Databázové systémy (IDS)

2021/2022

13 - Restaurace

Projektová dokumentace

Lucie Svobodová (xsvobo1x)
Jakub Kuzník (xkuzni04)

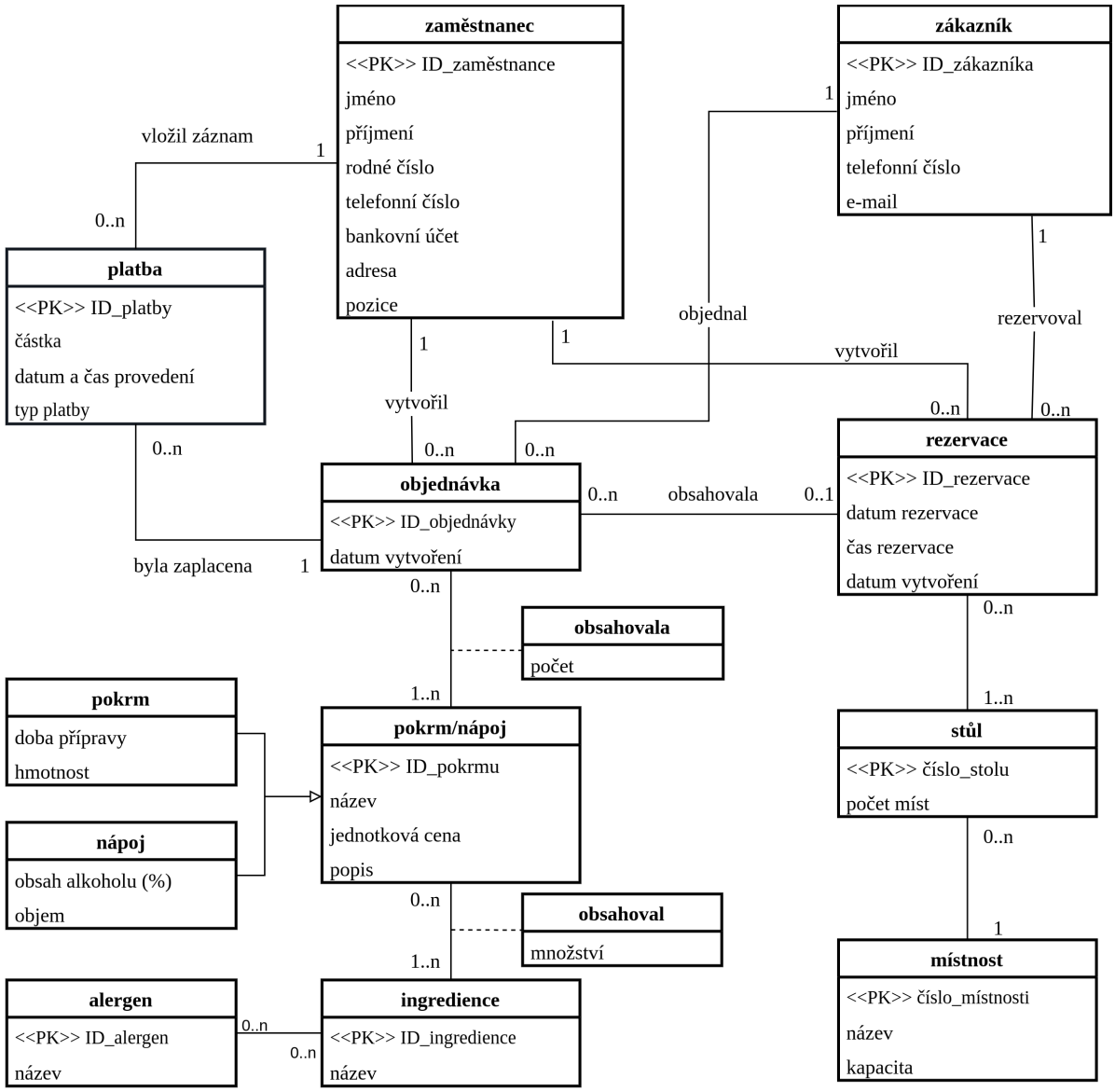
22. 4. 2022

Obsah

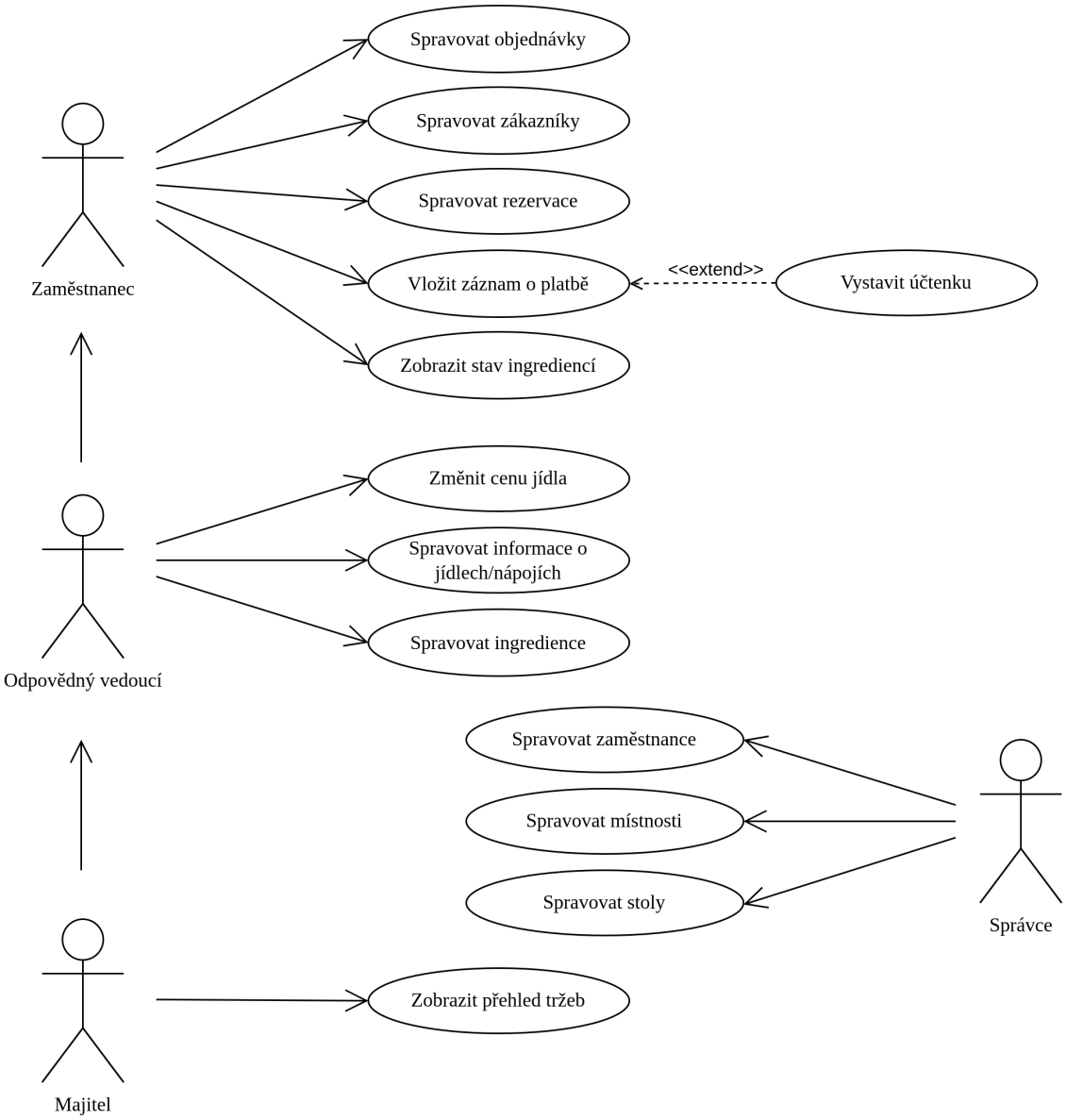
1 Zadání - Informační systém pro restauraci

