VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Databázové systémy – Konceptuální model

# Hotel

1. května 2017

Marek Šipoš
Patrik Sztefek

# 1 Zadání

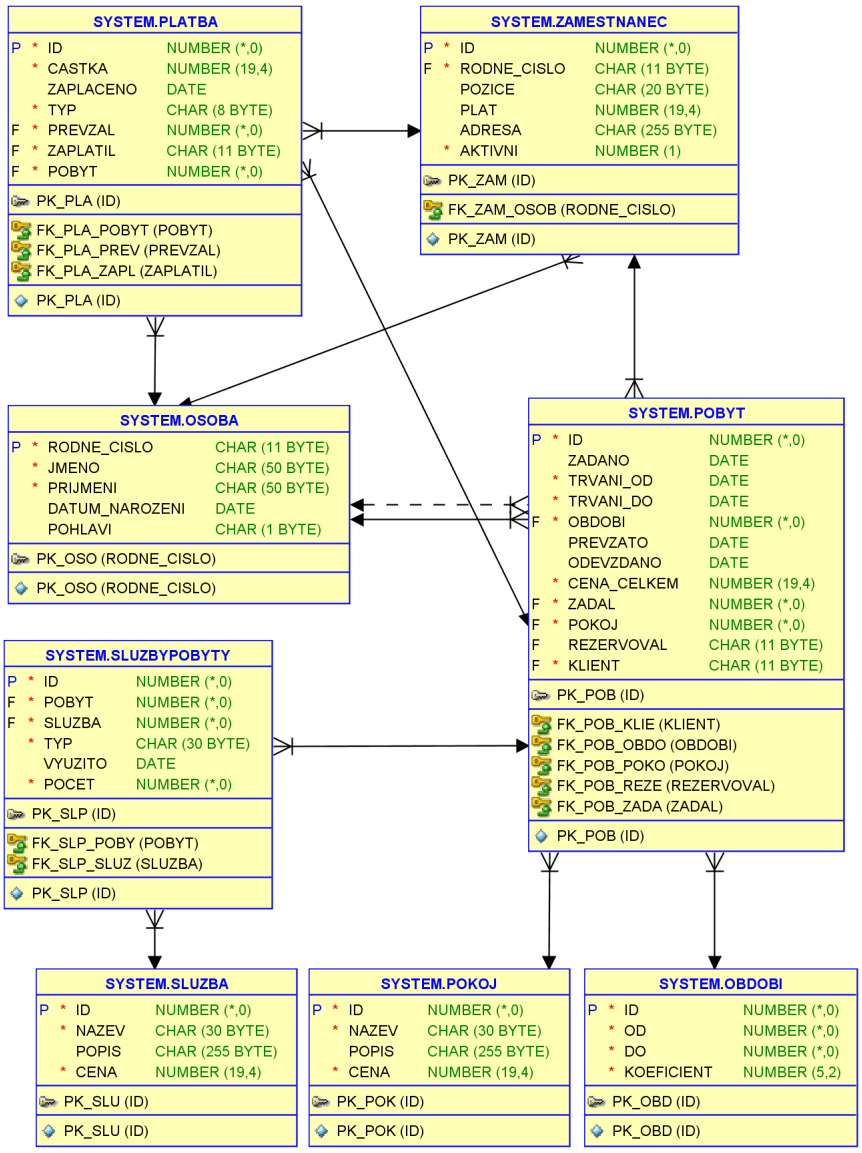
## 1.1 Model informačního systému – Hotel

Navrhněte IS hotelu, který by poskytoval přehled o dostupnosti pokojů, údaje o hostech, jejich pobytech v hotelu, požadavcích na služby, platby za pokoje, atd. Zákazníci mohou provádět rezervace pokojů (tím pádem musí zadat své osobní údaje). Klient si může v rámci jedné rezervace objednat více pokojů, třeba i na jiné datum. Hotelu stačí mít informace pouze o jednom klientovi, který zaštiťuje celý pobyt v hotelu (o ostatních účastnících pobytu nemusí být dostupné žádné informace). Pobyt může být vytvořen na základě rezervace, nebo k rezervaci pokoje vůbec nemusí dojít, pokud klient přijde přímo na recepci hotelu. Jednotlivé typy pokojů mají různé ceny, cena pokoje se navíc může lišit podle období (turistická sezóna), na které si klient pokoj objednává. Cena pokoje se zároveň odvíjí od toho, zda si host pokoj rezervuje dopředu, nebo až na místě. Klient si může přiobjednat k danému pobytu služby v hotelu navíc, jako např. pronájem bazénu, a to i vícekrát. IS dále bude schopen evidovat skutečně absolvované pobyty - tj. které pokoje klient od kdy do kdy obýval a jaké služby skutečně využil (od x do x na kterém pokoji). U každého pobytu je evidováno, který zaměstnanec převzal platbu za pobyt.

