MMORPG hra

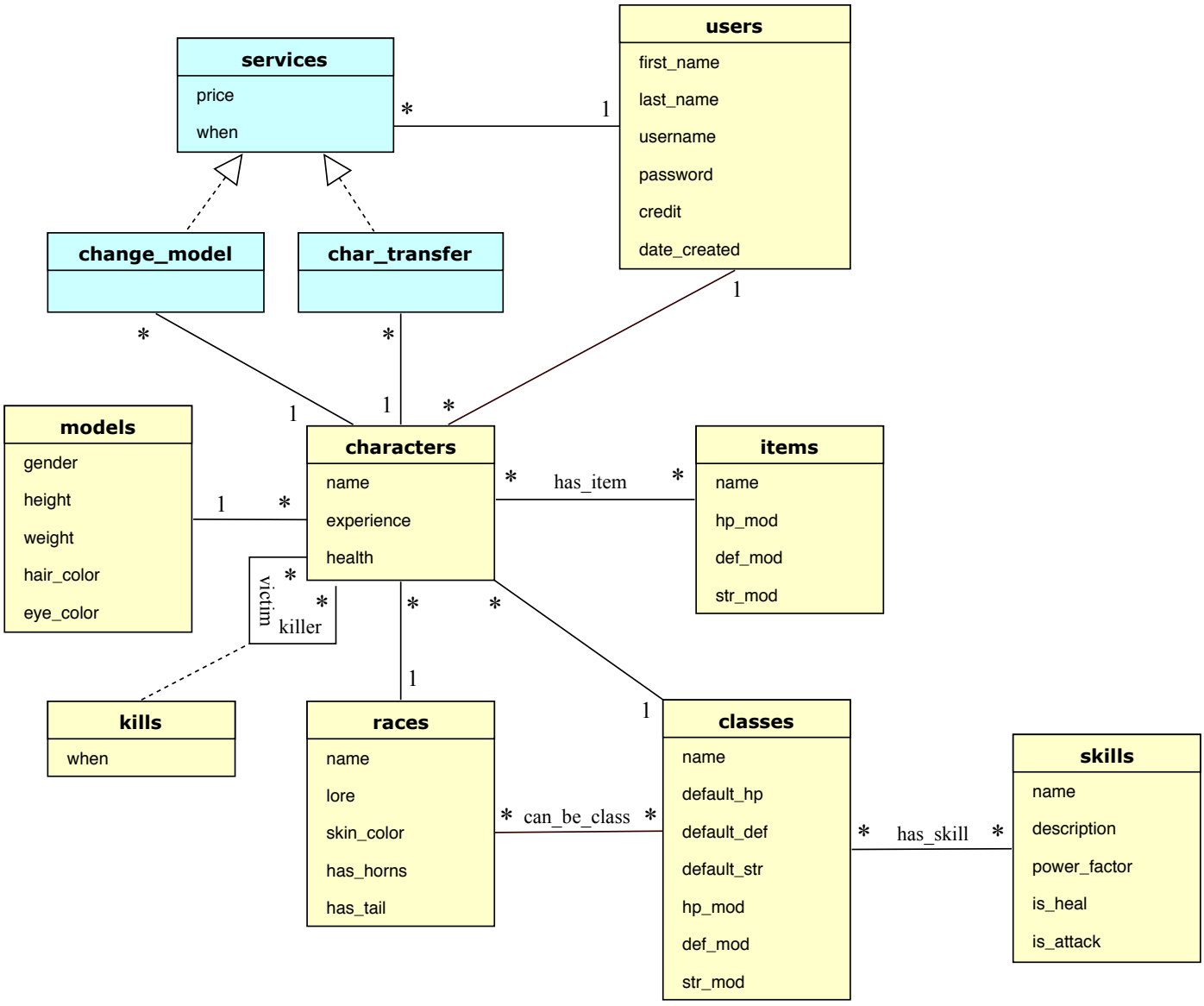
Záverečná správa
Semestrálny projekt na Databázy (2)