IS byl vytvořen pro restauraci, které napomůže zlepšit přehled o rezervacích a objednávkách. Restaurace je členěna do více místností, kde se nachází stoly pro hosty. Zákazník si po telefonické dohodě nebo přes e-mail může zarezervovat jeden nebo více stolů v místnostech, anebo celé místnosti, případně i celou restauraci pro různé společenské akce. Zákazník sám s IS nijak nepracuje. Součástí rezervace může být také objednávka nápojů a jídel. Systém umožňuje zaměstnancům restaurace vkládat objednávky spolu s informacemi, který zaměstnanec danou objednávku vložil a pro koho je určena. Systém uchovává i informace o platbách za jednotlivé objednávky, přičemž jedna objednávka může být zaplacena více platbami. Systém dále eviduje veškeré prodávané jídlo a pití včetně seznamu alergenů a konkrétního složení, aby měli zaměstnanci přehled, kolik surovin je potřeba do skladu nakoupit.

2 Entity relationship diagram



3 Use case diagram



4 Implementace

Na začátku skriptu jsou pomocí klauzule `DROP` zahozeny případně již existující tabulky a triggery. Poté se pomocí příkazů `CREATE TABLE` vytvoří veškeré tabulky, přičemž integritní omezení se pro přehlednost nachází uvnitř těchto `CREATE TABLE` klauzulí. Reprezentace generalizace/specializace byla realizována pomocí jedné tabulky (`Pokrm_napoj`) pro obě specializace (`Pokrm`, `Napoj`), protože tyto specializace jsou disjunktní a totální. Po vytvoření všech tabulek následuje sekce s množinou klauzulí `INSERT INTO`, které naplní relační databází vzorovými daty, aby bylo možné realizovat operace nad těmito tabulkami. Dále následuje vytváření dotazů a dalších objektů databáze.

4.1 Triggery

V rámci projektu byly implementovány dva databázové triggery. Použití těchto triggerů je ukázáno na několika vzorových příkladech.

```
TRIGGER Zmena_platby
TRIGGER Kontrola_casu_rezervace
```

Trigger `Zmena_platby` slouží k uchovávání informací o všech změnách sloupce `cena` tabulky `Platby`. Není žádoucí, aby byly částky plateb měněny po vložení záznamu, proto byla vytvořena tabulka `Zmeny_castek_plateb`, do které je po aktualizaci nebo odstranění částky u nekteré z plateb vložena informace o uživateli, který změnu provedl, datu provedení a ID platby. Také je uložena původní částka a částka nová.

Druhým triggerem je trigger `Kontrola_casu_rezervace`. Ten při vložení nového záznamu do tabulky `Rezervace` zkontroluje, zda je zadané datum rezervace validní, tedy že je buď dnešní, nebo budoucí. Pokud je zadané datum menší než datum dnešní, je vyvolána výjimka.

4.2 Procedury

Součástí projektu byla implementace dvou procedur. Obě procedury využívají kurzory, datový typ odkazující se na řádek tabulky `table%ROWTYPE` i ošetření výjimek. Skript také obsahuje ukázku použití daných procedur na několika příkladech.

```
PROCEDURE Uctenka (ID IN INT)
PROCEDURE Vykonnost_zamestnance (ID IN INT)
```

Procedura `Uctenka()` na standardní výstup vypíše účtenku za objednávku zadanou v argumentu této procedury. Je vypsáno ID objednávky, dále jednotlivé pokrmy a nápoje, které tato objednávka obsahuje, včetně jejich počtu, jednotkové ceny a celkové ceny za daný pokrm/nápoj. Na závěr je vypsaná celková částka za danou objednávku.

Procedura `Vykonnost_zamestnance()` na standardní výstup vypíše informaci o procentu objednávek vytvořených zaměstnancem s ID zadaným v argumentu. Procedura vypíše výjimku, pokud zadaný zaměstnanec nepracuje na pozici číšníka (a tedy nemůže vytvářet objednávky), nebo pokud daný zaměstnanec nebyl nalezen.

4.3 Definice přístupových práv

Přístupová práva byla přidělena vedoucím týmu **xsvobo1x** druhému členovi **xkuzni04** na každou tabulku pomocí klauzule `GRANT ALL`. Dále byla přidělena práva na volání procedur `Uctenka()` a `Vykonnost_zamestnance` a na používání materializovaného pohledu `Jidelni_listek`.

