VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

IDS - Databázové systémy

Dokumentace k projektu

Restaurace

2016/2017

Juraj Korček (xkorce01)
Jan Kubica (xkubic39)

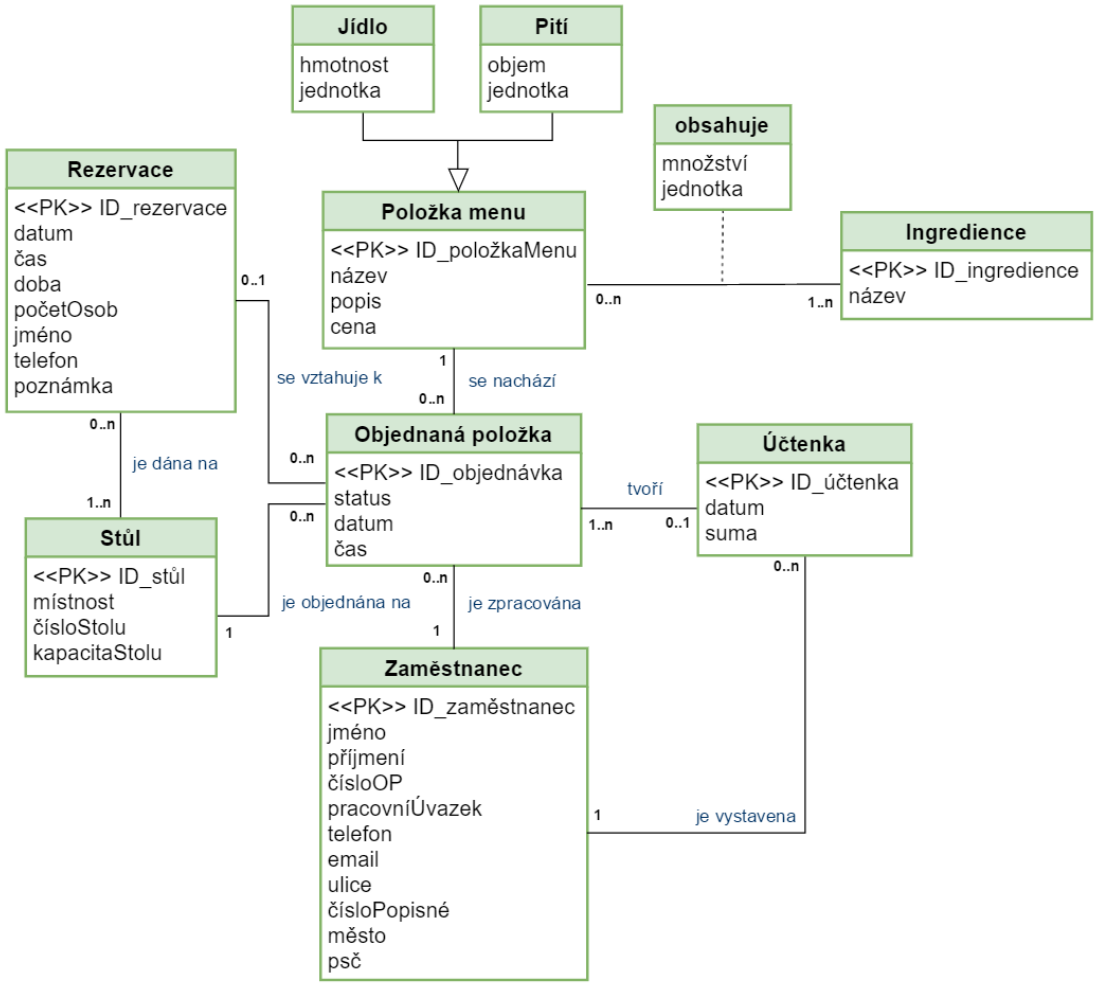
30. dubna 2017

1 Zadání

Vytvořte IS pro restaurační zařízení, který napomůže k zjednodušení a zpřehlednění jeho provozu. Restaurace je členěna do více místností a má přední a zadní zahrádku a poskytuje běžné stravovací služby veřejnosti. Od informačního systému se požaduje, aby, krom jiného, umožnil správu rezervací a objednávek. Rezervovat je možné jeden nebo více stolů v místnostech či na zahrádkách, anebo celé místnosti, případně i celou restauraci pro různé společenské akce. Součástí rezervace také může být objednávka nápojů a jídel. Systém musí umožňovat zaměstnancům restaurace vkládat objednávky spolu s informacemi, který zaměstnanec danou objednávku vložil a pro koho je určena. Když se zákazníci rozhodnou zaplatit, musí jim systém vystavit účtenku. Po zaplacení pak příslušný zaměstnanec vloží záznam o platbě do systému. Systém by měl také poskytovat podrobný přehled tržeb za vybrané období. Přístup k této funkci bude mít pouze majitel. V neposlední řadě musí systém evidovat veškeré prodávané jídlo a pití (včetně složení), přičemž majitel a odpovědný vedoucí mají možnost měnit ceny jídla a pití nebo přidávat a odebírat položky.

