ŽILINSKÁ UNIVERZITA V ŽILINE
Fakulta riadenia a informatiky

SEMESTRÁLNA PRÁCA

Pokročilé databázové systémy

Nemocničný informačný systém

Vypracovali:

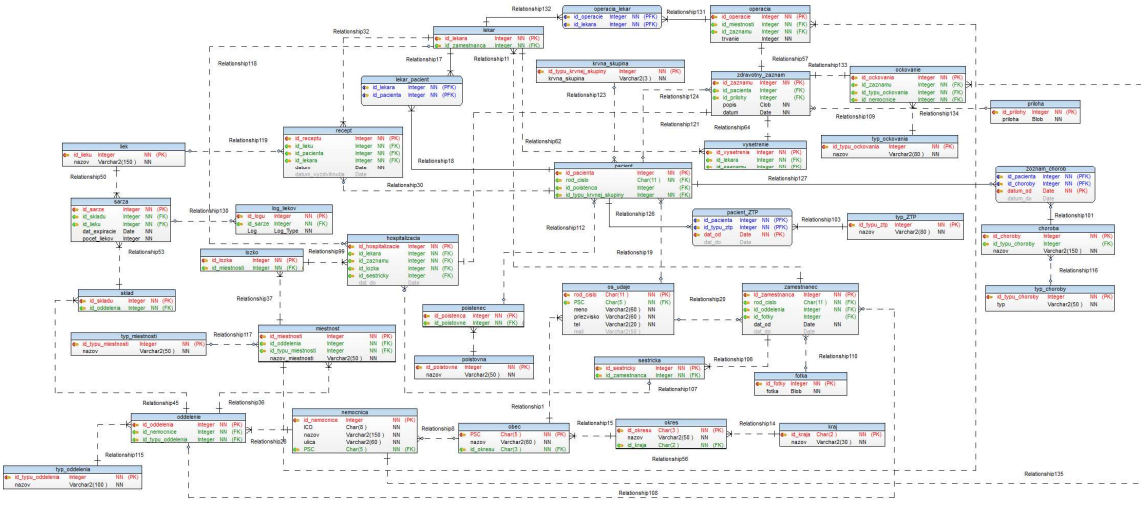
Marek Šútora	(5ZIB11)	**Akademický rok:** 2022/2023
Samuel Slivovský	(5ZIB11)	**Študijný program:**
Iveta Šinálová	(5ZIB11)	Biomedicínska informatika
Tomáš Štulrajter	(5ZIB11)	
Tomáš Labát	(5ZIB11)	

Obsah

Popis semestrálnej práce

Témou našej semestrálnej práce je nemocničný informačný systém spolu s grafickým rozhraním pre zamestnancov. Grafické rozhranie sme vytvorili pomocou frontendového javascript frameworku React a backendovú časť sme vytvorili pomocou frameworku Node.js. Súbory týkajúce sa tejto semestralnej práce sa v našom projekte nachádzajú v priečinku SemestrálnaPráca_PDS, kde sú rozdelené do priečinkov podľa toho čoho sa týkajú. Na uloženie dát pre aplikáciu sme využili školský databázový server obelix, ktorý beží na databázovom systéme Oracle. Na testovanie boli v práce využité dáta, ktoré sa nachádzajú v priečinku Testovacie_data.

Dátový model



Obrázok 1 Dátový model

Dátový model bol vytvorený v programe Toad Data Modeller a nachádza sa v priečinku Toad_Model.

os_udaje

Tabuľka osobných údajov pacientov a zamestnancov nemocnice.

zamestnanec

Tabuľka všetkých zamestnancov nemocnice, spolu s dátumom nástupu a odchodu. Každý zamestnanec má tiež pridelené oddelenie, na ktorom pracuje.

fotka

Tabuľka, ktorá v sebe uchováva informáciu o fotke zamestnanca. Fotka je vo formáte BLOB.

sestricka

Tabuľka, ktorá reprezentuje sestričku.

pacient

Tabuľka reprezentujúca konkrétneho pacienta.

pacient_ZTP
Tabuľka ktorá určuje typ ZŤP konkrétneho pacienta a od kedy do kedy ho má.

typ_ZTP
Uchováva v sebe všetky typy ZŤP.

zoznam_chorob
Tabuľka, ktorá určuje zoznam chorôb, konkrétneho pacienta. Ku každej chorobe je aj dátum od kedy do kedy danú chorobu mal.

choroba
Pomocou tejto tabuľky vieme určiť, ku ktorému typu choroby patrí konkrétny názov choroby.

