VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

DATABÁZOVÉ SYSTÉMY

Dokumentácia 5. časti projektu

Šlesár Michal `xslesa01`
Belko Erik `xbelko02`

3. mája 2021

Obsah

1 Zadanie - Bug Tracker **2**

2 Model prípadu užitia **3**

3 ER diagram **4**

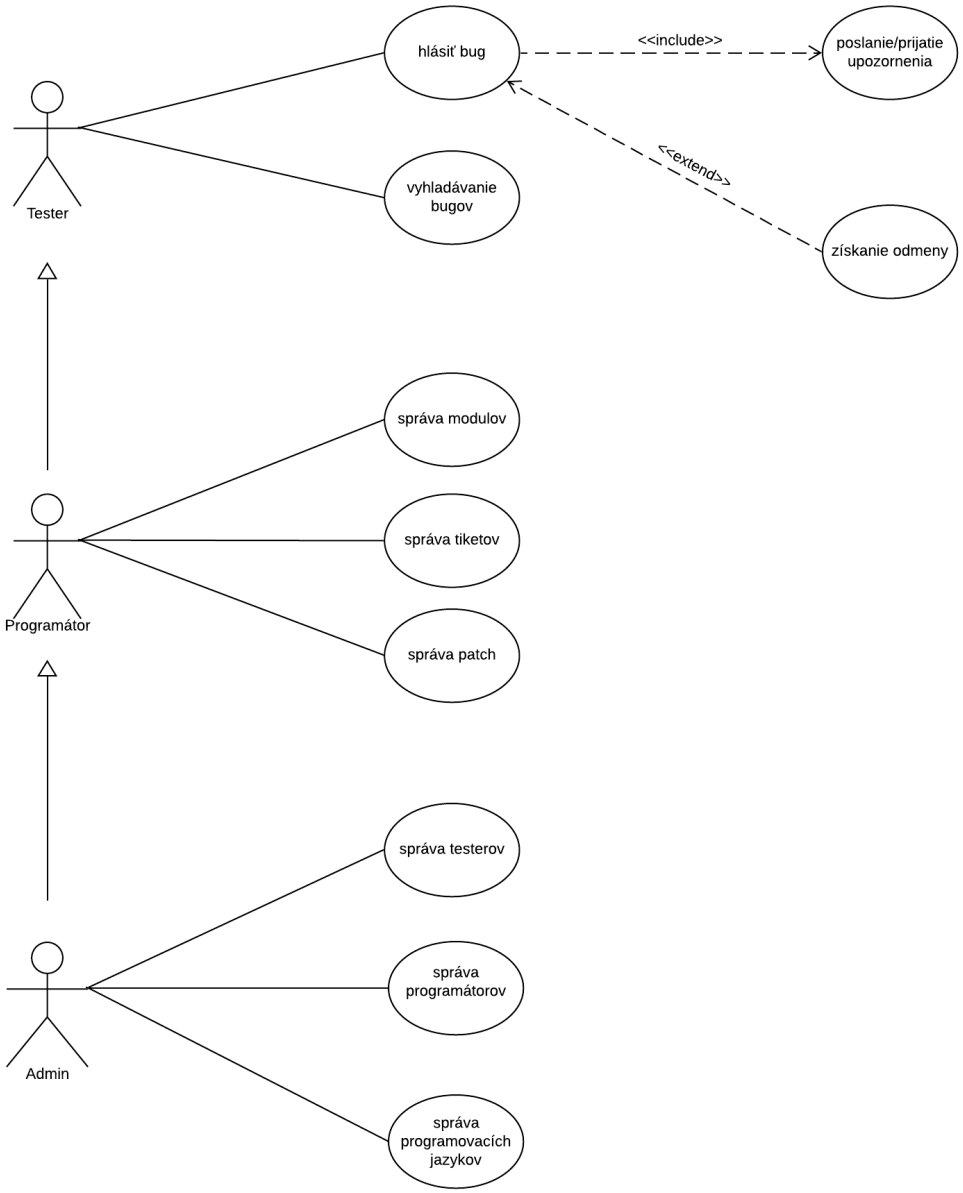
4 Implementácia 2. a 3. časti **5**
4.1 DROP 5
4.2 CREATE 5
4.3 INSERT a SELECT 5

5 Implementácia 4. časti **6**
5.1 TRIGGER 6
5.2 PROCEDURE 6
5.3 EXPLAIN PLAN a INDEX 6
5.3.1 Plán dotazu bez použitia indexu 7
5.3.2 Plán dotazu s použitím indexu 7
5.4 MATERIALIZED VIEW 7
5.5 PRÍSTUPOVÉ PRÁVA 7

