V práci sme výkonnostne analyzovali nasledujúci príkaz:

**select**

**(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 sqldveloper, 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 44. Tabuľky sú prechádzané metódou TABLE ACCESS – FULL, ktorá je použitá celkovo 4x, čo zodpovedá celkovému počtu selectov v príkaze (bez posledného selectu z dual). 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. Pri joinovaní sa najskôr zoradila tabuľka lekar, ktorá sa následne joinovala s tabuľkou os\_údaje. Keďže ale tieto 2 tabuľky nemajú spoločný stĺpcec, použila sa metóda MERGE JOIN CARTESSIAN. Výsledok sa joinoval s tabuľkou zamestnanec prostredníctvom HASH MATCH. Na záver sa opäť vykonala metóda SORT AGGREGATE na konečné získanie veku.

Následne sme vykonali nasledujúce optimalizácie:

1. Pridali sme indexy nad PK tabuľkami, 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:

* Prestalo sa pristupovať na tabuľku os\_udaje, nakoľko z nej prakticky nezískavame žiadne dáta, pretože údaje o rodných číslach je možné získať z tabuľky zamaestnanec.
* Spojenie tabuliek lekar a zamestnanec sa zmenilo na NESTED JOIN – vonkajšou tabuľkou bol lekar (prístup cez TABLE ACCESS FULL), vnútornú tabuľku predstavoval zamestnanec, ku ktorému bol prístup veľmi rýchly kvôli indexu nad jeho PK (id\_zamestnanca) (metóda ACCESS BY INDEX ROWID, index sa prechádzal metódou INDEX UNIQUE SCAN, pretože PK má vlastnosť unikátnosti).

1. 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 – týmto považujeme tento select sa optimalizovaný.