typ_choroby
Obsahuje všetky typy chorôb, ktoré sa následne priraďujú k jednotlivým chorobám.

krvna_skupina
Tabuľka obsahujúca všetky typy krvných skupín, ktoré sú následne priradené pacientovi.

zdravotny_zaznam
Tabuľka obsahuje všetky zdravotné záznamy pacienta spolu s popisom a dátumom záznamu. Zastrešuje všeobecnú informáciu o očkovaniach, vyšetreniach, operáciách a hospitalizáciách pacientov. Každý záznam môže mať aj prílohu.

priloha
Obsahuje prílohu ku konkrétnemu záznamu.

ockovanie
Tabuľka ktorá v sebe uchováva informácie o všetkých očkovaniach, kde je uchované aj v ktorej nemocnici sa očkovanie robilo, aký má typ a o ktorý záznam sa jedná.

typ_ockovania
Tabuľka, ktorá obsahuje názvy typov očkovaní, ktoré sú následne priraďované k jednotlivým očkovaniam.

vysetrenie
Tabuľka, ktorá predstavuje vyšetrenie pacienta lekárom.

operacia
Tabuľka, ktorá predstavuje operáciu pacienta. Obsahuje informáciu o zázname pacienta a taktiež o miestnosti a trvaní danej operácie.

operacia_lekar
Tabuľka, ktorá zabezpečuje vzťah M:N medzi operáciou a lekárom.

lekar
Tabuľka, v ktorej sa nachádzajú všetci zamestnanci, ktorí sú lekármi.

lekar_pacient
Tabuľka, v ktorej sú uložení ku každému lekárovi jeho pacienti.

recept
Tabuľka, ktorá predstavuje recept predpísaný lekárom pacientovi. Obsahuje taktiež informácie o lieku, dátume jeho vydania a následného vyzdvihnutia.

liek
Tabuľka, ktorá predstavuje názvy všetkých liekov používaných v databáze.

sarza
Tabuľka, ktorá predstavuje jednu šaržu nachádzajúcu sa v sklade. Obsahuje informáciu o lieku, jeho počte v danej šarži a taktiež o dátume exspirácia.

log_liekov
Tabuľka, v ktorej sa zaznamenávajú informácie o zamestnancoch, ktorí pridávali alebo odoberali lieky z danej šarže. Informácia o zamestnancovi sa uchováva v atribúte Log, ktorého typ je objekt Log_Type.

Log_Type
Objekt, ktorý je tvorený atribútmi – id_zamestnanca, datum a zmena_poctu. id_zamestnanca predstavuje id zamestnanca, ktorý manipuloval s danou šaržou, datum je dátum kedy k manipulácii došlo a zmena_poctu je zmena v počte liekov v danej šarži. Táto zmena môže byť buď kladná (v prípade pridania lieku) alebo záporná (v prípade odobrania lieku). Obsahuje taktiež 2 metódy – na výpis a taktiež na triedenie objektu. Triedenie sa vykonáva na základe atribútu zmena_poctu. Kód k definícií tohto objektového typu spolu s jeho funkciami sa nachádza v priečinku Objekt.

sklad
Tabuľka, ktorá zabezpečuje priradenie skladu k danému oddeleniu.

miestnost
Tabuľka, ktorá predstavuje miestnosť v oddelení nemocnice. Obsahuje taktiež aj názov miestnosti a id jej typu.

typ_miestnosti
Tabuľka, ktorá obsahuje názov typu miestnosti, ktoré sú následne prideľované k miestnostiam.

oddelenie
Tabuľka, ktorá predstavuje oddelenie v danej nemocnici.

typ_oddelenia
Tabuľka, ktorá obsahuje názvy jednotlivých typov oddelení, ktoré sú následne priraďované k oddeleniu.

lozko
Tabuľka prostredníctvom ktorej sú k jednotlivým miestnostiam priraďované lôžka.

hospitalizacia
Tabuľka, ktorá predstavuje hospitalizáciu pacienta. Obsahuje informáciu o zázname pacienta, lôžku, na ktorom je pacient hospitalizovaný, o sestričke, ktorá sa hospitalizovaného pacienta stará a o dátume kedy bol prepustený.

nemocnica
Tabuľka, ktorá ukladá informácie o nemocnici. Konkrétne jej IČO, názov, ulicu a PSČ.

obec
Tabuľka, v ktorej sú k PSČ priradené názvy obcí a id okresu, do ktorého obec patrí.

okres
Tabuľka, v ktorej sa nachádzajú názvy okresov a id krajov, do ktorých dané okresy patria.

kraj
Tabuľka, ktorá obsahuje názvy krajov.

poistovna
Tabuľka, v ktorej sú uložené názvy poisťovní.

poistenec
Tabuľka, pomocou ktorej je priradený poistenec ku poisťovni.

