ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA PEDAGOGICKÁ
KATEDRA VÝPOČETNÍ A DIDAKTICKÉ TECHNIKY

PRŮVODCE VOLNĚ DOSTUPNÝM NÁSTROJEM DATOVÉHO MODELOVÁNÍ SQL DATABÁZE

BAKALÁŘSKÁ PRÁCE

Jakub Heidtke
Informatika se zaměřením na vzdělávání

Vedoucí práce: Mgr. Denis Mainz

Plzeň, 2016

Prohlašuji, že jsem diplomovou práci vypracoval samostatně s použitím uvedené literatury a zdrojů informací.

V Plzni, 13. dubna 2016

.....

vlastnoruční podpis

Tímto bych rád poděkoval Mgr. Denisu Mainzovi za metodické vedení při vypracování bakalářské práce.

ZDE SE NACHÁZÍ ORIGINÁL ZADÁNÍ KVALIFIKAČNÍ PRÁCE.

OBSAH

Seznam zkratek

DB - Databáze (Data Base)

DBMS – Database Management Systém, neboli Systém Řízení Báze Dat (SŘBD)

EP – Execution Plan, Plán spuštění. Query optimizer zobrazí sérii příkazů, které je třeba vykonat k výsledku dotazu.

ETL – Extract Transform Load. O ETL hovoříme většinou v kontextu práce s daty z jiného zdroje. Například jiný databázový server, sešit Excel, či FF.

FF – Flat file je jednoduchá databáze v textovém souboru (plochý soubor), v takovém souboru se většinou nachází jedna tabulka.

RS – Result set/ Record set je set řádek který nám vrátí databázový systém po spuštění SQL dotazu, který obsahuje i metadata s názvem sloupců případně i datový typ.

PK – Primary Key, Primární klíč jednoznačně určuje záznam v dané tabulce. Může být tvořen jedním sloupcem, či jako složenina více sloupců.

FK - Foreign Key. Máme-li nad sloupcem cizí klíč, hodnoty ve sloupci odkazují do jiné tabulky, kde je tato hodnota primárním klíčem.

SP - Stored Procedure, Uložená procedura. Blok příkazů na straně databázového serveru, který se vykoná při volání procedury.

NF - Normální Forma

ÚVOD

Tato práce se snaží nastínit základní možnosti datového modelování, administraci a programování na straně databázového serveru. Seznámit se základními pojmy relační databáze, jako například kardinalita, entita, tabulka, či index. Bude zde poskytnut přehled možností jazyka SQL. Vysvětleny budou pojmy jako podmínka, cyklus, kurzor z programovacího jazyka databáze. Hlavním cílem práce je na praktickém příkladu představit hlavní funkce a programové možnosti volně dostupného nástroje datového modelování, jako například nástroj pro tvorbu datových modelů. Práce popisuje i vytváření uživatelů a přidělování uživatelských práv. Na konkrétním příkladu je ukázán návrh jednoduchého docházkového systému.

1 DATOVÉ MODELOVÁNÍ SQL DATABÁZE A SOUVISEJÍCÍ POJMY

1.1 SYSTÉM ŘÍZENÍ BÁZE DAT (SŘBD)

Speciální software pro přístup k údajům v databázi se nazývá česky systém řízení báze dat, případně anglicky Database Management System. Uživatel, případně aplikační program potom nemusí znát fyzickou strukturu uložení údajů, protože k údajům v databázi přistupují prostřednictvím systému řízení báze dat. Komunikace klienta nebo aplikačního programu SŘBD probíhá pomocí jazyka SQL. (Lacko, 2003)

1.2 RELAČNÍ DATABÁZE

Relační databáze je databáze založená na relačním modelu. Relační databáze byla poprvé představena roku 1969. Otcem relačního modelu byl Edgar F. Codd, který byl výzkumníkem firmy IBM a zabýval se výzkumem nových metod zpracování velkého množství dat.

Relační databáze je založena na vztazích, jenž uživatel vidí jako tabulky. Tabulky se skládají z atributů, neboli sloupců a z řádek, které chápeme jako záznamy. Každý záznam je identifikován atributem, který obsahuje unikátní hodnotu. Sloupce mají určen svůj datový typ a doménu, která definuje přípustné hodnoty daného sloupce. Některé sloupce často tvoří tzv. cizí klíče, které chápeme tak, že uchovávají informace o relacích mezi jednotlivými tabulkami. (Hernandez, 2006)

1.3 DATOVÉ MODELY

Datové modelování je proces používaný k definování požadavků na data, která jsou součástí procesů uvnitř organizace. Analytici, kteří tvoří model, musí úzce spolupracovat businessmany a s potencionálními uživateli informačního systému, aby byli schopni podchytit všechna současná, případně možná budoucí očekávání. Datový model popisuje nejen datové elementy, ale i jejich strukturu a vztahy. Datové modely jsou progresivní. Neexistuje nic jako finální model pro obchod, či aplikaci. Zpravidla se postupem času mění a rozvíjí s ohledem na to, jak se mění obchodní procesy.

1.3.1 ERA MODELY

ERA znamená Entity Relationship Atribut. Entita je typ objektu (osoba, věc, místo…), o kterém chceme vést záznamy. Důležitým rysem je, jak jsou jednotlivé objekty v entitě

vzájemně odlišitelné. Například zaměstnanec má přiděleno jedno svoje zaměstnanecké číslo. Toto číslo je tedy atribut sloužící k jednoznačné identifikaci zaměstnance. Relationships neboli relace popisují vztahy mezi entitami, které v datovém modelování zachycujeme. Například budova je rozdělena do několika oddělení. Atribut je vlastnost charakterizující entitu. Může to být například jméno, rodné číslo, nebo adresa. Atributy mohou být identifikační, nebo popisné. Dále mohou být povinné, či nepovinné. V datovém modelování rozeznáváme 3 typy modelů:

1.3.2 KONCEPTUÁLNÍ MODEL

Konceptuální model popisuje sémantiku. Je to vlastně první krok k definování požadavků na data. Na této úrovni vůbec neřešíme použité technologie, zajímá nás pouze teoretický pohled na věc. Takto vzniklý model je poté přeložen do logického (Lacko, 2003).

1.3.3 LOGICKÝ MODEL

Logický model popisuje datové struktury, které mohou být naimplementovány do databáze. Na této úrovni se již mohou volit například datové typy (Lacko, 2003).

1.3.4 FYZICKÝ MODEL

Posledním krokem je transformování do fyzického modelu, kde jsou data organizována do tabulek (Lacko, 2003).

1.4 VZTAHY MEZI OBJEKTY

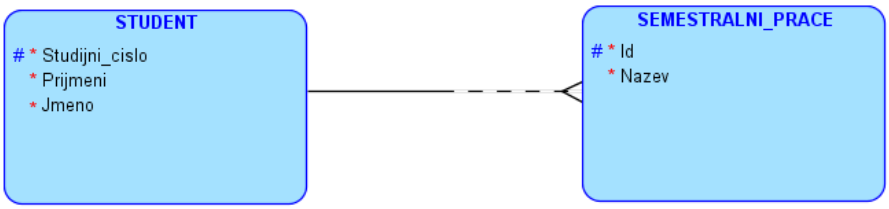
Vztahy zachycují skutečnosti mezi dvěma, či více entitami. Například student studuje předmět.

1.4.1 KARDINALITA

Jeden student však může studovat více předmětů a zároveň jeden předmět může být studován více studenty. V takovém případě má vztah kardinalitu N:M. O něco běžnější je typ kardinality 1:N, kdy například jeden student napsal více semestrálních prací, ale každá ze semestrálních prací byla vždy vytvořena jen jedním studentem. Kardinalita 1:1 znamená, že jednomu záznamu v tabulce odpovídá maximálně jeden záznam z jiné tabulky.

1.4.2 POVINNOST VÝSKYTU

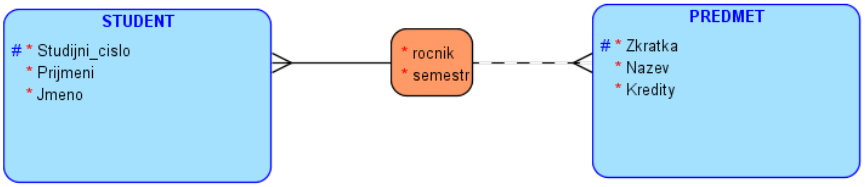
U vztahu definujeme rovněž povinnost výskytu. U výše uvedeného příkladu by z hlediska student > semestrální práce bylo třeba mít povinnost nepovinnou, jinak bychom se dostali do situace, že nemůžeme zaevidovat nového studenta, protože ještě nemá napsanou žádnou semestrální práci. Na druhou stranu relace semestrální práce > student by měla mít výskyt povinný. Nemůže tak nastat situace, že budeme mít semestrální práci, kterou nenapsal žádný student. Tento vztah by pomocí ERA modelu vypadal takto.



1 nepovinný vztah 1:N

Povinnost výskytu je vyznačena přerušovanou čarou. Symbol vraní nohy na straně semestrální práce ukazuje kardinalitu více.

1.4.3 ATRIBUTY VZTAHU



2 vztah N:M

Každý vztah může mít své atributy. Na příkladu je ukázán vztah N:M mezi entitami student a předmět. Každý student musí studovat alespoň jeden předmět a zároveň může existovat předmět, který zatím nebyl studován žádným studentem. Doplňující informace o tom, ve kterém ročníku a semestru student předmět studuje, jsou atributy relace.

1.4.4 ARITA

Pojem arita souvisí s algebraickou operací kartézského součinu. Aritou označujeme počet entit, mezi kterými je dána relace (PAVLOVSKÁ, 2011). Podle počtu entit vstupujících do vztahu, říkáme, že je arita n-ární. Ve výše uvedeném vztahu vidíme aritu 2, neboli

binární, jelikož do vztahu vstupují pouze dvě entity. Mohou nastat i situace, kdy arita bude ternární.

1.5 DATABÁZOVÉ OBJEKTY

Databáze Oracle rozeznává objekty, které jsou spojeny s určitým schématem, ale i objekty, které se schématem spojené nejsou. Vytvoření schéma je způsobem jak logicky seskupit objekty. Schéma je vlastně takový kontejner objektů. Využít schémata můžeme v přidělování uživatelských práv, kdy každý uživatel bude mít přiřazeno nějaké schéma. Přistupovat k objektům, nebo je měnit, potom může pouze autorizovaný uživatel (Database Documentation, 2005).

1.5.1 TABULKA

Tabulka je základním objektem databáze. Tabulky popisují entity, o kterých chceme uchovávat informace. Tabulku tvoří sloupce, kde každý může být jiného datového typu.

Každý sloupec má své záhlaví, ve kterém je možné vidět název sloupce. Řádky reprezentují jednotlivé záznamy vložené do tabulky. Tabulky mohou být různých typů.

- Neindexovaná tabulka tzv. „Heap organized table“ je tabulka u které není definovaný primární klíč, tedy ani index a data tedy nejsou ukládány na disk v žádném konkrétním pořadí (Database Documentation, 2005).
- Indexem organizovaná tabulka je každá tabulka, u které určen primární klíč. Zde jsou data ukládána na disk v pořadí daném indexem. Dotazování a využití disku je s indexovanými tabulkami efektivnější.
- Externí tabulka je tabulka určená pouze ke čtení. Její data se nenachází uvnitř databáze, ale v nějakém externím zdroji například v nějakém textovém souboru, nejčastěji CSV. Takové soubory bývají označovány jako „Flat file“.
- V Systémové tabulce nalezneme informace jako například systémový čas, nebo můžeme zobrazit/měnit/přidělovat přístupová práva. Systém Oracle vlastně ukládá informace sám o sobě a o své činnosti do těchto tabulek.
- Permanentní tabulka je každá tabulka vytvořena běžným způsobem (příkazem CREATE TABLE název_tabulky).
- Dočasná tabulka se používá například tam, kde je výhodnější výsledek vnořeného dotazu uložit do dočasné tabulky v hlavním dotazu dotazovat na ní, takto se vnořený dotaz proveden pouze jednou. Dočasné tabulky existují pouze v paměti RAM. Přístup k nim je tedy mnohem rychlejší.

1.5.2 DOTAZ

Dotaz, neboli query není vlastně objektem jako takovým, protože je to příkaz, který posíláme serveru, který není nikde uložen. Chceme-li dotaz uložit, za účelem jeho opakovaného spouštění, uložíme ho jako pohled. Po spuštění nám databáze vrátí výsledek v podobě result setu. Provedení každého dotazu předchází Execution plan, který si můžeme zobrazit. Příkazy plánu ukazují sekvenci operací, které Oracle vykoná při spuštění dotazu. To je užitečné při optimalizaci dotazů

1.5.3 POHLED

Pohled, neboli view je pouze logická tabulka definovaná sql dotazem, jejíž základem je jedna, nebo více tabulek nebo jiných pohledů. Pohled můžeme použít například, chceme-li povolit uživateli přístup pouze k některým záznamům v tabulce, či zobrazit jen některé sloupce.

1.5.4 SESTAVA

Sestavou rozumíme report. Report je výstupem nějaké analýzy naší databáze. Základem reportu je jeden či více result setů. Report pak není nic jiného než formátovaný výsledek vzniklých RS

1.5.5 INDEX

Indexy slouží k organizaci a řazení dat. Je-li nad tabulkou nastaven index Selektivní dotazy, trvají zpravidla kratší dobu. Díky uspořádanosti dat Oracle ví lépe kde hledat. Základním indexem je primární klíč, pokud je v tabulce klíč definován, tabulka má automaticky index.