Lukáš Gajdošech
9. mája 2018

1 O dokumente

Dokument predstavuje záverečnú správu k projektu odovzdávaného v rámci predmetu Databázy (2). Opisuje vytvorený systém MMORPG hry, v ktorej sú kontá hráčov a ich postavy. Postavy majú svoju rasu, triedu, schopnosti a aj predmety. S väčšinou týchto entít vieme manipulovať pomocou CRUD operácii. Medzi postavami môžeme v systéme odsimulovať jeden útok či liečenie, alebo aj celý súboj. Víťazstvá postáv v súbojoch sú evidované, využívajú sa na niekoľko štatistík a ich majitelia sú odmeňovaní. Hráči môžu zo svojich kont pristupovať aj k plateným službám, ako prevod postavy a zmena vzhľadu.

2 Dátový model



Obr. 1: Entitno relačný model systému MMORPG hry

Poznámka: tento model prešiel od pôvodného modelu viacerými vylepšeniami, ktoré popíšem nižšie

Na *obrázku 1* možno vidieť entitno relačný model systému MMORPG hry. Systém uchováva účty hráčov (`users`). Každý hráč môže mať viacero postáv (`characters`). Každá postava má model vzhľadu (`models`), ktorý popisuje vzhľadové črty postavy a pohlavie. Viacero postáv môže mať jeden model vzhľadu.

Postava má okrem mena aj ďalšie údaje, ako napríklad aktuálny počet bodov skúsenosti. Z tohto údaja sa dá vypočítať jej level. Pre jednoduchosť uvažujeme, že každých 100 bodov je jedna úroveň. Rozhodol som sa pre takéto vylepšenie, lebo je vďaka tomu level vždy jednoznačný a nemôže sa stať, že sa napríklad postave pridajú body a *zabudne* sa zvýšiť level. V rámci implementácie mi to ušetrilo veľa starostí. V zásade to šetrí aj miesto v pamäti a napriek tomu užívateľ nie je ukrátený o informáciu o leveli (vždy sa vypočíta). Maximálny level je 20, teda 20 000 bodov skúsenosti. Postava má ďalej aj aktuálnu úroveň zdravia (samozrejme nemôže prekročiť svoje maximum) a je nejakej rasy (`races`) a triedy (`classes`).

Vzťah medzi týmito dvoma tabuľkami hovorí o tom, ktorá rasa môže byť akej triedy. Triedy sú vo vzťahu so schopnosťami (`skills`), pričom jedna trieda môže mať viacero schopností a jednu schopnosť môže mať viacero tried. Pri schopnostiach je okrem názvu, popisu a faktoru sily aj informácia o tom, či ide o liečivú alebo útočnú schopnosť.

Trieda zároveň určuje východzie (default) hodnoty zdravia, obrany a sily. To sú základné vlastnosti postavy tejto triedy na nultom leveli. Zároveň je tu aj informácia o modifikátoroch týchto vlastností, čo sú nejaké konštanty, o ktoré sa jednotlivé vlastnosti pri dosiahnutí nového levelu zvýšia. Postava má aj predmety (`items`), pričom jedna postava môže mať viacero predmetov a jeden predmet môže mať viacero postáv (v hre môže existovať nekonečne veľa kópií daného predmetu). Oproti pôvodnému návrhu som vypustil typy predmetov, keďže v princípe nič nemenia a bola by to zbytočná komplikácia. Aj pri predmetoch sú modifikátory vlastností postavy, ale tu predstavujú nejakú percentuálnu hodnotu, o ktorú tento predmet vlastnosť zvyšuje, respektíve znižuje, či nemení. Pôvodne som uvažoval, že sa aktuálne vlastnosti postáv budú uchovávať v tabuľke (`stats`). Nakoniec som ale túto myšlienku vypustil a vlastnosti sa vždy dynamicky vypočítajú (napr. pred súbojom).