PL/SQL
V našej práci sme v rámci využitia PL/SQL použili procedúry a funkcie, v ktorých využívame kolekcie a recordy. Všetky tieto súbory sa nachádzajú v priečinku PL_SQL.

Procedúry
Procedúry využívame v našej práci na generovanie náhodných dát do niektorých tabuliek. Používame ich aj na vkladanie dát do tabuľky sarza a taktiež pri vkladaní dát typu BLOB do tabuliek fotka a príloha.

Procedúry na generovanie operácií
Procedury, ktoré slúžia na generovanie náhodných dát do tabuliek opeacia, operacia_lekar a zdravotny_zaznam. Nachádzajú sa v súbore operacia_insert.sql

generovanie_operacii_p
Procedúra, ktorá slúži na generovanie náhodných dát do tabuľky **operacia**.

generovanie_zdr_zazn_p
Procedúra, ktorá vytvorí pre generovanú operáciu zdravotný záznam.

generovanie_operacia_lekar_p
Procedúra ktorá slúži na generovanie dát do tabuľky **operacia_lekar** tak, aby bol priradený lekár zo správneho oddelenia.

generovanie_lekar_pacient_operacie_p
Procedúra ktorá zabezpečí vytvorenie vzťahu medzi lekárom a pacientom pre tabuľku **operacia**. Pomocou for cyklu voláme pre každý riadok funkciu **existuje_lekar_pacient_f**. Táto funkcia je bližšie popísaná v sekcií Funkcie.

generovanie_vysetreni_p
Procedúra ktorá slúži na generovanie dát do tabuľky **vysetrenie**. Nachádza sa v súbore vysetrenie_insert.sql.

insert_hospitalizacie
Procedúra ktorá slúži na generovanie dát do tabuľky **hospitalizacia**. Nachádza sa v súbore hospitalizacia_insert.sql.

recept_insert
Procedúra ktorá slúži na generovanie dát do tabuľky **recept.** Nachádza sa v súbore recept_insert.sql.

sarza_insert_proc
Procedúra ktorá slúži na vloženie dát do logovacej tabuľky **log_liekov** po vložení dát do tabuľky **sarza**. Nachádza sa v súbore sarza_insert_update.sql.

sarza_update_proc
Procedúra ktorá slúži na aktualizáciu tabuľky **sarza**, taktiež vloží záznam do logovacej tabuľky **log_liekov**. Nachádza sa v súbore sarza_insert_update.sql.

uprava_sarze
Procedúra na vytvorenie nových náhodných záznamov do tabuľky **log_liekov** na základe zmeny počtu liekov pre danú šaržu. Nachádza sa v súbore sarza_insert_update.sql.

insert_BLOB_into_table
Procedúra, slúžiaca na vkladanie BLOBov do tabuliek fotka a príloha. Prvým krokom na načítanie BLOBu do databázy je nahratie súboru na server cezar.fri.uniza.sk do zložky /media/Obelix.Bloby_student/labat_sp (čo predstavuje login našej semestrálne práce). Takýto súbor sa automaticky mapuje na fyzickú cestu C:\Bloby_student\labat_sp\ . Ďalším krokom je vytvorenie directory, ktoré predstavuje fyzickú cestu k súboru. Následne si v rámci procedúry vytvoríme BFILE, ktorý pomocou konštruktora namapujeme na vytvorený directory a názov súboru, ktorý sme vložili na server cezar. Ako ďalšie vytvoríme smerník na BLOB, pomocou konštruktora EMPTY_BLOB(), hodnota je však stále prázdna. Súbor otvoríme, získame jeho dĺžku a načítame jeho hodnotu do premennej, ktorú následne vložíme do tabuľky.

Funkcie
Funkcie sme v našej práci využili na generovanie XML dokumentov pre požadovanú nemocnicu, ktorá sa odovzdáva prostredníctvom parametra. Jednotlivé tieto funkcie je možné nájsť taktiež v priečinku PL_SQL/ XML_Funkcie. Taktiež sme si vytvorili pomocnú funkcie, ktoré využívame pri generovaní dát prostredníctvom procedúr.

existuje_lekar_pacient_f
Funkcia ktorá slúži na kontrolu či existuje vzťah medzi lekárom a pacientom.

getHospitalizacieNemocnice_f
Funkcia ktorá slúži na vytvorenie XML dokumentu pre hospitalizácie danej nemocnice.

getOckovaniaNemocnice_f
Funkcia ktorá slúži na vytvorenie XML dokumentu pre očkovania danej nemocnice.

getOperacieNemocnice_f

Funkcia ktorá slúži na vytvorenie XML dokumentu pre operácie danej nemocnice.

getVysetreniaNemocnice_f

Funkcia ktorá slúži na vytvorenie XML dokumentu pre vyšetrenia danej nemocnice.