## 1.2 Tvorba skriptu relační databáze – část 4 a 5

- Vytvoření alespoň dvou netriviálních databázových triggerů vč. jejich předvedení, z toho právě jeden trigger pro automatické generování hodnot primárního klíče nějaké tabulky ze sekvence (např. pokud bude při vkládání záznamů do dané tabulky hodnota primárního klíče nedefinována, tj. NULL)
- Vytvoření alespoň dvou netriviálních uložených procedur vč. jejich předvedení, ve kterých se musí (dohromady) vyskytovat alespoň jednou kurzor, ošetření výjimek a použití proměnné s datovým typem odkazujícím se na řádek či typ sloupce tabulky (table_name.column_name%TYPE nebo table_name%ROWTYPE)
- Explicitní vytvoření alespoň jednoho indexu tak, aby pomohl optimalizovat zpracování dotazů, přičemž musí být uveden také příslušný dotaz, na který má index vliv, a v dokumentaci popsán způsob využití indexu v tomto dotazy (toto lze zkombinovat s EXPLAIN PLAN, vizte dále)
- Alespoň jedno použití EXPLAIN PLAN pro výpis plánu provedení databázového dotazu se spojením alespoň dvou tabulek, agregační funkcí a klauzulí GROUP BY, přičemž v dokumentaci musí být srozumitelně popsáno, jak proběhne dle toho výpisu plánu provedení dotazu, vč. objasnění použitých prostředků pro jeho urychlení (např. použití indexu, druhu spojení, atp.), a dále musí být navrhnut způsob, jak konkrétně by bylo možné dotaz dále urychlit (např. zavedením nového indexu), navržený způsob proveden (např. vytvořen index), zopakován EXPLAIN PLAN a jeho výsledek porovnán s výsledkem před provedením navrženého způsobu urychlení
- Definice přístupových práv k databázovým objektům pro druhého člena týmu
- Vytvoření alespoň jednoho materializovaného pohledu patřícího druhému členu týmu a používajícího tabulky definované prvním členem týmu (nutno mít již definována přístupová práva), vč. SQL příkazů/dotazů ukazujících, jak materializovaný pohled funguje.

# 2 Schéma databáze




Obrázek 1: Diagram tříd

## 2.1 Generalizace/Specializace

Specializaci je v databázi využita pro snížení zbytečné redundance dat. Tabulka `Osoba` obsahuje všechna data, která jsou potřeba pro identifikaci libovolné osoby. Je tedy zbytečné, aby tabulka `Zamestnanec` tyto informace obsahovala také. Z toho důvodu obsahuje tabulka `Zamestnanec` cizí klíč `rodne_cislo`, které odpovídá primárnímu klíči tabulky `Osoba`.

# 3 Implementace

Databázový skript pracuje v tomto pořadí:

- Zrušení případných již existujících tabulek
- Zrušení případných již existujících sekvencí
- Tvorba tabulek, primárních i cizích klíčů a omezení s podmínkami
- Tvorba sekvencí
- Vytvoření (nebo přepsání již existujících) triggerů
- Vytvoření netriviálních procedur
- Naplnění tabulek ukázkovými daty
- Spuštění ukázkových výběrových dotazů
- Spuštění výše vytvořených procedur
- Tvorba indexu pro optimalizaci a spuštění EXPLAIN PLAN
- Přidělení práv druhému členovi týmu
- Materializovaný pohled

Úlohy týkající se projektu 4 a 5 jsou popsány níže.

## 3.1 Triggery

Triggery se ve skriptu starají o dvě věci:

- Automatické přiřazování ID nově vkládaným položkám
- Automatický výpočet ceny pobytu

