VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DATABÁZOVÉ SYSTÉMY

2018/2019

DOKUMENTÁCIA K PROJEKTU

ZADANIE Z IUS Č. 67 - FOTBALOVÝ KLUB

René Bolf (xbolfr00), Radoslav Grenčík (xgrenc00) V Brně 29.4.2019

ZADANIE

Vytvořte informační systém pro profesionální fotbalový klub. Klub se skládá z několika týmů (ženský, mužský, junioři, kadeti, apod.), přičemž v každém týmu je určitý počet hráčů a několik trenérů. O hráčích i trénérech se uchovávají údaje o jejich jméně, adrese bydliště a věku. U hráčů je dále důležitá výška, váha, post a číslo dresu. Hráč může hrát za více týmů (např. za kadety i juniory zároveň), přičemž v každém týmu může hrát na jiném postu a s jiným číslem dresu. U trenéra nás dále zajímá jeho funkce v týmu (hlavní trenér, asistent, apod.). Je běžné, že jeden trenér plní různé funkce v různých týmech v rámci klubu. Klub vlastní několik sportovišť, které mohou být různého typu (hřiště, hala, posilovna) a rovněž si může sportoviště pronajímat. U sportoviště nás kromě typu a vlastnictví (stačí rozlišit, zda je ve vlastnictví klubu nebo ne) zajímá jeho název, adresa, kapacita diváků a kapacita hráčů. Každý tým organizuje tréninky, které mohou být pravidelné nebo jednorázové. Trénink se vždy koná na nějakém sportovišti a vede ho jeden z trenérů týmu, který trénink v systému vytváří. Kromě tohoto trenéra se tréninku mohou účastnit také další trenéři a určitý počet hráčů (maximálně do kapacity sportoviště). Hráči se na trénink přihlašují sami nebo je přihlašuje trenér organizující trénink. Systém rovněž umožňuje trenérovi volbu automatického přihlašování všech hráčů v týmu (v takovém případě mají hráči možnost se odhlásit). Kromě tréninků se týmy účastní zápasů proti jiným týmům. Zápas se hraje buď doma (v takovém připadě nás zajímá, na kterém sportovišti klubu se koná) nebo venku (v takovém případé postačí evidovat adresu konání). Zápas se koná v určitý den a čas a za tým se ho může účastnit maximálně počet hráčů určený kapacitou soupisky daného zápasu. Tyto hráče vybírá a přihlašuje hlavní trenér týmu. Samozřejmostí je uložení výsledku zápasu. Klub dále hráčům zajišťuje vybavení, o kterém eviduje typ, cenu a hráče (popřípadě trenéra nebo celý tým), kterému je přiděleno.

1. *Datový model (ERD) a model případů užití* – Datový model (ER diagram) zachycující strukturu dat, resp. požadavky na data v databázi, vyjádřený jako diagram tříd v notaci UML nebo jako ER diagram v tzv. Crow's Foot notaci a model případů užití vyjádřený jako diagram případů užití v notaci UML reprezentující požadavky na poskytovanou funkcionalitu aplikace používající databázi navrženého datového modelu. Datový model musí obsahovat alespoň jeden vztah generalizace/specializace (tedy nějakou entitu/třídu a nějakou její specializovanou entitu/podtřídu spojené vztahem generalizace/specializace; vč. použití správné notace vztahu generalizace/specializace v diagramu).
2. *SQL skript pro vytvoření základních objektů schématu databáze* – SQL skript vytvářející základní objekty schéma databáze, jako jsou tabulky vč. definice integritních omezení (zejména primárních a cizích klíčů), a naplňující vytvořené tabulky ukázkovými daty. Vytvořené schéma databáze musí odpovídat datovému modelu z předchozí části projektu a musí splňovat upřesňující požadavky zadání.