> *Príklad: Hráč **Leeroy** má postavu **Jenkins**. Táto postava je triedy **bojovník**. Pre túto triedu je východzia sila 50 a modifikátor sily je +7. Na prvom levely má teda **Jenkins** silu 50, no po dosiahnutí druhej úrovne sa zvýši na 50+7 = 57. Podobne pri treťom levely to bude už 57 + 7 = 64. **Jenkins** ale získa aj nový predmet – **Arthasov meč**, ktorý zvyšuje silu o 15%. Jeho výsledná sila (dymamicky vypočítaná) v tomto momente teda bude 64 * 1.15 ≅ 74.*

Medzi dvoma postavami je vzťah (`kills`), ktorý identifikuje vraha a obeť spolu so záznamom času. To slúži na štatistické účely, teda výpis ktorá postava zabila najviac iných postáv za posledný týždeň a podobne.

Hráč má prístup aj k dvom službám (`services`). Konkrétne prevod postavy (`char_transfer`) a zmena vzhľadu postavy (`change_model`). Tá je realizovaná tak, že sa postave buď zmení prislúchajúci model vzhľadu na iný, už existujúci, alebo sa vytvorí celkom nový. Za služby platí užívateľ kreditmi. Pôvodne som na uchovanie informácie o kúpe uvažoval väzobnú tabuľku (`payments`). Pri implementácii som si ale uvedomil, že každá entita zo services sa viaže práve k jednej platbe, keďže sa to vždy týka nejakej konkrétnej postavy (pri prevode aj zmene vzhľadu). Medzi užívateľmi a službami pôvodný vzťah m:n kvôli tomunedáva zmysel. Čas a cena platby sa preto uchováva priamo v tabuľke (`services`).

3 Zoznam funkcií

Poznámka: Funkcie vychádzajú zo zadania témy projektu (https://goo.gl/AG7Ubd).

Systém umožňuje vylistovať, pridávať, odoberať, mazať, upravovať a zobrazovať detail (CRUD operácie):

- hráčov,
- postáv,
- tried,
- rás (iba vylistovať)

