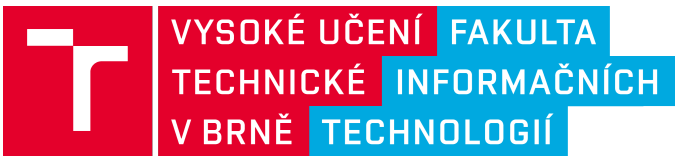

Vysoké učení technické v Brně

Fakulta informačních technologií

Databázové systémy

24. dubna 2022

Projektová dokumentace

SQL databáze

Projekt č.24 Videopůjčovna

Autoři:

Koval Maksym xkoval20

Ivan Golikov xgolik00

Obsah

1 Úvod **2**
1.1 Motivace . . . 2
1.2 Název projektu . . . 2
1.3 Zadání . . . 2

2 část - Vytvoření objektů databáze **2**
2.1 Tvorba tabulek . . . 2
2.2 Plnění tabulek . . . 3

3 Dotazy SELECT **3**
3.1 Spojeni dvou tabulek . . . 3
3.2 Spojeni tří tabulek . . . 4
3.3 2x dotazy GROUP BY . . . 4
3.4 dotaz z predikátem EXISTS . . . 4

4 část - Vytvoření pokročilých objektů databáze **5**
4.1 Trigery . . . 5
4.2 Procedury . . . 5
4.3 Explain plan . . . 6
4.4 Definici přístupových práv . . . 7
4.5 View . . . 8

1 Úvod

1.1 Motivace

Účelem dokumentace je popsat proces vývoje, použití a problematiku projektu. V této dokumentaci naleznete postup řešení tohoto projektu a popis realizace jeho částí.

1.2 Název projektu

Projekt **č.24 Videopůjčovna**

1.3 Zadání

IS videopůjčovny. Videopůjčovna půjčuje kazety registrovaným zákazníkům.
Systém bude využíván jednak pracovníky videopůjčovny, jednak samotnými zákazníky.
Musí umožnit zákazníkům výběr požadovaného titulu podle vhodných kritérií a zjistit, zda je k dispozici (volné). Zaměstnanci potom realizují vlastní výpůjčku a vrácení zapůjčených titulů.
Systém musí zajistit vystavení účtu. Cena je závislá na době zapůjčení a zvyšuje se progresivně s dobou. Ceník se může měnit.

2 část - Vytvoření objektů databáze

2.1 Tvorba tabulek

Náš skript vytváří tabulky s následujícími názvy:

- Ceník
- Půjčeni
- Žánr
- Klient
- Zaměstnance
- Kazeta

Všechny tabulky kromě tabulek **Žánr** a **Ceník** mají své vlastní unikátní klíče typu `INT`. Výše uvedené tabulky odkazují na unikátní klíče tabulek **Kazeta** a **Půjčeni** resp.

Tabulky obsahují další pole typu `VARHAR` o délce 50 znaků pro informace o uživatelských jménech, jejich e-mailech nebo názvech filmů atd. Pomocí příkazu `CHECK` jsme také ovládali pole **datum od**, **datum od** a **množství**.

2.2 Plnění tabulek

K naplnění tabulek jsme použili příkaz `INSERT INTO` a `VALUES`. K zápisu daty jsme použili příkaz `TO_DATE()`

Několik příkladů:

```
INSERT INTO zamestnance
VALUES (7708170420, 'Adam', 'Bitcoin', 'Manezer', 'bitcoin@gmail.com',
TO_DATE('1979-11-27', 'yyyy-mm-dd'));

INSERT INTO klient (jmeno, prijmeni, email, datum_narozeni)
VALUES ('Janicka', 'Kovar', 'kovar@gmail.com',
TO_DATE('2000-12-01', 'yyyy-mm-dd'));

INSERT INTO kazeta (titul, datum_premiery, mnozstvi, cena)
VALUES ('Avatar', TO_DATE( '2009-12-09', 'yyyy-mm-dd'), 20, 50);

INSERT INTO pujceni (id_klienta, id_kazety, id_zamestnance)
VALUES (1, 1, 7911210300);

INSERT INTO cenik (cenik_id, datum_od, datum_do, celkem)
VALUES (1, TO_DATE('2007-01-01', 'yyyy-mm-dd'), TO_DATE('2007-02-01', 'yyyy-mm-dd'), 120);

INSERT INTO zanr (zanr_id, nazev_zanru) VALUES (1, 'Action');
```

3 Dotazy SELECT

3.1 Spojeni dvou tabulek

Výběr všech žánrů každé kazety:

```
SELECT kazeta.titul, zanr.nazev_zanru AS zanr
FROM kazeta, zanr
WHERE kazeta.id_kazeta = zanr.zanr_id;
```

Výběr klientů a zaměstnanců, kteří je obsluhovali:

```
SELECT zamestnance.jmeno, zamestnance.prijmeni, klient.jmeno, klient.prijmeni
FROM klient, zamestnance, pujceni
WHERE pujceni.id_klienta = klient.klient_id AND
pujceni.id_zamestnance = zamestnance.rodne_cislo;
```