Předpokládáme-li, že se v tabulce bude vyhledávat na základě jiných sloupců než je PK, definujeme další indexy. Index může být nad jedním, či více sloupců. Takto vytvořený index vyžaduje extra místo na disku, protože není nic jiného, než další tabulka, ve které jsou záznamy řazeny podle daného sloupce. Při použití velkého množství indexů nám může nastat problém s tzv. „přeindexováním“. Výběrové dotazy sice běží rychleji, Ale co se stane, chceme-li do tabulky data vložit.? Nevkládá se pouze jeden záznam do této jedné tabulky, ale i do všech tabulek reprezentujících indexy a to na konkrétní pozice.

1.5.6 FUNKCE

Pomocí funkcí je možné modifikovat výsledek dotazu.

Funkce pracující s jedním řádkem

Funkce dopočítává výslednou hodnotu v konkrétním sloupci pro každý řádek, může sloužit k přetypování. Může přijmout jeden, či více argumentů a vrátit jednu hodnotu. Tyto argumenty mohou být sloupec, nebo výraz.

Jedná se například o funkce pro práci s řetězci (UPPER(), LOWER()…), čísly, daty (YEAR()).

Příklad:

```
SELECT  UPPER(jmeno), UPPER(prijmeni)
FROM zamestnanec;
```

Dotaz nám vrátí jména a příjmení všech zaměstnanců vypsané velkým písmem.

Seskupovací funkce

Seskupvací funkce umožňují seskupit více záznamů do méně řádek.

Count() – počet

Avg() – aritmetický průměr

Sum() – suma

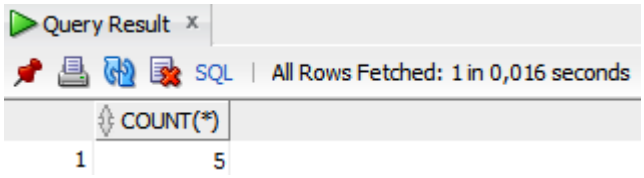
Min() – minimální hodnota

Max() – maximální hodnota

Příklad:

```
SELECT COUNT(*)
FROM zamestnanec;
```

Vrátí celkový počet zaměstnanců.

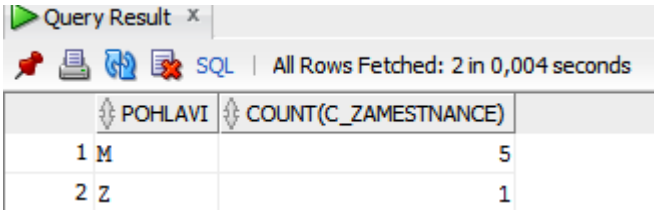


3 výsledek agregačního dotazu s count

Příklad 2:

```
SELECT pohlavi, COUNT(c_zamestnance)
FROM zamestnanec
GROUP BY pohlavi;
```

Vrátí RS se dvěma záznamy. V jednom bude počet zaměstnaných mužů, ve druhém počet zaměstnaných žen.



4 výsledek dotazu s count a group by

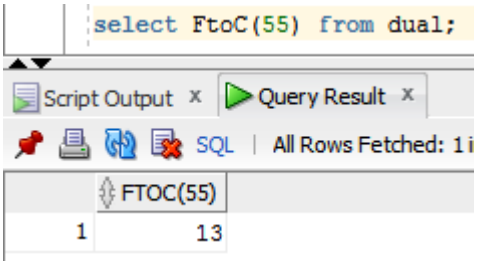
Uživatelem definované funkce

User defined functions jsou funkce, které si můžeme nadefinovat sami pomocí PL/SQL, nebo SQL.

Příklad:

```
CREATE OR REPLACE FUNCTION FtoC(F in Number)
return Number is
begin
return round((F-32)*(5/9),0);
end;
/
```

Funkce převede hodnotu teploty ve Fahrenheitech do stupňů celsia. Volání této funkce pak může vypadat takto:

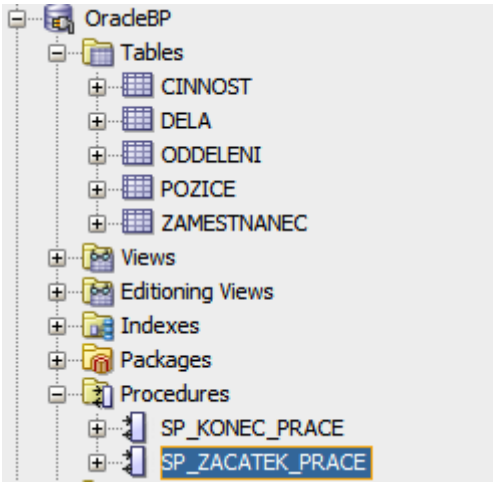


5 výsledek funkce

Část from dual se v Oracle používá tam, kde nechceme vybírat data z žádné specifické tabulky.

1.5.7 ULOŽENÁ PROCEDURA

Stored Procedure je pojmenovaný program, který je uložen v databázi a slouží k vykonání jedné, nebo sérii úloh. Procedura může, nebo nemusí vracet hodnotu.



6 strom procedur v SQL Developeru

Skládá se ze dvou částí. Specifikace a těla.

Specifikace

```
CREATE OR REPLACE PROCEDURE
název_procedury[(název_parametru datový_typ, ...)]
is
```

Tělo

```
 [zde je možno deklarovat proměnné]
BEGIN
  příkazy;
END;
/
```

Procedura může být spuštěna těmito způsoby:

- anonymním blokem

```
Begin
     název_procedury(parametry);
End;
/
```

- příkazem Exec

```
Exec název_procedury(parametry);
```

- příkazem Call

```
Call název_procedury(parametry);
```

1.5.8 SP S VÝSTUPNÍMI PARAMETRY

Parametrům je možné nastavit, zda mají být vstupní klíčovým slovem in, nebo výstupní klíčovým slovem out. Klíčové slovo se používat nemusí, neboť nedefinujeme-li parametr jako výstupní je automaticky brán jako vstupní. Výstupní parametr se používá tam, kde je potřeba, aby procedura vracela hodnotu. Pomocí výstupních parametrů může procedura vracet jednu, nebo více hodnot. Parametr může být rovněž vstupně výstupní, definujeme-li ho s oběma klíčovými slovy (název_parametru IN OUT datový_typ). V tomto případě může parametr plnit obě funkce.

Příklad vytvoření procedury s výstupním parametrem:

```
CREATE OR REPLACE PROCEDURE spzamestnanec_jmeno
(pc_zamestnance IN NUMBER, pjmeno OUT NUMBER)IS
BEGIN
     SELECT jmeno INTO pjmeno
     FROM zamestnanec
     WHERE pc_ zamestnance = pc_ zamestnance;
END;
/
```

Procedura selektuje jméno z tabulky zaměstnanců na základě vloženého parametru čísla zaměstnance a tuto hodnotu ukládá do výstupního parametru.

Příklad použití procedury s výstupním parametrem v anonymním bloku:

```
DECLARE
vjmeno varchar(20);
BEGIN
     spzamestnanec_jmeno(1, vjmeno);
     dbms_output.putline('Jmeno zamestnance je' ||
     vjmeno);
END LOOP;
END;
```

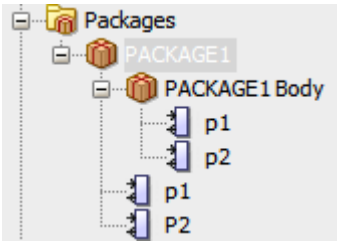
```
/
```

Nejdříve je zde deklarována proměnná, která se předává proceduře jako výstupní parametr. Procedura tedy do této proměnné vrátí hodnotu, která se vypisuje na konzoli.

1.5.9 BALÍK

Package je databázový objekt, který zapouzdřuje proměnné, konstanty, procedury, funkce a kurzory do jednoho celku. Balík sám o sobě nemůže být volán.

Balíky se používají pro zvýšení výkonu, protože když poprvé voláme podprogram z balíku, načítá se do paměti celý balík. To redukuje počet přístupů na disk.



7 strom balíku v SQL Developeru

Specifikace

Ve specifikaci balíku se definují proměnné, které budou přístupné v celém balíku, procedury a funkce.

Příklad:

```
create or replace PACKAGE PACKAGE1 AS
procedure p1 (A NUMBER, B NUMBER);
procedure P2;
END PACKAGE1;
```

Tělo

V těle balíku se nadefinované struktury implementují do konkrétních příkazů.

Příklad:

```
create or replace PACKAGE BODY PACKAGE1 AS
  procedure p1 (A number , B number)AS
  c number;
  BEGIN
    c := A + B;
```

```
    DBMS_OUTPUT.PUT_LINE(c);
  END p1;
  procedure p2 IS
    BEGIN
      DBMS_OUTPUT.PUT_LINE('API procedure p2');
    END;
END PACKAGE1;
```

1.5.10 TRIGGER

Trigger neboli spouštěč je vlastně uložená procedura, která se automaticky spustí kdykoliv je vykonána nějaká DML operace nad tabulkou nebo pohledem.

Základní rozdělení triggerů v PL/SQL je:

- trigger na na úrovni příkazu. Tělo triggeru je provedeno pouze jednou. Tyto triggery neobsahují OLD a NEW kvalifikátory.
- trigger na úrovni řádku. Tělo těchto triggerů je prováděnu postupně pro každý řádek tabulky, nebo pohledu. Tyto triggery obsahují kvalifikátory OLD a NEW.

Přiklad syntaxe:

```
CREATE OR REPLACE trigger název_triggeru
before / after
insert / update / delete
[for each row]
[when podmínka]
[declare]
proměnné, kurzory
begin
  příkazy;
end;
/
```

Dále se triggery dělí na:

- Before - Příkazy before triggeru se provedou ještě před vykonáním DML operace. Je tak možné například ověřit jsou-li data validní, před tím než proběhne jejich insert do tabulky.

- After - Příkazy after triggeru se provedou po vykonání DML operace. Jeho využití je například, chceme-li zaznamenávat informace o tom, kdo kdy vykonal jako operaci s daty. Tomuto zaznamenávání se říká auditování.
- Instead of - Instead of trigger jak název napovídá, nám vykoná jiný příkaz na místo, toho co byl spuštěn. Instead of trigger nalezne své využití například jako Instead of update on nějaký_pohled, kdy nechceme, aby se updatovala data přes pohled, který je založen na více tabulkách, protože by vedlo ke ztrátě integrity dat. Program tedy na místo updatování pohledu může vést k updatu tabulek tvořících pohled.

Kvalifikátory OLD a NEW

Při práci s triggery na úrovni řádků je často třeba rozeznat "starou" nebo "novou" hodnotu k tomu slouží tyto kvalifikátory. Používají se s dvojtečkou a následně názvem sloupce.

Příklad:

OLD:název_sloupce odkazuje na hodnotu, kterou měl sloupec například před updatováním, nebo smazáním.

NEW:název_sloupce odkazuje na hodnotu, kterou sloupec získal po updatování, nebo insertu.

1.6 SQL

Structured Query Language je jazyk vyvinutý pro práci s databází. Jazyk SQL můžeme rozdělit na příkazy DDL a DML. V Oracle je jazyk sql case non-sensitive, to znamená, že nezáleží na velikosti písmen.

DDL (Data Definition Language) je část jazyka SQL, ve které se vytváří objekty jako tabulky, pohledy atd.

1.6.1 DML

Data Manipulation Language jsou všechny příkazy pracující s daty.

Select

Příkaz sloužící k výběru dat například z tabulky, nebo pohledu. Za slovem SELECT specifikujeme názvy sloupců, které chceme zobrazit, nebo hvězdičku, která znamená, že chceme zobrazit všechny sloupce. Za každým sloupce můžeme rovněž definovat jeho alias. To znamená, jaký název sloupce se nám zobrazí v RS. Za slovem FROM následuje

název tabulky, nebo pohledu ze kterého chceme data vybírat, může se zde nacházet i více tabulek spojených joinem.

Nejjednodušší select může vypadat takto:

```
Select * from název_tabulky;
```

Tento select vypíše všechna data ze zvolené tabulky.

Za slovem WHERE následuje podmínka definující kritéria, podle kterých se má z tabulky vybírat.

Příklad:

```
SELECT  *
FROM zamestnanci
WHERE c_zamestnance = 6
```

Vrátí všechny záznamy zobrazující všechny sloupce z tabulky zaměstnanci kde číslo zaměstnance je rovno šesti.

Za částí GROUP BY můžeme definovat, jak se výsledný dotaz má seskupit. Používá se nejčastěji s agregačními funkcemi (SUM(), AVG(), COUNT(), MIN(), MAX()...) tak, že na jeden sloupec použijeme funkci a ostatní sloupce uvedeme v GROUP BY.

Funkce se mohou do sebe vnořovat.

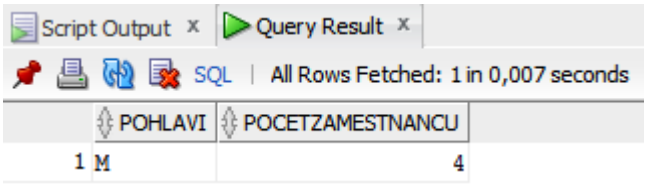
Část HAVING se můžeme použít, chceme-li aby výsledek dotazu byl dle nějaké podmínky vyfiltrovaný.