Výstupy

Všetky výstupy sa nachádzajú v súbore vystupy.sql, ktorý sa nachádza v priečinku Vystupy. Tieto výstupy sú tam usporiadané postupne ako sú očíslované nižšie v texte.

Štatistické výstupy

Cieľom týchto výstupov bolo vytvoriť jednoduché zobrazenie dôležitých údajov pre jednotlivé oddelenia. Výsledok sme docielili pomocou použitia grafov a tabuliek.

1. Počet pacientov oddelenia:

Vypísanie počtu všetkých pacientov ktorí sa objavili na vybranom oddelení. Informuje zamestnancov oddelenia o tom, koľko pacientov je na ich oddelení. Výstup je zobrazený ako číslo.

2. Počet zamestnancov oddelenia:

Vypísanie počtu zamestnancov oddelenia ktorí pracujú alebo pracovali v danom roku. Informuje zamestnancov koľko kolegov majú/mali v danom roku. Vedia si teda pozrieť či im kolegovia pribudli, prípadne odbudli. Výstup je zobrazený ako číslo.

3. Počet vykonaných hospitalizácií:

Vypísanie počtu vykonaných hospitalizácií v danom roku pre vybrané oddelenie. Informuje zamestnancov o tom, koľko rôznych hospitalizácií bolo v danom roku vykonaných na ich oddelení. Výstup je zobrazený ako číslo.

4. Počet vykonaných operácií:

Vypísanie počtu vykonaných operácií v danom roku pre vybrané oddelenie. Informuje zamestnancov o tom, koľko rôznych **operácií** bolo v danom roku vykonaných na ich oddelení. Výstup je zobrazený ako číslo.

5. Počet vykonaných vyšetrení:

Vypísanie počtu vykonaných vyšetrení v danom roku pre vybrané oddelenie. Informuje zamestnancov o tom, koľko rôznych **vyšetrení** bolo v danom roku vykonaných na ich oddelení. Výstup je zobrazený ako číslo.

6. Podiel mužských a ženských pacientov:

Vypísanie percentuálneho podielu medzi ženami a mužmi u pacientov. Informuje zamestnancov o tom, koľko pacientov predstavuje ženskú a koľko mužskú časť. Vedia si takto zistiť majoritné pohlavie. Výstup je zobrazený ako koláčový graf.

7. Rozdelenie pacientov podľa veku:

Vypísanie počtu pacientov vzhľadom na vek. Informuje zamestnancov o počte pacientov pre danú vekovú kategóriu. Zamestnanci si takto vedia zistiť s akými vekovými kategóriami sa stretávali najčastejšie. Výstup je zobrazený ako stĺpcový graf.

8. Výplaty v mesiacoch:

Vypísanie súčtu výplat zamestnancov v mesiacoch pre vybrané oddelenie a rok. Informuje zamestnancov o súčte výplat vo vybranom roku rozdelených po mesiacoch. Zamestnanci sa takto vedia informovať ako veľmi sa zmenili výplaty v minulých rokoch. Výstup je zobrazený ako stĺpcový graf.

9. Najlepšie zarábajúci zamestnanci:

Vypísanie piatich najlepšie zarábajúcich zamestnancov. Informuje zamestnancov o ich najlepšie zarábajúcich kolegoch v danom roku. Výstup je zobrazený ako tabuľka.

10.Počet typov krvnej skupiny u pacientov:

Vypísanie počtov typov krvných skupín u pacientov na vybranom oddelení. Zamestnancov takto informuje aké krvné skupiny sú pre ich oddelenie najčastejšie. Výstup je zobrazený ako koláčový graf

PORADIE	ZAMESTNANEC	Zárobok
1	Jozef Gubbins	14290.95
2	Beniamino Follett	14283.83
3	Arlene Pantridge	14219.9
4	Randie Langthorne	14046.34
5	Sande Reedie	13991.9

Obrázok 2 Zobrazenie štatistických výstupov

Ostatné výstupy

Tieto výstupy sú zobrazované pomocou tabuľky. Pri niektorých týchto výstupoch je možné najprv zadať určité parametre ako napríklad počet alebo percentuálny podiel prvkov.

