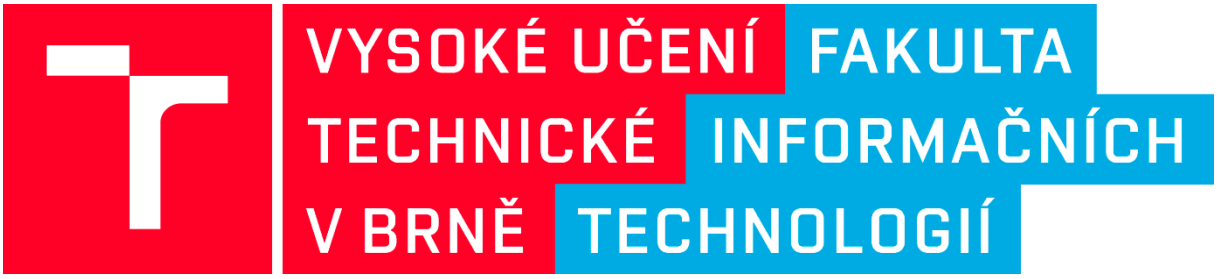
VYSOKÉ UČENÍ TECHNICKÉ v BRNĚ

Fakulta Informačných technológii



Databázové systémy

2016/2017

Konceptuálny model

Zadanie č. 49 – Bug Tracker

Jozef Urbanovský (xurban66)
Adrián Tomašov (xtomas32)