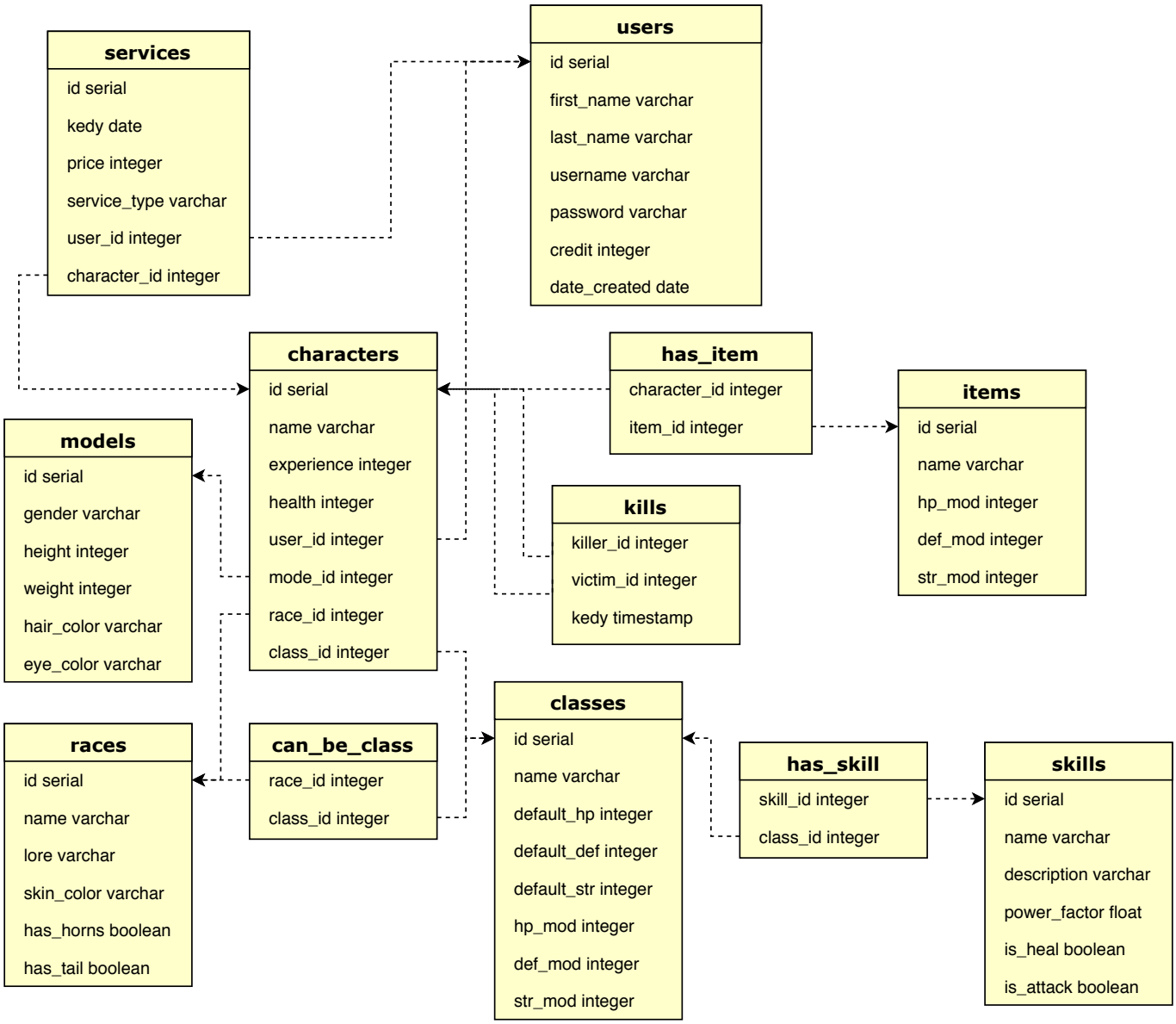
Ďalej systém umožňuje

- pridať nejakej postave predmet
- pridať triedu k rase
- pridať / vymazať schopnosť ku triede
- vylistovať schopnosti triedy
- vypísať poplatky hráča
- odsimulovať jeden útok alebo liečenie, prípadne aj celý súboj v krokoch s pozastavením, aby bolo možné vidieť živý priebeh. Hráči si volia akcie náhodne a ich postavy bojujú, až dokým niektorá z nich neumrie.
- previesť postavu z jedného účtu na druhý (služba hráča)
- zmeniť vzhľad postave (služba hráča)
- vyhodnotiť najväčšieho zabijáka v zvolenom mesiaci, pričom ocenení sú aj až 3 hráči, ktorí ho zabili najčastejšie (pomstitelia)

Systém zobrazuje nasledovné reporty/štatistiky:

- pre každý týždeň a každý mesiac vypíše postavu, ktorá za toto obdobie porazila najviac postáv, pričom ich počet aj vypíše (zvlášť pre obe pohlavia)
- Hráčov v každom týždni rozdelí podľa toho, či percentil ich zabití tvorí 90% a zvyšok. Pre obe vzniknuté skupiny zvlášť identifikuje deň v týždni a hodinu, kedy porazili najviac postáv.

4 Relačná databáza



Obr. 2: Relačný model dát systému MMORPG hry

Obrázok 2 zobrazuje výslednú relačnú databázu, ktorá vznikla transformovaním entitno relačného modelu z obrázku 1. Podmnožiny `services` boli transformované stratégiou *všetky množiny do jednej tabuľky*, ale namiesto stĺpcov s boolean hodnotou (typu `is_change_model`) som sa rozhodol pre stĺpec `service_type`, ktorý obsahuje slovný identifikátor danej služby. Spravil som to tak kvôli tomu, aby bol systém jednoducho rozšíriteľný o ďalšie typy služieb. Kredity a ceny služieb uchovávame v `integeroch`, keďže v našej hre sú iba celočíselné sumy hernej meny, netreba `numeric`. Stĺpce vyjadrujúce časový záznam som pomenoval `kedy`, lebo `when` je rezervované `SQL` slovo.