11.Pacienti s najviac chorobami

Získa top X pacientov s najviac chorobami, kde X je zadávaný parameter.

12.Pacienti s najviac operáciami

Získa top X percent pacientov s najviac operáciami.

13.Pacienti s najviac hospitalizáciami

Získa top X percent pacientov s najviac hospitalizáciami.

14.Najlepšie platený zamestnanci

Získa top X najlepšie platených zamestnancov pre každé oddelenie.

15.Počty očkovaní podľa typu pre pacientov

Pre všetkých pacientov sa vypíše koľkokrát podstúpili očkovanie daného typu.

16. Všetci zamestnanci všetkých oddelení

Vypíše všetkých zamestnancov pre každé oddelenie.

17. Najčastejšie choroby roka

Získa top X najčastejších chorôb vybraného roka.

18. Neobsadené lôžka na najbližší týždeň na oddelení

Vypíše všetky neobsadené lôžka na najbližší týždeň pre vybrané oddelenie.

19. Najviac predpisované lieky roka

Vypíše 10% najviac predpisovaných liekov vybraného roka.

20. Lieky s počtom menej ako

Lieky k oddeleniam v nemocnici, ktorých je v sklade menej ako 5.

21. Menovci medzi pacientami a lekármi

Vypíše krstné meno pacienta a lekára ak sú rovnaké a navyše ich priezviská.

22. Operácie podľa počtu lekárov a trvania

Vypíše informácie o operáciách, ktoré spĺňajú podmienku, že počet lekárov a trvanie je väčšie ako zadané parametre.

23. Kraje podľa počtu operovaných

Vráti zoznam krajov usporiadaných podľa počtu pacientov, ktorí v nich boli operovaní.

XML Report

Pre každú nemocnicu sme vytvorili možnosť vygenerovania XML dokumentov pre výpis všetkých svojich operácií, hospitalizácií, očkovaní alebo vyšetrení. K jednotlivým záznamom sa okrem špecifických informácií pre daný typ záznamu vypíšu aj dodatočné informácie akými sú dátum a popis. Kódy k funkciám na generovanie týchto XML dokumentov sa nachádzajú v priečinku PL_SQL/XML_Funkcie. Tieto XML dokumenty v našej práci ale konvertujeme pomocou funkcie getClobVal() na dátový typ CLOB aby bolo možné ľahko získať tento dokument ako textový reťazec a zobraziť ho používateľovi v jeho webovom prehliadači.

Analýza výkonnosti selectu vzhľadom na indexy

V práci sme výkonnostne analyzovali nasledujúce príkazy:

1.)

```
select sum(vek) from
(select extract(year from sysdate) - extract(year from datum_narodenia) as vek
from (select to_date(substr(rod_cislo, 5, 2) || '.' || (case when substr(rod_cislo, 3, 1) = '5' then '0' when substr(rod_cislo, 3, 1) = '6' then '1' end)
|| substr(rod_cislo, 4, 1) ||  '.19' || substr(rod_cislo, 1, 2), 'DD.MM.YYYY') as datum_narodenia
from os_udaje join zamestnanec using(rod_cislo)
join lekar using(id_zamestnanca ))) /
(select count(distinct id_zamestnanca) from lekar) as priemerny_vek
from dual;
```

Tento príkaz select vráti priemerný vek lekára, ktorý vypočítava pomocou vnorených selectov ako súčet vekov všetkých lekárov podelený ich počtom. Na analýzu sme využili funkciu Autotrace programu sql developer, kde sme sledovali parameter „cost", ktorý predstavuje metriku nákladnosti vykonaného príkazu.

V prípade, že nad relevantnými stĺpcami nie je vytvorený žiadny index, odhaduje optimalizátor cost na hodnotu 14. Tabuľky sú prechádzané metódou TABLE ACCESS – FULL. Najskôr sa vykonal spodný select, kde sa kvôli použitiu klauzuly distinct vo funkcii count() vykonalo sortovanie tabuľky lekar podľa id_zamestnanca (metóda SORT GROUP BY). Na výsledok sa použil ďalší sort, tentokrát SORT AGGREGATE, ktorý sa často volá v prípade použitia agregačnej funkcie, ktorá vracia len 1 riadok, v našom prípade count().