Příklad:

```
SELECT pohlavi, COUNT(c_zamestnance) as
PocetZamestnancu
FROM zamestnanec
WHERE c_zamestnance <> 1
GROUP BY pohlavi
HAVING pohlavi='M';
```

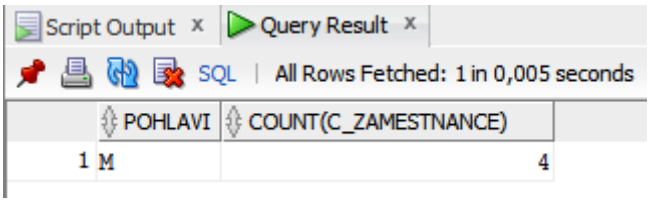
Takto napsaný dotaz nejprve vybere všechny záznamy z tabulky zaměstnanec, kde se číslo zaměstnance nerovná 1 a selectuje pouze sloupec pohlaví a číslo zaměstnance. Poté seskupí sloupec pohlaví, což znamená, že v zobrazení odstraní duplicity ve sloupci pohlaví

a zobrazí tedy jen dva řádky. Jeden pro M a druhý pro Z, protože ve sloupci se pohlaví se jiná hodnota nenachází. Do dalšího sloupce vypíše počet zaměstnanců pro dané pohlaví. V poslední řadě z RS vyřadí řádek kde pohlaví je M. Výsledkem dotazu je tedy jen jeden řádek.



8 výsledek dotazu s having a aliasem

V příkladu je použit alias, protože jinak by název sloupců s počtem zaměstnanců vypadal takto.



9 výsledek dotazu s having

Úplně poslední volitelnou součásti selectu je ORDER BY, za kterém následují názvy sloupců oddělené čárkou, podle kterých se má výsledný RS řadit. Implicitně je nastaven na vzestupné řazení. To lze změnit přidáním slova DESC, které značí řazení sestupné.

Distinct

Klíčové slovo distinct nám odstraní duplicitní řádky z RS. Umisťuje se přímo za slovo select.

Insert

Příkazy insert slouží ke vkládání dat do tabulky.

Příklad:

```
INSERT INTO název_tabulky [seznam sloupců] VALUES
(hodnota1, hodnota2)
```

Za název tabulky můžeme definovat posloupnost sloupců, hodnoty v závorce pak budou vkládány v tomto pořadí. Nedefinujeme-li sloupce Oracle, předpokládá, že v závorce budou hodnoty pro všechny sloupce z definice tabulky.

Příklad:

```
INSERT INTO CINNOST VALUES (5,'Projekt2','Projekt
dva se týká');
```

Vloží jeden záznam do tabulky činnost.

Vkládat do tabulky je možný i výsledek nějakého dotazu select.

Příklad:

```
INSERT INTO cinnost
SELECT  *
FROM Temp_cinnost;
```

V příkladu byla definovaná dočasná tabulka, se kterou byly například vykonávány určité DML příkazy. SQL kód příkazu pak ukazuje nahrání všech dat z dočasné tabulky do tabulky permanentní.

Update

SQL příkaz UPDATE slouží ke změně už existujících dat v nějaké tabulce.

Důležité je zde pokaždé správně definovat klauzuli WHERE jinak po vykonání příkazu dojde ke změně všech záznamů tabulky. Za slovem UPDATE se nachází název tabulky, kterou chceme updatovat. Za slovem SET následuje název sloupce, kterému chceme

přiřadit novou hodnotu. Sloupců zde může být víc, v takovém případě za novou hodnotu prvního sloupce vložíme čárku a můžeme psát název a hodnotu dalšího sloupce.

Příklad:

```
UPDATE ZAMESTNANEC
SET C_UCTU = '254868755/5110'
WHERE C_ZAMESTNANCE = 6;
```

SQL příkaz updatuje v tabulce zaměstnanec sloupec číslo účtu všem zaměstnancům jejich číslo je 6.

Delete

Delete slouží k mazání v tabulce. Opět je nutné definovat WHERE jinak dojde k vymazání obsahu celé tabulky.

Příklad:

```
DELETE
FROM zamestnanec
WHERE c_zamestnance = 6;
```

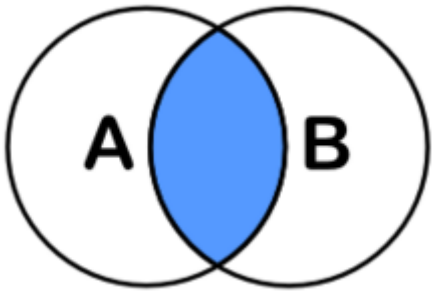
Příkaz smaže z tabulky zaměstnanec všechny záznamy, kde ve sloupci c_zamestnance nalezne hodnotu 6.

Join

Joinem definujeme spojení tabulek v sekci FROM. Velmi často totiž potřebujeme přistupovat k datům z více tabulek najednou. Join je způsob jak získat data z více tabulek v jediném RS.

Inner Join

Vnitřní spojení nám vybere záznamy ze dvou tabulek, které se shodují v jednom, či více sloupcích.



10 inner join

Příklad:

```
SELECT z.JMENO,
  z.PRIJMENI,
  p.NAZEV_POZICE
FROM ZAMESTNANEC z
INNER JOIN POZICE p
ON p.C_POZICE = z.POZICE_C_POZICE;
```

Tento select nám vrátí jméno, příjmení a pozici všech zaměstnanců. Pro zjednodušení kódu je zde použit alias “z” pro tabulku zaměstnanec a “p” pro tabulku pozice.

Předpokládejme, že v tabulce zaměstnanec existuje záznam s pracovníkem, který momentálně nemá přiřazenou žádnou pozici. Inner join tedy tento záznam úplně vypustí.

Script Output | Query Result

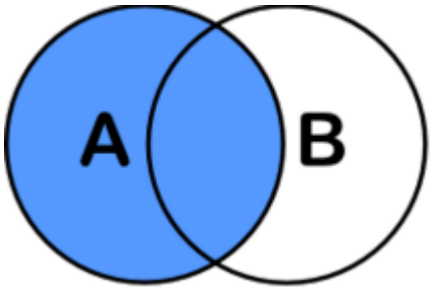
SQL | All Rows Fetched: 6 in 0,009 seconds

	JMENO	PRIJMENI	NAZEV_POZICE
1	Jan	Zkušební	Ředitel
2	Petr	Pan	Manažer oddělení
3	Jan	Bílek	Vedoucí
4	Eduard	Kachl	Řadový pracovník
5	Petr	Pan	Řadový pracovník
6	Jana	Testová	Řadový pracovník

11 výsledek dotazu s inner join

Left join

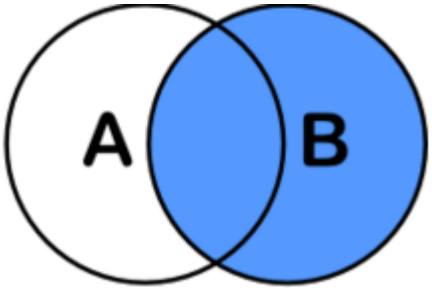
Jak je vidět na obrázku Left Outer Join nám oproti Inner Joinu přidává i všechna data z tabulky zapsané vlevo. V našem případě by to tedy byla tabulka zaměstnanec. Tento typ spojení by nám vyřešil problém pracovníka bez přidělené pozice a zobrazil ho ve výsledném RS s hodnotou null ve sloupci pozice.



12 left join

Right join

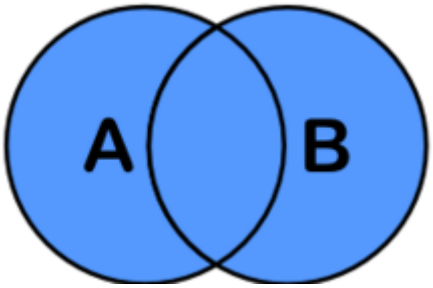
Right join je vlastně obdobou left joinu. S jediným rozdílem že tabulka obsahující všechna data bude tabulka zapsaná na pravé straně.



13 right join

Full Join

Full Outer Join vybírá data z obou spojovaných tabulek a u obou zobrazí i hodnoty, které nemají společné.



14 full join

Cross Join

Cross join je nežádoucím typem joinu, který vznikne špatným zápisem. Jedná se vlastně o kartézský součin obou tabulek.

Množinové operace s dotazy

UNION – Tento příkaz umožňuje sjednocení dvou dotazů do jediného výsledného RS.

Příklad:

```
SELECT *
FROM ZAMESTNANEC
WHERE C_zamestnance =  1
UNION
 SELECT *
FROM ZAMESTNANEC
WHERE C_zamestnance =  2
```

Sjednotí oba RS a vrátí jako jeden se dvěma záznamy.

MINUS – Příkaz minus odečte obsah druhého dotazu od prvního. Z výsledku prvního dotazu tedy vrátí pouze řádky, které se nenachází ve výsledku druhého dotazu.

INTERSECT – Tento příkaz představuje průnik obou dotazů. Z výsledku prvního dotazu tedy vrátí pouze záznamy, které obsahuje i výsledek druhého dotazu.

1.7 TRANSAKČNÍ ZPRACOVÁNÍ

Oracle nám umožňuje pracovat s daty v transakcích. Jsme li připojeni k databázi, veškeré DML operace jsou ukládány nejprve do žurnálu až po commitu jsou opravdu uložené do databáze a viditelné pro ostatní uživatele. Oracle umožňuje vnořování transakcí.

1.7.1 COMMIT

Commit je příkaz jehož zavoláním říkáme, že všechny DML operace, které jsme při práci s daty provedly, jsou správně a mohou být provedeny.

1.7.2 ROLLBACK

Příkaz rollback znamená “zahodit” všechny změny které jsou v žurnálu a nejsou tedy ještě uloženy do DB, například protože někde nastala chyba.

1.7.3 ZÁMKY

Jedním z prostředků zachování integrity dat v databázovém systému Oracle jsou zámky. Při transakčním zpracování, může snadno dojít k události, že jeden uživatel upravil data v nějaké tabulce, ale práci s nimi nedokončil příkazem commit, nebo rollback. Tyto data

jsou pro ostatní uživatele nedostupná. Pokouší-se se někdo k nim přistoupit, Oracle ho nechá čekat, dokud nejsou data odemknuta.

1.8 NORMÁLNÍ FORMY

Normální formy se zabývají odstraňováním redudance v databázi. Definují pravidla, která musí být splněna, aby databáze splňovala danou normální formu. Pro splnění normální formy vyššího řádu je vždy nutno splnit všechny NF nižšího řádu.

1.8.1 **0.NF**

Tabulka je v 0.NF, pokud obsahuje ne-atomické sloupce, nebo pokud se některé sloupce opakují.

1.8.2 **1.NF**

Tabulka je v 1.NF, pokud jsou všechny sloupce atomické. To znamená, že je nejde dále rozdělit. Ne-atomickým sloupcem může být například sloupec, ve kterém je sloučeno jméno a příjmení dohromady. Tabulka v 1.NF má definovaný primární klíč (Lacko, 2003).

1.8.3 **2.NF**

Tabulka je v 2.NF, jsou-li všechny neklíčové atributy plně závislé na primárním klíči. Je-li primárním klíčem pouze jeden sloupec, je tato podmínka automaticky splněna. Pokud existuje neklíčový atribut, který není plně závislý na primárním klíči, je nutné tabulku rozdělit do dvou relací (Lacko, 2003).

1.8.4 **3.NF**

Tabulka je ve 3.NF pokud všechny neklíčové atributy jsou navzájem nezávislé (Lacko, 2003).

1.8.5 **BCNF**

Boyce/Coddova normální forma aplikuje 3.NF do primárního klíče. Říká tedy, že všechny neklíčové atributy jsou navzájem nezávislé. Opět pokud má tabulka pouze je jeden sloupec jako primární klíče a nachází se ve 3.NF je tato podmínka splněna triviálně (Date, c2005).

1.9 INTEGRITNÍ OMEZENÍ

Integritní omezení (constraints) vymezují korektnost DB. Říkají, jaká data se mohou v DB nacházet a jaká ne.

1.9.1 PRIMÁRNÍ KLÍČ (PK)

PK je nejznámějším IO říká nám, že v tabulce se již nesmí nacházet záznam se stejným PK.

1.9.2 UNIQUE

Unique constraint funguje podobně jako primární klíč. Toto integritní omezení volíme tam, kde chceme zajistit unikátnost hodnoty v celé tabulce, ale nechceme sloupec zahrnout do primárního klíče. Výhodou oproti PK je, že sloupec s Unique umožňuje mít ve sloupci hodnotu null.

1.9.3 CIZÍ KLÍČ (FOREIGN KEY)

Foreign key constraint je omezení cizího klíče. Říká, že ve sloupci se mohou nacházet pouze hodnoty, které obsahuje tabulka, na kterou odkazuje.

1.9.4 CHECK

Check constraint nám umožňuje vkládanou hodnotu testovat nějakou podmínkou. Pokud není podmínka splněna, Oracle vrací chybovou hlášku.

