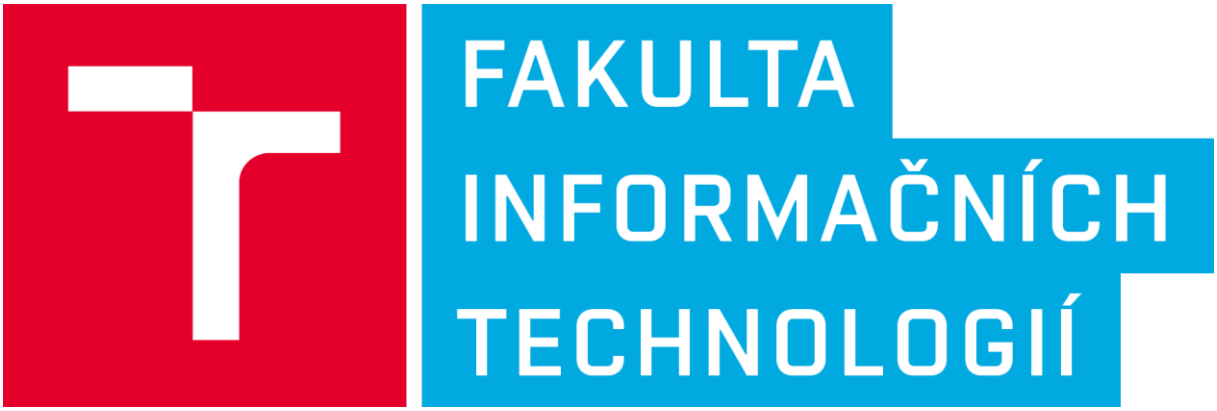
**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**

Fakulta informačních technologií




# DATABÁZOVÉ SYSTÉMY

## Dokumentace k projektu

## Zadání č. 65 – Pohřební ústav

David Valecký (xvalec01), Martin Kladňák (xkladn00) 29. dubna 2019

# Obsah

# Úvod

Na projektu jsme pracovali průběžně, komunikovali jsme přes platformu jménem Discord. Spolupráce probíhala bez problémů, pracovali jsme s verzovacím systémem na Google Drive.

# Zadání

***65. Pohřební ústav***

*Najal si Vás pohřební ústav Rekviem. V IS si chtějí uchovávat informace o zesnulých a pohřbech. K projednání pohřbu je nutno donést list o prohlídce zemřelého (u všech, pokud jich je více), případně fotografii nebožtíka, určit typ obřadu dle církve (u zpopelnění je třeba uvést adresu vybraného krematoria) a kdy a jak dlouho se pohřeb bude konat. Systém musí umožnit přiřadit k pohřbu potřebné zaměstnance ústavu a vytisknout parte. Následně je zesnulý umístěn na volnou pozici v příslušném hrobě (v kolumbáriu či v urnovém háji) a nebo je rozptýlen na pietní loučce. Hřbitov je rozdělen na sektory a za pravidelnou údržbu okolí hrobových míst v sektorech zodpovídá přiřazený zaměstnanec. V rámci sektoru nás zajímají jednotlivá hrobová místa, která mohou být různého typu: jednohrob (2 místa), dvojhrob (4 místa), hrobka (n míst), místo v urnovém háji, schránka v kolumbáriu. O hrobě potřebujeme vědět jeho lokaci v rámci sektoru, maximální kapacitu a zda a kdy byl uhrazen poplatek za pronájem hrobového místa. Dále si uchováváme dobu, po kterou je hrob uhrazen. Skutečnost, že už není zaplacený pronájem hrobu, je oznámena na aktuální kontaktní adresu uvedenou v IS při hrobovém místě. Po uplynutí 6 - měsíční doby bez zaplacení se stává celé hrobové místo volné. Stejně tak po 10 letech od pohřbení zesnulého se stává jeho pozice v hrobě "volná". Avšak systém si musí uchovávat historii umístnění zesnulých.*

# Použité tabulky v databázi

Tabulky v naší databázi jsou následující:

**Zákazník**
**Zesnulý**
**Zaměstnanec**
**Sektor**
**Hrob**
**Pohřeb**

Všechny tabulky, kromě Sektoru, mají svůj primární klíč jako NUMBER, který začíná na číslu 1 a zvyšuje se o 1. Sektor má primární klíč jakožto svoje označení typu VARCHAR2: ‚A' ‚B' ‚C'

Tabulky Zákazník, Zaměstnanec a Zesnulý vychází ze specializace/generalizace z nadřazené tabulky Člověk v návrhu. V implementaci jsme zvolili tento způsob, zavedení jednotlivých odvozených tabulek jako každou samostatně se všemi atributy i s těmi společnými.