5 Organizácia kódu

Pri tvorbe aplikácie som sa v tomto smere veľmi inšpiroval vzorovým projektom. Rovnako je preto aplikácia naprogramovaná v Jave, využíva vzory *Row Data Gateway, Transaction Script*, ale napríklad aj návrhový vzor *Singleton.* Prístup do DB je riešený cez *JDBC* a zdrojový kód je rozdelený do balíkov. Namiesto konvencie doménového spôsobu pomenovania som ich ale kvôli väčšej prehľadnosti usporiadal jednoduchšie.

`root`

Balík obsahuje triedu `Main`, v ktorej sa otvorí spojenie a uloží v `DbContext` a taktiež spustí hlavné menu. Obsahuje aj metódu `runScript(String filePath)`, ktorá slúži na spustenie `sql` skriptu (napr. `create_script.sql`).

`root.rdg`

Veľmi podobné ako vzorový projekt. Akurát tu nie je `Row Data Gateway` pre každú tabuľku v databáze, keďže nie so všetkými aplikácia priamo pracuje. Používam abstraktnú triedu `BaseGateway` zo vzorového projektu a `BaseFinder` som iba rozšíril o metódu `findByStr`, ktorá umožňuje vyhľadávať aj podľa stĺpcu s reťazcom. Zopár tried pre tabuľky ale nededí z `Row Data Gateway`. Nad niektorými totiž nepotrebujeme vykonávať všetky CRUD operácie, ale iba ich vyberať, prípadne aktualizovať (napr. `items`). Niektoré triedy museli byť mierne premenované, kvôli rezervovaným slovám jazyku Java (napr. `Class` na `Klassa`). Sú tu aj `Row Data Gateway` pre vypočítané tabuľky, ako `Stats` alebo `JackStatistic`.

`root.ts`

Tento balík obsahuje kód zložitejších doménových funkcií. Obsahuje navyše aj dvojicu tried pre špeciálne výnimky, ktoré môžu pri transakciách nastať. Tie sú samozrejme realizované vzorom `Transaction Script` a delíme ich do troch kategórii. `PlayerServices` sa týkajú platených služieb hráča a `GameActions` herných akcií, čiže súbojov, prideľovaniu bodov skúsenosti a podobne. Ešte sú tu `OtherActions`, ktoré sa starajú napríklad o pridávanie dát do väzobných tabuliek (operácie ako pridaj schopnosť triede).

`root.ui`

Balík s kódom užívateľského rozhrania. Realizácia je prakticky totožná so vzorovým projektom. Teda jednoduché rozhranie v príkazovom riadku, v ktorom môžeme vyberať z číselných možností. Do abstraktnej triedy Menu som navyše pridal niekoľko reťazcových konštánt s takzvanými `ANSI Escape Codes`. Tie umožňuje farebný výpis, čo sprehľadňuje rozhranie.

`root.sql`

Tento balík obsahuje kód `SQL` skriptov. Konkrétne `create_script.sql` slúžiaci na vytvorenie štruktúry tabuľky a `generate_script.sql`, ktorý tabuľku naplní dátami, vytvorí potrebné funkcie, jeden materializovaný pohľad a aj niekoľko indexov.