4.4 Materializovaný pohled

V projektu implementovaný materializovaný pohled je vytvořen členem týmu **xkuzni04**. Materializovaný pohled zobrazuje některé sloupce z tabulky `Pokrm_napoj`, proto je nazván `Jídelni_listek`. Díky `CACHE` databáze postupně optimalizuje čtení z pohledu, `BUILD IMMEDIATE` zajišťuje naplnění pohledu ihned po jeho vytvoření, `ENABLE QUERY REWRITE` umožní použití pohledu pro optimalizaci přesně stejného dotazu a díky `REFRESH ON COMMIT` se změny v materializovaném pohledu projevují po commitu master tabulky.

Ukázka použití materializovaného pohledu:

```
SELECT * FROM Jidelni_listek;
```

NAZEV	CENA
Kuřátko supreme s jasmínovou rýží, restovaným cukrovým hráškem a baby karotkou na sezamovém oleji	268
Cheesecake z bílé čokolády	129
Domácí třešňová limonáda	65
Cappuccino	59

```
INSERT INTO Pokrm_napoj (nazev, doba_pripravy, cena)
VALUES ('Burger s trhanym veprovym masem v BBQ omacce', '20 minut', 180);

COMMIT;
SELECT * FROM Jidelni_listek;
```

NAZEV	CENA
Kuřátko supreme s jasmínovou rýží, restovaným cukrovým hráškem a baby karotkou na sezamovém oleji	268
Cheesecake z bílé čokolády	129
Domácí třešňová limonáda	65
Cappuccino	59
Burger s trhaným vepřovým masem v BBQ omáčce	180

4.5 Explain plan a Index

Řešení klauzule `EXPLAIN PLAIN` bylo spojeno s požadavkem na optimalizaci pomocí vytvoření vlastního `INDEXU`. Optimalizován byl následující dotaz, který se nachází uvnitř klauzule `EXPLAIN PLAN FOR`.

```
EXPLAIN PLAN FOR
    SELECT COUNT(*) pocet_objednavek, jmeno, prijmeni, ID_zamestnanec
    FROM Objednavka O NATURAL JOIN Zamestnanec ZA
    GROUP BY ID_zamestnanec, jmeno, prijmeni, ID_zamestnanec
    HAVING COUNT(*) > 1
    ORDER BY jmeno;
```

Ve výsledné tabulce, která byla vytvořena, vidíme, že byly postupně vykonány operace `SELECT`, poté `FILTER`, `GROUP BY` a `HASH JOIN`.

PLAN_TABLE_OUTPUT

```
Plan hash value: 1747799445

-------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     4 |   248 |     7  (15)| 00:00:01 |
|*  1 |  FILTER             |             |       |       |            |          |
|   2 |   SORT GROUP BY     |             |     4 |   248 |     7  (15)| 00:00:01 |
|*  3 |    HASH JOIN        |             |     4 |   248 |     6   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| OBJEDNAVKA  |     4 |    52 |     3   (0)| 00:00:01 |
|   5 |     TABLE ACCESS FULL| ZAMESTNANEC |     4 |   196 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Pro snížení ceny byl použit následující index. Ten v tabulce nahradil jedno z `TABLE ACCESS FULL` a ulehčil vyhledávání, což se projevilo na cenách procesoru. Mezi další možnosti, jak by šel následující dotaz optimalizovat, je například zrušení klauzule `HAVING BY`, nebo vytvoření druhého indexu pro zaměstnance, či vytvoření kombinací indexu pro zaměstnance a objednávku.

```
CREATE INDEX id_zam ON Objednavka(ID_zamestnanec);
```

PLAN_TABLE_OUTPUT

```
Plan hash value: 2805734978

-------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     4 |   248 |     4  (25)| 00:00:01 |
|*  1 |  FILTER             |             |       |       |            |          |
|   2 |   SORT GROUP BY     |             |     4 |   248 |     4  (25)| 00:00:01 |
|   3 |    NESTED LOOPS     |             |     4 |   248 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS FULL| ZAMESTNANEC |     4 |   196 |     3   (0)| 00:00:01 |
|*  5 |     INDEX RANGE SCAN | ID_ZAM      |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```