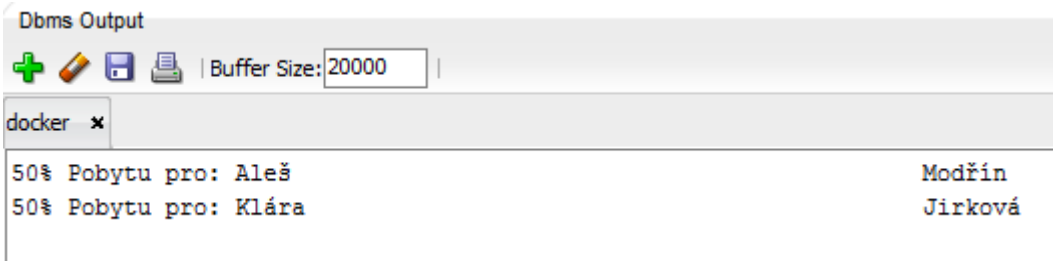
Automatické přiřazování ID funguje tak, že se nejdříve vytvoří sekvence, která hodnotu uchovává, a ta se poté metodou `nextval` získá a použije ke vložení do sloupce ID. Takto to je provedeno u všech tabulek kromě tabulky `osoba`, která obsahuje jako primární klíč rodné číslo.

Trigger na výpočet ceny pobytu vypočítá celkovou cenu pobytu na základě délky pobytu, období, ve kterém pobyt začíná (liší se cenový koeficient) a ceny pokoje. Volá se kdykoliv, když je do databáze přidán nový pobyt. Trigger navíc vhodně přiřadí aktuální období ke vkládanému pobytu.

## 3.2 Procedury

### 3.2.1 staff_stats

Tato procedura slouží pro jednoduché zobrazení výkonnosti zaměstnanců při zřizování pokojů zákazníkům. Po jejím spuštění se vypíše procentuální podíl na pronajatých pokojích na každého zaměstnance. Procedura nepřijímá žádné parametry.




Obrázek 2: Příklad výpisu procedury staff_stats

### 3.3 Optimalizace dotazu pomocí indexu, EXPLAIN PLAN

Optimalizační nástroj *Oracle Optimizer* se stará o vnitřní optimalizaci dotazů. Podle struktury cílových dat dokáže vhodně navrhnout postup, který se pro vykonání dotazu použije. Pro analýzu práce tohoto optimalizačního nástroje umožňuje databáze Oracle zavolat **EXPLAIN PLAN** pro tzv. *plánování dotazu*, což umožňuje zobrazit jednotlivé dílčí kroky při jeho vykonávání a také jejich paměťovou a výpočetní náročnost.

Plánování dotazu bylo v tomto případě využito při dotazu, který kombinuje spojení dvou tabulek, agregační funkci COUNT a klauzuli GROUP BY. Výsledkem je výpis všech pokojů a počty, kolikrát v nich byl někdo ubytován.

```
SELECT Pokoj.ID, Pokoj.Nazev, COUNT(Pobyt.pokoj) AS pocet_ubytovani
FROM Pokoj LEFT JOIN Pobyt ON (Pokoj.ID = Pobyt.pokoj)
GROUP BY Pokoj.ID, Pokoj.Nazev ORDER BY Pokoj.ID;
```

Provedení plánování dotazu pomocí EXPLAIN PLAN zobrazilo tento postup při jeho vykonávání:

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|---|---|---|---|---|---|---|
| **0** | SELECT STATEMENT | | 8 | 464 | 5 (20) | 00:00:01 |
| **1** | SORT GROUP BY | | 8 | 464 | 5 (20) | 00:00:01 |
| * **2** | HASH JOIN OUTER | | 8 | 464 | 4 (0) | 00:00:01 |
| **3** | TABLE ACCESS FULL | POKOJ | 6 | 270 | 2 (0) | 00:00:01 |
| **4** | TABLE ACCESS FULL | POBYT | 4 | 52 | 2 (0) | 00:00:01 |

Tabulka 1: Plánování dotazu před explicitním vložením indexu

Jak lze v tabulce 1 vidět, databázový stroj usoudil, že není potřeba nad takovým množstvím dat (vyjádřeným sloupcem **Rows**) používat jakoukoliv indexaci. Zde je stručné vysvětlení toho, jakým způsobem je provedení dotazu naplánováno[1] (řádek po řádku podle **Id**):

1. SELECT STATEMENT – Samotný výběrový dotaz jako celek. Vrací řádky tak, aby vyhovovaly klauzuli `WHERE`.
2. SORT GROUP BY – Operace pro třídění a slučování
3. HASH JOIN OUTER – Operace spojení tabulek pomocí hash tabulky, která je vytvořena nad tabulkou s menším počtem záznamů.
4. TABLE ACCESS FULL (POKOJ) – Běžný přístup k datům tabulky `Pokoj`
5. TABLE ACCESS FULL (POBYT) – Běžný přístup k datům tabulky `Pobyt`

[1] Řádky v tabulkách označené hvězdičkou ve sloupci `Id` značí paralelní operace.