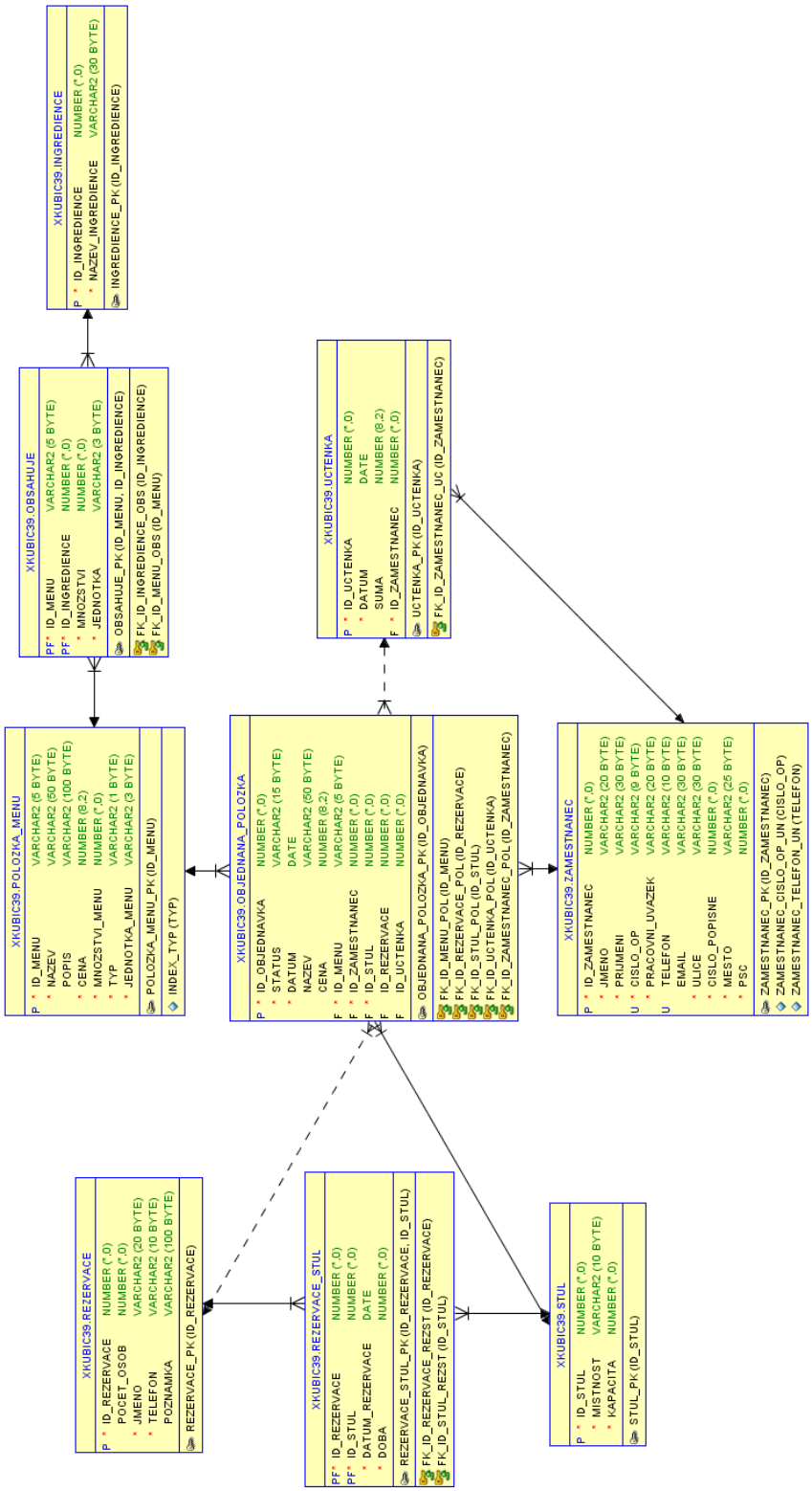
2 Návrh zpracování

2.1 ER diagram



Obrázek 1: ER diagram

2.2 Schéma databáze



Obrázek 2: Schéma databáze generované v Oracle SQL Developeru

2.3 Generalizace/specializace

Realizaci vztahu generalizace/specializace jsme řešili spojením dvou entitních množin (jídla a pití) do jedné výsledné tabulky, přičemž byl přidán sloupec **typ** pro rozlišení dané kategorie. Tento sloupec pak nabývá pro každou položku menu buď to hodnoty `J` pro jídlo, anebo `P` pro pití. Pro ošetření správné jednotky byla přidána podmínka `CHECK`, která povoluje jídlu nastavit jednotku pouze z množiny { *g, kus* }, pití pak { *ml, cl, dcl, l* }. K tomuto řešení jsme přistoupili pro její snadnější a logičtější implementaci v databázi.

3 Implementace

3.1 Triggery

V rámci našeho řešení byly vytvořeny celkem tři triggery. Jako první např. **`TRG_cena_objednavka`**, který zajišťuje *po vytvoření* objednané položky překopírování názvu a ceny z tabulky `polozka_menu` do tabulky `objednana_polozka`. Takto jsme postupovali z důvodu možnosti rychlejšího zpracování dotazů a taktéž kvůli obecně časté změně názvu či ceny položky menu v budoucnu a následné integritě dat.