Následne sa vykonal vrchný select. Vzhľadom na neprítomnosť akýchkoľvek indexov sa na prístup k tabuľkám použila výlučne metóda TABLE ACCESS FULL. Na joinovanie sa použila metóda HASH JOIN. Môžeme si všimnúť, že na tabuľku os_udaje sa nikdy nepristupovalo, nakoľko z nej prakticky nezískavame žiadne dáta, pretože údaje o rodných číslach je možné získať z tabuľky zamestnanec. Plán vykonania v základnom stave vyzerá takto:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				14
SORT		AGGREGATE	1	
HASH JOIN			1	8
Access Predicates				
ZAMESTNANEC.ID_ZAMESTNANCA=LEKAR.ID_ZAMESTNANCA				
TABLE ACCESS	LEKAR	FULL	1	3
TABLE ACCESS	ZAMESTNANEC	FULL	2000	5
SORT		AGGREGATE	1	
VIEW	VW_DAG_0		1	4
SORT		GROUP BY	1	4
TABLE ACCESS	LEKAR	FULL	1	3
FAST DUAL			1	2

Obrázok 3 Plán vykonania 1

Následne sme vykonali nasledujúce optimalizácie:

Pridali sme indexy nad PK tabuliek, ktoré select používa (*os_udaje* (rod_cislo)), *zamestnanec* (id_zamestnanca, *lekar*(id_lekara)), ktoré sa štandardne vytvárajú automaticky pri vzniku tabuliek. Tento základný krok znížil cost na hodnotu 9. To nezmenilo execution plan v dolnom selecte, ale v hornom nastali nasledujúce zmeny:

Spojenie tabuliek lekar a zamestnanec sa zmenilo na NESTED LOOPS – vonkajšou tabuľkou bol lekar (prístup cez TABLE ACCESS FULL), vnútornú tabuľku predstavoval zamestnanec, ku ktorému bol rýchly prístup kvôli indexu nad jeho PK (id_zamestnanca) (metóda ACCESS BY INDEX ROWID, index sa prechádzal metódou INDEX UNIQUE SCAN, pretože PK má vlastnosť unikátnosti).

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				9
SORT		AGGREGATE	1	
NESTED LOOPS			1	3
NESTED LOOPS			1	3
TABLE ACCESS	LEKAR	FULL	1	3
INDEX	PK_ZAMESTNANEC	UNIQUE SCAN	1	0
Access Predicates				
ZAMESTNANEC.ID_ZAMESTNANCA=LEKAR.ID_ZAMESTNANCA				
TABLE ACCESS	ZAMESTNANEC	BY INDEX ROWID	1	0
SORT		AGGREGATE	1	
VIEW	VW_DAG_0		1	4
SORT		GROUP BY	1	4
TABLE ACCESS	LEKAR	FULL	1	3
FAST DUAL			1	2

Obrázok 4 Plán vykonania 2

Pridali sme indexy nad FK tabuliek *zamestnanec* (rod_cislo) a *lekar* (id_zamestnanca), čím sme opäť výrazne znížili cost, a to na hodnotu 4. Nastali nasledujúce zmeny:

Nastala zmena prístupu k tabuľke lekar pri dolnom aj hornom selecte, kde sa namiesto TABLE ACCESS FULL použila metóda INDEX FULL SCAN, pretože prechádzanie indexu je rýchlejšie ako prechádzanie tabuľky.

Pridávaním ďalších indexov, ani funkčných by už výkon testovaného príkazu nevylepšilo. Výsledný plán vykonania je nasledujúci:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				4
SORT		AGGREGATE	1	
NESTED LOOPS			1	1
NESTED LOOPS			1	1
INDEX	IX_RELATIONSHIP11	FULL SCAN	1	1
INDEX	PK_ZAMESTNANEC	UNIQUE SCAN	1	0
Access Predicates				
ZAMESTNANEC.ID_ZAMESTNANCA=LEKAR.ID_ZAMESTNANCA				
TABLE ACCESS	ZAMESTNANEC	BY INDEX ROWID	1	0
SORT		AGGREGATE	1	
VIEW	VW_DAG_0		1	1
SORT		GROUP BY	1	1
INDEX	IX_RELATIONSHIP11	FULL SCAN	1	1
FAST DUAL			1	2

Obrázok 5 Plán vykonania 3

2.)

```
select nazov_lieku, pocet_predpisani, poradie from
   (select l.nazov as nazov_lieku, count(*) as pocet_predpisani, rank() over(order by count(*) desc) as poradie
         from liek l join recept r on(l.id_lieku = r.id_lieku)
            where to_char(datum, 'YYYY') = '2020'
               group by l.nazov, l.id_lieku
         ) where poradie <= 0.10*(select count(*) from liek join recept using(id_lieku) where to_char(datum, 'YYYY') = '2020');
```

Tento príkaz vráti 10% najviac predpisovaných liekov v roku 2022. V tomto príkaze sa pracuje s tabuľkami *liek* a *recept* a v základnom stave sa vykonáva za prítomnosti indexov nad všetkými PK a FK týchto tabuliek (v prípade receptu sú to indexy s jedným atribútom nad jednotlivými cudzími kľúčmi).

