Databázové systémy

2021/2022

Obsah obrázku text

Popis byl vytvořen automaticky

Dokumentace k projektu

Dokumentace popisující finální schéma databáze

Kristián Kičinka (xkicin02)

Václav Valenta (xvalen29) Brno, 01. 05. 2022

# Obsah

Obsah

[2. Návrh databáze 3](#_Toc102353563)

[2.1. ER diagram 3](#_Toc102353564)

[2.2. Use Case diagram 4](#_Toc102353565)

[3. Vytvoření databáze 5](#_Toc102353566)

[3.1. Odstranění všech objektů 5](#_Toc102353567)

[3.2. Vytvoření tabulek 6](#_Toc102353568)

[3.3. Vyvtoření propojení 6](#_Toc102353569)

[4. Triggery 7](#_Toc102353570)

[4.1. Aktivita karty 7](#_Toc102353571)

[4.2. Konverze transakcí do EUR 8](#_Toc102353572)

[5. Procedury 9](#_Toc102353573)

[5.1. Změna limitu karty podle společnosti 9](#_Toc102353574)

[5.2. Změna platu podle pozice 10](#_Toc102353575)

[6. Select 11](#_Toc102353576)

[6.1. Dva dotazy využívající spojení dvou tabulek 11](#_Toc102353577)

[6.2. Dotaz využívající spojení tří tabulek 11](#_Toc102353578)

[6.3. Dva dotazy s klauzulí GROUP BY a agregační funkcí 12](#_Toc102353579)

[6.4. Jeden dotaz obsahující predikát EXISTS 12](#_Toc102353580)

[6.5. Jeden dotaz s predikátem IN s vnořeným selectem 13](#_Toc102353581)

[7. Explain plan 14](#_Toc102353582)

[7.1. Explain plan před vytvořením indexu 14](#_Toc102353583)

[7.2. Vytvoření indexu 14](#_Toc102353584)

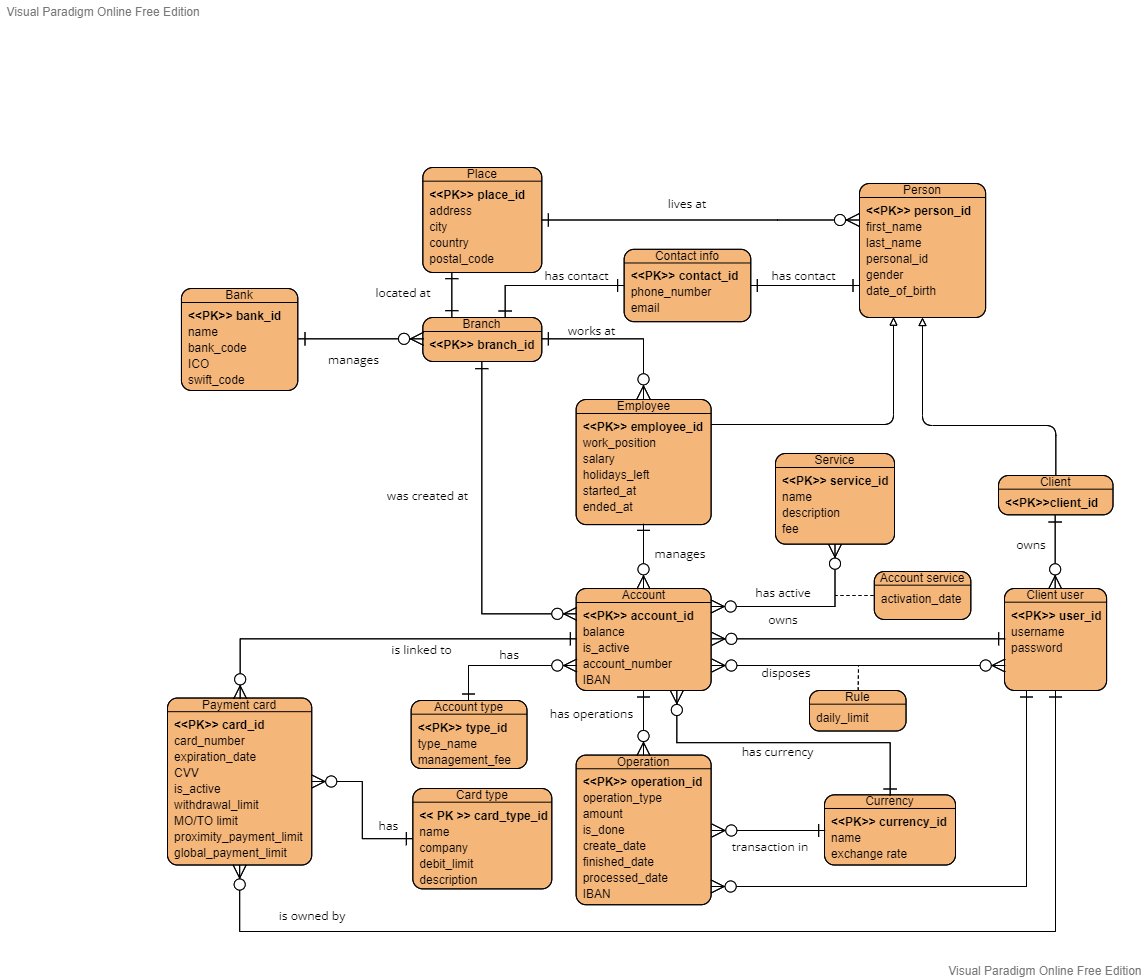
[7.3. Explain plan po vytvoření indexu 14](#_Toc102353585)