3.2 Spojeni tří tabulek

Výběr jakou kazetu si klient vzal:

```
SELECT pujceni.id_pujceni, klient.jmeno, klient.prijmeni, kazeta.titul AS titul_kazety
FROM klient, kazeta, pujceni
WHERE klient.klient_id = pujceni.id_klienta AND
kazeta.id_kazeta = pujceni.id_kazety;
```

3.3 2x dotazy GROUP BY

Vyber: kolikrát byla každá kazeta vypůjčena:

```
SELECT kazeta.titul, COUNT(*) AS cislo_pronajmu
FROM kazeta, pujceni
WHERE kazeta.id_kazeta = pujceni.id_kazety
GROUP BY kazeta.titul
ORDER BY cislo_pronajmu DESC;
```

Vyber: kolik kazet si každý klient vzal:

```
SELECT klient.jmeno, klient.prijmeni , COUNT(pujceni.id_kazety) AS cislo_kazet
FROM kazeta, pujceni, klient
WHERE klient.klient_id = pujceni.id_klienta AND
kazeta.id_kazeta = pujceni.id_kazety
GROUP BY klient.jmeno, klient.prijmeni
ORDER BY cislo_kazet DESC;
```

3.4 dotaz z predikátem EXISTS

Vyber kazet, který nebyly vypůjčeni:

```
SELECT *
FROM kazeta
WHERE NOT EXISTS (SELECT * FROM pujceni WHERE kazeta.id_kazeta = pujceni.id_kazety);
```

4 část - Vytvoření pokročilých objektů databáze

4.1 Trigery

Implementovali jsme dva typy triggeru.

První typ je inkrementální trigger s nazvem **seq_<nazev tabulky>**.
Vytvořili jsme takové triggery pro tabulky **Kazeta, Klient, Půjčeni**.
Jsou spuštěny, když je do tabulky přidáno nové pole. Generují nové unikátní ID pro tyto tabulky, počínaje od jedné, zvyšující ID o 1 pro každé nové pole vybrané tabulky pomocí příkazu `NEXTVAL`.

Druhy typ je dekrementální trigger s názvem **kazeta_count_minus**.
Tento trigger se spustí, když se do tabulky **Půjčeni** přidá pole a odečte 1 od počtu kazet s daným názvem.
Příklad triggeru pro tabulku **Kazeta**:

```
CREATE SEQUENCE seq_kazeta INCREMENT BY 1 START WITH 1;
CREATE OR REPLACE TRIGGER kazeta_id
BEFORE INSERT ON Kazeta
FOR EACH ROW
BEGIN
:NEW.id_kazeta := seq_kazeta.nextval;
END;
```

Příklad triggeru **kazeta_count_minus**:

```
CREATE OR REPLACE TRIGGER kazeta_count_minus
AFTER INSERT ON pujceni
FOR EACH ROW
BEGIN
UPDATE kazeta SET mnozstvi = mnozstvi - 1 WHERE id_kazeta = :NEW.id_kazety;
END;
```

4.2 Procedury

V našem projektu jsme vytvořili dva procedury: **new_batch** a **the_best_seller**. První postup přidává zadané množství do zadaných kazet, čímž se simuluje zásoba kazet. Druhý postup vybírá nejlepšího prodejce za daný časové interval. Obě procedury spouštíme pomocí programu Oracle SQL Developer.

Procedura **new_batch** akceptuje název kazety a přidá její počet kusů, které mají být přidány pomocí funkce `UPDATE`.

Příklad procedury **new_batch**:

```
CREATE OR REPLACE PROCEDURE new_batch (pocet INT, titul_kazety VARCHAR) AS

BEGIN
IF pocet < 1
THEN
DBMS_OUTPUT.PUT_LINE('Error executing the new_batch procedure.');
ELSE
UPDATE kazeta SET mnozstvi = mnozstvi + pocet WHERE titul = titul_kazety;
END IF;
END;
```

Procedura **the_best_seller** akceptuje datum od a datum do, mezi kterými chceme vybrat nejlepšího prodejce. Dále pomocí funkce `COUNT` spočítáme celkový počet prodaných kazet za dané období a pro každého prodejce. Funkce zobrazí výsledek jako procent z celkového počtu, zaokrouhlené na dvě čísla za tečkou.

Příklad výstupu:

Connecting to the database homework.
Procento prodej od 01.01.07 do 31.01.07:
*Prodavac **Borek Ethereum** Prodavac: 75 %*
*Prodavac **Barbora Solana** Prodavac: 25 %*
Process exited.
Disconnecting from the database homework.

Samotnou funkci nebudeme do dokumentace kopírovat z důvodu její objemnosti.