# Vyžadované SELECT

Za úkol bylo implementovat tyto příkazy SELECT:

## 2x JOIN TWO TABLES:

Vypíše zákazníky, kterým vypršela doba, do které mají uhradit hrob

```
SELECT Zakaznik.zakJmeno AS JmenoZakaznika, Zakaznik.zakPrijmeni AS
PrijmeniZakaznika, hrob.hrobUhrazenoDo AS UhrazenoDo, Zakaznik.zakTelefon
AS Telefon
FROM Zakaznik JOIN Hrob ON Zakaznik.zakId = Hrob.zakID
WHERE Hrob.hrobUhrazenoDo IS NOT NULL AND Hrob.hrobUhrazenoDo <=
'13.04.2019';
```

Vypíše jména zaměstnanců, kteří vykonávali pohřeb v posledním roce

```
SELECT Zamestnanec.zamJmeno AS JmenoZamestnance, Zamestnanec.zamPrijmeni AS
PrijmeniZamestnance, Pohreb.pohDatumKonani AS DatumKonani
FROM Zamestnanec JOIN Pohreb
ON Zamestnanec.zamId = Pohreb.zamId
WHERE Pohreb.pohDatumKonani >= '13.04.2018' AND Pohreb.pohDatumKonani <=
'13.04.2019';
```

## JOIN THREE TABLES:

Vypíše datum, kdy měl/i pohřeb zesnulí ležící v určitém hrobě

```
SELECT
Zesnuly.zesJmeno AS JmenoZesnuleho, Zesnuly.zesPrijmeni AS
PrijmeniZesnuleho, Pohreb.pohDatumKonani AS DatumKonani
FROM Zesnuly, Hrob, Pohreb
WHERE Zesnuly.zesID = Pohreb.zesID AND Zesnuly.zesID = hrob.zesID AND
Hrob.hrobRada = 2 AND Hrob.hrobSloupec = 2 AND Hrob.sekID = 'A';
```

## 2x GROUP BY a agregační funkce:

Vypíše počet pohřbených v hrobech typu obyčejný hrob sestupně

```
SELECT Hrob.hrobRada AS Řada ,Hrob.hrobSloupec AS Sloupec, Hrob.sekID AS
Sektor, COUNT(*) AS PocetPohrbenych
FROM Hrob, Zesnuly
WHERE Hrob.zesId = Zesnuly.zesid AND Hrob.hrobTyp = 'hrob'
GROUP BY Hrob.hrobRada, Hrob.hrobSloupec, Hrob.sekID
ORDER BY PocetPohrbenych DESC;
```

Vypíše počet zákazníků, kteří mají nejvíc objednaných pohřbů

```
SELECT Zakaznik.zakPrijmeni AS Prijmeni, COUNT(*) AS Nejcastejsi_zakaznik
FROM Zakaznik,Pohreb
WHERE Zakaznik.zakID = Pohreb.zakID
GROUP BY Zakaznik.zakPrijmeni;
```

## EXISTS:

Vypíše všechny volné hroby

```
SELECT Hrob.hrobRada AS Rada, Hrob.hrobSloupec AS Sloupec, Hrob.sekID AS
Sektor
FROM Hrob
WHERE NOT EXISTS(
    SELECT Hrob.hrobId
    FROM Zakaznik
    WHERE Hrob.zakID = Zakaznik.zakID
);
```

## IN s vnořeným SELECT:

Vypíše jména zesnulých, kteří jsou v hrobech v sektoru A

```
SELECT Zesnuly.zesJmeno AS JmenoZesnuleho, Zesnuly.zesPrijmeni AS
PrijmeniZesnuleho
FROM Zesnuly JOIN Hrob
ON Zesnuly.zesId = Hrob.zesID
WHERE Hrob.sekID IN (SELECT Hrob.sekID FROM Hrob WHERE Hrob.sekID = 'A' AND
Hrob.hrobTyp = 'hrob');
```

# Triggery

První trigger slouží pro inkrementaci identifikátoru, aby se při insertu, přesněji před insertem, správně nastavilo ID.

Druhý trigger je pro kontrolu telefonního čísla při vkládání nebo upravení do tabulky Zákazník. Tělo triggeru kontroluje správnost zapsání telefonního čísla, v našem případě je to číslo s českou nebo slovenskou předvolbou. Při špatném zadání je zde využita tato funkce pro vyvolání chyby.

```
Raise_application_error(-20320,'Špatně zadané telefonní číslo!');
```

# Procedury

```
1)
CREATE OR REPLACE PROCEDURE Pocet_Pohrbenych_V_Hrobe(Sektor_hrob IN
VARCHAR2, Rada_hrob IN NUMBER, Sloupec_hrob IN NUMBER)
```

**POPIS:** První procedura slouží k vypsání počtu pohřbených v zadaném hrobě pomocí pozice v rámci sektoru (název sektoru, řada a sloupec hrobu), což jsou vstupní parametry. Vypíše i jména zesnulých uložených v těchto hrobech. Pokud je tabulka prázdní vypíšeme oznámení.