[8. Přidání práv druhému členovi týmu 15](#_Toc102353586)

[9. Materialized view 15](#_Toc102353587)

# Návrh databáze

## ER diagram



ER diagram se skládá z následujících základních entit: *Account, Person, Employee, Client, Client\_user, Operation, Payment\_card, Place* a *Contact\_info*.

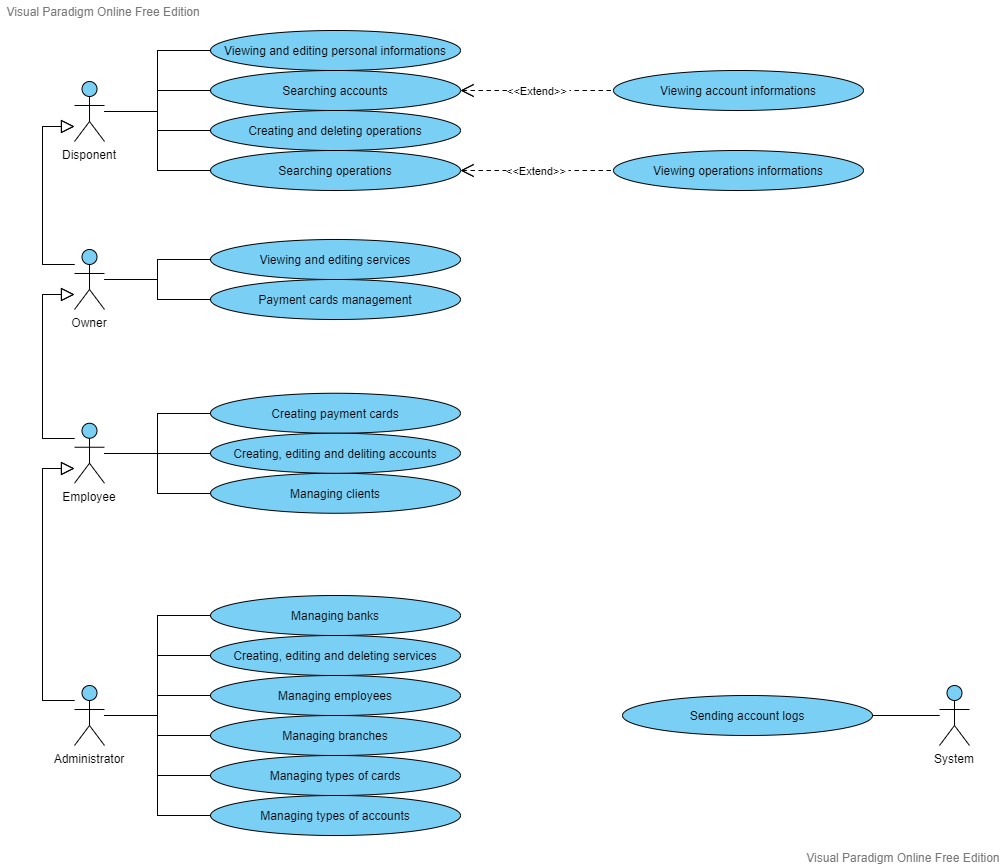
Hlavní prvkem diagramu je entita *Account*. Eviduje položky týkající se účtu klienta. Má nejvíce vazeb na jiné entity. S entitou *Client user* má dvě vazby – 1:N vazba pro vlastníka účtu a N:N vazbu pro disponenty účtu.

