VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Databázové systémy

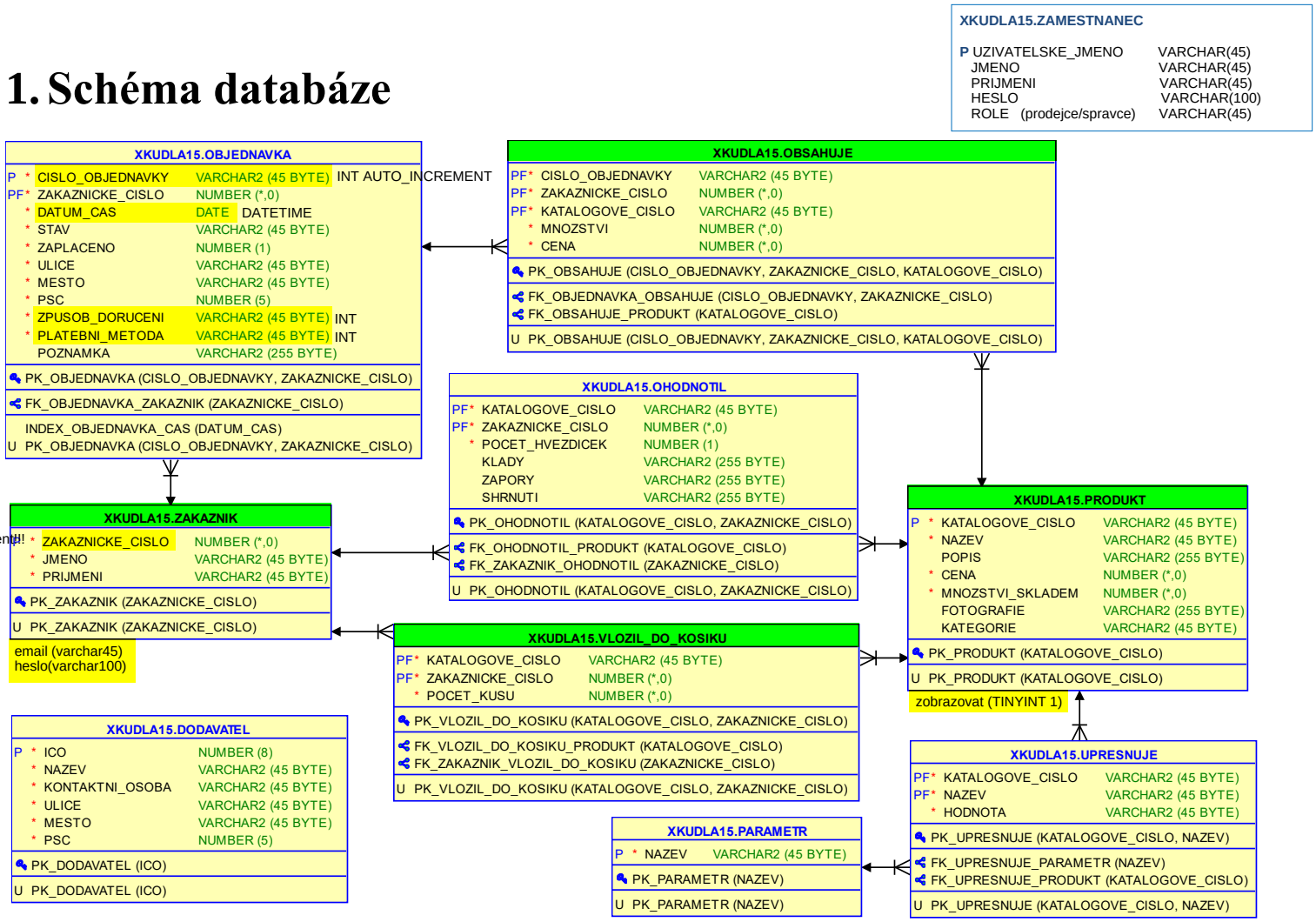
Projekt 4. + 5. část

Varianta 27 - Internetový obchod

Kudláč Vladan, xkudla15
Musil Marek, xmusil65

Obsah

1. Schéma databáze



2. Triggery

V naší databázi jsme se rozhodli pro vytvoření dvou triggerů. Pro vypočítání nového zákaznického čísla a pro kontrolu IČO.

2.1. Zákaznické číslo

Jelikož Oracle DB nemá vestavěnou funkci auto inkrement, tak jsme pro tuto funkci vytvořili trigger, který se spustí před vložením nového zákazníka do databáze. Číslo nového zákazníka je určeno tak, že se nalezne nejvyšší zákaznické číslo v databázi a novému se přiřadí toto maximum zvětšené o jedna.

2.2. IČO

Při vkládání nebo úpravě záznamu v tabulce `Odběratelé` dojde ke spuštění triggeru pro kontrolu hodnoty sloupce `ico`. IČO má obsahovat maximálně 8 číslic, což je ošetřeno použitým datovým typem `NUMERIC(8)`. Trigger zajišťuje kontrolu dělitelnosti. Nejprve zjistí součet – první až sedmou číslici vynásob čísly 8, 7, 6, 5, 4, 3, 2 a součiny sečte. Poté spočítá zbytek – součet modulo 11. Pokud je zbytek 0, pak musí být nejpravější číslice 1, je-li 1, pak 0, jinak musí být nejpravější číslice rovna 11-zbytek.