Jako druhý trigger byl implementován **`TRG_status_zaplaceno_castka`**. Tento trigger se spouští *po vytvoření* účtenky a umožňuje konkrétní označené objednané položky (status `oznaceno`) zaplatit (status `zaplaceno`), daným položkám pak přidělit cizí klíč na konkrétní účtenku a vypočítat celkovou sumu potřebnou pro právě vytvořenou účtenku.

Třetí trigger **`TRG_cislo_rezervace`** se spoutší *před vytvořením* rezervace a nastavuje primární klíč ze sekvence složením aktuálního data a unikátního čísla v rámci dne. Předpokládá se, že unikátní číslo by pak bylo pomocí procedury `reset_seq` vynulováno každý den o půlnoci.

3.2 Procedury

Procedura **`reset_seq`** umožňuje resetování počítadla pro pořadí rezervace. Nejprve se načítá následující hodnota počítadla, která se poté od počítadla odečítá a tím se počítadlo inicializuje na nulu. Nakonec se určí minimální hodnota počítadla.

Procedura **`inc_prices`** vykonává plošnou změnu cen danou procentem, které je předáno přes parametr. Procedura využívá explicitní kurzor, datové typy odkazující na sloupec tabulky a ošetření výjimek. Pokud uživatel zadá procento menší než -99%, je na výstup vypsáno chybové hlášení.