2 PRŮVODCE HLAVNÍMI FUNKCEMI VOLNĚ DOSTUPNÝM NÁSTROJEM PRO DATOVÉ MODELOVÁNÍ S POUŽITÍM VLASTNÍHO PŘÍKLADU DATABÁZE

Jako nástroj pro datové modelování jsem zvolil nástroj Oracle Database 11g Express Edition (XE), jenž je v současnosti jeden z nejvyužívanějších nástrojů datového modelování a je volně dostupný po registraci na stránkách Oraclu viz odkaz níže.

http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html

Tento produkt je, dle žebříčku DB-Engines,v současnosti hodnocen jako nejpopulárnější databázový systém. Hodnocení je založeno na počtu zmínění systémů na webových stránkách, četnosti vyhledávání v Google, četnosti technických diskusí (např. na stránce Stack Overflow), počtu pracovních nabídek, počtu zmínění na profesních a sociálních sítích (Complete Ranking, 2016).

Tento produkt je k dispozici zdarma a může být nainstalován na libovolný server. Verze express je omezuje velikost databáze na 11GB, může pracovat maximálně s 1GB operační paměti a pouze jedním CPU. Poté máme možnost upgradu na některou z placených verzí a to Standard edition nebo Enterprise edition.

Program pro práci s Oracle databází se nazývá SQL developer. Tento nástroj umožňuje modelování i na relační vrstvě (ERA model) pomocí Data modeleru. Tento model lze poté převést do fyzické vrstvy (SQL příkazů).

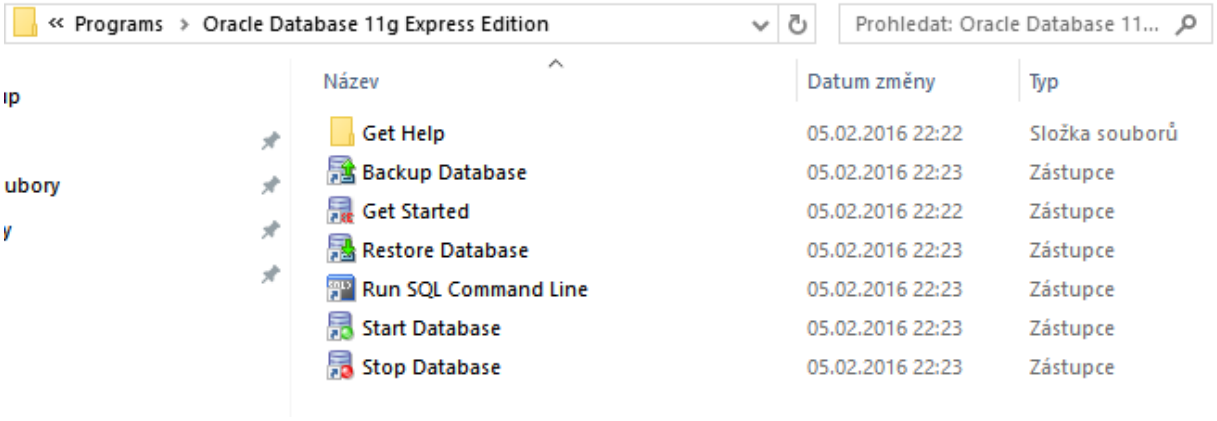
Některé alternativní nástroje

- MS-ACCESS – Access je základním nástrojem pro vytváření menších databází, který je obsažen v balíku MS-Office. Kromě vytvoření databáze umožňuje vytváření formulářových aplikací a datových sestav. Pracuje s programovacím jazykem VBA (Visual Basic for Applications).
- MS-SQL Server – SQL server je softwarové řešení od společnosti Microsoft pro větší databáze. Jeho součástí je Database Engine, neboli SŘBD. Balík dále obsahuje nástroj na tvorbu reportů Reporting Services (SSRS), komponentu pro ETL procesy Integration Services (SSIS) a Analysis Services (SSAS), což je nástroj pro analýzu velkého množství dat, kde se pracuje s tzv. datovými kostkami. Podobně jako Oracle nabízí verzi Express, která je zdarma.
- MySQL Workbench – stejně jako v případě SQL developeru se jedná o programový nástroj umožňující modelovat data i na relační vrstvě (ERA model)

s možností převést datový model lze poté převést do fyzické vrstvy. Výhodou databázového systému MySQL oproti Oracle je nižší cena. Je k dispozici ve verzi Free a v Placené verzi Enterprise, která je více stabilní. Nevýhodou pak jsou menší programové a administrační možnosti.

2.1 INSTALACE DATABÁZE

Po stažení 316MB zip souboru, je nutno instalační soubory nejprve extrahovat. Při samotné instalaci se kontrolují systémové požadavky, administrátorská práva zjišťuje se rovněž, je-li přítomna nějaká jiná instance Oracle XE service. Následuje zvolení adresáře a hesla pro databázi. Po skončení instalace je nutnost databázi spustit. Učiníme tak v nabídce start > všechny programy > Oracle Database 11g Express Edition > Start Database.



15 adresář Oracle

Otevře se okno s příkazovým řádkem a po chvíli jsme informováni, že služba Oracle Srvice XE byla úspěšně spuštěna.

Nyní se můžeme do databáze přihlásit pomocí Run SQL Command Line. Po otevření okna s příkazovým řádkem zadáme:

1. Příkaz “connect”
2. Jako uživatelské jméno (user-name) zadáme “SYSTEM”
3. Heslo zvolené při instalaci databáze

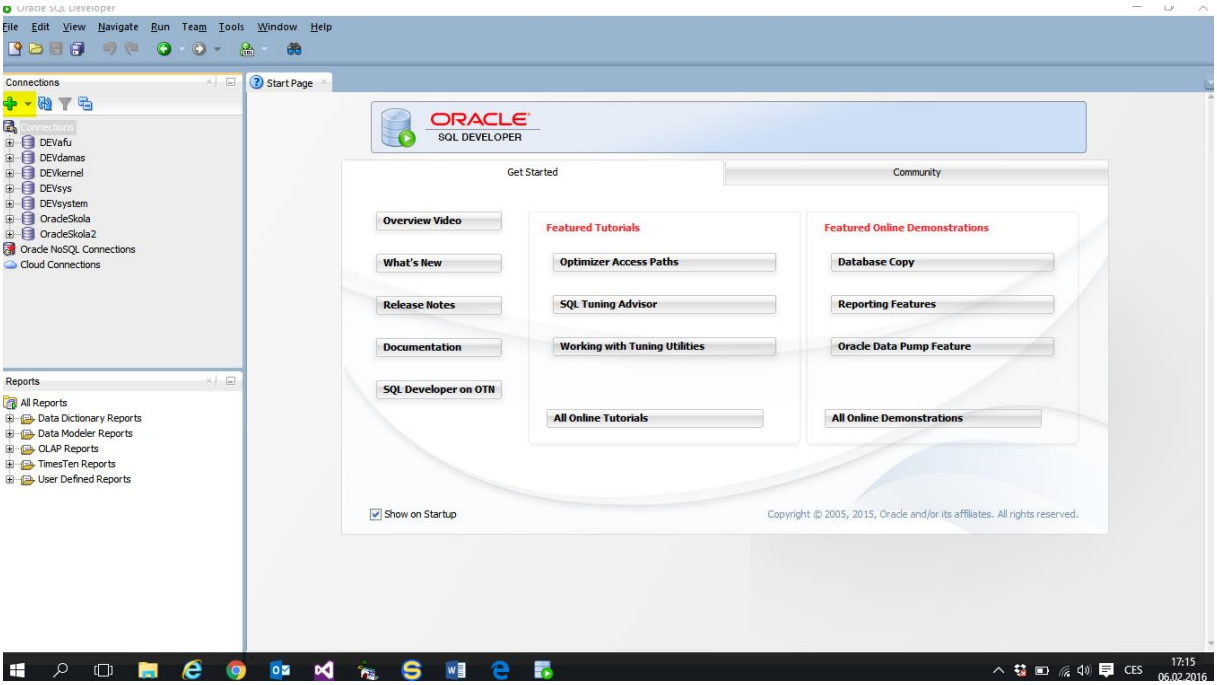
V příkazovém řádku se vypíše stav Connected a nyní už můžeme pracovat s databází pomocí příkazového řádku SQL*Plus. Tento způsob práce vyžaduje mnoho zkušeností. Lepším řešením je používání rozhraní SQL Developer.

2.2 INSTALACE SQL DEVELOPERU

Z níže uvedeného odkazu máme možnost stáhnout SQL developer pro různé operační systémy, pokud volíme verzi pro windows, máme ještě na výběr verzi zahrnující Javu (JDK 8 included), případně pokud máme Javu na našem stroji nainstalovanou, můžeme volit verzi JDK 8 required. Stažený soubor není nutno instalovat, stačí extrahovat do námi zvoleného adresáře a spustit.

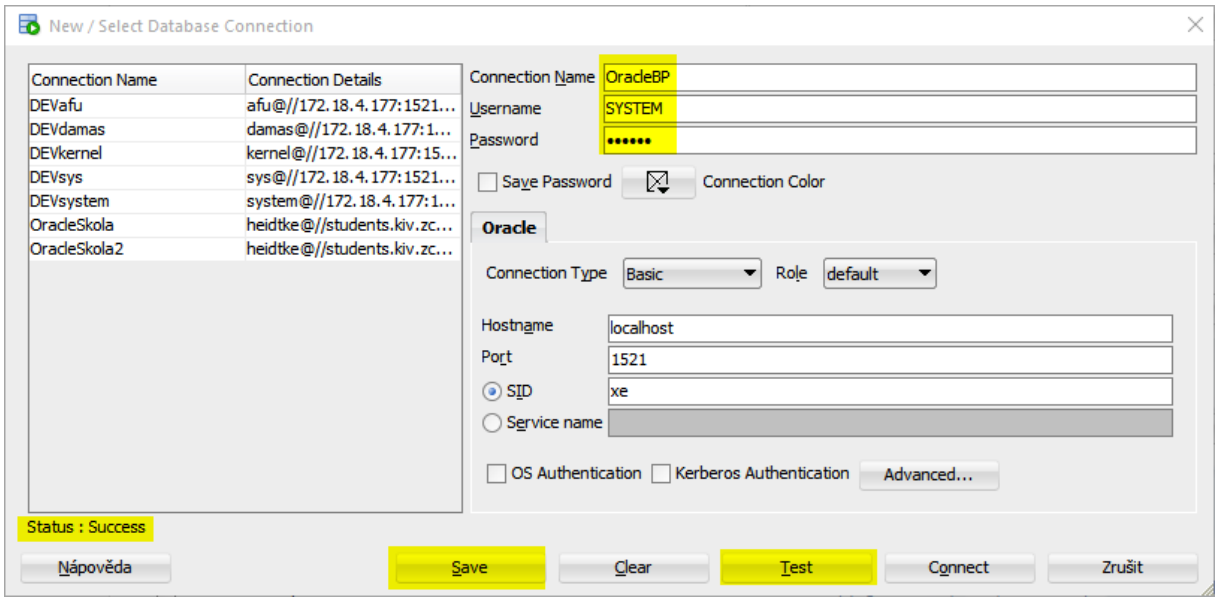
http://www.oracle.com/technetwork/developer-tools/sql-developer/downloads/index.html

Posledním krokem je nastavení nového připojení. Učiníme tak pomocí tlačítka zeleného křížku zobrazeného níže.



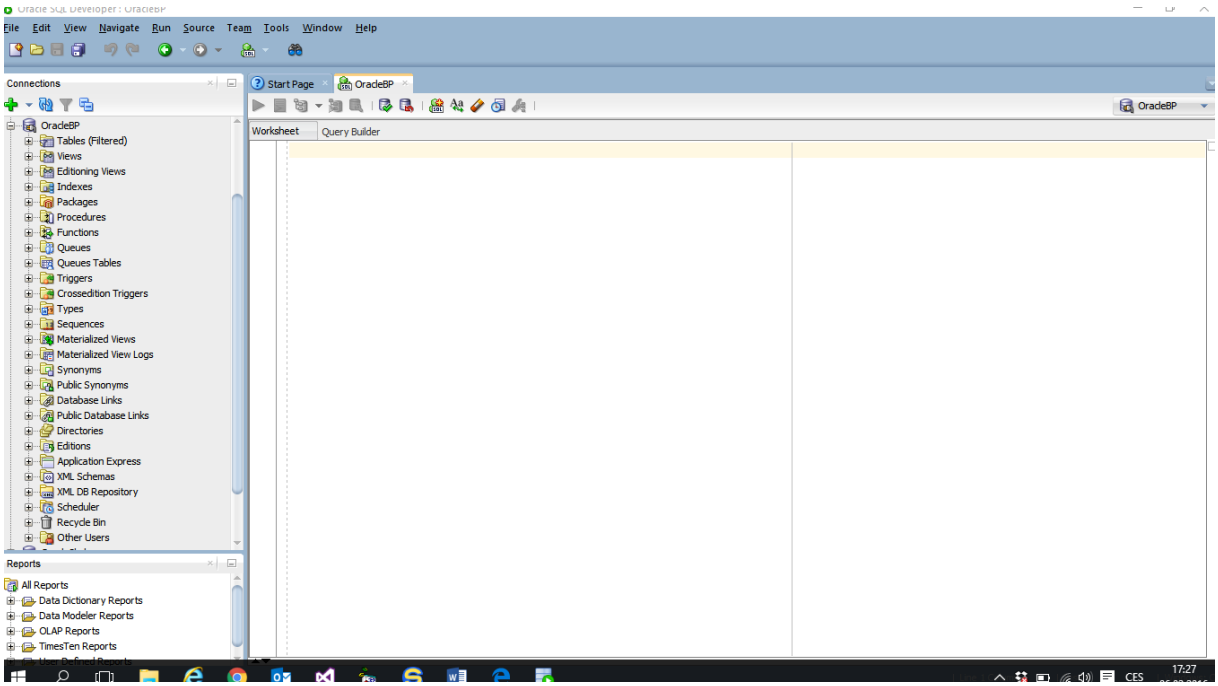
16 úvodní strana SQL Developeru

Zobrazí se okno pro konfiguraci připojení, kde opět vyplníme „Username“ a „Password“ a spojení pojmenujeme v poli Connection Name. Správnost konfigurace otestujeme tlačítkem Test a objeví se status Success. Po úspěšném ověření si spojení uložíme tlačítkem Save.