1 Zadanie - Bug Tracker

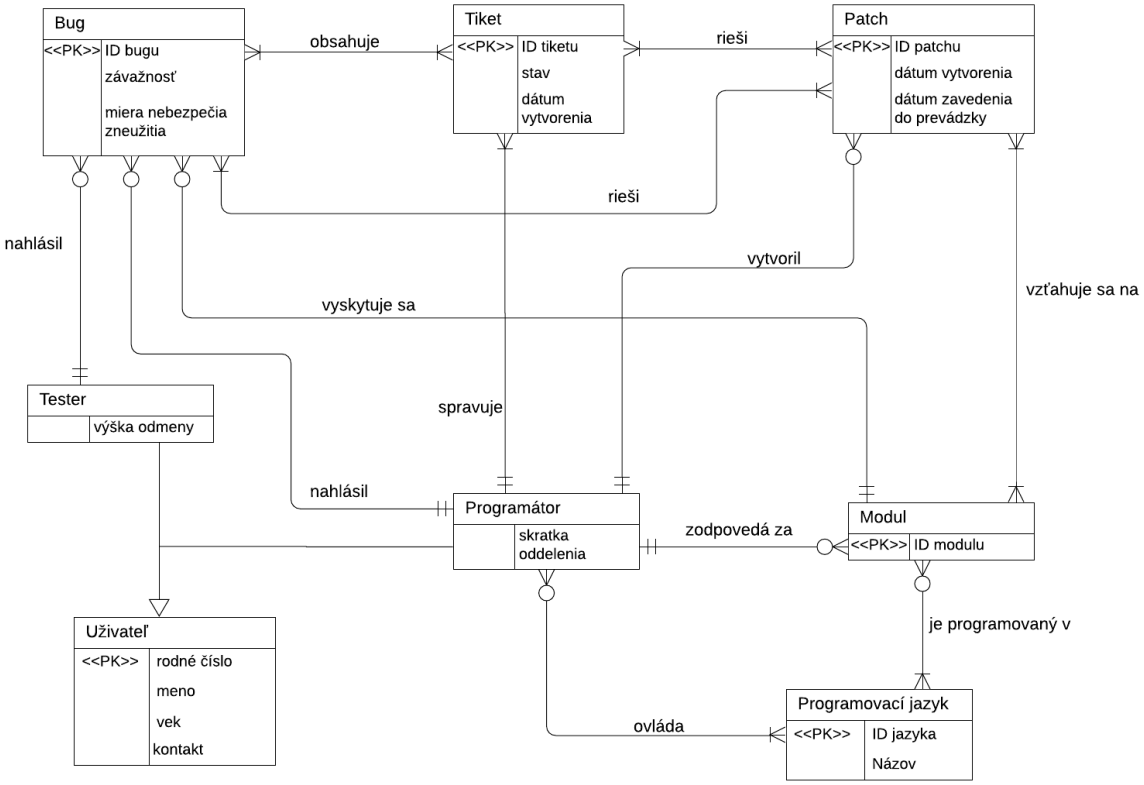
Vytvorte informačný systém pre hlásenie a správu chýb a zranitelností systému. Systém umožňuje uživateľom hlásiť bugy, ich závažnosti a moduly, v ktorých sa vyskytli, vo forme tiketov. Tikety môžu obsahovať hlásenie o viac než jednom bugu a ten istý bug môže byť nahlásený viacerými uživateľmi. Bug môže (ale nemusí) byť zraniteľnosťou a v tomto prípade zaevidujeme aj potencionálnu mieru nebezpečia zneužitia proti zraniteľnosti. V prípade nahlásenia bugov, odošle systém upozornenie programátorovi, ktorý zodpovedá za daný modul, pričom môže zodpovedať za viacero modulov. Programátor potom daný tiket zaberie, prepne jeho stav na "V riešení" a začne pracovať na oprave vo forme patchu. Patch je charakterizovaný dátumom vydania a musí byť schválený programátorom zodpovedným za modul, ktorý môže byť v rôznych programovacích jazykoch. Jeden patch môže riešiť viacero bugov a súčasne riešiť viacero tiketov a vzťahuje sa na niekoľko modulov. Okrem dátumu vytvorenia patchu taktiež evidujte dátum zavedenia patchu do ostrej prevádzky. Každý uživateľ je charakterizovaný základnými informáciami (rodné číslo, meno, vek, kontakt). Uživateľ môže byť registrovaný buď ako tester alebo ako programátor. U testera sa uchováva informácia o výške jeho odmeny, ktorá značí jeho odmenu za nahlásenie bugu. U programátora sa uchováva informácia o oddelení firmy (respektíve skratky oddelenia), v ktorom programátor pracuje a je charakterizovaný jazykmi ktorými disponuje. V prípade opravenia bugu, môžu byť testeri upozornení na danú opravu a následne odmenení peňažnou hodnotou prislúchajúcou výške ich odmeny.

