# **Implementace**

Náš SQL skript vytvoří dle námi navrchnutého ER diagramu model databáze, který je pak následně naplněn ukázkovými daty. K takto vytvořené databázy byly přiloženy ukázkové SELECT skript znázorňující práci s databází. Na konec ve čtvrté části zadání byly dodány SQL skripty pro vytvoření pokročilých objektů schématu databáze. Popis jednotlivých požadovaných vlastností je popsán níže.

#### **Triggery**

V skirptu existují dva triggery a to trigger REZERVACE\_PROHLIDKY a trigger ZAJEM\_O\_NEMOVITOST. Přičemž první z nich informuje o tom, když se do tabulky PROHLIDKA pokouší vložit záznam či se daný zaznám aktualizuje, který nesplňuje jedno z těchto logických pravidel:

- Prohlídku může na určitou nemovitost ve stejný čas uskutečňovat více zaměstanců avšak zaměstanec nemůže ve stejný čas provádět více prohlídek.
- Zájemce si nemůže ve stejný čas naplánovat více prohlídek než právě jednu.
- (Předpokládá se, že prohlídka trvá 30 minut).

Druhý trigger s názvem ZAJEM\_O\_NEMOVITOST informuje o tom, když se do tabulky pokouší vložit či aktualizovat záznam, který nesplňuje následující pravidlo:

 Zájemce ne,ůže podat nabídku na nemovitost, kterou již vlastní. Respektive ji nabízí.

### Procedury

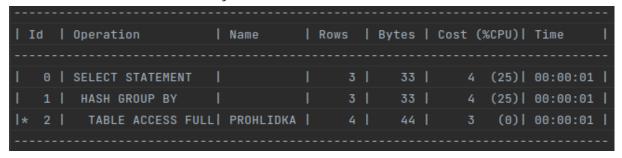
Ve skriptu existují také dvě procedury a to zaměstnanec\_smlouva\_prohlidka a provize\_prodej.

První z nich je zaměřena na výpočet výkonnosti zaměstnanců (kolik dokáže průměrně jeden zaměstnanec udělat prohlídek a podepsat smluv). Procedura začíná součtem všech zaměstnanců, prohlídek a smluv. Poté už stačí jen udělat průměr na jednoho zaměstnance, ošetřit dělení nulou a vypsat výsledek.

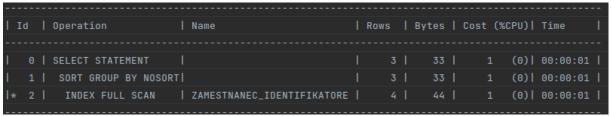
Druhá slouží k výpočtu mzdy podle prodaných nemovitostí a podle provize prodejce. Deklarujeme si pomocí %TYPE proměnnou id\_zamestnance podle typu použitého v tabulce. Následně si vytvoříme CURSOR obsahující všechny ceny nemovitostí které prodal zaměstnanec zadaný parametrem procedury. Následuje vyhledání id\_uživatele, a spuštění loopu na CURSOR. Zde už pouze pomocí provize zadané parametrem vypočítáme celkovou částku vyplácenou zaměstnanci.

## Explain plan

Pomocí funkce explain plan testujeme jeden z našich selectů. Po první spuštění jsme zjistili, jak je daný select náročný na výkon procesoru a jeho paměťovou náročnost. Pořadí tabulek na které odkazuje.



Poté si pomocí optimalizační klauzule seřadíme používané sloupečky příkazy where, nebo order a explain plan. Znovu spustíme.



Zde můžeme vidět, že po této optimalizaci je náš výběr mnohem méně náročný.

### Přidělení práv

Pomocí příkazu GRANT ALL nad všemi tabulkami byly přiděleny všechna práva kolegovi s loginem xkanko00. Příkaz GRANT ALL všem uživatelům v živé produkci by byl nevhodný. V odůvodněných případech by měl smysl pro vyvolené jednotlivce.

# Materializovaný pohled

Před implementací materializovaného pohledu pro kolegu jsme vytvořili materializovaný log nad tabulkou Smlouva. K tomuto řešení jsme se odhodlali, abychom mohli použít FAST REFRESH ON COMMIT, tudíž změny, které by nastali v materializovaném pohledu, by se projevili přímo v něm a nemuseli bychom vytvářet nový meterializovaný pohled. Změny provedené v logu se aktualizují až po provedení příkazu COMMIT.