Entita *Client* je zde z toho důvodu, aby jedna osoba mohla operovat, či vlastni více účtů – pokaždé s unikátním uživatelským jménem a heslem.

Náš systém se věnuje databázovému návrhu jedné banky – je však potřebné uchovávat základní údaje o konkurenčních bankách. V našem případě se jedná o *název*, *kód banky*, *IČO* a *swift kód* banky.

Pro generalizaci byl použit první způsob popsaný na přednáškách. Tedy tabulka *Client* i *Employee* obsahuje cizí klíč Person\_ID, kterým se odkazuje na tabulku *Person*.

## Use Case diagram



V diagramu užití evidujeme 5 typů uživatelů. Jedná se o disponenta, vlastníka, zaměstnance, administrátora a systém. Jednotliví uživatelé mají dostupnou řadu funkcí a oprávnění se kterými mohou ovládat informační systém.

Disponent účtu má dostupné jen základní operace s přiřazeným účtem, na rozdíl od vlastníka, který má více možností, jako například správu platebních karet propojených s účtem.

Zaměstnanec má druhé nejvyšší oprávnění v systému. Dokáže operovat s právy dostupnými pro klienty, navíc může vytvářet a upravovat účty i platební karty.

Administrátor je uživatel s nejvyššími oprávněními. Dokáže prakticky vytvářet a editovat jakýkoliv záznam

Jednotlivé role tvoří hierarchii a dědí od sebe navzájem práva. Samostatná role je pak role systému, která slouží pro logování a zasílání výpisu účtů.

# Vytvoření databáze

## Odstranění všech objektů

Program musí po spuštění uvolnit všechny zdroje které mohou ale i nemusí být obsazené. Z tohoto důvodu ze zavolá Pl/SQL script, který se o vše potřebné postará.

Nejprve si vytvoří pole pro jednotlivé typy objektů v databázi a následně se pokusí všechny odstranit, pokud se mu to nepodaří z důvodu, že tato tabulka neexistuje, tato chyba je ignorována.

Následující příklad popisuje právě popsanou situaci (zjednodušenou na 3 názvy tabulek):

BEGIN

DECLARE

type array\_t is varray(40) of varchar2(100);

array array\_t := array\_t('client', 'account', 'bank');

BEGIN

FOR i IN 1..table\_array.count

LOOP

-- DROP all tables if possible

EXECUTE IMMEDIATE ('DROP TABLE ' || array(i) || ' CASCADE CONSTRAINTS');

EXCEPTION

WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF;

END;

END LOOP;

END;

END;

/

Mimo jiné se vytvoří všechny sekvence používané pro inkrementaci ID

## Vytvoření tabulek

Po úspěšné inicializaci a vyčištění všech dat začne program vytvářet všechny tabulky – jedná se o 17 tabulek. Každá tabulka používá pro identifikaci své ID, jako primární klíč (např. tabulka *person*, obsahuje *<<PK>> person\_id*).

Tabulky obsahují integritní omezení, aby vyhovovaly podmínkám. Všechny jsou zapsány ve formátu *CHECK regexp\_like* – například:

E- mail: ^\w{3,}(\.\w+)?@(\w{2,}\.)+\w{2,3}$

Telefon: ^((\+)?[0-9]{3})?[0-9]{9}$

Swift kód: ^([A-Z]{4}(CZ|SK)([0-9]|[A-Z]){2})$

Některé tabulky obsahují default hodnoty, např. tabulka *branch* má nastavenou defaultní banku

INSERT INTO branch

VALUES (SEQ\_BRANCH\_ID.nextval, DEFAULT, 6, 6);

## Vyvtoření propojení

Po vytvoření tabulek, program propojuje tabulky pomocí *ALTER TABLE ADD CONSTRAINT* tak, jak je popsáno v ER diagramu.