Po explicitním vyžádání indexu ve sloupci `Pokoj` tabulky `Pobyt` a opětovném plánování již zmíněného dotazu byl zobrazen následující postup:

| Id | Operation | Name | Rows | Bytes | Cost (%CPU) | Time |
|---|---|---|---|---|---|---|
| **0** | SELECT STATEMENT | | 8 | 464 | 3 (34) | 00:00:01 |
| **1** | SORT GROUP BY | | 8 | 464 | 3 (34) | 00:00:01 |
| **2** | NESTED LOOPS OUTER | | 8 | 464 | 2 (0) | 00:00:01 |
| **3** | TABLE ACCESS FULL | POKOJ | 6 | 270 | 2 (0) | 00:00:01 |
| * **4** | INDEX RANGE SCAN | PERFORMANCEINDEX | 1 | 13 | 0 (0) | 00:00:01 |

Tabulka 2: Plánování dotazu po explicitním vložením indexu

Na první pohled lze vidět, že se paměťová náročnost (**Bytes**) mírně snížila, zatímco se procesorový čas (**Cost (%CPU)**) mírně zvýšil. Níže jsou stručně vysvětleny body, ve kterých se jednotlivé kroky oproti neindexované variantě liší:

2. NESTED LOOPS OUTER – Optimalizátor se zde rozhodl spustit iteraci a JOIN sestavit postupným iterováním nad dvěma seznamy. Jakmile si prvky vyhovovaly podle určených pravidel (vyjádřených klauzulí pro spojení `LEFT JOIN` **`ON`**), došlo ke spojení
4. INDEX RANGE SCAN – Přístup k indexovaným záznamům ve vzestupném pořadí. Používá se zde explicitně vytvořený index. Velmi rychlá operace

## 3.4 Přístup k databázi druhým členem týmu

Zadáním úkolu bylo nastavit uživatelská práva druhého člena týmu tak, aby měl přístup k tabulkám prvního člena týmu. Nad tímto se pak měl vytvořit materializovaný pohled, který ukazuje, jak to funguje.

### 3.4.1 Přístupová práva

Za předpokladu, že je druhý člen týmu zaměstnancem hotelu, mu byla nastavena práva následovně:

- Žádný přístup do tabulky `Obdobi`
- Možnost čtení (SELECT) z tabulky `Zamestnanec`
- Plný přístup do ostatních tabulek

Řadový zaměstnanec nemá přístup do tabulky `Obdobi`, protože je to v hotelovém systému součást vnitřní logiky, ke které nemusí zaměstnanec žádným způsobem přistupovat. Případné změny a návrh na nová období by pak byl předmětem činnosti systémové administrátora, hotelového ekonoma nebo někoho v podobné pozici.

Číst z tabulky `Zamestnanec` může řadový zaměstnanec proto, aby si mohl zkontrolovat své údaje a zároveň i číst případné údaje svých kolegů – za účelem spolupráce (např. sekretářka si bude potřebovat zobrazit adresu zaměstnance za účelem korespondence). Rozdělení více různých rolí různým zaměstnancům by mohlo být předmětem diskuze, je to však mimo rozsah tohoto projektu. Řadový zaměstnanec nemůže jakkoliv tabulku `Zamestnanec` modifikovat (např. si nemůže zvýšit plat).

Řadový zaměstnanec také nemá povolen přístup k proceduře `staff_stats`, jelikož taková informace by měla být přístupná pouze vedení hotelu.

### 3.4.2 Materializovaný pohled

Pro materializovaný pohled byly nejprve vytvořeny materializované logy, ve kterých se uchovávají změny v tabulkách, aby spíše než klauzuli `complete refresh` bylo možné použít klauzuli `fast refresh on commit`. Následně byl vytvořen samotný materializovaný pohled a ilustrována jeho funkčnost před a po přidání nových dat do tabulky s následným provedením `commit`. Vytvořený pohled měl tyto vlastnosti:

- `BUILD IMMEDIATE` – okamžité naplnění daty
- `ENABLE QUERY REWRITE` – optimalizace případného opakovaného dotazu

# 4 Závěrem

Skript byl testován pomocí nástroje *Oracle SQL Developer* na školním databázovém serveru *Oracle Database 12c* verze 12.1.0.2.0 na adrese `gort.fit.vutbr.cz` a také na lokálním serveru prostřednictvím aplikace Docker a kontejneru *sath89/oracle-12c*.

Skript je možné spouštět opakovaně (postará se o vhodné přepsání).