Brno 1. 5. 2017

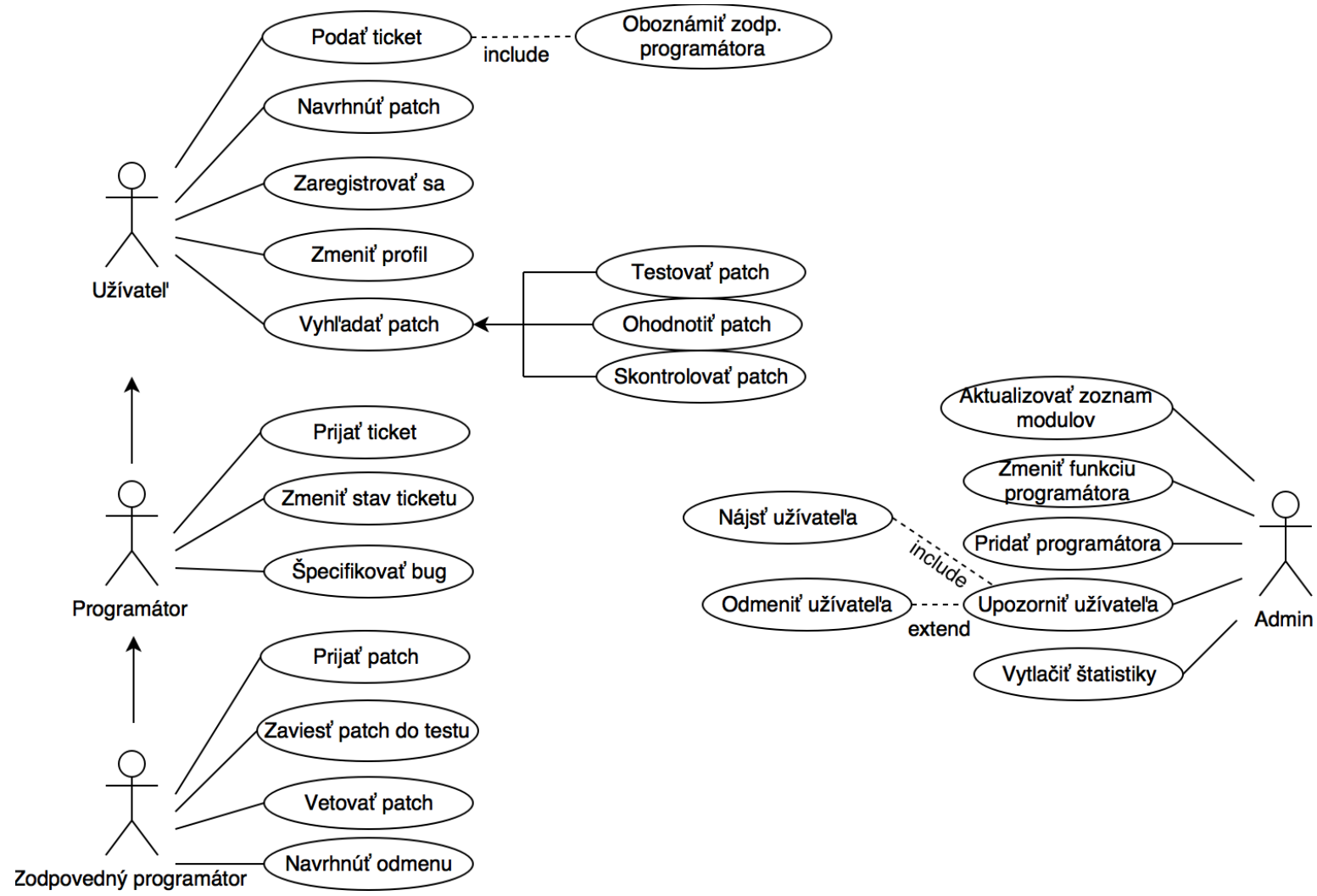
1 Zadanie

Vytvořte informační systém pro hlášení a správů chyb a zranitelností systému. Systém umožňuje uživatelům hlásit bug, jejich závažnosti a moduly, ve kterých se vyskytly, ve formě tiketů. Tikety mohou obsahovat hlášení o více než jednom bugu a stejný bug může být zahlášen více uživateli současně. Bug může (ale nemusí) být zranitelností a v tomto případě zaevidujeme i potenciální míru nebezpeční zneužití této zranitelnosti. V případě zahlášení bugů, odešle systém upozornění programátorovi, který zodpovídá za daný modul, přičemž může odpovídat za více modulů. Programátor pak daný tiket zabere, přepne jeho stav na .V řešení. a začne pracovat na opravě ve formě Patche. Patch je charakterizován datem vydání a musí být schválen programátorem zodpovědným za modul, které mohou být v různých programovacích jazycích. Jeden Patch může řešit více bugů a současně řešit více tiketů a vztahuje se na několik modulů. Samotní uživatelé mohou rovněž tvořit patche. Takové patch však musí projít silnější kontrolou než jsou zavedeny do systému. Kromě vytvoření patch rovněž evidujte datum zavedení patche do ostrého provozu. Každý uživatel a programátor je charakterizován základními informacemi (jméno, věk, apod.), ale současně i jazyky, kterými disponuje, apod. V případě opravení bugů, mohou být uživatele upozorněni na danou opravu a případně být odměněni peněžní hodnotou (podle závažnosti bugu či zranitelnosti).