3. *SQL skript s několika dotazy SELECT* – SQL skript, který nejprve vytvoří základní objekty schéma databáze a naplní tabulky ukázkovými daty (stejně jako skript v bodě 2) a poté provede několik dotazů SELECT dle upřesňujících požadavků zadání.
4. *SQL skript pro vytvoření pokročilých objektů schématu databáze* – SQL skript, který nejprve vytvoří základní objekty schéma databáze a naplní tabulky ukázkovými daty (stejně jako skript v bodě 2), a poté zadefinuje či vytvoří pokročilá omezení či objekty databáze dle upřesňujících požadavků zadání. Dále skript bude obsahovat ukázkové příkazy manipulace dat a dotazy demonstrující použití výše zmiňovaných omezení a objektů tohoto skriptu (např. pro demonstraci použití indexů zavolá nejprve skript EXPLAIN PLAN na dotaz bez indexu, poté vytvoří index, a nakonec zavolá EXPLAIN PLAN na dotaz s indexem; pro demostranci databázového triggeru se provede manipulace s daty, která vyvolá daný trigger; atp.).
5. *Dokumentace popisující finální schéma databáze* – Dokumentace popisující řešení ze skriptu v bodě 4 vč. jejich zdůvodnění (např. popisuje výstup příkazu EXPLAIN PLAN bez indexu, důvod vytvoření zvoleného indexu, a výstup EXPLAIN PLAN s indexem, atd.).