V tomto základnom stave je cost rovný 49. Plán vykonania bol takýto:

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT				49
VIEW			2640	29
Filter Predicates				
PORADIE<=.1*				
WINDOW		SORT	2640	29
HASH		GROUP BY	2640	29
HASH JOIN			4982	27
Access Predicates				
L.ID_LIEKU=R.ID_LIEKU				
TABLE ACCESS	RECEPT	FULL	4982	20
Filter Predicates				
TO_CHAR(INTERNAL_FUNCTION(R.DATUM),'YYYY')='2020'				
TABLE ACCESS	LIEK	FULL	3484	7
SORT		AGGREGATE	1	
TABLE ACCESS	RECEPT	FULL	150	20
Filter Predicates				
TO_CHAR(INTERNAL_FUNCTION(RECEPT.DATUM),'YYYY')='2022'				

Obrázok 6 Plán vykonania 4

Vidíme, že žiadny zo základných indexov sa pri vykonaní nepoužil. Napriek tomu, že v príkaze sa použité tabuľky joinujú dvakrát za dvoch rôznych okolností, reálne sa vykonal len jeden HASH JOIN, pretože na získanie hodnoty count(*) nie je join potrebný, keďže z tabuľky *liek* nepotrebujeme získať žiadne dodatočné údaje, a taktiež by spojenie neovplyvnilo získaný počet riadkov, pretože každý recept sa musí viazať na konktrétny liek. Zmena číselného udania roku za reťazcovú formu ‘2022’ v podmienke where nemala účinok na výslednú hodnotu cost.

Následne sme vytvorili a testovali nasledujúce indexy:

Pre tabuľku *liek*:

create index LIEK0 on liek (nazov)

create index LIEK1 on liek (nazov, id_lieku)

create index LIEK2 on lien (id_lieku, nazov)

Pre tabuľku *recept*:

create index RECEPT1 on recept (to_char(datum, 'YYYY')

create index RECEPT2 on recept (id_lieku, to_char(datum, 'YYYY'))

create index RECEPT3 on recept (to_char(datum, 'YYYY'), id_lieku)

create index RECEPT4 on recept (id_receptu, id_lieku, id_pacienta, id_lekara, to_char(datum, 'YYYY'))

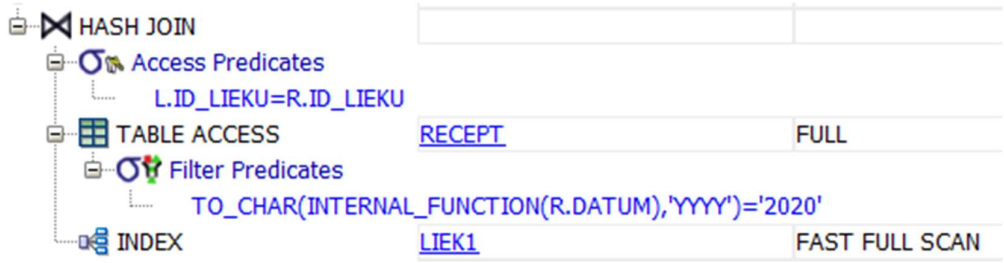
create index RECEPT4 on recept (to_char(datum, 'YYYY'), id_receptu, id_lieku, id_pacienta, id_lekara)

Pri tvorbe indexov sme vyskúšali rôzne kombinácie atribútov a v rámci nich sme menili aj ich poradie a sledovali sme výsledný účinok na hodnotu cost.

V prípade indexov na tabuľku *liek*, mohli sme vzhľadom na malý počet atribútov tejto tabuľky (2) vyskúšať všetky možné indexy (index nad PK sa vytvoril implicitne). Tieto indexy ale nemajú na výkon príkazu veľký vplyv:

LIEK0 sa pri vykonávaní príkazu nepoužije

LIEK1 a LIEK2 sa pri vykonávaní použijú, pričom na poradí atribútov nezáleží, dôležitá je prítomnosť oboch atribútov. Index sa v tomto prípade použije namiesto samotnej tabuľky pri joinovaní oboch tabuliek, a využije sa metóda FAST FULL SCAN.



Obrázok 7 Plán vykonania 6

Vplyv na výsledný cost je však minimálny, a jeho hodnota sa zníži o 2 body na 47.