4.3 Explain plan

Použili jsme funkci plánu `EXPLAIN PLAN` k analýze konkrétního výběru „kolik kazet si každý klient vzal“. Tato funkce nám poskytuje informace o čase stráveném na tomto výběru, o zatížení CPU serveru atd.

```
EXPLAIN PLAN FOR
SELECT klient.jmeno, klient.prijmeni , COUNT(pujceni.id_kazety) AS cislo_kazet
FROM kazeta, pujceni, klient
WHERE klient.klient_id = pujceni.id_klienta AND
kazeta.id_kazeta = pujceni.id_kazety
GROUP BY klient.jmeno, klient.prijmeni
ORDER BY cislo_kazet DESC;
SELECT * FROM table (DBMS_XPLAN.DISPLAY);
```

Vytvoření dvou indexů **klient_index** a **pujceni_index** pro tabulky **Klient** a **Půjčeni** se sloupci, na kterých náš select pracuje, pomocí příkazu `CREATE INDEX`.

```
CREATE INDEX klient_index ON klient (jmeno, prijmeni, klient_id);
CREATE INDEX pujceni_index ON pujceni (id_klienta, id_kazety);
```

Jak vidíme z výsledků, zátěž CPU se znatelně snížila.

Bez indexu:

```
-------------------------------------------------------------------------------
| Id  | Operation             | Name     | Rows  | Bytes | Cost (%CPU)| Time
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |          |    18 |  1494 |     7  (15)| 00:00:01
|   1 |  SORT ORDER BY        |          |    18 |  1494 |     7  (15)| 00:00:01
|   2 |   HASH GROUP BY       |          |    18 |  1494 |     7  (15)| 00:00:01
|*  3 |    HASH JOIN          |          |    18 |  1494 |     6   (0)| 00:00:01
|   4 |     VIEW              | VW_GBF_7 |     9 |   630 |     3   (0)| 00:00:01
|   5 |      TABLE ACCESS FULL| KLIENT   |     9 |   603 |     3   (0)| 00:00:01

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|   6 |     TABLE ACCESS FULL | PUJCENI  |    18 |   234 |     3   (0)| 00:00:01
-------------------------------------------------------------------------------
```

S indexem:

```
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |               |    18 |  1494 |     3  (34)| 00:00:01 |
|   1 |  SORT ORDER BY      |               |    18 |  1494 |     3  (34)| 00:00:01 |
|   2 |   HASH GROUP BY     |               |    18 |  1494 |     3  (34)| 00:00:01 |
|*  3 |    HASH JOIN        |               |    18 |  1494 |     2   (0)| 00:00:01 |
|   4 |     VIEW            | VW_GBF_7      |     9 |   630 |     1   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN| KLIENT_INDEX  |     9 |   603 |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
|   6 |     INDEX FULL SCAN | PUJCENI_INDEX |    18 |   234 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------
```

4.4 Definici přístupových práv

Přístupová práva jsme udělili pomocí příkazu `GRANT ALL ON` pro tabulky a taky `GRANT EXECUTE ON` pro procedury.

Příklad:

```
GRANT ALL ON cenik TO xkoval20;
GRANT ALL ON kazeta TO xkoval20;
GRANT ALL ON klient TO xkoval20;
GRANT ALL ON pujceni TO xkoval20;
GRANT ALL ON zamestnance TO xkoval20;
GRANT ALL ON zanr TO xkoval20;

GRANT EXECUTE ON the_best_seller TO xkoval20;
GRANT EXECUTE ON new_batch TO xkoval20;
```

4.5 View

Pomocí příkazu `VIEW` jsme vytvořili pohled **kazety_comedy** s filmy v kategorii "Komedie". Vytvořili jsme také jeho materializovaného "klona" **mat_kazety_comedy**. Materializovaný pohled je vytvořen příkazem `MATERIALIZED VIEW` a je fyzicky uložen na disku serveru, takže přístup k němu je rychlejší. Po změnách v naší databázi můžeme aktualizovat naše pohledy pomocí příkazu `COMMIT`.

Příklad `VIEW`:

```
CREATE VIEW kazety_comedy AS
SELECT K.titul
FROM XKOVAL20.kazeta K, XKOVAL20.zanr Z
WHERE K.id_kazeta = Z.zanr_id and Z.nazev_zanru = 'Comedy';
```

Příklad `MATERIALIZED VIEW`:

```
CREATE MATERIALIZED VIEW mat_kazety_comedy
REFRESH ON COMMIT AS
SELECT K.titul
FROM XKOVAL20.kazeta K, XKOVAL20.zanr Z
WHERE K.id_kazeta = Z.zanr_id and Z.nazev_zanru = 'Comedy';
```

Vystyp do `COMMIT`:

TITUL
————————————————————-
Avatar
The Lord of the Rings: The Return of the King
Spider-Man
Harry Potter and the Sorcerers Stone

```
INSERT INTO zanr(zanr_id, nazev_zanru) VALUES (2, 'Comedy');
COMMIT;
```

Vystyp po `COMMIT`:

TITUL
————————————————————-
Avatar
Kung Fu Panda
The Lord of the Rings: The Return of the King
Spider-Man
Harry Potter and the Sorcerers Stone