OBSAH

ÚVOD 5

1 MODEL PRÍPADOV POUŽITIA 6

2 ER DIAGRAM 7

3 FINÁLNA SCHÉMA DATABÁZY 8

4 RIEŠENIE VZŤAHU GENERALIZÁCIA/ŠPECIALIZÁCIA 9

5 IMPLEMENTÁCIA PROJEKTU Č. 4 10

5.1 TRIGGERY 10

5.2 PROCEDÚRY 10

5.3 EXPLAIN PLAN A VYTVORENIE INDEXU 11

5.4 PRÍSTUPOVÉ PRÁVA 13

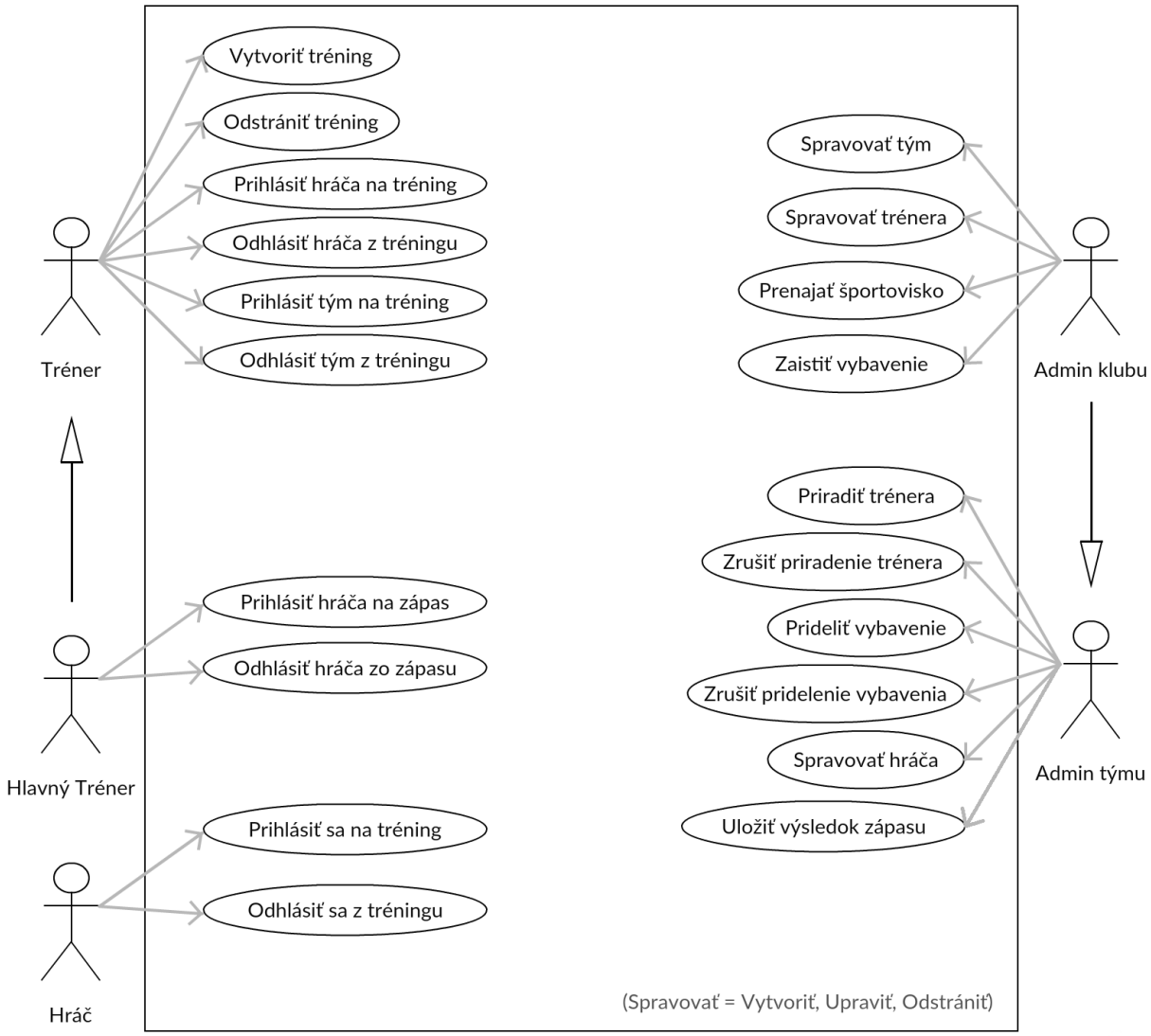
