

# Implementace

Náš SQL skript vytvoří dle námi navrhnutého ER diagramu model databáze, který je pak následně naplněn ukázkovými daty. K takto vytvořené databázi 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 skriptu 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 záznam aktualizuje, který nesplňuje jedno z těchto logických pravidel:

- Prohlídku může na určitou nemovitost ve stejný čas uskutečňovat více zaměstnanců avšak zaměstnanec 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 vypsát 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_uzivatele`, 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.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3	33	4 (25)	00:00:01
1	HASH GROUP BY		3	33	4 (25)	00:00:01
* 2	TABLE ACCESS FULL	PROHLIDKA	4	44	3 (0)	00:00:01

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

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
0	SELECT STATEMENT		3	33	1 (0)	00:00:01
1	SORT GROUP BY NOSORT		3	33	1 (0)	00:00:01
* 2	INDEX FULL SCAN	ZAMESTNANEC_IDENTIFIKATORE	4	44	1 (0)	00:00:01

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 projeví přímo v něm a nemuseli bychom vytvářet nový materializovaný pohled. Změny provedené v logu se aktualizují až po provedení příkazu COMMIT.