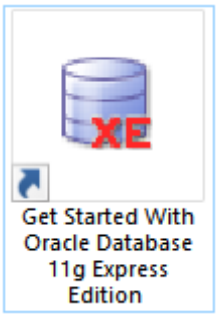
17 konfigurace připojení

Nyní již máme vše potřebné nainstalované a nakonfigurované a můžeme pracovat s naší databází pomocí SQL developeru. Po otevření spojení OracleBP vidíme na levé straně všechny databázové objekty rozřazeny do příslušných kategorií. Na pravé straně je k dispozici connection session, kde můžeme přímo psát SQL a PL/SQL příkazy.



18 connection session

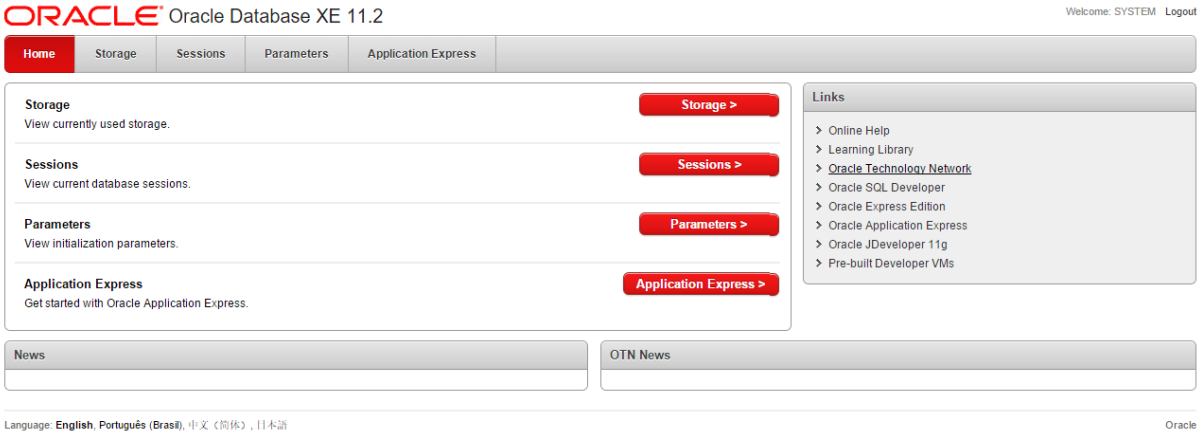
2.3 WEBOVÝ KLIENT PRO MANAGEMENT DATABÁZE



19 zástupce

Po instalaci databáze se na ploše vytvoří zástupce odkazující na adresu: 127.0.0.1:8080

Zde jsou k dispozici různé informace o námi vytvořeném informačním systému.

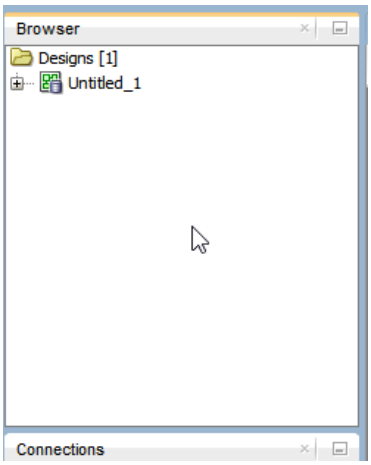


20 webový klient

Můžeme zde například monitorovat využití diskového prostoru. Nalezneme zde některé údaje o proběhlých či stávajících připojeních, parametry databáze pro administrační účely, a můžeme zde i vytvořit nového uživatele.

2.4 DATA MODELER

V kartě View > Data Modeler vybereme možnost Browser a otevře se nám nové okno.

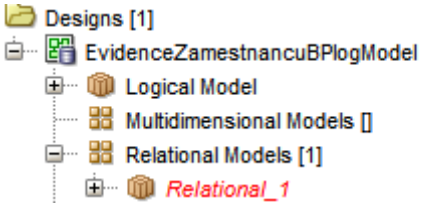


21 data modeler browser

Pravým tlačítkem nad složkou Designs zvolíme New Design. Nyní máme k dispozici nástroj na vytvoření datového modelu.

Data modeler ve výchozím nastavení nezobrazuje atributy relace, abychom je viděly v pravým kliknutím do plátna datového modelu vybereme Show > Relationship Attributes.

SQL developer zvýrazňuje červenou barvou objekty, u kterých byla provedena změna, a nebyly uloženy.



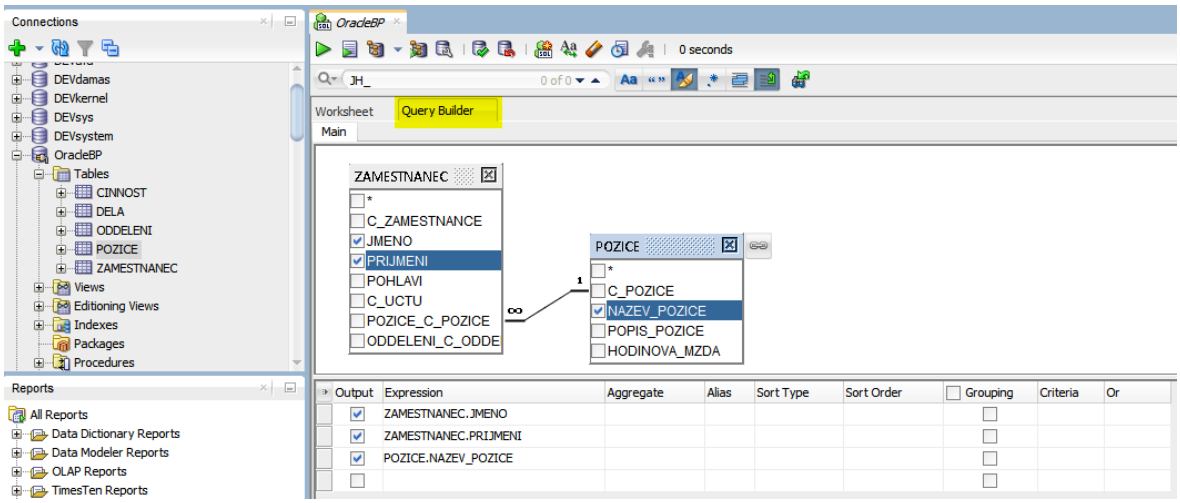
22 změna v relačním schématu

Příklad vytvoření ERA je popsán v jedné z dalších částí (kapitole 3.1)

2.5 VYBRANÉ FUNKCE SQL DEVELOPERU

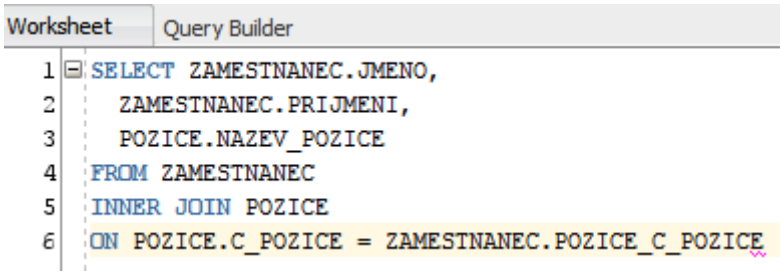
2.5.1 QUERY BUILDER

Tento nástroj slouží k usnadnění vytváření SQL dotazů. Uživatel tak nemusí úplně znát syntaxi SQL. Nalezneme ho po překliknutí z karty Worksheet na kartu Query Builder. Poté můžeme přetáhnout ze seznamu tabulky, ze kterých chceme vybírat, zaškrtnout sloupce které chceme zobrazit, nastavit filtry, řazení, typ joinu a mnoho dalšího.



23 query builder

Po opětovném přechodu na kartu Worksheet se nám dotaz ukáže v podobě SQL.



24 převedené SQL z query builderu

Z nějakého důvodu mu ovšem chybí středník na konci. Středník je v Oracle povinný za všemi příkazy.

2.5.2 VYTVÁŘENÍ OBJEKTŮ

Objekty můžeme vytvářet v GUI, nebo pomocí DDL příkazů.

Vytvoření tabulky s GUI

Tabulku je možné vytvořit po pravém kliknutí na složku tables. Volba Create table.

Příklad vytvoření tabulky s SQL

```
CREATE TABLE ZAMESTNANEC
(
C_zamestnance INTEGER NOT NULL,
Jmeno VARCHAR2(50 CHAR) NOT NULL,
Prijmeni VARCHAR2(50 CHAR) NOT NULL,
C_uctu VARCHAR2(30 CHAR),
```

```
POZICE_C_pozice INTEGER NOT NULL,
ODDELENI_C_oddeleni INTEGER NOT NULL
);
```

2.5.3 EDITACE OBJEKTŮ

Editace struktury tabulky

Vytvořenou tabulku můžeme upravit příkazem Alter Table.

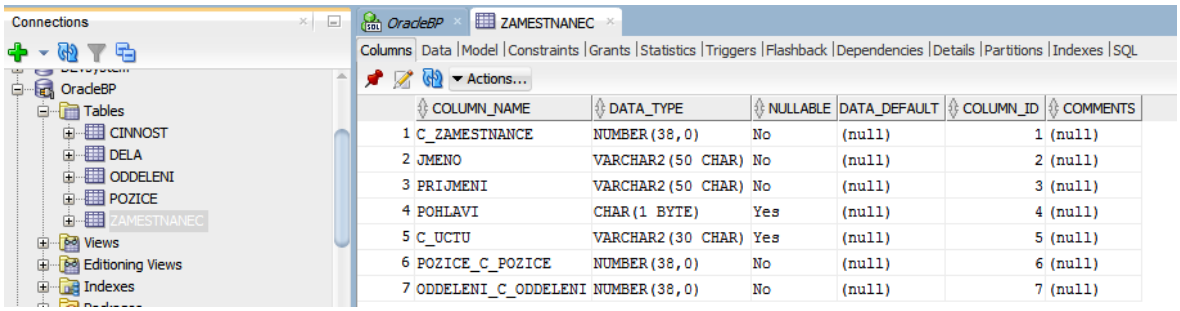
Příklad:

```
ALTER TABLE ZAMESTNANEC ADD CONSTRAINT
ZAMESTNANEC_PK PRIMARY KEY ( C_zamestnance );
```

Tento příkaz definuje v tabulce zaměstnanec primární klíč na sloupec c_zamestnance. Příkazem alter můžeme měnit i ostatní objekty jako pohledy indexy funkce atd.

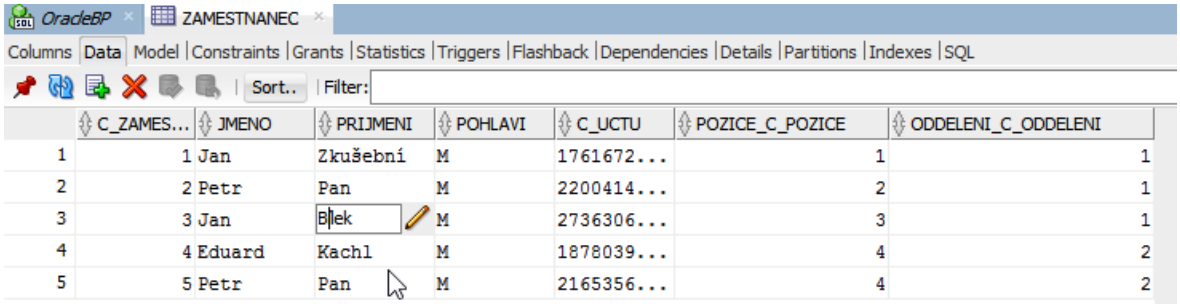
2.5.4 EDITACE DAT

Po kliknutí na konkrétní tabulku se zobrazí její struktura. Názvy sloupců a jejich datové typy. Pod hlavní kartou s názvem tabulky máme na výběr mimo jiné kartu data.



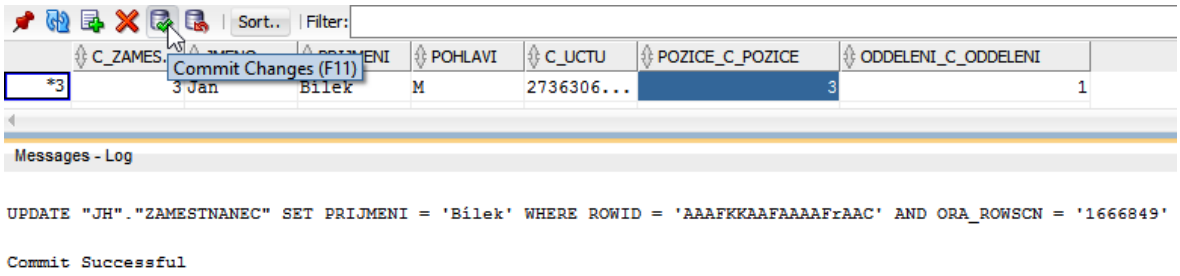
25 design view

V kartě data. Můžeme provádět DML příkazy bez nutnosti znalosti jazyka SQL. Například dvojklikem na vybranou buňku můžeme editovat její obsah.