5.5 MATERIALIZOVANÝ POHĽAD 13

ZÁVER 14

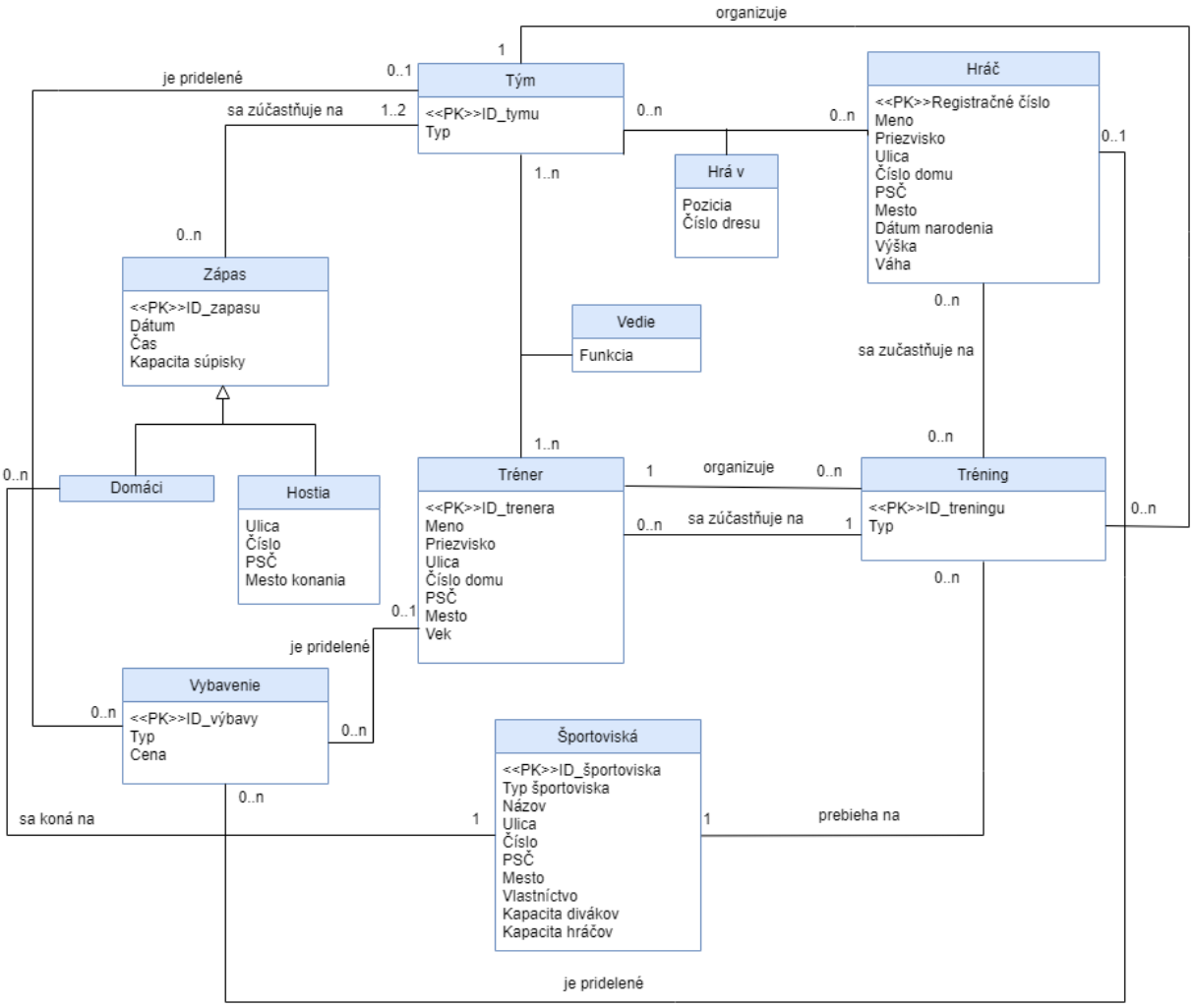
Úvod

Našim cieľom bolo vytvoriť funkčný databázový systém pre futbalový klub, naučiť sa pracovať s nástrojom ORACLE SQL Developer, upevniť a ozrejmiť si látku preberanú na predmete Databázové systémy a získať množstvo skúsenosti pri vypracovávaní projektu.

