli stresu a nervozite z preťaženia servera. Vývoj projektu prebiehal na školskom serveri v prostredí od JetBrains - Data Grip. Pri vypracovávaní sme využili aj menšiu konzultáciu s pedagógmi.