Následně jsou vytvořené dvě pomocné tabulky pro N:N propojení.

Jedná se o tabulku *account\_service*, která propojuje tabulku *account* a *service* a eviduje služby aktivované pro určitý účet.

Druhá tabulka *rules* slouží pro nastavení pravidel disponentům účtu. Propojuje tabulky *account* a *client\_user*.

Před naplněním tabulek daty se vytvoří triggery a procedury, které budou popsány v následující kapitole.

# Triggery

## Aktivita karty

Tento trigger při změně údajů o kartě, nebo při vytvoření nové karty, kontroluje datum expirace. Pokud toto datum je větší než dnešní datum, nastaví aktivitu karty na hodnotu 0. Jinak změní hodnotu na 1.

Obsah obrázku text

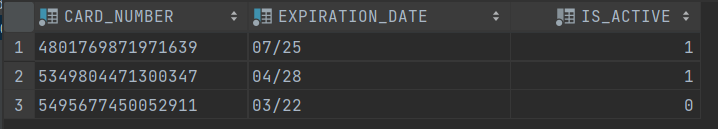
Popis byl vytvořen automaticky

UPDATE PAYMENT\_CARD

SET EXPIRATION\_DATE = '03/22'

WHERE PAYMENT\_CARD\_ID = 3;

Při změně údajů o platební kartě s id 3, se aktivuje trigger a nastaví aktivitu karty na hodnotu 0.



## Konverze transakcí do EUR

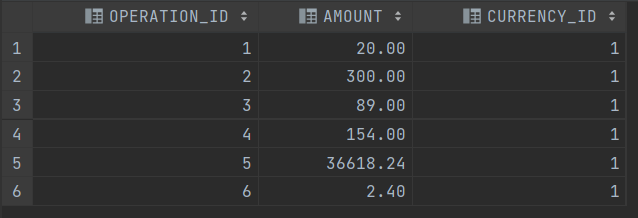
Druhý trigger slouží pro konverzi transakcí s operací do defaultní měny (EUR). Při vytvoření operace se aktivuje trigger a pokud je v jiné měně než v eurech, hodnota se přepočítá podle hodnoty v tabulce *currency*.

Obsah obrázku stůl

Popis byl vytvořen automaticky

INSERT INTO operation VALUES (SEQ\_OPERATION\_ID.nextval, 'payment', **59.99**, '12.1.2022', '13.1.2022', '14.1.2022', 0, 'CZ5262106701002216739313', 3, **2**, 4);

Po vložení nového záznamu v měně CZK se aktivuje trigger a převede hodnotu do EUR

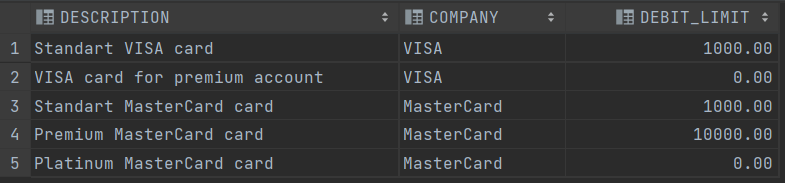


# Procedury

## Změna limitu karty podle společnosti

CHANGE\_DEBITS\_LIMITS\_BY\_COMPANY(cardCompany varchar, debitLimit number)

Tato procedura při zavolání změní limity kreditních karet vybrané společnosti. Procedura používá kurzor na vybrání všech karet společnosti. Pokud je hodnota *debitLimit* null, změní se limity na 0.



BEGIN

CHANGE\_DEBITS\_LIMITS\_BY\_COMPANY('VISA', 50);

END;

/

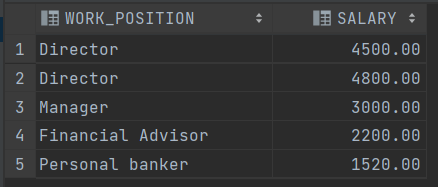
Obsah obrázku text

Popis byl vytvořen automaticky

## Změna platu podle pozice

INCREASE\_SALARY(position VARCHAR, percentage NUMBER)

Procedura změní plat v procentech všem zaměstnancům vybrané pozice.