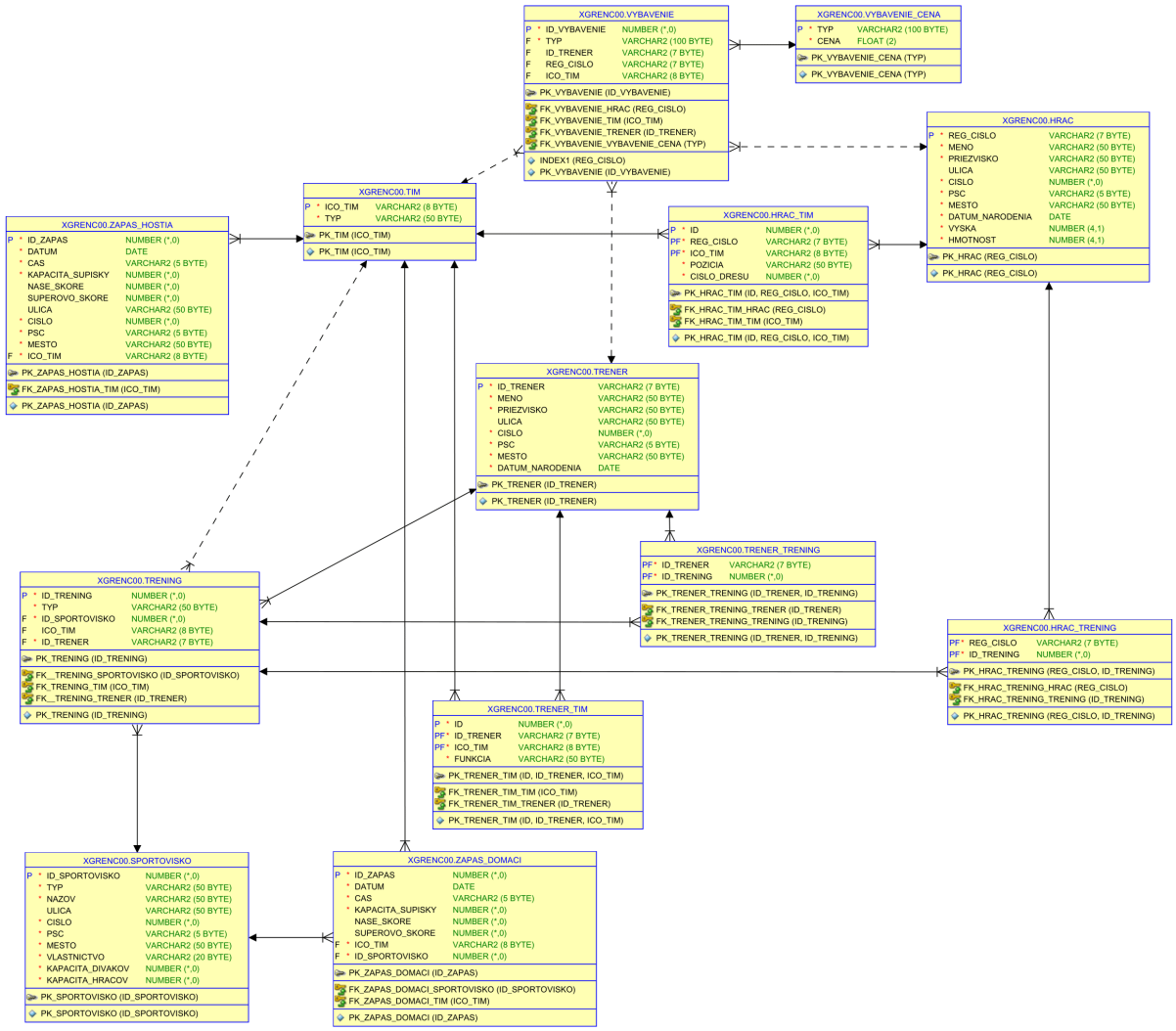
1 MODEL PRÍPADOV POUŽITIA



2 ER DIAGRAM


organizuje
1
je pridelené
0..1
Tým
<<PK>>ID_tymu
Typ
sa zúčastňuje na
1..2
0..n
0..n
Hráč
<<PK>>Registračné číslo
Meno
Priezvisko
Ulica
Číslo domu
PSČ
Mesto
Dátum narodenia
Výška
Váha
0..1
1..n
Hrá v
Pozicia
Číslo dresu
0..n
Zápas
<<PK>>ID_zapasu
Dátum
Čas
Kapacita súpisky
Vedie
Funkcia
sa zučastňuje na
0..n
1..n
0..n
0..n
Domáci
Hostia
Ulica
Číslo
PSČ
Mesto konania
Tréner
<<PK>>ID_trenera
Meno
Priezvisko
Ulica
Číslo domu
PSČ
Mesto
Vek
1
organizuje
0..n
Tréning
<<PK>>ID_treningu
Typ
0..n
sa zúčastňuje na
1
0..n
0..n
0..1
je pridelené
Vybavenie
<<PK>>ID_výbavy
Typ
Cena
0..n
0..n
0..n
Športoviská
<<PK>>ID_športoviska
Typ športoviska
Názov
Ulica
Číslo
PSČ
Mesto
Vlastníctvo
Kapacita divákov
Kapacita hráčov
sa koná na
1
1
prebieha na
je pridelené