Využíváme *CURSOR*, s kterým následně operujeme ve smyčce.

```
2)
CREATE OR REPLACE PROCEDURE
Pocet_pohrbu_zamestnance_za_casove_obdobi(Zamestnanec_ID IN NUMBER,
Odkdy IN DATE, Dokdy IN DATE)
```

**POPIS:** Druhá procedura slouží k vypsání počtu pohřbů zaměstnance v zadaném časovém období. Vstupními parametry jsou ID zaměstnance a časové rozmezí.

Stejně jako v první proceduře využíváme *CURSOR*.

# EXPLAIN PLAN

EXPLAIN PLAN slouží pro výpis posloupnosti vykonávání operací optimizátorem Oracle. Také vypíše informace o výkonnosti (využití CPU) pro každou z operací a čas jejich trvání.

```
EXPLAIN PLAN FOR
    SELECT Pohreb.zamID, Zamestnanec.zamJmeno, Zamestnanec.zamPrijmeni,
COUNT(*) AS ObslouzenePohrby
    FROM Zamestnanec, Pohreb
    WHERE Zamestnanec.zamID = Pohreb.zamID
    GROUP BY Pohreb.zamID,Zamestnanec.zamJmeno, Zamestnanec.zamPrijmeni;
```

Náš plán zpracovává SELECT pro počet obsloužených hrobů zaměstnanci.

```
-----------------------------------------------------------------------------------
| Id  | Operation           | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |             |     5 |   400 |     7  (15)| 00:00:01 |
|   1 |  HASH GROUP BY      |             |     5 |   400 |     7  (15)| 00:00:01 |
|*  2 |   HASH JOIN         |             |     5 |   400 |     6   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| ZAMESTNANEC |     2 |   134 |     3   (0)| 00:00:01 |
|   4 |    TABLE ACCESS FULL| POHREB      |     5 |    65 |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------
```

Zde vidíme pořadí, ve kterém EXPLAIN PLAN vykonává operace a následně, ke kterým tabulkám, jak přistoupil. Ke všemu jsou vypsané hodnoty operací.

# INDEX

```
CREATE INDEX ObslouzeniPohrbu ON Pohreb(zamID);
```

Při častém využití vypisování z nějaké tabulky můžeme využít INDEX pro zrychlení přístupu. V častém upravováním tabulky naopak můžeme přístup zpomalit.

Nejčastěji pro vypisování z jeho tabulky budeme využívat zaměstnancovo ID, proto jsme zvolili INDEX takto.

Když EXPLAIN PLAN spustíme podruhé se zavedením tohoto indexu, přístup snížil cenu, ale zvýšil chvilkové zatížení procesoru.

```
-----------------------------------------------------------------------------------------------
| Id  | Operation           | Name              | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |                   |     5 |   400 |     4  (25)| 00:00:01 |
|   1 |  HASH GROUP BY      |                   |     5 |   400 |     4  (25)| 00:00:01 |
|   2 |   NESTED LOOPS      |                   |     5 |   400 |     3   (0)| 00:00:01 |
|   3 |    TABLE ACCESS FULL| ZAMESTNANEC       |     2 |   134 |     3   (0)| 00:00:01 |
|*  4 |    INDEX RANGE SCAN | OBSLOUZENIPOHRBU  |     3 |    39 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------
```

Bylo zvoleno skenování INDEX v cyklu, a to zjednodušilo přístup.

Výrazněji by šlo vidět zlepšení při větším počtu záznamů v tabulce.

# Přístupové práva

Přístupové právo jsme využili při vytváření práv pro zaměstnance. Nový zaměstnanec má přístup ke všem tabulkám kromě tabulky zaměstnanec. K této tabulce je mu přístup odepřen, aby nemohl pracovat s citlivými údaji dalších zaměstnanců.

# Materializovaný pohled

Pohled je objekt, který si lze představit jako tabulku v databázi. Rozdíl mezi materializovaným pohledem a nematerializovaným je v tom, že materializovaný pohled se uloží na lokální disk a je rychlejší, pokud se tento pohled bude často volat. V našem projektu jsme si vytvořili pohled na hroby zákazníků. Ten vytváří data okamžitě. Nové změny v tabulce se v pohledu projevují až s příkazem COMMIT.

# Závěr

Skript byl testován na školním serveru Oracle 12c. Skript byl napsaný ve vývojovém prostředí Oracle SQL Developer, kde to bylo i otestováno.