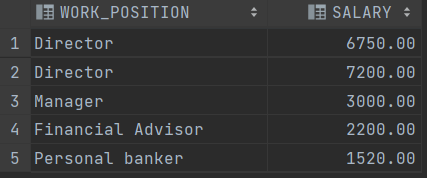


BEGIN

INCREASE\_SALARY('Director', 50);

end;

/



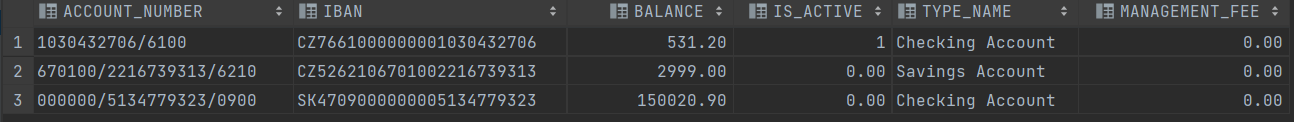
# Select

## Dva dotazy využívající spojení dvou tabulek

Zobrazení základních informací o účtech

SELECT ACCOUNT\_NUMBER, IBAN, BALANCE, IS\_ACTIVE, TYPE\_NAME, MANAGEMENT\_FEE

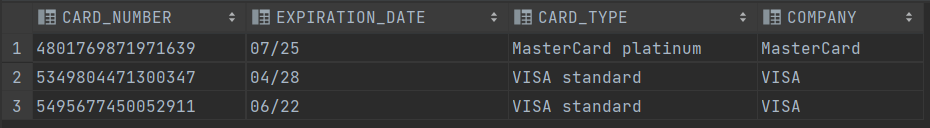
FROM account NATURAL JOIN account\_type;



Zobrazení detailů o platebních kartách

SELECT CARD\_NUMBER, EXPIRATION\_DATE, CARD\_TYPE.NAME as CARD\_TYPE, COMPANY

FROM payment\_card NATURAL JOIN card\_type;



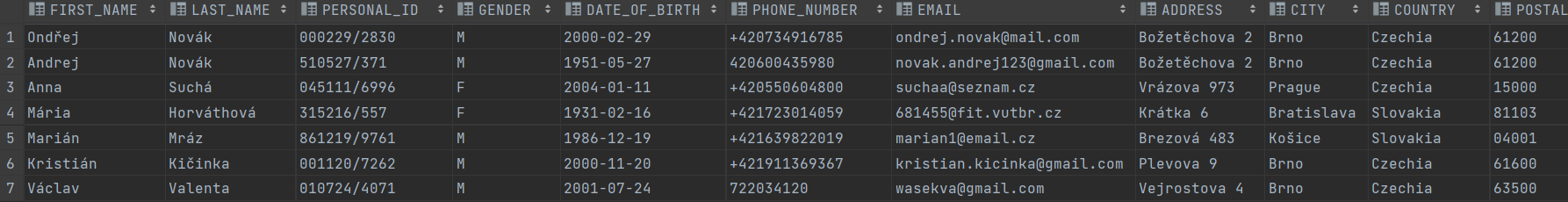
## Dotaz využívající spojení tří tabulek

Veškeré informace o osobách v systému

SELECT FIRST\_NAME, LAST\_NAME, PERSONAL\_ID, GENDER,DATE\_OF\_BIRTH,

PHONE\_NUMBER,EMAIL, ADDRESS, CITY, COUNTRY, POSTAL\_CODE

FROM person NATURAL JOIN place NATURAL JOIN contact\_info;

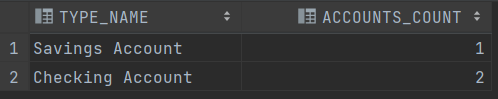


## Dva dotazy s klauzulí GROUP BY a agregační funkcí

Zobrazení počtu účtů podle typu

SELECT TYPE\_NAME, COUNT(\*) as Accounts\_count

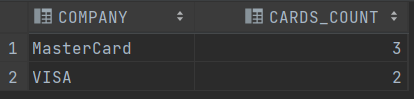
FROM ACCOUNT\_TYPE NATURAL JOIN ACCOUNT GROUP BY TYPE\_NAME;



Zobrazení počtu karet podle společnosti

SELECT COMPANY, COUNT(\*) as CARDS\_COUNT

FROM CARD\_TYPE GROUP BY COMPANY;