3 FINÁLNA SCHÉMA DATABÁZY



4 RIEŠENIE VZŤAHU GENERALIZÁCIA/ŠPECIALIZÁCIA

Vzťah generalizácia/špecializácia sme riešili tak, že sme si vytvorili separátnu tabuľku pre každý podtyp, kde každá táto tabuľka obsahuje všetky stĺpce z nadtypu. Toto riešenie sme zvolili preto, lebo máme taký vzťah generalizácia/špecializácia, kde jeden podtyp neobsahuje žiadne svoje stĺpce (obsahuje len stĺpce z nadtypu). Kvôli tomu sa nám nehodí riešenie s jednou tabuľkou pre nadtyp a separátnymi tabuľkami pre každý podtyp. Takisto sa nám nehodí ani vzťah typu jedna tabuľka pre nadtyp a jedna tabuľka pre podtypy a ani vzťah iba s jednou tabuľkou pretože potrebujeme modelovať vzťah jedného z našich dvoch podtypov s inou tabuľkou (viz. 2 - ER diagram - vzťah medzi "Zápas domáci" a "Športovisko"). Kvôli týmto dôvodom sme zvolili práve riešenie so separátnymi tabuľkami pre každý podtyp. Toto riešenie taktiež vyhovuje kvôli tomu, že náš vzťah typu generalizácia/špecializácia je totálny a disjunktný a zvolené riešenie sa hodí práve pre takýto typ vzťahu. Takisto nebudeme ani vykonávať veľa operácií so spoločnými dátami oboch podtypov, kde by zvolené riešenie nebolo vyhovujúce.

5 IMPLEMENTÁCIA PROJEKTU Č. 4

5.1 TRIGGERY

Ako prvé sme implementovali databázové triggery. Naimplementovali sme ich až osem. Sedem z nich slúži na auto inkrementáciu primárneho kľúča v tabuľkách TRENING, TRENER_TIM, HRAC_TIM, VYBAVENIE, ZAPAS_DOMACI, ZAPAS_HOSTIA a SPORTOVISKO. Pre účel auto inkrementácie sme si vytvorili pre každý trigger jednu sekvenciu, ktorá si pamätá číslo posledného pridaného primárneho kľúča. Posledný z triggerov slúži na automatické pridelenie vybavenia "brankárske rukavice" hráčovi, ktorý je vložený do tabuľky HRAC_TIM a v stĺpci "pozicia" je hodnota "brankar". Tento trigger takisto automaticky vytvorí vybavenie "brankárske rukavice" v tabuľke VYBAVENIE_CENA ak takéto vybavenie ešte neexistuje. Činnosť triggerov je predvedená v skripte.

5.2 PROCEDÚRY

Prvá procedúra SUM_VYBAVENIE vypočíta sumu vybavenia, ktoré je pridelené konkrétnemu členovi futbalového klubu (členom môže byť hráč, tréner alebo tím). Procedúra má dva parametre a to "vlastnik" a "ID_vlastnik". Parameter "vlastnik" označuje stĺpec v tabuľke VYBAVENIE a je case-insensitive. Pokiaľ užívateľ zadá niečo iné ako "reg_cislo", "ID_trener" alebo "ICO_tim" procedúra skončí nami deklarovanou výnimkou "bad_column". V procedúru sú vytvorené až tri kurzory, ale používa sa vždy len jeden. Kurzor sa vyberie na základe parametra "vlastnik". Parameter "ID_vlastnik" je identifikačné číslo konkrétneho člena klubu. Pokiaľ je zadané nesprávne a konkrétny kurzor je prázdny, procedúra skončí nami deklarovanou výnimkou "not_found".

Druhá procedúra SUM_SCORE spočíta počet strelených a inkasovaných gólov celého futbalového klubu za určité obdobie. Procedúra má 2 parametre "date1" a "date2", ktoré majú dátový typ ZAPAS_DOMACI.DATUM%TYPE odkazujúci sa na tabuľku ZAPAS_DOMACI. Pokiaľ je parameter "date1" väčší ako parameter "date2" procedúra skončí nami deklarovanou výnimkou "bad_params". Pokiaľ používateľ zadá parametre "date1" a "date2" také, pre ktoré neexistuje v databáze záznam, procedúra o tomto vypíše informáciu. V procedúre sú vytvorené dva kurzory, pretože procedúra bude spočítavať góly v dvoch tabuľkách a to ZAPAS_DOMACI a ZAPAS_HOSTIA. Ďalej je vhodné spomenúť ošetrenie prípadu, keď je v stĺpci "NASE_SKORE" alebo "SUPEROVO_SKORE" v niektorej zo spomenutých tabuliek hodnota "null". Tento prípad môže nastať ak sa zápas ešte neodohral, ale už bol zapísaný do databázy. Vždy by sa v takomto prípade mala vyskytovalť hodnota "null" v oboch stĺpcoch "NASE_SKORE" aj "SUPEROVO_SKORE". Tento prípad sme vyriešili zanoreným cyklom a vhodnou ukončujúcou podmienkou (viz skript).