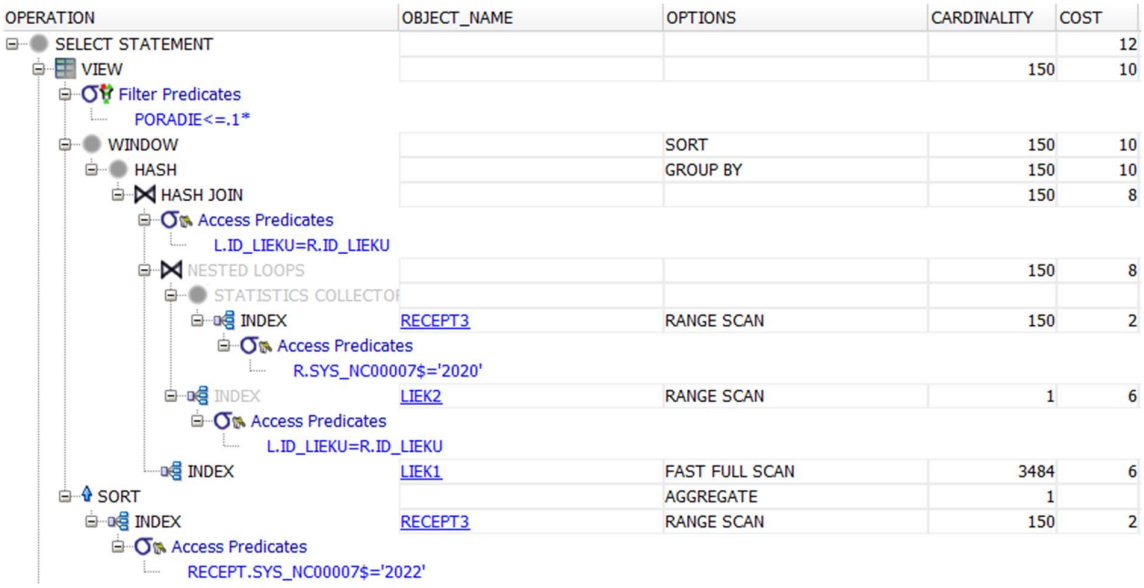
Indexy nad tabuľkou *recept* mali na výsledné náklady príkazu oveľa väčší vplyv, pretože cielili na podmienku *where*. Hodnoty cost pri použití jednotlivých indexov sú uvedené v tabuľke:

Tabuľka 1 Indexy recept

INDEX	COST
RECEPT1	34
RECEPT2	36
RECEPT3	12
RECEPT4	48
RECEPT5	13

Vidíme, že poradie atribútov je v tomto prípade veľmi dôležité. Indexy RECEPT3 a RECEPT4 sa líšia len pozíciou atribútu *to_char(datum, 'YYYY')*, ale keďže z hľadiska príkazu ide o dôležitý atribút, tak táto pozícia rozhoduje o využiteľnosti indexu.

Skombinovaním indexov nad oboma tabuľkami sme docielili náklady príkazu v hodnote 12, čo bola najnižšia hodnota, aká sa nám podarila dosiahnuť. Výsledný plán vykonania s použitím vytvorených indexov vyzeral nasledovne:

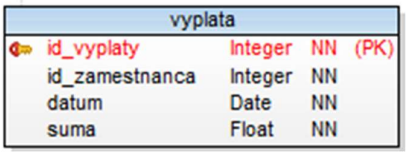


Obrázok 8 Plán vykonania 7

Najzásadnejší rozdiel oproti pôvodnému plánu vykonania bolo použitie indexu RECEPT3, pomocou ktorého je možné rýchlo pristupovať k tabuľke *recept* za pomoci metódy INDEX RANGE SCAN a tým zrýchliť join oboch tabuliek pri zadanej podmienke.

Prístup k dátam na vzdialenom serveri

V našej semestrálnej práci pristupujeme k externým dátam, ktoré sa nachádzajú na školskom serveri asterix. Prepojenie sme vytvorili pomocou objektu database link. Tieto dáta predstavujú informácie o výplatách zamestnancov a nachádzajú sa v tabuľke vyplata, ktorá slúži na uchovávanie výšok a dátumov výplat. Keďže tieto výplaty sa budú aktualizovať raz za mesiac rozhodli sme sa vytvoriť materializovaný pohľad vyplaty_mw, ktorý v sebe tieto dáta uchováva a automaticky sa aktualizuje každý mesiac. Kód k vytvoreniu databázového linku a materializovaného pohľadu sa nachádza v priečinku DB_Link. Štruktúru tabuľky vyplata je možné vidieť nižšie.



Zoznam obrázkov

Zoznam tabuliek