## Jeden dotaz obsahující predikát EXISTS

Zobrazení služeb aktivních účtů

SELECT NAME, DESCRIPTION, FEE

FROM SERVICE

WHERE EXISTS(

SELECT ACCOUNT\_NUMBER

FROM ACCOUNT

JOIN ACCOUNT\_SERVICE AccService on ACCOUNT.ACCOUNT\_ID = AccService.ACCOUNT\_ID

WHERE SERVICE.SERVICE\_ID = AccService.SERVICE\_ID

AND IS\_ACTIVE = 1);

Obsah obrázku text

Popis byl vytvořen automaticky

## Jeden dotaz s predikátem IN s vnořeným selectem

Zobrazení zaměstnanců pracujících v bankách jejichž kód je větší než 6000

SELECT FIRST\_NAME, LAST\_NAME, PERSONAL\_ID, WORK\_POSITION, SALARY

FROM PERSON

JOIN EMPLOYEE ON PERSON.PERSON\_ID = EMPLOYEE.PERSON\_ID

JOIN BRANCH ON EMPLOYEE.BRANCH\_ID = BRANCH.BRANCH\_ID

JOIN BANK ON BANK.BANK\_ID = BRANCH.BANK\_ID

WHERE BANK.NAME IN (SELECT NAME FROM BANK WHERE BANK.BANK\_CODE > 6000);

Obsah obrázku text

Popis byl vytvořen automaticky

# Explain plan

## Explain plan před vytvořením indexu

EXPLAIN PLAN FOR

SELECT SERVICE.NAME, SERVICE.FEE, COUNT(A2.ACCOUNT\_ID) AS ACCOUNTS\_COUNT

FROM SERVICE

JOIN ACCOUNT\_SERVICE "AS" on SERVICE.SERVICE\_ID = "AS".SERVICE\_ID

JOIN ACCOUNT A2 on "AS".ACCOUNT\_ID = A2.ACCOUNT\_ID

GROUP BY SERVICE.NAME, SERVICE.FEE

ORDER BY ACCOUNTS\_COUNT DESC;

Explain plan provede akci select – zobrazení počtu aktivních služeb

Obsah obrázku text, počítač, elektronika

Popis byl vytvořen automaticky

## Vytvoření indexu

CREATE INDEX "service\_index" ON "ACCOUNT\_SERVICE" ("SERVICE\_ID");

CREATE INDEX "account\_index" ON "ACCOUNT\_SERVICE" ("ACCOUNT\_ID");

## Explain plan po vytvoření indexu

Obsah obrázku text

Popis byl vytvořen automaticky

# Přidání práv druhému členovi týmu

-- Creates privileges procedure

CREATE OR REPLACE PROCEDURE ADD\_PRIVILEGES(user VARCHAR)

IS

tableName USER\_TABLES.table\_name%type;

CURSOR select\_all\_tables IS SELECT table\_name

FROM USER\_TABLES;

BEGIN

OPEN select\_all\_tables;

LOOP

FETCH select\_all\_tables INTO tableName;

EXIT WHEN select\_all\_tables%NOTFOUND;

EXECUTE IMMEDIATE 'GRANT ALL ON ' || tableName || ' TO ' || user;

end loop;

CLOSE select\_all\_tables;

end;

/

Pro účely přidání práv byla vytvořena výše uvedená procedura. Po zavolání procedury se přidají práva na všechny objekty databáze zvolenému uživateli.

# Materialized view

CREATE MATERIALIZED VIEW show\_disponents\_accounts

AS

SELECT DISTINCT CLIENT\_USER.USER\_ID,

PERSON\_ID,

FIRST\_NAME,

LAST\_NAME,

ACCOUNT.ACCOUNT\_NUMBER,

ACCOUNT.BALANCE

FROM ACCOUNT

JOIN RULES ON ACCOUNT.ACCOUNT\_ID = RULES.ACCOUNT\_ID

JOIN CLIENT\_USER ON RULES.USER\_ID = CLIENT\_USER.USER\_ID

JOIN CLIENT ON CLIENT\_USER.CLIENT\_ID = CLIENT.CLIENT\_ID

NATURAL JOIN PERSON;

Materializovaný pohled je jakýmsi hybridem mezi klasickou tabulkou a pohledem. Podobá se vyrovnávací paměti. Jeho cílem je zvýšit rychlost databázového systému. Zvyšuje rychlost dotazů.