2 Model prípadu užitia



Model prípadu užitia (use case diagram) sa skladá z troch aktérov - Administrátor, Programátor, Tester. Administrátor dedí všetky prípady užitia zvyšných aktérov, ale navyše okrem nich môže spravovať (pridávať, upravovať, mazať, prehľadávať) účty testerov a programátorov a spravovať dostupné programovacie jazyky. Programátor môže vykonávať činnosti ktorými disponuje aj tester a navyše môže spravovať moduly, tikety, patche a prijímať upozornenia nahlásených bugov. Tester môže nahlasovať bugy (s týmto nahlásením sa zároveň odošle upozornenie), prehľadávať už existujúce bugy a prípadne získať odmenu za nahlásenie bugu.

3 ER diagram



V ER diagrame sa nachádza hlavná entita uživateľ, z ktorej dedia dve entity, programátor a tester (generalizácia). Tieto dve entity môžu robiť odlišné úlohy. Tester prakticky môže iba nahlasovať bugy v moduloch a ďalej už entity nijako neovplyvňuje. Programátor prípadne môže taktiež nahlasovať bugy v moduloch, ovláda špecifické programovacie jazyky a zodpovedá za moduly. Modul je napísaný v nejakom, prípadne vo viacerých programovacích jazykoch a zodpovedá zaň práve jeden programátor. Zároveň sa uchováva informácia o tom ktoré programovacie jazyky programátor ovláda. Po nahlásení bugu sa vytvorí tiket ktorý tento bug obsahuje a je pridelený práve jednému programátorovi ktorý ho spravuje. Programátor taktiež vytvára patche ktoré riešia konkrétne tikety, respektíve bugy. Tieto patche sa následne vzťahujú na konkrétny modul systému.

4 Implementácia 2. a 3. časti

4.1 DROP

Najskôr bolo treba implementovať DROP, ktorý zabezpečí to, aby neprišlo ku kolíziám, tým že zruší už vytvorené tabuľky.

4.2 CREATE

Implementovali sme vytvorenie tabuliek podľa zadaného ER diagramu. Tabuliek je celkovo 14, majú príslušné primárne alebo cudzie kľúče. V implementácii sme použili aj kontrolovanie zadaných hodnôt pomocou regulárnych výrazov.

4.3 INSERT a SELECT

Insert sme použili na vloženie hodnôt do tabuliek. Zadaných hodnôt je primerane veľa, a pomohli nám pri implementácii a kontrolovaní správneho chodu scriptu.

Selectov máme v implementácii veľa, časť z nich sa nachádza v 4. časti projektu. Tých ktoré sú čisto použité iba na 3. časť projektu je 7 a využívajú rôzne tabulky. Dva selecty sú implementované pre spojenie 2 tabuliek, jeden select je pre spojenie 3 tabuliek. Následne sme implementovali dva selecty s využitím klauzuly GROUP BY s agregačnou funkciou, jeden select s využitím predikátu EXISTS, a taktiež jeden select s využitím predikátu IN a vnoreného selectu.

5 Implementácia 4. časti

V tejto časti sme implementovali databázový trigger, procedúru, materializovaný pohľad, index a skúmali plán uskutočnenia dotazu. Taktiež sme pridelili práva druhému členovi tímu.

5.1 TRIGGER

Implementovali sme dva triggery. Prvý trigger `"stary_jazyk_update"` nepovoľuje pridávať starý jazyk Pascal medzi programovacie jazyky, pri pridávaní ho nahradí za jazyk C, id jazyka neovplyvňuje.

Druhý trigger `"tiket_vyrieseny"` po vyriešení a následnom vymazaní tiketu vloží tiket naspäť s hodnotou stavu 'Vyriešený', aby v tabuľke zostal údaj o vyriešenom tikete. Ostatné hodnoty sú získané zo starého (vymazaného) tiketu.

5.2 PROCEDURE

Procedúry sme vytvorili tiež dve. Prvá procedúra `"starsie_tikety"` spočíta koľko tiketov nie je z aktuálneho kalendárneho roku a výsledok vypíše. V procedúre je použitý kurzor, ktorý vyberie hodnoty `"datum_vytvorenia"` z tiketov, dátumy sú neskôr porovnávané v cykle. Je tu použitý aj výpis ktorý oznámi počet hľadaných hodnôt. Taktiež je vyriešené ošetrenie výnimiek a príslušný výpis.