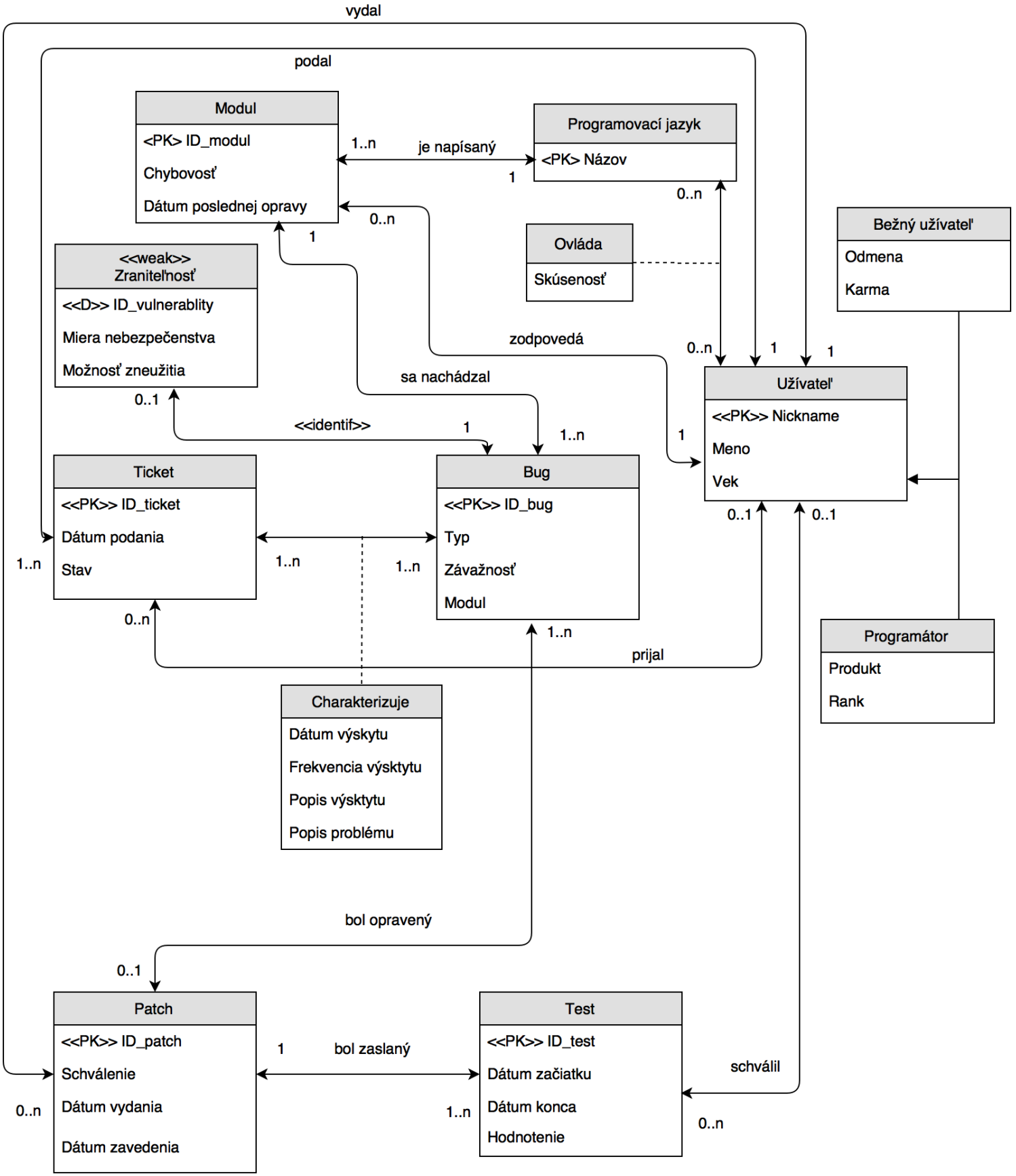
1.1 Upresnenie zadania

Informačný systém je navrhovaný ako bug tracker pre operačný systém, kde sú možné extenzívne testy. Programátor je ako zamestanec danej firmy, ktorá spravuje a vyvýja operačný systém. Každý patch prejde kontrolou programátora, ktorý je zodpovedaný za modul, v ktorom sa daný bug vyskytol a následne je prijatý, prípade vetovaný a zaslaný na repozitáre. Ktokoľvek s prístupom k repozitárom môže otestovať patch a následne ho ohodnotiť. Každý, kto chce podať ticket, musí byť zaregistrovaný. Po oprave bugu, špecifikovaného v danom tickete, by sa mal človek,ktorý podal ticket podielať na jeho teste, nakoľko sám našiel túto chybu. Každý užívateľ a programátor má karmu za prácu, ktorú vykonal. Podľa jeho karmy je vyžadované silnejšie testovanie, prípadne menej extenzívne. Beta testeri sú užívatelia a programátori, ktorí sa tak označili v systéme a chcú pomáhať s rozvojom a verifikovaním patchov.