5.3 EXPLAIN PLAN A VYTVORENIE INDEXU

Pre demonštráciu použitia EXPLAIN PLAN sme zvolili nasledovný SELECT so spojením 2 tabuliek, klauzulou GROUP BY a agregačnou funkciou:

```
select h.reg_cislo, h.meno, h.priezvisko, count(distinct v.ID_vybavenie) as pocet_vybaveni
from vybavenie v join hrac h on h.reg_cislo = v.reg_cislo
group by h.reg_cislo, h.meno, h.priezvisko
having count(distinct v.ID_VYBAVENIE) > 1;
```

Výstup EXPLAIN PLAN vyzerá nasledovne:

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
Plan hash value: 226534526

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |           |     8 |   512 |     4  (25)| 00:00:01 |
|*  1 |  FILTER                       |           |       |       |            |          |
|   2 |   HASH GROUP BY               |           |     8 |   512 |     4  (25)| 00:00:01 |
|   3 |    NESTED LOOPS               |           |     8 |   512 |     3   (0)| 00:00:01 |
|   4 |     NESTED LOOPS              |           |    17 |   512 |     3   (0)| 00:00:01 |
|   5 |      TABLE ACCESS FULL        | VYBAVENIE |    17 |    85 |     3   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
|*  6 |       INDEX UNIQUE SCAN       | PK_HRAC   |     1 |       |     0   (0)| 00:00:01 |
|   7 |     TABLE ACCESS BY INDEX ROWID| HRAC      |     1 |    59 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(COUNT(*)>1)
   6 - access("H"."REG_CISLO"="V"."REG_CISLO")

Note

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------------
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

Postup vykonávania nášho príkazu SELECT je nasledovný:

1) Bod č. 5 TABLE ACCESS FULL VYBAVENIE alebo aj TABLE FULL SCAN číta celú tabuľku VYBAVENIE - všetky riadky a stĺpce tak ako sú uložené na disku. Je to jedna z najdrahších operácií a preto sa ju budeme snažiť eliminovať.
2) Bod č. 6 INDEX UNIQUE SCAN PK_HRAC vykoná B-tree prechod indexom PK_HRAC. Databáza použije túto operáciu ak unikátne obmedzenie zaistí, že výsledkom vyhľadávania bude iba jedna zhoda.
3) Bod č. 4 NESTED LOOPS spojí dve tabuľky načítaním výsledku z prvej tabuľky a jeho spojením s každým riadkom druhej tabuľky. Toto sa vykoná pre každý riadok z prvej tabuľky.

4) Bod č. 7 TABLE ACCES BY INDEX ROWID HRAC získa riadok z tabuľky HRAC zapomoci ROWID získaného z predchádzajúceho prehľadávania indexu.
5) Bod č. 3 NESTED LOOPS spojí dve tabuľky načítaním výsledku z prvej tabuľky a jeho spojením s každým riadkom druhej tabuľky. Toto sa vykoná pre každý riadok z prvej tabuľky.
6) Bod č. 2 HASH GROUP BY zoskupí výsledok zapomoci hashovacej tabuľky. Táto operácia vyžaduje veľké množstvo pamäti. Výsledok nie je zoradený.
7) Bod č. 1 FILTER filtruje výsledok.

Ako môžme vidieť vykonávanie nášho SELECTu nie je optimálne pretože tabuľka VYBAVENIE sa skenuje celá po riadkoch a toto je veľmi drahá operácia. Cena (COST) nášho príkazu SELECT je vysoká (až 4). Preto sme v tabuľke VYBAVENIE vytvorili pre stĺpec REG_CISLO, ktorý sa používa pri spájaní tabuliek index - INDEX1 (viz. skript). Tento cenu vykonania nášho dotazu znížil o polovicu (zo 4 na 2). Po zavedení indexu vyzeral výstup EXPLAIN PLAN nasledovne:

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
Plan hash value: 2514019871