Druhá procedúra `"info_o_module"` so vstupným číselným parametrom `"zadane_id"` nájde daný modul podľa zadaného id. Následne vypíše informácie: názov modulu a rodné číslo zodpovedného programátora za daný modul. Poprípade napíše ak daný modul neexistuje. V procedúre sú použité tri kurzory, na získanie id, názvu a rodného čísla zodpovedného programátora z tabuľky modul. V tejto procedúre sú využité aj štyri premenné s dátovým typom odkazujúcim se na typ stĺpca tabulky, cyklus a výpis hľadaných hodnôt. Nakoniec je implementované ošetrenie výnimiek a príslušný výpis.

5.3 EXPLAIN PLAN a INDEX

Plán je použitý dvakrát, raz bez použitia indexov a druhýkrát s použitím indexov. V oboch plánoch je select, ktorý využíva GROUP BY s agregačnou funkciou. Select zistuje: vekový priemer programátorov ktorých jazykov, je vyšší ako 20 rokov a koľko programátorov daný jazyk ovláda? Funkcia filtruje jazyky podľa priemerného veku programátorov, ktorí v ňom programujú.

Indexy sú implementované dva. `"index_programator"` vytvára rýchlejší prístup k dátam `"rodne_cislo"` a `"vek"` z tabuľky `"uzivatel"`. `"index_programator_jazyk"` zasa vytvára rýchlejší prístup k dátam `"jazyk_id"` a `"programator_rodne_cislo"` z tabuľky `"programator_jazyk"`. Na následujúcej strane sú ukážky plánov pred a po použití indexov. Možno si všimnúť že bez indexu sa prehľadáva celá tabuľka a s indexom sa prehľadáva len určitá časť definovaná indexom. Takže používanie indexov je väčšinou efektívnejšie. Vidno to v tabuľke v stĺpci využitia procesoru `Cost (%CPU)`.

5.3.1 Plán dotazu bez použitia indexu

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                      |     5 |   435 |    10  (20)| 00:00:01 |
|   1 |  SORT ORDER BY           |                      |     5 |   435 |    10  (20)| 00:00:01 |
|*  2 |   FILTER                 |                      |       |       |            |          |
|   3 |    HASH GROUP BY         |                      |     5 |   435 |    10  (20)| 00:00:01 |
|*  4 |     HASH JOIN            |                      |     5 |   435 |     8   (0)| 00:00:01 |
|*  5 |      HASH JOIN           |                      |     5 |   335 |     5   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL  | jazyk                |     3 |   141 |     3   (0)| 00:00:01 |
|   7 |       INDEX FAST FULL SCAN| programator_jazyk_id |     5 |   100 |     2   (0)| 00:00:01 |
|   8 |      TABLE ACCESS FULL   | uzivatel             |     6 |   120 |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

5.3.2 Plán dotazu s použitím indexu

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                      |     5 |   435 |     7  (29)| 00:00:01 |
|   1 |  SORT ORDER BY           |                      |     5 |   435 |     7  (29)| 00:00:01 |
|*  2 |   FILTER                 |                      |       |       |            |          |
|   3 |    HASH GROUP BY         |                      |     5 |   435 |     7  (29)| 00:00:01 |
|*  4 |     HASH JOIN            |                      |     5 |   435 |     5   (0)| 00:00:01 |
|*  5 |      HASH JOIN           |                      |     5 |   335 |     4   (0)| 00:00:01 |
|   6 |       TABLE ACCESS FULL  | jazyk                |     3 |   141 |     3   (0)| 00:00:01 |
|   7 |       INDEX FULL SCAN    | index_programator_jazyk |     5 |   100 |     1   (0)| 00:00:01 |
|   8 |      INDEX FULL SCAN     | index_programator    |     6 |   120 |     1   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------
```

5.4 MATERIALIZED VIEW

Vytvorenie materializovaného pohľadu sme použili na zobrazenie mien užívateľov ktorí sú iba testeri. Následne sme si dali dvakrát zobrazit materializovaný pohľad pred a po pridaní hodnoty do tabuľky. Materializovaný pohľad novo pridanú položku nezobrazuje. Toto vzniklo tým, že materializovaný pohľad pri svojom vytvorení vytvorí novú tabuľku, kde uloží dáta ktoré obsahuje. Tieto dáta sa v praxi obnovujú periodicky, ale my sme toto obnovenie nenastavili. Je zakomentované v našej implementácii materializovaného pohľadu `--REFRESH ON COMMIT`.

5.5 PRÍSTUPOVÉ PRÁVA

Prístupové práva sme pridelili druhému členovi tímu, t.j. `xbelko02`. Prístupové práva na všetky tabuľky, na obidve procedúry a aj na MATERIALIZED VIEW.