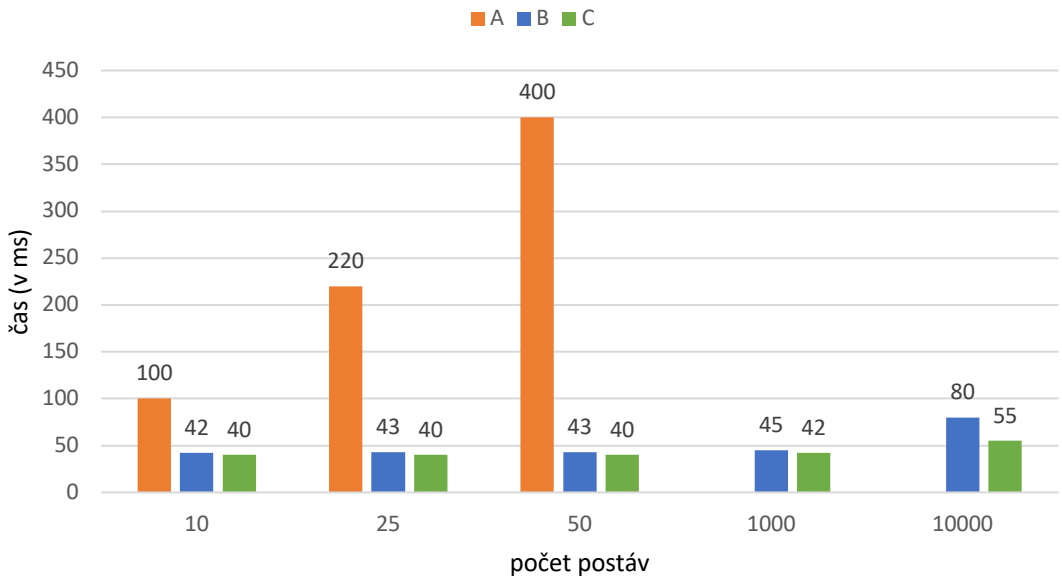
6 Optimalizácia

Menším zlepšením je vytvorenie `indexu` nad tabuľkou `services` podľa stĺpcu `user_id`. Z tejto tabuľky vyberáme pri výpise poplatkov za služby pre hráča. Pre veľké dáta (veľa záznamov o službách) sa začínala prejavovať neefektívnosť sekvenčného prehľadávanie. Vytvorenie `indexu` prinieslo zopár ušetrených milisekúnd:

Veľkosť tabuľky services (v riadkoch)	rýchlosť	
	Bez použitia indexu (v ms)	S použitím indexu (v ms)
1000	0.5	0.05
10 000	2	0.05
100 000	35	0.05

Tab. 1: Porovnanie rýchlosti výberu služby podľa id hráča

Okrem toho sa mi ale podarila omnoho významnejšia optimalizácia. Pri výpise postavy mi prišlo vhodné mať aj slovnú informáciu o tom, akej je triedy. Pri simulovaní bojov alebo aj iných herných udalostiach je to dôležitá informácia a chceme vedieť, či je postava napríklad *Warrior* a nie len vidieť jej `class_id`. Ponúka sa niekoľko riešení. Jednoznačne najhorším je sa pri každom načítaní nejakej postavy pozrieť aj do tabuľky `classes` a vybrať si meno jej triedy (riešenie A). Takto som to pôvodne urobil, no potom som si uvedomil, že napríklad pri výpise 10 000 postáv je to neúnosný počet samostatných dopytov. Druhým riešením (riešenie B) je pri výbere z `characters` `joinovať` v `selecte` tabuľku `classes`. To už je omnoho rýchlejšie. Napadlo mi však ešte rýchlejšie riešenie a navyše pri druhom mi trocha vadilo, že sa pri *jednoduchej* operácií nad tabuľkou `characters` musíme pozerať aj niekam inam. Podľa mňa teda najlepšie riešenie (riešenie C) je na začiatku programu načítať všetky triedy a vytvoriť v `Jave` mapu `(class_id ↦ class_name)`. To výber masívne zrýchlilo a navyše to je operácia nezávislá od počtu postáv v tabuľke `characters`. Výber hodnoty z mapy je totiž zanedbateľne rýchla operácia. Rozdiely v rýchlosti zobrazuje nasledujúci graf. Ďalšiu významnú optimalizáciu spomínam aj v nasledujúcej časti 7...



