# Datový model (ERD) a model případů užití

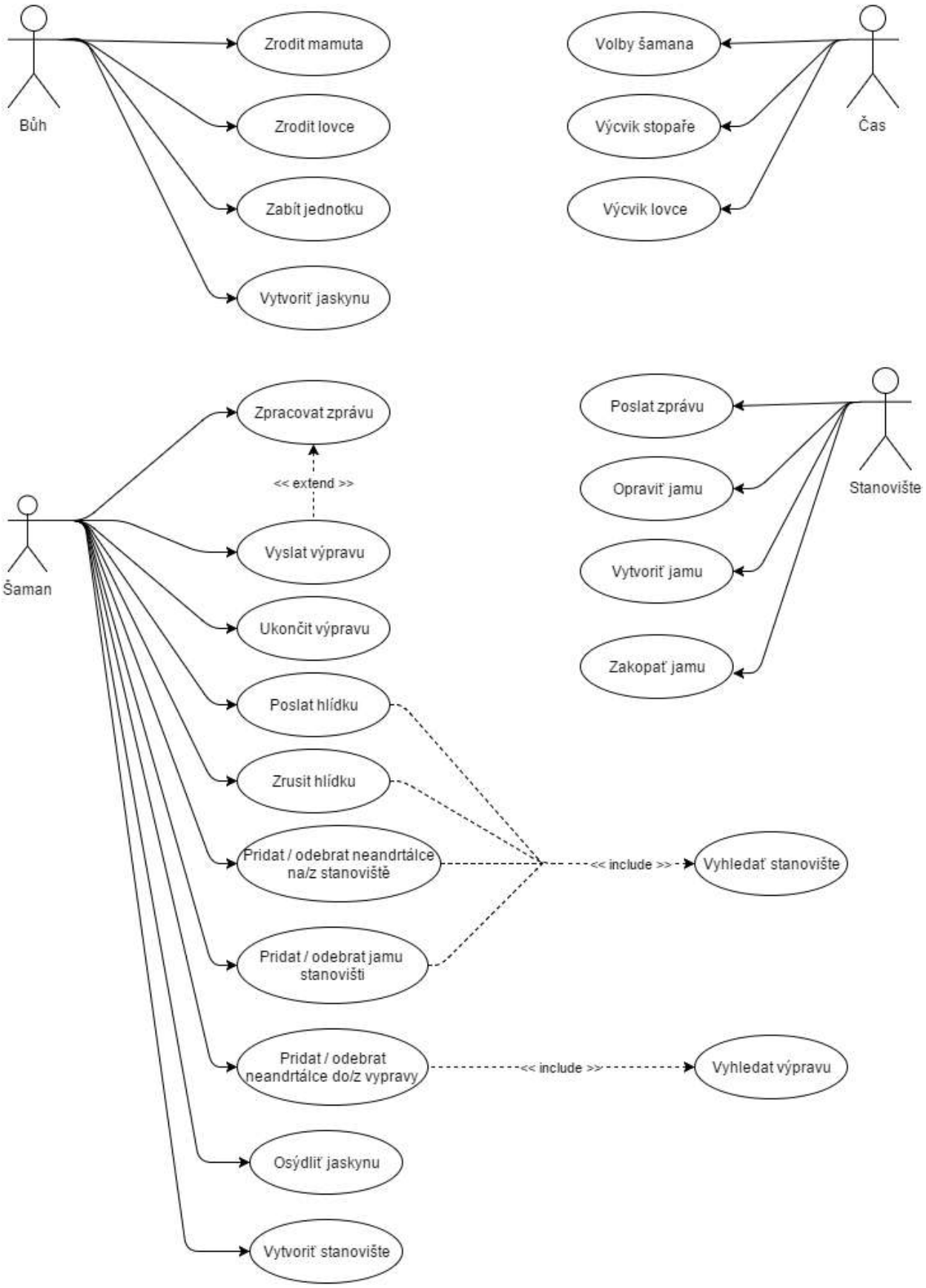
## Zadání č. 46 – Lovci mamutů

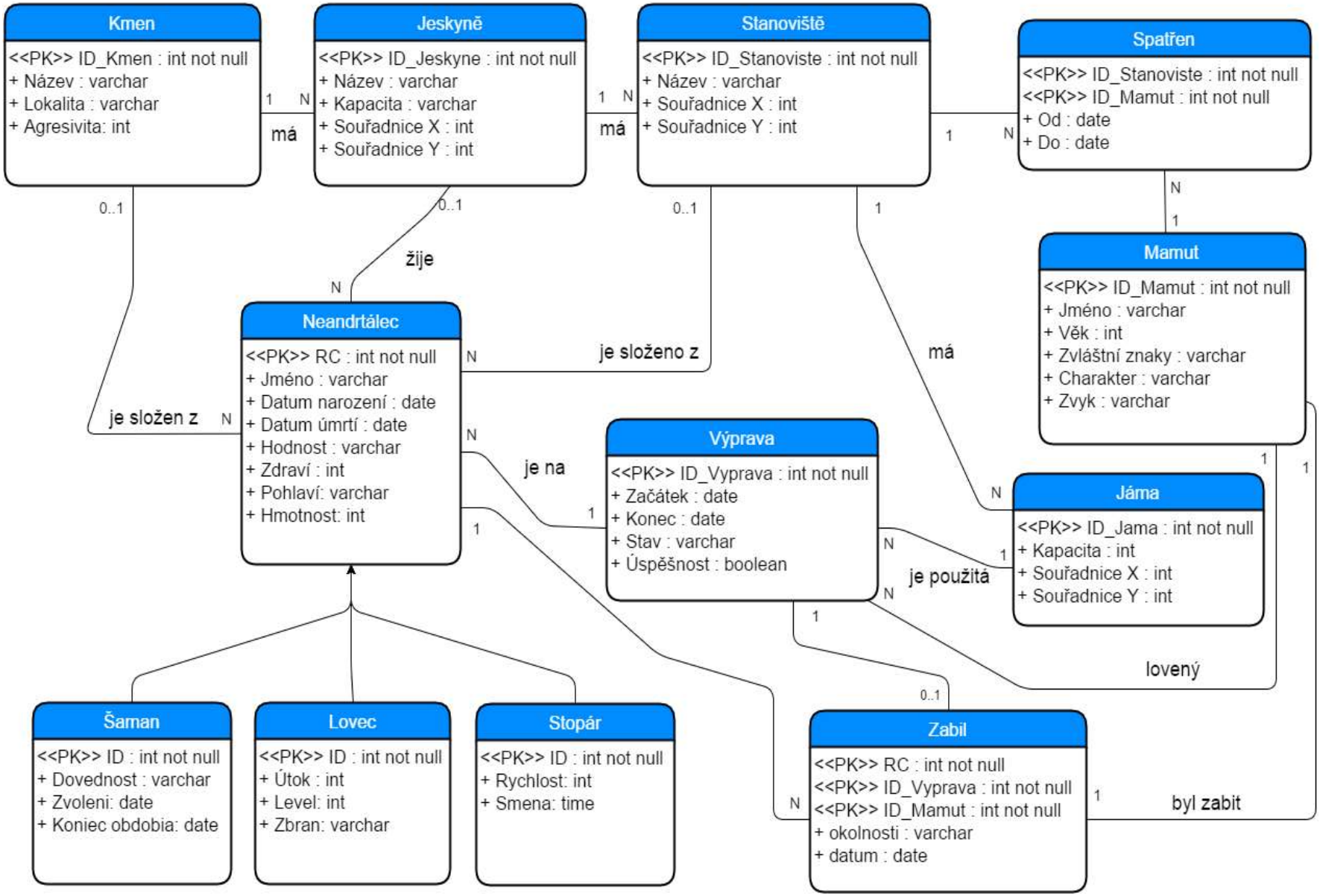
Zadání

Navrhněte informační systém pro kmen lovců mamutů. Lovci žijí v jeskyni a udržují vzdálená stanoviště, na kterých hlídky monitorují výskyt mamutů. Po krajině jsou rozmístěny mamutí jámy. Pokud hlídka vystopuje mamuta, pošle kouřovou zprávu. Zpráva obsahuje informace jako počet spatřených mamutů a výčet již známých dříve spatřených mamutů. Jeskynní ústředí zprávy eviduje a postupně zpracovává. Rozhoduje, zda na základě zprávy vyšle loveckou výpravu, které volné mamutí jámy výpravě přidělí (nemělo by se stát, že by se dvě výpravy snažily nalákat své mamuty do stejné jámy), které neandrtálce do výpravy zapojí. Na zpracovanou zprávu odpovídá kouřovou zprávou "ack". Rozhodnutí se provádí na základě informací ve zprávě, informací o neandrtálcích (zdraví, dovednosti, a pod.), informací o známých mamutech (zvláštní znaky, zvyky, charakterové vlastnosti) informacích o historii lovů (který neandrtálec kdy a za jakých okolností skolil které mamuty, v jaké jámě, který mamut kdy a za jakých rozšlapal které