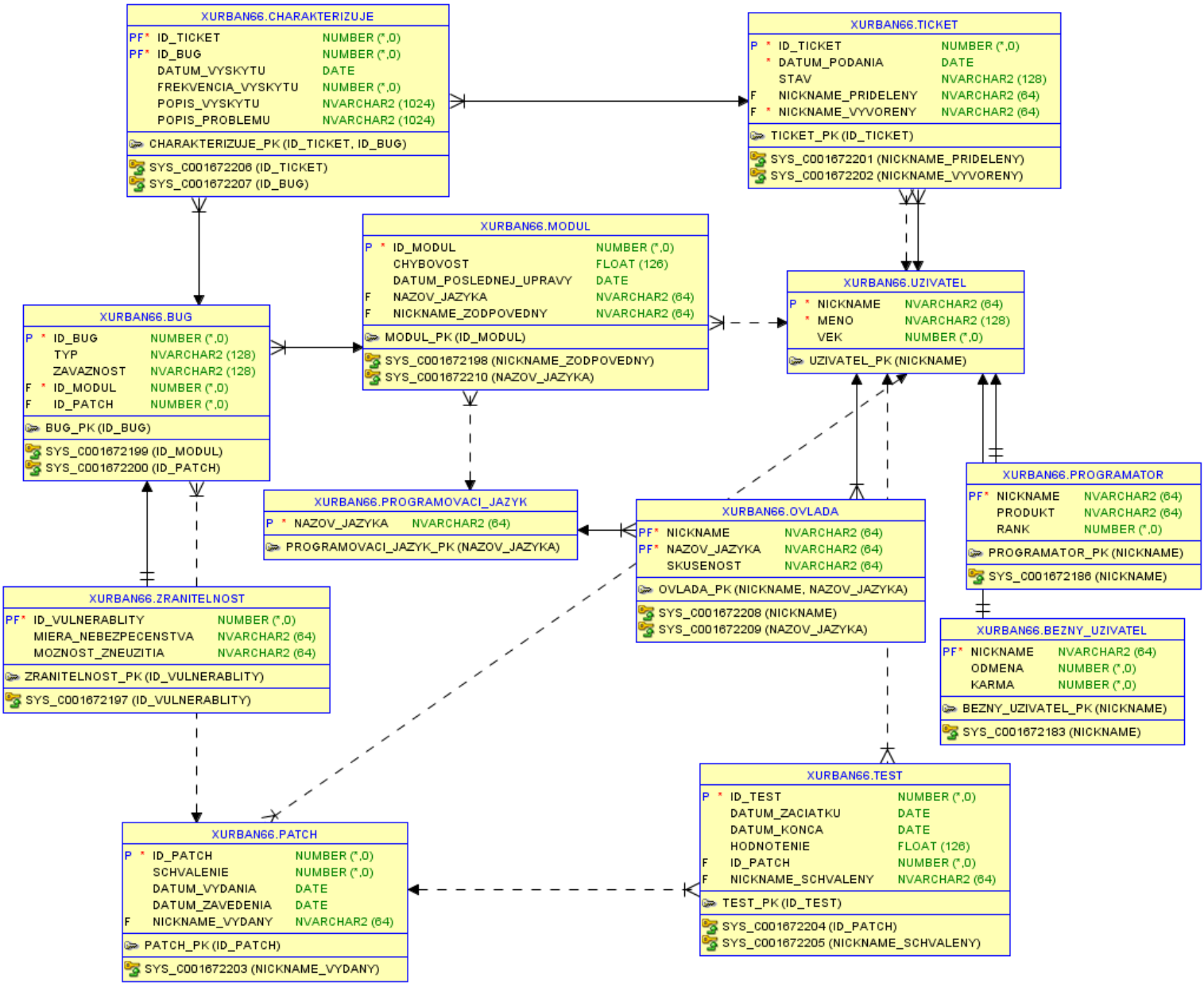
2 Use case diagram



3 ER diagram


vydal
podal
Modul
<PK> ID_modul
Chybovosť
Dátum poslednej opravy
1..n
je napísaný
1
Programovací jazyk
<PK> Názov
0..n
0..n
1
Ovláda
Skúsenosť
Bežný užívateľ
Odmena
Karma
<<weak>>
Zraniteľnosť
<<D>> ID_vulnerablity
Miera nebezpečenstva
Možnosť zneužitia
0..1
zodpovedá
0..n
1
1
sa nachádzal
Užívateľ
<<PK>> Nickname
Meno
Vek
<<identif>>
1
1..n
1
0..1
0..1
Ticket
<<PK>> ID_ticket
Dátum podania
Stav
1..n
1..n
1..n
Bug
<<PK>> ID_bug
Typ
Závažnosť
Modul
0..n
1..n
prijal
Programátor
Produkt
Rank
Charakterizuje
Dátum výskytu
Frekvencia výsktytu
Popis výsktytu
Popis problému
bol opravený
0..1
Patch
<<PK>> ID_patch
Schválenie
Dátum vydania
Dátum zavedenia
1
bol zaslaný
Test
<<PK>> ID_test
Dátum začiatku
Dátum konca
Hodnotenie
schválil
0..n
1..n
0..n

4 Finálne schéma databáze



5 Popis riešenia

5.1 Generalizácia/Špecializácia

Generalizácia/Špecializáciu sme použili pri tabuľke `Uzivatel`, ktorý zobecňuje ako zamestnanca v danej firme – `Programator`, tak aj užívateľa tohto systému, ktorý len bugy nahlasuje, prípadne sa podiela na tvorbe patchov – `Bezny_uzivatel`. Nakoľko obe generalizované entity obsahujú vo svojich tabuľkách rozdielne položky a môžu prevádzať rozidelne operácie, nebolo vhodné aby boli súčasťou tabuľky `Uzivatel`.

5.2 Implementácia

SQL skript má za úlohu prvotne vytvoriť prevedené entity z ER diagramu do tabuliek relačnej databáze, naplniť tabuľky ukážkovými dátami a zadefinovať pokročilé obmedzenia a objekty. Skript následne obsahuje príklady manipulácie s dátami a požiadavkami, demonštrujúce použitie vyššie spomínaných obmedzení a objektov. Pre obmedzenia sme použili ako TRIGGER tak aj CHECK.

5.3 Triggery