3. Procedury

Databáze obsahuje dvě uložené procedury. Výpočet celkové ceny objednávky a výpis statistik objednávek v určitém časovém období.

3.1. Cena objednávky

Procedura je volána s jedním parametrem, který obsahuje číslo objednávky, pro kterou se má spočítat celková cena. Tento výpočet se provede tak, že se spojí tabulka `Objednávka` a `Obsahuje`, jejíž číslo objednávky odpovídá předanému číslu parametrem a vynásobí se cena za kus zboží množstvím a celková cena se vypíše na `dbms_output`. Při nenalezení objednávky se vypíše informace o nenalezení,

3.2. Statistiky

Procedura statistika zobrazí celkový počet objednávek a počet nezaplacených objednávek za období zadané parametry od do na `dbms_output`. Procedura využívá explicitní kurzor. Za pomocí něj prochází jednotlivé řádky tabulky, které byly vytvořeny v zadaném časovém rozmezí a u každého řádku zjistí, jestli je zaplacen nebo nezaplacen a započítá jej do statistiky. Pokud jsou všechny objednávky zaplacené, počet nezaplacených nevypíše. Procedura upozorní, pokud v zadaném období nebyla provedena žádná objednávka. Časový údaj OD musí předcházet údaji DO, jinak procedura generuje chybu s kódem `-20001`.

4. Explain plan

Chtěli bychom zjistit, kolik objednávek provedli jednotliví zákazníci od 1. 3. 2018 (a jestli nějaké vůbec provedli). K tomu potřebujeme spojit tabulku `Zakaznik` a `Objednavka` a použít agregační funkci `COUNT`. Řádky se agregují podle sloupce `zakaznicke_cislo`.

4.1. Explain plan popisovaného dotazu bez využití indexu

```
-------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     6 |   210 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY      |             |     6 |   210 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS OUTER|             |     6 |   210 |     3   (0)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL| OBJEDNAVKA  |     6 |   132 |     3   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN| PK_ZAKAZNIK |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Spojení tabulek (left join) se provede pomocí metody **`NESTED LOOPS`**, každý řádek jedné tabulky se porovnává se všemi řádky druhé tabulky. Podmínku na datum novější než 1. 3. 2018 řeší průchodem celé tabulky (**`TABLE ACCESS FULL`**) a kontrolou hodnoty sloupce `datum_cas`. Sloupec `zakaznicke_cislo` je primárním a cizím klíčem, lze tedy využít přístup **`INDEX UNIQUE SCAN`**, který přistupuje k jedinečným hodnotám pomocí B-stromu.
Při výskytu **`TABLE ACCESS FULL`** bychom měli zpozornět, neboť průchod rozsáhlou tabulkou je neefektivní. V tomto dotazu je způsoben prací se sloupcem `datum_cas`. Hodnoty sloupce nabývají rozdílných hodnot, unikátní ale nejsou, nicméně sloupec `datum_cas` je ideálním kandidátem pro indexování.

4.2. Po vytvoření indexu

```
-----------------------------------------------------------------------------------------------------------
| Id | Operation                           | Name                 | Rows | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|  0 | SELECT STATEMENT                    |                      |    6 |   210 |     3  (34)| 00:00:01 |
|  1 |  HASH GROUP BY                      |                      |    6 |   210 |     3  (34)| 00:00:01 |
|  2 |   NESTED LOOPS OUTER                |                      |    6 |   210 |     2   (0)| 00:00:01 |
|  3 |    TABLE ACCESS BY INDEX ROWID BATCHED| OBJEDNAVKA           |    6 |   132 |     2   (0)| 00:00:01 |
|* 4 |     INDEX RANGE SCAN                | INDEX_OBJEDNAVKA_CAS |    6 |       |     1   (0)| 00:00:01 |
|* 5 |    INDEX UNIQUE SCAN                | PK_ZAKAZNIK          |    1 |    13 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------
```

Po vytvoření indexu se již kvůli příkazu WHERE neprochází celá tabulka. Namísto toho se řádky dohledávají po použití indexu (**`TABLE ACCESS BY INDEX ROWID`**). Hodnoty `datum_cas` však nejsou unikátní a je tak ještě nutné projít řádky se stejnou hodnotou (**`INDEX RANGE SCAN`**). Již teď je dotaz efektivnější, klesla celková cena dotazu. S přibývajícími řádky by význam použití indexu rostl.

5. Materializovaný pohled

Materializovaný pohled jsme vytvořili jeden s názvem `Mskicaky` a to pro správu produktů spadajících do kategorie Skicáky. Tento pohled má nastaven parametr `REFRESH ON COMMIT`, což znamená, že při každé úpravě tabulky produktů se vygeneruje nový pohled. Tento pohled je možné zobrazit jednoduchým příkazem `SELECT`.