domorodce, a pod.) informací o probíhajících lovech (které zprávy jsou právě řešeny vyslanou skupinou, které jámy jsou právě přiděleny ke kterému lovu, kteří neandrtálci jsou volní, a pod.). (Systém pracuje podobně jako počítač. Operátor ovládá systém zvukovými příkazy, informace jsou vyhledávány a zobrazovány na stěny jeskyně či zapisovány na břidlicové destičky týmem malovačů, vyhledávačů, nosičů, a kamenotepců.)

Andrej Tichý - xtichy09@stud.fit.vutbr.cz

Anastasiia Stoika – xstoik00@stud.fit.vutbr.cz


Bůh
Zrodit mamuta
Zrodit lovce
Zabít jednotku
Vytvoriť jaskynu
Čas
Volby šamana
Výcvik stopaře
Výcvik lovce
Šaman
Zpracovat zprávu
<< extend >>
Vyslat výpravu
Ukončit výpravu
Poslat hlídku
Zrusit hlídku
Pridat / odebrat neandrtálce na/z stanoviště
Pridat / odebrat jamu stanovišti
Pridat / odebrat neandrtálce do/z vypravy
Osýdliť jaskynu
Vytvoriť stanovište
<< include >>
Vyhledať stanovište
<< include >>
Vyhledat výpravu
Stanovište
Poslat zprávu
Opraviť jamu
Vytvoriť jamu
Zakopať jamu


Kmen
<<PK>> ID_Kmen : int not null
+ Název : varchar
+ Lokalita : varchar
+ Agresivita: int
Jeskyně
<<PK>> ID_Jeskyne : int not null
+ Název : varchar
+ Kapacita : varchar