Databázové triggery sme implementovali a využili pri automatickej inkrementácii primárneho kľúča v tabuľke `Ticket` a `Bug`. Oba triggery sa spúšťajú vždy pred vkladaním dát do danej tabuľky a sú realizované pomocou sekvencie, pre pamätanie si čísla posledného pridaného primárneho kľúča. Naša databáza obsahuje atribúty, ktoré majú predpísaný formát a preto bolo nutné naimplementovať trigger, ktorý sa postará o overenie ich správnosti. Spomínaný trigger kontroluje, či sa atribút `Skusenost` tabuľky `Ovlada` zhoduje s daným formátom a spúšťa sa vždy keď sa vkladá alebo upravuje atribút `Skusenost` . Posledný implementovaný trigger je určený na automatické prepočítavanie a aktualizovanie atribútu `Chybovost` v tabuľke `Modul` po každom pridaní dát do tabuľky `Bug`. Tento trigger vypočíta podiel nahlásených bugov v danom module ku všetkým bugom, ktoré sú v systéme zaznamenané.

5.4 Procedúry

Procedúra `Vyuzivanie_prog_jazykov` slúži na vypísanie štatistík využitia programovacích jazykov užívateľmi databáze. Využíva kurzor, aby sme boli schopní jednoduchšie iterovať cez viacero riadkov tabuľky. Procedúra vypočíta individuálne pre každý programovací jazyk koľko ľudí zaregistrovaných v systéme ho ovláda a následne vypočíta podiel ľudí ovládajúcich daný jazyk ku všetkým ľudom ovládajúcim aspoň 1 jazyk. Výstupom procedúry je percentuálne zastúpenie ľudí, ktorý ovládajú daný jazyk v systéme. Ako ďalšiu procedúru sme naimplementovali `Ticket_opraveny`, ktorý má ako argument číslo ticketu, ktorý chceme overiť či už bol vyriešený a jeho bugy opravené. Do kurzoru si nahráme spojenú tabuľku `Bug` a `Charakterizuje` z ktorej si následne vytiahneme atribúty `ID_bug` a `ID_patch` potrebné pre procedúru. V prípade ak je ticket vyriešený a jeho bugy opravené vypíše sa číslo ticketu, dátum podania a dátum jeho zavedenia. V opačnom prípade sa sa vyvolá výnimka, vypíše číslo ticketu a číslo bugu, ktorý je zatiaľ neopravený.