Procedura **`spocti_trzbu`** slouží k vypočtení tržby v určitém období daném dvěma daty předanými jako parametr. I zde jsou využity datové typy odkazující na sloupec tabulky a také je ošetřena výjimka, kdy je v prvním parametru zadáno pozdější datum než je datum v druhém parametru.

3.3 Vytvoření indexu společně s explain plan

Pro optimalizaci byl vybrán následující SQL dotaz:

```
SELECT nazev AS "Nazev jidla", COUNT(nazev) AS "Pocet ingrediencii"
FROM Polozka_menu NATURAL JOIN Obsahuje NATURAL JOIN Ingredience
WHERE typ = 'J'
GROUP BY nazev
ORDER BY COUNT(nazev) DESC;
```

Aby bylo možné určit cenu tohoto dotazu, byl použit `EXPLAIN PLAN`, jehož výpis můžete vidět níže.

```
-----------------------------------------------------------------------------------------
| Id  | Operation             | Name           | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |                |    24 |   888 |     6  (34)| 00:00:01 |
|   1 |  SORT ORDER BY        |                |    24 |   888 |     6  (34)| 00:00:01 |
|   2 |   HASH GROUP BY       |                |    24 |   888 |     6  (34)| 00:00:01 |
|*  3 |    HASH JOIN          |                |    24 |   888 |     4   (0)| 00:00:01 |
|*  4 |     TABLE ACCESS FULL | POLOZKA_MENU   |     6 |   198 |     3   (0)| 00:00:01 |
|   5 |     INDEX FULL SCAN   | SYS_C001483240 |    24 |    96 |     1   (0)| 00:00:01 |
```

Jako prostředek pro urychlení byl vybrán index nad sloupcem `typ` tabulky `polozka_menu`. Po aplikování indexu byl znovu použit `EXPLAIN PLAN`, který jasně demonstruje, že zavedení indexu optimalizovalo dotaz. Zde je možné vidět, jak se celková cena dotazu snížila a zatížení procesoru zvýšilo.

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name         | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |              |    24 |   888 |     5  (40)| 00:00:01 |
|   1 |  SORT ORDER BY                           |              |    24 |   888 |     5  (40)| 00:00:01 |
|   2 |   HASH GROUP BY                          |              |    24 |   888 |     5  (40)| 00:00:01 |
|*  3 |    HASH JOIN                             |              |    24 |   888 |     3   (0)| 00:00:01 |
|   4 |     TABLE ACCESS BY INDEX ROWID BATCHED  | POLOZKA_MENU |     6 |   198 |     2   (0)| 00:00:01 |
|*  5 |      INDEX RANGE SCAN                    | INDEX_TYP    |     6 |       |     1   (0)| 00:00:01 |
```

3.4 Přístupové práva

Přístupové práva pro druhého člena týmu byly definovány jako pro *zaměstnance-číšníka*. *Číšník* byl měl mít práva výběru, změny, vložení a mazání pro tabulky **`rezervace`**, **`stul`**, **`rezervace_stul`**, **`objednana_polozka`**. Číšník by měl mít možnost výběru z tabulky **`polozka_menu`**, ale modifikovat ji může pouze provozní. Pro tabulku **`uctenka`** byly povoleny práva výběru, změny i vložení.

3.5 Materializovaný pohled

Pro druhého člena týmu byl vytvořen materializovaný pohled, který umožňuje *práci s rezervacemi*. Před vytvořením materializovaného pohledu musely být definovány přístupové práva pro tabulky patřící prvnímu členovi týmu. Kvůli optimalizaci byly vytvořené logy pro tabulky využívané materializovaným pohledem. Tyto logy umožňují zaznamenávat změny, a proto se nemusí při vložení dat procházet tabulka celá. Pro další optimalizaci byly při vytváření materializovaného pohledu použity operace: `REFRESH FAST ON COMMIT`, `ENABLE QUERY REWRITE` a `CACHE`.