+ Souřadnice X : int
+ Souřadnice Y : int
Stanoviště
<<PK>> ID_Stanoviste : int not null
+ Název : varchar
+ Souřadnice X : int
+ Souřadnice Y : int
Spatřen
<<PK>> ID_Stanoviste : int not null
<<PK>> ID_Mamut : int not null
+ Od : date
+ Do : date
Mamut
<<PK>> ID_Mamut : int not null
+ Jméno : varchar
+ Věk : int
+ Zvláštní znaky : varchar
+ Charakter : varchar
+ Zvyk : varchar
Neandrtálec
<<PK>> RC : int not null
+ Jméno : varchar
+ Datum narození : date
+ Datum úmrtí : date
+ Hodnost : varchar
+ Zdraví : int
+ Pohlaví: varchar
+ Hmotnost: int
Výprava
<<PK>> ID_Vyprava : int not null
+ Začátek : date
+ Konec : date
+ Stav : varchar
+ Úspěšnost : boolean
Jáma
<<PK>> ID_Jama : int not null
+ Kapacita : int
+ Souřadnice X : int
+ Souřadnice Y : int
Šaman
<<PK>> ID : int not null
+ Dovednost : varchar
+ Zvoleni: date
+ Koniec obdobia: date
Lovec
<<PK>> ID : int not null
+ Útok : int
+ Level: int
+ Zbran: varchar
Stopár
<<PK>> ID : int not null
+ Rychlost: int
+ Smena: time
Zabil
<<PK>> RC : int not null
<<PK>> ID_Vyprava : int not null
<<PK>> ID_Mamut : int not null
+ okolnosti : varchar
+ datum : date
má
žije
je složen z
je složeno z
je na
je použitá
lovený
byl zabit