5.5 Explain plan a index

Explain plan bol demonštrovaný na jednoduchom `SELECT` dotaze, ktorý spája 2 tabuľky s využitím agregačnej funkcie `AVG` a klauzule `GROUP BY`. Prvotne sme spustili explain plan nad dotazom bez optimalizácie, pomocou indexu. Následne sme index zadefinovali a overili výsledky spustením explain planu nad rovnakým dotazom.

Bez použitia indexu:

```
----------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     5 |  1375 |     7  (15)| 00:00:01 |
|   1 |  HASH GROUP BY      |             |     5 |  1375 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN         |             |     5 |  1375 |     6   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| PROGRAMATOR |     5 |   395 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| UZIVATEL    |    10 |  1960 |     3   (0)| 00:00:01 |
----------------------------------------------------------------------------------
```

S použitím indexu:

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |               |     5 |  1375 |     6  (17)| 00:00:01 |
|   1 |  HASH GROUP BY                      |               |     5 |  1375 |     6  (17)| 00:00:01 |
|*  2 |   HASH JOIN                         |               |     5 |  1375 |     5   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL                | PROGRAMATOR   |     5 |   395 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS BY INDEX ROWID BATCHED| UZIVATEL    |    10 |  1960 |     2   (0)| 00:00:01 |
|   5 |     INDEX FULL SCAN                 | INDEX_EXPLAIN |    10 |       |     1   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

Ako môžeme vidieť na príklade výstupov z tabuliek DBMS_XPLAN, tak pri použití indexu sa znížila cena – počet prístupov na disk, ale zvýšlo sa využitie procesoru. Pri takom malom objeme dát tento dotaz trval veľmi krátky čas, takže optimalizácia na ňom nieje zreteľná.

5.7 Materializovaný pohľad

Pri implementácii materializovaného pohľadu patriacemu druhému členovi týmu, ktorý používa tabuľky definované prvým členom týmu sme si vytvroili materializované logy, v ktorých sa uchovávajú zmeny hlavnej tabuľky a aby sa mohol pri zmenách tabuľky používať fast refresh on commit, ktorý nam urýchli tento dotaz pohľadu. Po vytvorení spomínaných logov, sa implementoval materializovaný pohľad a predviedla sa jeho funkčnosť tým, že vypísali čo materializovaný pohľad obsahoval, následne pridali dáta do tabuľky z ktorej daný pohľad čerpal, potvrdili zmeny príkazom `COMMIT` a výpisom tabuľky z ktorej pohľad vychádzal, dokázali, že pohľad sa zmenil. Pohľad bol nastavený tak, aby využíval cache pre optimalizáciu načítaných tabuliek , aby sa plnil ihneď po vytvorení a povolenie pre použitie optimalizátoru. Výsledný materializovaný pohľad vyzerá nasledovne: `CACHE BUILD IMMEDIATE REFRESH FAST ON COMMIT ENABLE QUERY REWRITE`.

6 Záver

Skript bol otestovaný v prostredí Oracle na školskom serveri Gort. Bol vytvorený a testovaný vo vývojovom prostredí Oracle SQL develeoper. K úspešnému zvládnutiu tohto projektu sme si naštudovali látku z prednášok IDS, dokumentácie Oracle Database 12c a rôzne iné internetové zdroje opisujúce prácu s databázou, napríklad W3Schools, StackOverflow, atď...