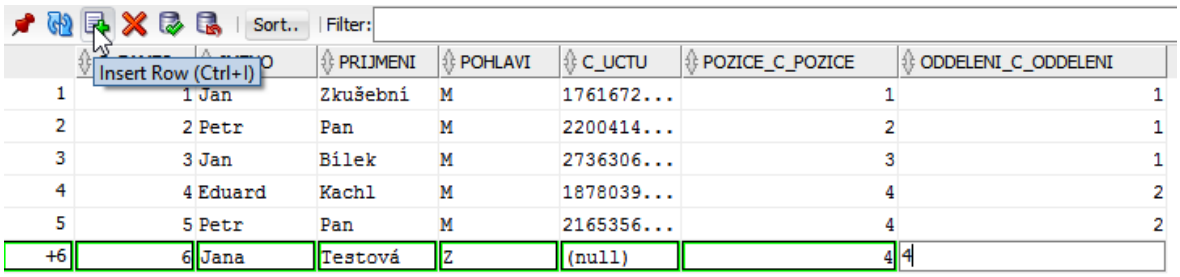
26 editace dat

Provedeme-li změnu, řádek tabulky, kde byla provedena změna, se označí hvězdičkou a máme na výběr tlačítko commit a tlačítko rollback. Zvolíme-li commit Oracle si převede naše rozhodnutí do SQL, což je vidět v Message logu.



27 commit

Tlačítko se zeleným symbolem plus umožňuje Insert řádku do tabulky a to jak nakonec, tak na specifickou pozici.



28 insert

Tlačítko červený kříž slouží k mazání řádků.

2.6 PL/SQL

PL/SQL (Procedural Language SQL) je procedurální jazyk pro programování na straně databáze.

2.6.1 ANONYMNÍ SQL BLOK

Oproti procedurám a funkcím je anonymní blok část kódu, která nemá svůj název. Používá se například k testování kódu.

Declare

Declare je volitelná část anonymního bloku. Zde je možné deklarovat proměnné a kurzory, se kterými se pracuje v těle bloku.

Begin – End

Tělo anonymního bloku je vymezeno slovy begin a end. Za end následuje středník a zpětné lomítko. Zde se nachází samotné příkazy programu spolu s řídícími strukturami.

Exception

Exception je další nepovinná část anonymního bloku. Zde se definuje kód, který má být spuštěn dojde-li v těle bloku k výjimce.

Příklad:

Nejjednodušší anonymní blok může vypadat například takto:

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
/
```

2.6.2 DATOVÉ TYPY

Oracle obsahuje nesčetné množství datových typů. Jejich výčet můžeme naleznout na stránkách Oraclu.

Vybrané datové typy:

NUMBER – Datový typ, který umožňuje ukládání jak celých čísel, tak i čísel s plovoucí řádovou čárkou. Může obsahovat kladná, nebo záporná čísla v rozsahu

od 1 x 10^{-130} do 9.99...9 x 10^{125} s až 38-mi číslicemi.

název_sloupce NUMBER (precision, scale)

parametr precision určuje, kolik číslic se může nacházet vlevo od desetinné čárky.

Parametr scale určuje, kolik číslic se může nacházet vpravo od desetinné čárky.

Příklad:

NUMBER(4,2) umožňuje uložení čísla 1234,44.

NVARCHAR2 – Datový typ pro ukládán řetězců. Jeho maximální délka je 4000 bytů, neboli 4000 znaků. Písmeno N značí, že využívá Unicode. Unicode vznikl, jako snaha sjednotit kódování všech znaků všech jazyků do jedné kódové sady.

DATE – Date umožňuje uložit datum a čas s vteřinovou přesností v rozsahu

1.ledna 4712 př.n.l. až do 31.prosince 9999 našeho letopočtu.

%ROWTYPE – Datový typ jednoho záznamu, který reprezentuje řádek v tabulce. Umožňuje uložení celého řádku dat selektovaného z tabulky, nebo vytaženého z kurzoru.

2.6.3 PODMÍNKY

Podmínky stejně jako v jiných programovacích jazycích slouží k větvení programu. Příklady znázorňují možnosti zápisu.

Příklad 1:

```
IF podmínka THEN
   {...příkazy k provedení, je-li podmínka TRUE...}
END IF;
```

Příklad 2 :

```
IF podmínka THEN
     {...příkazy k provedení, je-li podmínka
TRUE...}
ELSE
   {...příkazy, které se provedou, je-li podmínka
FALSE...}
END IF;
```

Příklad 3:

```
IF podmínka1 THEN
     {...příkazy k provedení, je-li podmínka
TRUE...}
ELSIF podmínka2 THEN
   {...příkazy, které se provedou, je-li podmínka1
FALSE a druhá podmínka TRUE...}
ELSE
   {...příkazy, které se provedou, jsou-li obě
podmínky FALSE...}
END IF;
```

2.6.4 CYKLY

V Oracle máme na výběr 3 druhy cyklů, a to LOOP, FOR LOOP a WHILE LOOP. Cykly slouží, k opakování jednoho, či série příkazu pokud je splněna podmínka.

- LOOP je cyklem, u kterého je nutné definovat slovem EXIT, kdy má skončit, jinak bude smyčka nekonečná. Tento cyklus se dá využít jako repeat until.

Příklad:

```
LOOP
   příkazy;
   EXIT WHEN podmínka;
END LOOP;
```

Zde si musíme být jisti, že vyhodnocení podmínky skončí jako TRUE, alespoň jednou v průběhu cyklem.

- FOR LOOP je cyklus s pevným počtem opakování. Zde se inkrementuje čítač od nižšího čísla, do chvíle kdy dojde do vyššího čísla. Pokud uvedeme klíčové slovo tak probíhá dekrementace.

Příklad:

```
FOR čítač IN [REVERSE] nižší_číslo..vyšší_číslo
LOOP
     příkazy;
END LOOP;
```

- WHEN LOOP je cyklus s podmínkou na začátku. Příkazy se tedy provedou, pouze pokud je podmínka vyhodnocena jako TRUE

Příklad:

```
WHILE podmínka
LOOP
   příkazy;
END LOOP;
```

2.6.5 KURZORY

Kurzor je vlastně speciálním typem cyklu. Využívá se v případě, pracujeme-li s RS obsahujícím více než jeden řádek. Kurzor je nutno nejprve deklarovat, poté otevřít. Samotná práce s kurzorem potom probíhá tak, že se postupně příkazem FETCH předá první řádek do proměnné, se kterou se provedou další operace. Tento proces se opakuje,

dokud operace neproběhnou pro všechny řádky. Po ukončení práce s kurzorem bychom ho měli zavřít příkazem CLOSE.

Příklad:

```
        DECLARE
        Cursor c1 is SELECT prijmeni FROM zamestnanec;
        Vprijmeni NVARCHAR(50);
BEGIN
        Open c1;
        LOOP
        FETCH c1 into Vprijmeni;
        EXIT WHEN c1%notfound;
        DBMS_OUTPUT.OUTPUT_PUT_LUNE(Vprijmeni);
        END LOOP;
        CLOSE c1;
END;
```

Tento program vybere všechny příjmení z tabulky zaměstnanec a poté je postupně vypíše na konzoli.

2.6.6 VÝJIMKY

Výjimky jsou mechanismem, který se používá k ošetření chyb, které mohou nastat při běhu programu. V Oraclu je možné pracovat s předdefinovanými výjimkami, je ale možné definovat výjimky vlastní.

Některé vybrané výjimky:

- zero_divide – vznikne, pokoušíme-li se dělit nulou.
- invalid_cursor – vznikne, pokoušíme-li se pracovat s kurzorem, který není deklarován, nebo otevřen.
- cursor_already_open – vznikne, pokoušíme-li se otevřít kurzor, který již byl otevřen.
- no_data_found – pokoušíme-li se přiřadit data do proměnné například příkazem `SELECT INTO název_proměnné FROM název_tabulky;`

  a příkaz nevrátí žádná data.

- Too_many_rows – také vznikne, pokoušíme-li se přiřadit do proměnné data z příkazu SELECT. V tomto případě výjimka znamená že select vrátil více řádků, než jeden. V takovém případě je nutné použít kurzor.
- Invalid_number – chyba nastane například, pokoušíme-li se do sloupce datového typu number vložit ne numerickou hodnotu (‘abcd’).

Uživatelem definované výjimky

Tyto výjimky potřebujeme deklarovat, vyvolat a ošetřit. Výjimku můžeme vyvolat kdekoliv v programu, ať už v těle (mezi begin-end) nebo v části Exception (víceúrovňové výjimky). Výjimku vyvoláme klíčovým slovem Raise.

Příklad programu s výjimkou:

```
Declare
vyjimka exception;
Begin
     Raise vyjimka;
Exception
     WHEN výjimka THEN
     DBMS_OUTPUT.PUT_LINE(‘Nastala výjimka.’)
End;
/
```

2.6.7 KOLEKCE

Kolekce jsou speciálním typem proměnných, kterou tvoří skupina proměnných stejného datového typu. Každá položka zde má svůj index. Kolekce jsou obdobou jednorozměrných dynamických polí v jiných programovacích jazycích. Práce s kolekcemi je mnohem rychlejší, než modifikování obsahu tabulek pomocí SQL. Kolekce používají jako kolekce datových typů specifických k aplikaci, jako například kolekce záznamů z tabulky. Kolekci můžeme klasifikovat jako dense, což znamená, že všechny indexy nejnižším a nejvyšším jsou definovány, nebo jako sparse, což znamená, že v kolekci existují indexy, na kterých element není definován (Steven Feuerstein, 2016).

V Oraclu existují tři typy kolekcí:

- Associative Arrays

```
TYPE název_kolekce IS TABLE OF typ_položky INDEXED
BY typ_indexu;
```

- Nested Tables

```
TYPE název_kolekce IS TABLE OF typ_položky;
```

- Varray (varrying arrays)

  Zde můžeme parametrem limit specifikovat maximální možný počet položek v kolekci. Varrays jsou vždy plně naplněny, neboli tvoří pouze kolekce dense.

```
TYPE název_kolekce IS VARRAY (limit) OF
typ_položky;
```

Příklad proměnné kolekce na úrovni schématu:

```
CREATE TYPE nastup_t IS TABLE OF DATE;
```

Vytvoří kolekci typu Date. Proměnou potom můžeme deklarovat s názvem schématu, nebo bez něj, a to za předpokladu, jsme-li připojeni ke schématu, které je vlastníkem tohoto typu. Vlastníkem objektu je vždy schéma, které ho vytvoří.

```
DECLARE
     data_nastupu nastup_t;
--
DECLARE
     data_nastupu JH.nastup_t;
```

Příklad proměnné kolekce na úrovni balíku:

```
CREATE PACKAGE moje_typy IS
     TYPE OF retezce_t IS TABLE OF VARCHAR2(100);
END moje_typy;
```

Vytvoří kolekci řetězců typu Varchar2. Proměnou potom můžeme deklarovat s názvem balíku.

```
DECLARE
     retezce moje_typy.retezce_t;
```

Metody pro práci s kolekcí

Na metody se odkazuje tečkovou notací (název_proměnné.název_metody). Metody jsou buďto návratové, nebo metody měnící obsah kolekce.

COUNT: Vrací počet řádků/elementů v kolekci.

EXISTS : Vrací TRUE pokud je řádek na specifickém indexu definován (název_proměnné.EXISTS(index)).

FIRST/LAST: Vrací nejnižší, nebo nejvyšší hodnotu definovaného indexu v kolekci.

NEXT/PRIOR: Vrací element definovaný před, nebo za specifikovaným elementem.

LIMIT: Vrací maximální dovolený počet elementů ve Varray.

DELETE : Smaže jeden, nebo více řádek v kolekci. Nelze použít ve Varray.

EXTEND: přidá řádek na konec Nested Table nebo Vararray.

TRIM: Je jediným způsobem odstranění řádky, či více řádek z konce Varray. Tuto metodu lze použít i u Nested Table.

Příklad procházení dense kolekce:

```
BEGIN
     FOR radek_index IN 1 .. kolekce.COUNT
     LOOP
          {operace s kolekce(radek_index)}
     END LOOP;
END;
/
```

Tento způsob procházení kolekce je vhodný pouze, pokud všechny indexy mezi jedna až nejvyšším mají definovaný element. Pokud by v kolekci existoval jeden nedefinovaný index, program by končil chybou no_data_found.

Příklad procházení sparse kolekce:

```
DECLARE
     radek_index PLS_INTEGER := kolekce.FIRST;
BEGIN
     LOOP
     EXIT WHEN radek_index IS NULL;
     radek_index:= kolekce.NEXT(radek_index);
     {operace s kolekce(radek_index)}
```