# Implementace 4. časti projekt do předmětu IDS

## Triggery

V projektu jsme implementovali 4. triggery z různou triviálností:

1. `kmen_trig` – tento trigger implementuje automatické generovaní primárného klíče pokud při vstupu sekvence dát do tabulky `KMEN` je položka `ID_KMEN = NULL`. Při implementaci jsme využili inkrementální sekvenci `SEQ_ID_KMEN`. Trigger se volá vždy před vložením anebo aktualizováním dát.
2. `mamut_delete` – trigger ošetřuje situaci kdy chceme vymazat mamuta, přičemž vždy musíme korektně vymazat taky všechny záznamy v kterých je mamut zainteresovaný, tak aby v databázi nenastal nekonzistentní stav. Triger se zavolá po vymazání mamuta z tabulky `MAMUT`, přičemž se postupně vymažu taky všechny záznamy z tabulek `SPATREN`, `VÝPRAVA` a `ZABIL` pomocí cizího klíče.
3. mamut_insert – automatické generovaní primárného klíče pokud při vstupu sekvence dát do tabulky `MAMUT` je položka `ID_MAMUT = NULL`.
4. jeskyne_insert- automatické generovaní primárného klíče pokud při vstupu sekvence dát do tabulky `JESKYNE` je položka `ID_JESKYNE = NULL`.

Triggery 1, 2 a 3 jsou využívaný hlavně v klientský aplikaci.

## Procedury

Vytvořily jsme také dvě netriviální procedury.

První procedura se zaměřuje na ukázku dát, který neandertálec ulovil kterého mamuta na aktuální výpravě. Procedura je implementována tak, ze když uživatel zadá nějaké rodné číslo neandertálce, procedura ověřuje, jestli dané rodné číslo patří nove vytvořenému pohledu. Když ano, vypíše jméno lovce, na které výpravě spatřil mamuta, kterého mamuta a jeho znaky. Když ale rodné číslo nepatři do vytvořeného pohledu, vypíše chybové hlášeni.

Pro první proceduru byly použity:

- `VIEW aktual_lov_info` - tabulka, která obsahuje informaci o tom, který neandertálec na které výpravě spatřil kterého mamuta;
- `CURSOR` se vstupným parametrem - proměnné s datovým typem `(table_name.column_name%TYPE)`
- proměnná `table_name%ROWTYPE` - do které se vkládá vybraný řádek z výsledku cursoru

Druha procedura je vytvořena pro ukázku práce s parametry procedury typu `IN OUT` a `OUT`.

Na začátku použije jeden cursor, který při zadaní uživatelem rodného čísla šamana zapíše jméno šamana do parametru `INFO` typu `OUT`. Pak použije druhy cursor, který má stejný vstupní parametr, který vyhledá jeskyni, kde bydli šaman. Číslo jeskyně se zapíše do parametru `CISLO` typu `IN OUT`, tím samým změní obsah tohoto parametru.

Pro druhu proceduru byli pouzity:

- `VIEW saman_info` - zobrazí info o neandertálcích, který jsou šamany, kde bydli, datum narozeni..
- Dva cursory se stejným vstupním parametrem - proměnné typu `table_name.column_name%TYPE`

V proceduře byli použity také různá ošetřeni, jestli se podařilo otevřít cursor, nebo jestli zadané parametry uživatelem jsou v tabulce, se kterou pracuji.