Graf. 1: Porovnanie rýchlosti výberu postáv

Poznámka: pre počet postáv 1000 a viac už riešenie A neuvažujeme, pretože je nepoužiteľne pomalé

7 Vybraný riešený problém

Jednoznačne najväčšiu výzvu pre mňa predstavoval štatistický výpis:

> *Vyťaženosť podľa typu hráča: Hráčov zoraďte podľa toho, koľko postáv poprazili všetky ich postavy dokopy za týždeň. Rozdeľte hráčov podľa toho, či ich percentil tvorí 90% a viac a zvyšok. Pre obe skupiny identifikujte zvlášť deň v týždni a zvlášť hodinu, kedy porazili najviac postáv.*

V druhom štatistickom výpise (Jack Rozparovač) mi stačilo napísať vhodný SELECT s použitím agregačných funkcii. Tu som ale najskôr ani nevedel kde začať. No a keď už som aj dospel k fungujúcemu riešeniu, tak bolo pre veľké dáta *nepoužiteľne pomalé*... Poďme ale postupne.

Najskôr som sa rozhodol výpis „rozbiť" do niekoľkých menších a postupne vybudovať finálnu verziu. Vzniklo mi tak niekoľko pomocných SQL funkcii:

- `player_kills()`, vracia trojstĺpcovú tabuľku – *týždeň*, *id* hráča a *počet zabití* všetkých jeho postáv v týždni.
- `percentile_for_week(tyzden)` – vráti potrebný počet zabití, ktorý postavy hráča musia mať, aby bol jeho percentil v danom týždni 90 a viac.

Týmito dvoma jednoduchými funkciami som vyriešil veľkú časť problému a už samé o sebe poskytujú zaujímavé informácie. Napr. `player_kills()` môže byť použité ako samostatný štatistický výpis – hráči a zabitia ich postáv v týždňoch. Navyše vďaka nim vieme hráčov zoradiť a rozdeliť do dvoch skupín pre každý týždeň. To mi umožnilo vytvoriť odľahčenú verziu finálneho výpisu s počtom hráčov v danej skupine a súčtom ich zabití. Zatiaľ teda bez dňa a hodiny, kedy zabili najviac postáv:

```
-------------------------------------------------------------------------------------------------------
|squad:     |week:     |players    |kills:     |best_day:  |best_hour: |
-------------------------------------------------------------------------------------------------------
|CASUALS    |1          |278        |377        |LITE       |0          |
|PROS       |1          |76         |271        |LITE       |0          |
|CASUALS    |2          |333        |538        |LITE       |0          |
|PROS       |2          |38         |172        |LITE       |0          |
                                      ...
-------------------------------------------------------------------------------------------------------
```

ukážka „LITE" štatistického výpisu; CASUALS sú hráči s percentilom pod 90, PROS s 90 a viac

Takáto verzia ešte bežala ako-tak dobre aj na väčších dátach. V SELECTE som sa ale pre každého hráča zvlášť pýtal, či je jeho percentil 90 a viac a podľa toho ho zaradil do príslušnej skupiny. Zakaždým sa pritom volala funkcia `percentile_for_week()`. No a tá nie je zrovna rýchla. Z `kills` si vždy musí vybrať záznamy z daného týždňa, vytvoriť skupiny hráčov s ich zabitiami a nájsť hodnotu s percentilom 90 a viac. Už z tohto slovného popisu je zrejmé, že to nie je najlacnejšia operácia, no pre teraz to ignorujme a poďme ďalej.

V tomto štádiu som potreboval funkciu na zistenie najlepšieho dňa a hodiny v týždni pre každú skupinu zvlášť. Vyriešil som to veľmi priamočiaro, a to vytvorením dvoch funkcií – `best_day_hour_pros()` a `best_day_hour_casuals()`

Obe sa pozrú do `kills`, spravia GROUP BY nad týždňom, dňom a hodinou záznamov (pomocou funkcie `date_part()`) a záznamy zostupne usporiadajú podľa `count(*)`, teda podľa počtu zabití v danej hodine. V každom týždni nás ale zaujíma len top deň a top hodina, nakoniec sa teda spraví `DISTINCT ON` týždeň. Jediný rozdiel je v tom, že `best_day_hour_pros()` uvažuje iba postavy hráčov s percentilom 90 a viac, no a `best_day_hour_casuals()` ostatných.