```
        END LOOP;
    END;
    /
```

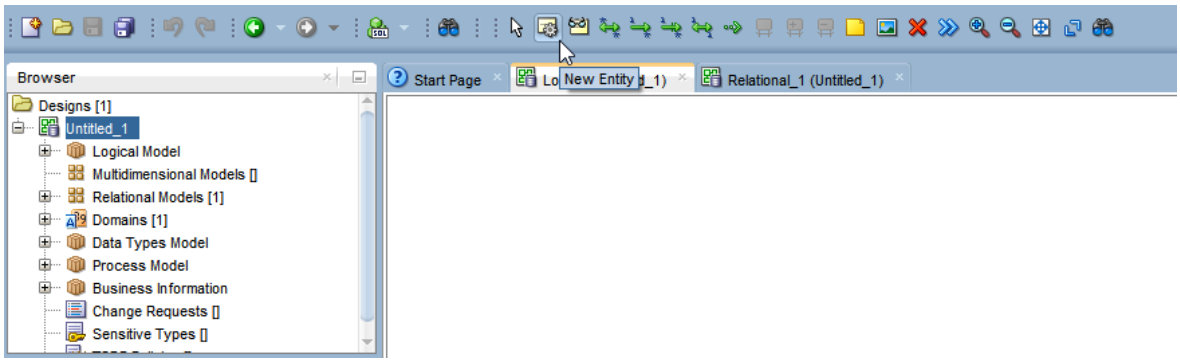
V programu je deklarován počáteční index kolekce. Cyklem loop se prochází následující element pomocí metody NEXT a cyklus se ukončí, když následující index neobsahuje žádný další element. Tímto způsobem se vyhneme tzv. „hluchým místům“ v kolekci a je tedy vhodný pro procházení kolekcí typu sparse. Podobného výsledku je možné dosáhnou použitím funkcí LAST a PRIOR. Kolekce se tak bude procházet od konce.

3 POSTUP K ULOŽENÍ PŘIPRAVENÉ DATABÁZE DO VYBRANÉHO DATABÁZOVÉHO SYSTÉMU

3.1 VYTVOŘENÍ MODELU

Příkladem bude datový model, který bude uchovávat informace o tom, kolik času tráví zaměstnanec na které činnosti. Činností může být buďto nějaký projekt, na kterém pracuje, ale i například dovolená, či nemoc. Dále bude možné zjistit, v jakém oddělení zaměstnanec pracuje, na jaké pozici a dle pozice jakou má hodinovou mzdu.

Přepneme se do data modeleru a tlačítkem New Entity vytvoříme požadované entity.

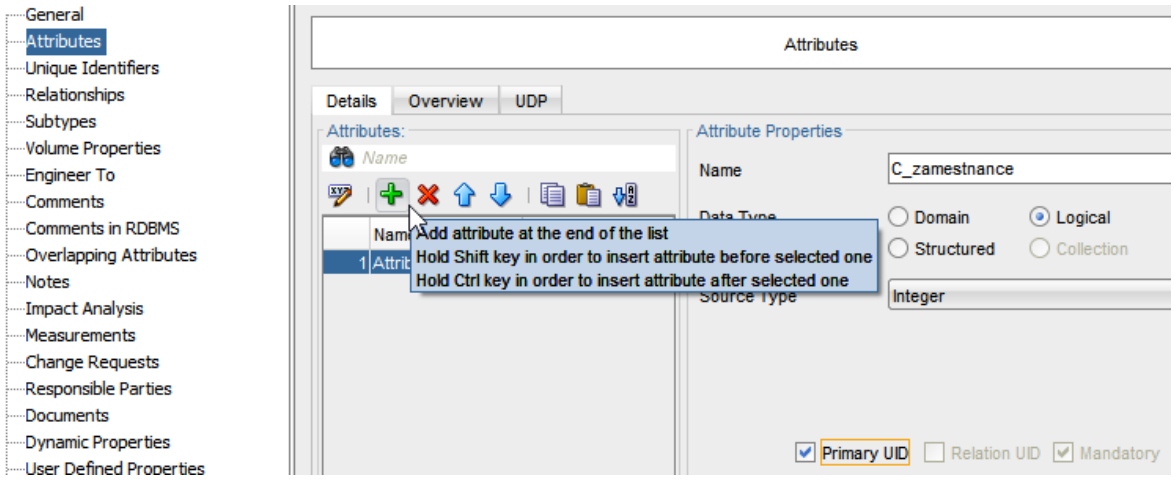


29 new Entity

Přepneme se do data modeleru a v logickém modelu tlačítkem New Entity vytvoříme požadované entity.

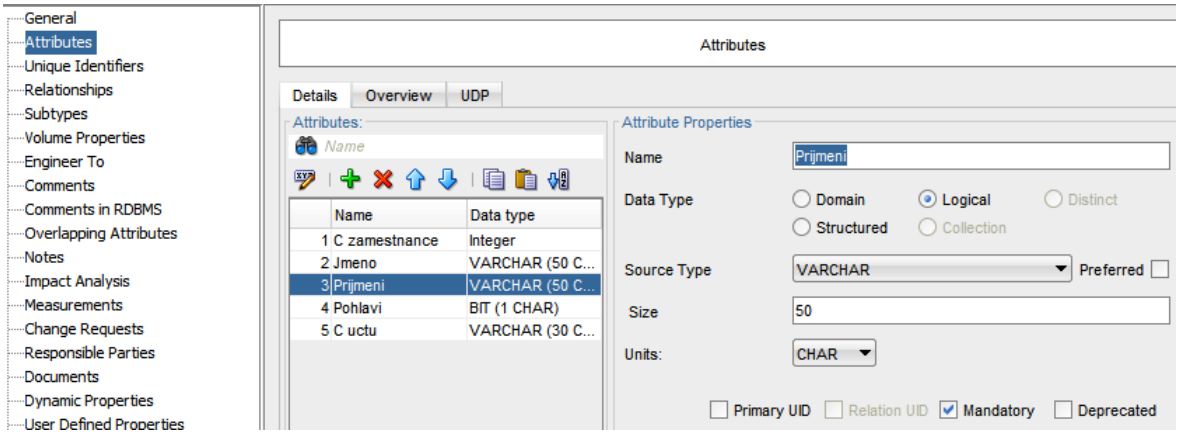
30 název entity

U každé vyplníme jméno.



31 vložení atributu entity

Přidáme atributy.

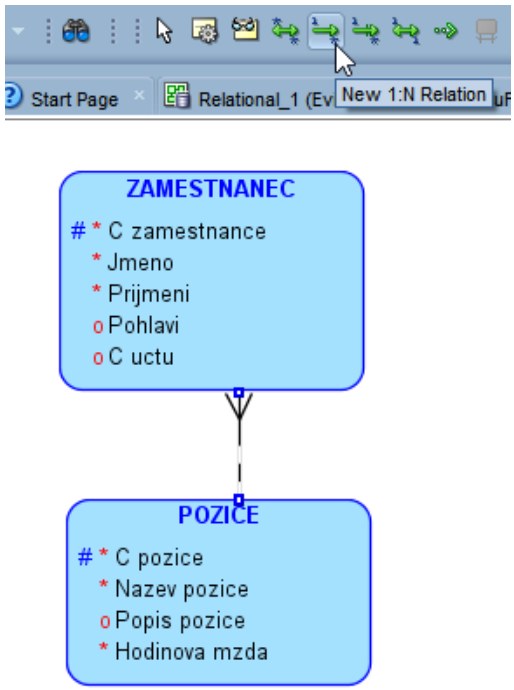


32 nastavení atributu entity

Každému atributu můžeme nastavit:

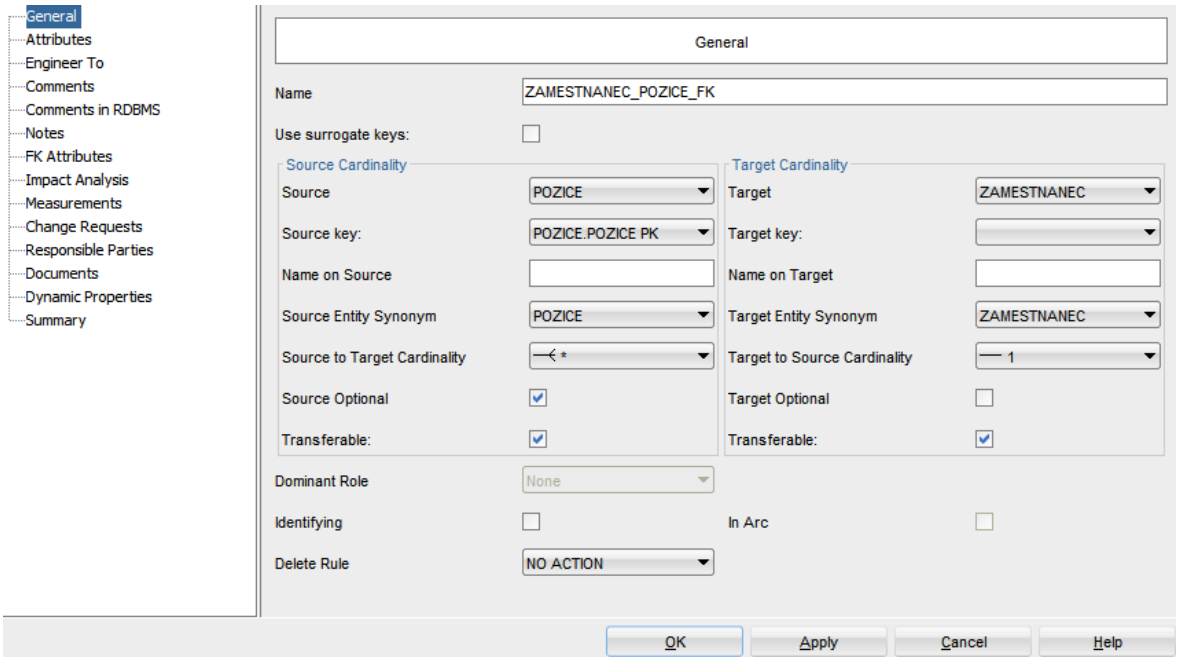
- datový typ
- je-li součástí primárního klíče (Primary UID)
- je-li jeho vyplnění povinné (Mandatory)

Předpokládáme, že ne jedné pozici může pracovat více zaměstnanců, volíme tedy relaci 1:N



33 výběr relace

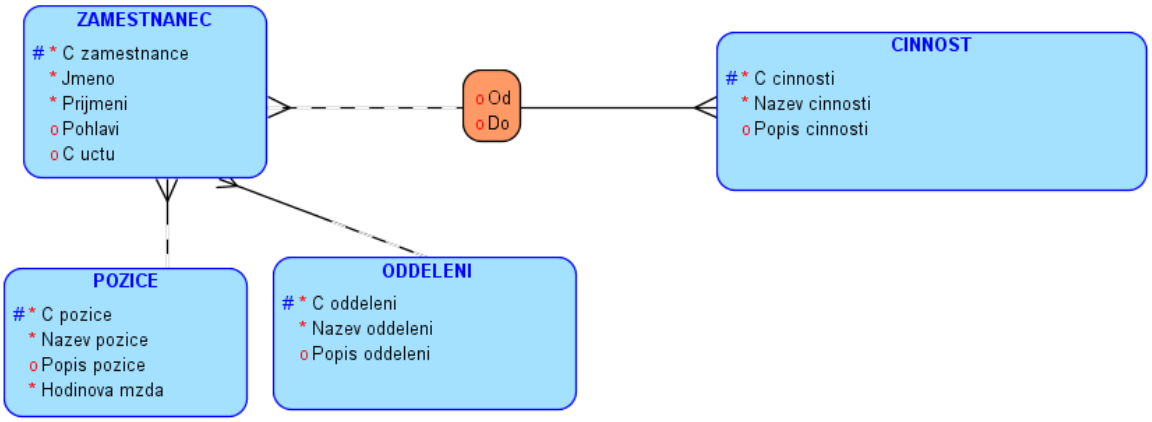
Vyplníme název relace.



34 nastavení relace

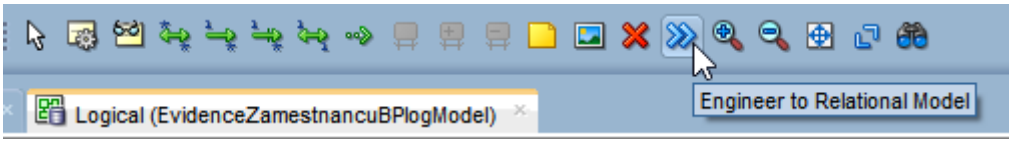
Mezi entitou zaměstnanec a činnost očekáváme vztah M:N, neboť jeden zaměstnanec může pracovat na více projektech a zároveň jeden projekt může být zpracováván více zaměstnanci.

Obdobným způsobem pokračujeme s entitami pozice a oddělení. Logický model by tedy měl vypadat následovně.



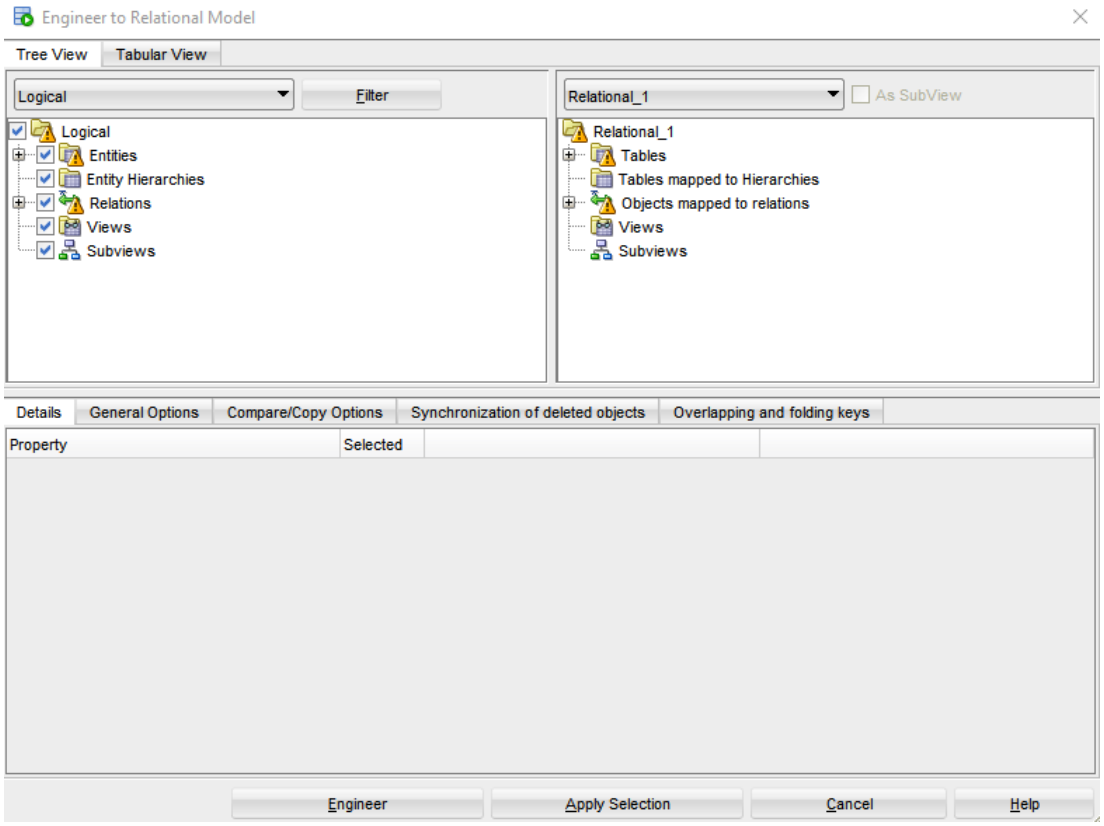
35 logický model

Tlačítkem Engineer to Relational Model převedeme logický model do relačního.



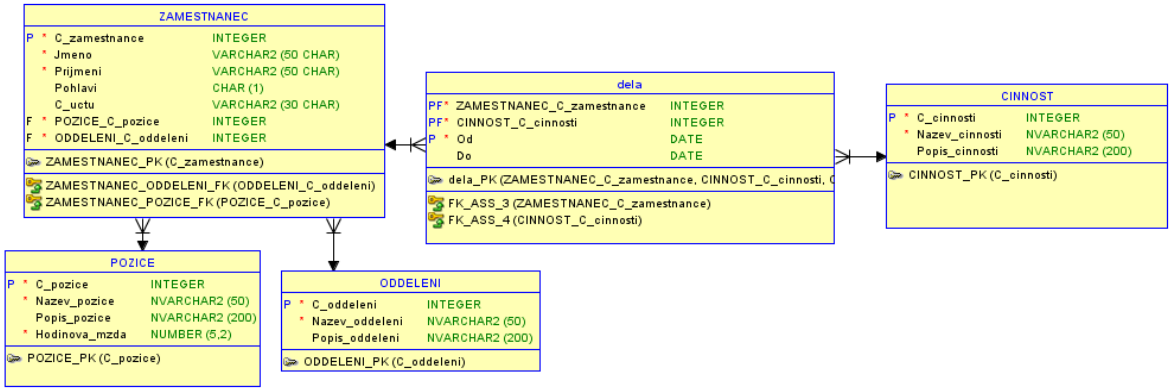
36 převedení do relačního modelu

Objeví se nám okno s možnostmi výběru entit, které chceme v relačním modelu použít. Necháme vše, jak je a zvolíme Engineer a získáme relační model.



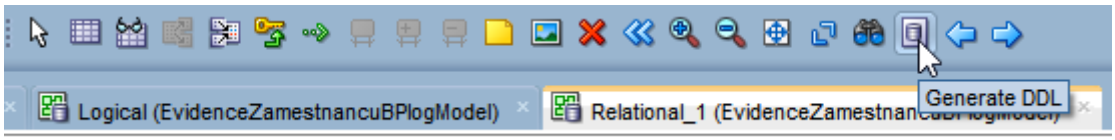
37 parametry převedení do relačního modelu

V takto vzniklém relačním modelu můžeme dělat další úpravy. V našem příkladu je entitě dělá rozšířit primární klíč, jinak by každý zaměstnanec mohl vykonávat pouze jednu činnost po celou dobu své existence. Dvojklikem na entitu „dela“ tedy přidáme atribut Od do primárního klíče.



38 relační model

Tlačítkem Generate DDL vygenerujeme z relačního modelu DDL skript obsahující všechny příkazy vytvářející tabulky a jejich omezení (constraints),



39 generování DDL

3.2 NASAZENÍ MODELU DO ORACLE 11G EXPRESS

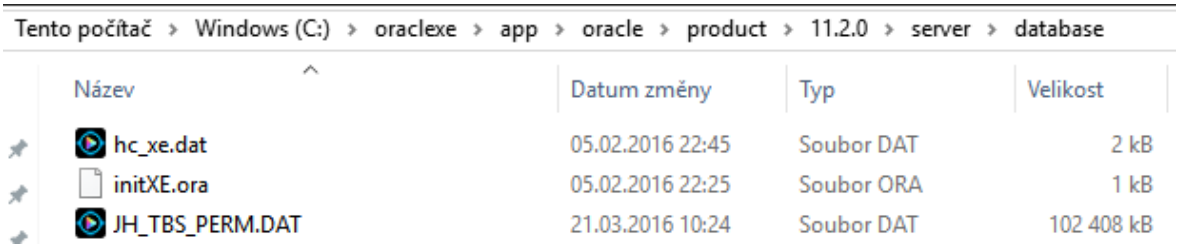
3.2.1 VYTVOŘENÍ TABULKOVÉHO PROSTORU

Příkaz CREATE TABLESPACE se používá, k alokování prostoru kam budou objekty daného schéma ukládány.

- PERMANENT TABLESPACE – obsahuje trvalé objekty, které jsou uloženy v data souborech.