-------------------------------------------------------------------------------------------
| Id  | Operation                     | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |         |     8 |   512 |     2  (50)| 00:00:01 |
|*  1 |  FILTER                       |         |       |       |            |          |
|   2 |   HASH GROUP BY               |         |     8 |   512 |     2  (50)| 00:00:01 |
|   3 |    NESTED LOOPS               |         |     8 |   512 |     1   (0)| 00:00:01 |
|   4 |     NESTED LOOPS              |         |    17 |   512 |     1   (0)| 00:00:01 |
|   5 |      INDEX FULL SCAN          | INDEX1  |    17 |    85 |     1   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
|*  6 |      INDEX UNIQUE SCAN        | PK_HRAC |     1 |       |     0   (0)| 00:00:01 |
|   7 |     TABLE ACCESS BY INDEX ROWID| HRAC    |     1 |    59 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(COUNT(*)>1)
   6 - access("H"."REG_CISLO"="V"."REG_CISLO")

Note

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------------------
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

Ako môžme vidieť tak v bode číslo 5 sa už neskenuje celá tabuľka, ale prehľadáva sa len nami vytvorený index (INDEX1) a to metódou INDEX FULL SCAN - prečíta celý index - všetky riadky po poradí (nie tak ako sú uložené na disku, ale poradie v indexe).

5.4 PRÍSTUPOVÉ PRÁVA

Prístupové práva pre druhého člena týmu sme vytvorili tak ako by boli v databáze vytvorené prístupové práva pre trénera. Vytvorili sme ich podľa prípadov použitia z modelu prípadov použitia (viz. 1 - model prípadov použitia). Tréner má všetky prístupové práva k tabuľkám TRENING, HRAC_TRENING a TRENER_TRENING, pretože môže vytvárať tréningy prihlasovať a odhlasovať hráčov a takisto seba na konkrétny tréning. Ďalej má práva SELECT, READ a COMMIT REFRESH ON k tabuľkám TIM, HRAC, TRENER, HRAC_TIM, TRENER_TIM, ZAPAS_DOMACI, ZAPAS_HOSTIA a SPORTOVISKO. Tréner má právo si pozerať obsah týchto tabuliek a takisto si vytvárať dotazy na tieto tabuľky. Právo COMMIT REFRESH ON je nevyhnutné na to aby si mohol vytvoriť materializovaný pohľad. Tréner nemá právo pristupovať k tabuľke VYBAVENIE ani spustiť procedúru SUM_VYBAVENIE, pretože o pridelení vybavenia nerozhoduje. Tréner má ešte právo EXECUTE na procedúre SUM_SCORE.

5.5 MATERIALIZOVANÝ POHĽAD

Materializovaný pohľad vytvára druhý člen týmu, ktorému sme pred tým pridelili spomínané prístupové práva. Bez týchto práv by tento pohľad nevedel vytvoriť. Pri vytváraní materializovaného pohľadu sme nastavili možnosť CACHE, ktorá optimalizuje čítanie z tohoto pohľadu a takisto možnosť BUILD IMMEDIATE, ktorá naplní materializovaný pohľad hneď po vytvorení. Materializovaný pohľad je vytvorený tak, aby sa aktualizoval príkazom COMMIT. Následne sme demonštrovali prácu s týmto pohľadom: vyvorili sme si príkaz SELECT na zobrazenie materializovaného pohľadu, potom sme do jednej z tabuliek, z ktorých je pohľad vytvorený vložili nový riadok a opäť spustili SELECT. Vidíme, že pohľad zostal nezmenený aj keď tabuľka bola zmenená. Aktualizovanie pohľadu docielime príkazom COMMIT a následne po spustení SELECTu vidíme, že pohľad bol aktualizovaný.

ZÁVER

Projekt sme riešili vo dvojici v prostredí ORACLE SQL Developer, vďaka čomu sme sa naučili s týmto nástrojom pracovať. Takisto nám projekt ozrejmil preberanú látku z predmetu Databázové systémy a získali sme veľa cenných skúseností, ktoré budeme môcť aplikovať v praxi.