Vďaka tomu som konečne mohol vybudovať celý štatistický výpis...:

squad:	week:	players	kills:	best_day:	best_hour:
CASUALS	1	278	377	Monday	10
PROS	1	76	271	Friday	6
CASUALS	2	333	538	Wednesday	16
PROS	2	38	172	Monday	12
		...			

ukážka „FULL“ štatistického výpisu; CASUALS sú hráči s percentilom pod 90, PROS s 90 a viac

...len aby som zistil, že pre počet postáv (riadky tabuľky `characters`) a zabití (`kills`) viac ako 100 je *nepoužiteľne pomalý*. Napríklad pri 1000 postavách a 10 000 zabitiach som sa ani po pol hodine nedočkal výsledku.

Po dňoch úpenlivého dumania mi napadlo (v skutočnosti mi to poradil Šaňo), že dáta pre uplynulé týždne sa už nikdy nezmenia. Je teda hlúposť ich vypočítavať vždy na novo a to dokonca v rámci rôznych `subselectov` viackrát. Riešením je preto zhmotnený pohľad. Samozrejme som ale nemohol spraviť zhmotnený pohľad z toho celého, keďže by to bolo strašne samoúčelné a navyše by som si musel vždy pri pregenerovaní databázy počkať kým sa vytvorí. Pričom neviem, ako dlho by to vlastne bolo, keďže ten pôvodný `SELECT` som nikdy nenechal dobehnúť (vraví sa, že beží dodnes...).

Musel som to preto nejako analyzovať. So spomínanými funkciami a výpismi som sa skúsil trocha pohrať, povyhadzovať nejaké `subselecty` a sledovať, aký to má vplyv na čas. Všimol som si, že sa na viacerých miestach opakovane zisťuje, či má nejaký hráč X (teda zabitia jeho postáv) v rámci týždňa Y percentil 90 a viac. Z toho sa dá jednoducho vytvoriť zoznam a teda dobrý kandidát na zhmotnený pohľad:

week:	player:_id
1	32
1	11
1	99
2	64
	...

zhmotnený pohľad `top_players` obsahujúci hráčov s percentilom zabití 90 a viac

V ňom máme uchovaný zoznam top hráčov, teda hráčov s percentilom zabití 90 a viac. No a to VŠETKO zjednodušuje a zrýchľuje. Pri hľadaní najlepšieho dňa a hodiny už nemusíme zakaždým vypočítavať do ktorej skupiny aktuálny hráč patrí, iba nazrieme do pohľadu. To isté pri finálnom výpise a rozdelení do skupín.

„LITE“ verziu výpisu toto zlepšenie zrýchlilo pri počte postáv a zabití 10 000 z 30 sekúnd na 1 sekundu! No a „FULL“ verzia je vďaka tomu použiteľná aj pre veľké dáta!

Poznámka 1: Dúfam, že „príbehový“ štýl tejto poslednej časti nebude nikomu vadiť. Považoval som to za najlepší prístup, akým opísať postup riešenia problému.

Poznámka 2: Kadencia slov so základom „zabiť“ v tejto časti myslím môže konkurovať každej detektívke alebo denníku vraha.

Poznámka 3: Pre jednoduchosť mám v databáze iba záznamy z roku 2018. `date_part('week', date)` (funkcia na zistenie týždňa z dátumu) totiž vracia poradové číslo týždňa v rámci roku. Ak by sme teda uvažovali dáta z viacerých rokov, musel by som nejako rozlišovať napr. medzi *1. týždňom* v roku 2017 a *1. týždňom* v roku 2018... To mi prišlo ako zbytočná komplikácia. Pekné riešenie by bolo počítať týždne od vzniku hry.
Na to by som ale potreboval nejaký vlastný kalendár ☺.