# Optimalizace pomocí indexu a EXPLAIN PLAN

Provedli jsme optimalizaci námi zvoleného selectu se spojením dvou tabulek a agregační funkcí `COUNT` s klauzulí `GROUP BY`. Nejprve jsme provedli `EXPLAIN PLAN` daného selectu.

```
SELECT jama.ID_JAMA, jama.SUR_X AS X, jama.SUR_Y AS Y,
count(jama.ID_JAMA) as pocet
FROM vyprava
LEFT OUTER JOIN jama ON vyprava.FK_JAMA = jama.ID_JAMA
GROUP BY jama.ID_JAMA, jama.SUR_X, jama.SUR_Y
ORDER BY 1;
```

```
-----------------------------------------------------------------------
| Id  | Operation           | Name    |Rows|Bytes| %CPU   | Time     |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         | 13 | 182 |  7 (15)| 00:00:01 |
|   1 |  SORT GROUP BY      |         | 13 | 182 |  7 (15)| 00:00:01 |
|*  2 |   HASH JOIN OUTER   |         | 13 | 182 |  6  (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| VYPRAVA | 13 |  39 |  3  (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| JAMA    | 40 | 440 |  3  (0)| 00:00:01 |
-----------------------------------------------------------------------
```

- `SELECT STATEMENT` – vykonaní dotazu select
- `SORT GROUP BY` - Třídí sadu výsledků na skupinu sloupců a agreguje vytříděný výsledek v druhém kroku . Tato operace vyžaduje velké množství paměti ke zhmotnění mezivýsledku. Zde by mněla mít optimalizace největší význam.
- `HASH JOIN OUTER` – vnější spojeni tabulek
- `TABLE ACCESS FULL` – znamená že se prochází celá tabulka bez jakýchkoliv indexů

```
-------------------------------------------------------------------------
| Id  | Operation           | Name    |Rows|Bytes| %CPU   | Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |         | 13 | 182 |  5 (20)| 00:00:01 |
|   1 |  SORT GROUP BY      |         | 13 | 182 |  5 (20)| 00:00:01 |
|*  2 |   HASH JOIN OUTER   |         | 13 | 182 |  4  (0)| 00:00:01 |
|   3 |    INDEX FULL SCAN  | I_FK_J  | 13 |  39 |  1  (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| JAMA    | 40 | 440 |  3  (0)| 00:00:01 |
-------------------------------------------------------------------------
```

- `INDEX FULL SCAN` – Optimalizace za použití indexu, kdy se použil index bez toho, abychom museli přistupovat do tabulky.

Když porovnáme provedení dotazu select před a po využití indexu je zřejmé snížení nutného čtení z disku za cenu vyššího vytížení CPU. V konečném důsledku se ale použití indexu dá považovat za optimalizaci, která by se projevila hlavně při vyšším počte záznamů.

# Materializovaný pohled a přístupová práva

Přístupová práva druhému členovi týmu jsme postupně definovali pro všechny tabulky, procedury a jeden materializovaný pohled pomocí.

- `GRANT ALL ON xtichy09.kmen TO xstoik00;`

# Pohled

Vytvořili jsme materializovaný pohled `mv1`, patrici druhému clenu tymu - `xstoik00`. Druhy clen tymu pak dává pravá na tento pohled prvnímu clenu tymu - `xtichy09`. První clen tymu to pak může zkusit zobrazit.

Po přidáni jednoho řádku do tabulky druhým cleném tymu, se pohled aktualizuje, co pak zas může ověřit první clen tymu pomoci `SELECT * FROM xtoik00.mv1`.

Pro vytvořeni materializovaného pohledu jsme na začátku vytvořili `MATERIALIZED VIEW LOG`, aby pak `MATERIALIZED VIEW` při změnách tabulky mohl používat `REFRESH FAST ON COMMIT`. Materializovanému pohledu kromě `REFRESH FAST ON COMMIT` jsme ještě nastavili této vlastnosti:

`CACHE, BUILD IMMEDIATE a ENABLE QUERY REWRITE.`

Pohled pracuje s tabulkou Kmen a vypisuje id kmenu a jeho název.