```
CREATE TABLESPACE JH_tbs_perm
DATAFILE 'JH_tbs_perm.dat'
SIZE 100M;
```

Tento příkaz vytvoří nový tabulkový prostor se jménem „JH_tbs_perm“, který bude ukládat data do souboru „JH_tbs_perm.dat“. Soubor má nastaveno omezení na 100MB.

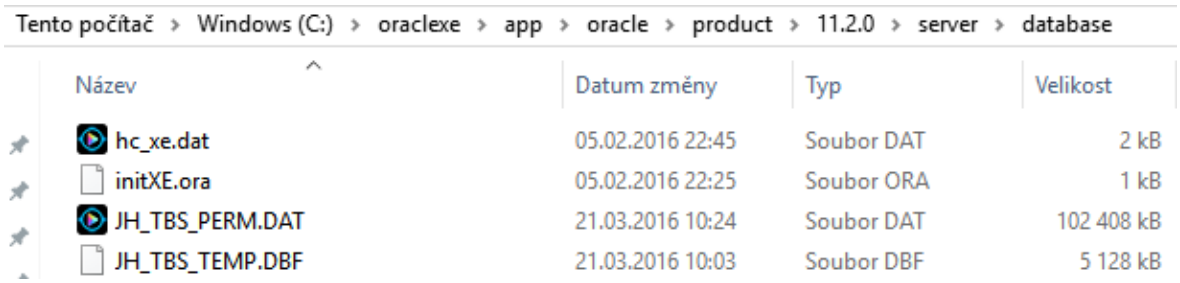


40 permanentní tabulkový prostor

- TEMPORARY TABLESPACE – obsahuje objekty, které jsou uloženy v dočasných souborech a existují pouze během session.

```
CREATE TEMPORARY TABLESPACE JH_tbs_temp
TEMPFILE 'JH_tbs_temp.dbf'
     SIZE 5M
          AUTOEXTEND ON;
```

Příkaz vytvoří dočasný tabulkový prostor s názvem „JH_tbs_temp“ který má jeden temp soubor s názvem „JH_tbs_temp.dbf”. Výchozí velikost je natavena na 5MB, je ale rovněž povoleno automatické rozšíření.



41 dočasný tabulkový prostor

3.2.2 VYTVOŘENÍ SCHÉMA

Aby bylo možné vytvořit schéma, je nutné vytvořit tabulkové prostory a nového uživatele.

```
CREATE USER JH
     IDENTIFIED BY 123456
     DEFAULT TABLESPACE JH_tbs_perm
     TEMPORARY TABLESPACE JH_tbs_temp.dbf
QUOTA 50M on JH_tbs_perm;
```

Příkaz vytvoří uživatele “JH”, jehož heslo je “123456”, který bude využívat dříve vytvořené tabulkové prostory, nastavení kvóty je zde nutné, jinak uživatel nebude moct vytvářet objekty v permanentním tabulkovém prostoru.

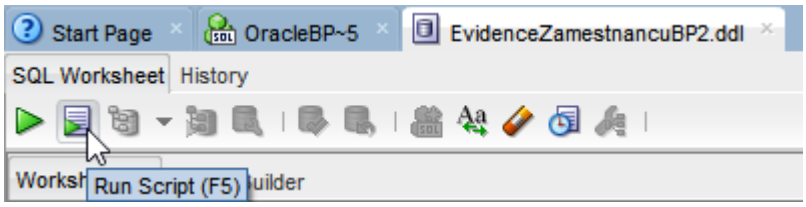
Přiřazení systémových privilegií novému uživateli.

```
GRANT create session TO JH;
GRANT create table TO JH;
GRANT create view TO JH;
GRANT create any trigger TO JH;
GRANT create any procedure TO JH;
```

Nyní máme k dispozici uživatele, se kterým je možné dále pracovat. Protože do teď jsme tvořili pod uživatelem SYSTEM, odhlásíme se a znovu přihlásíme jako nově vytvořený uživatel.

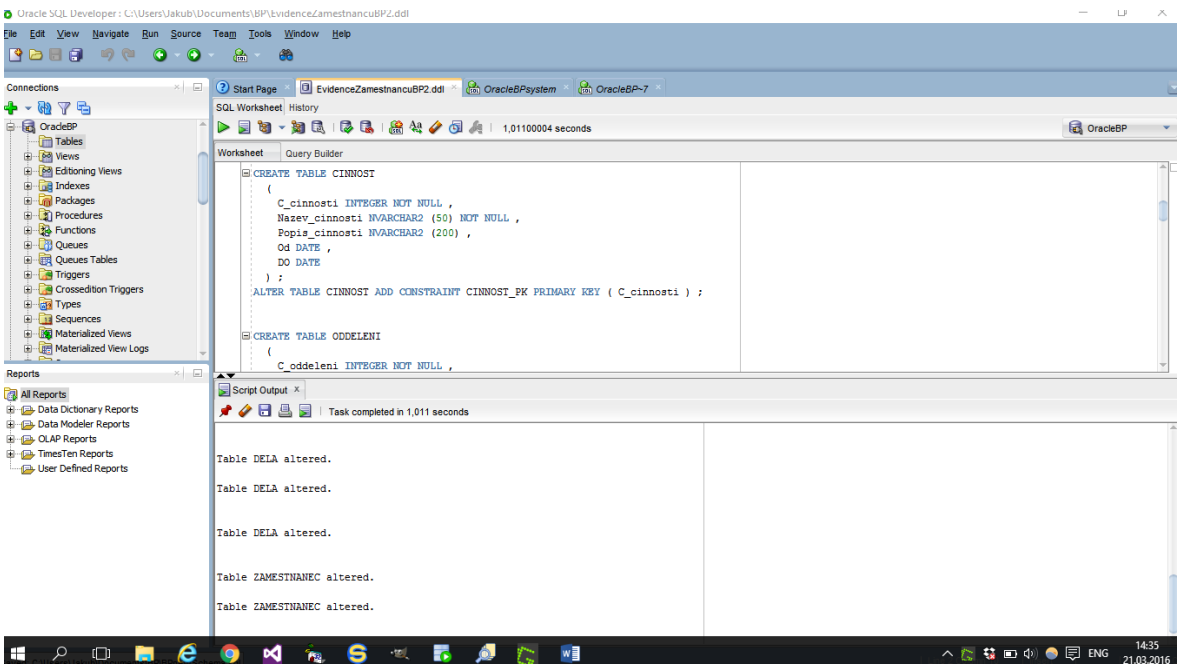
3.2.3 NAHRÁNÍ DDL

Jsme-li připojeni k databázi a máme-li připraven DDL skript, otevřeme si skript v SQL developeru a zvolímu Run script, případně klávesu F5.



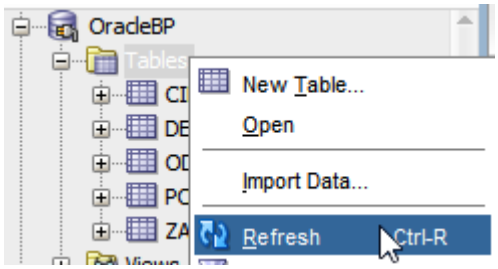
42 spuštění skriptu

Nyní ve script outputu vidíme, že spuštění scriptu proběhlo bez jakýchkoliv chyb.



43 script output

Zvolíme-li refresh nad stromovou strukturou tabulek, zobrazí se nám nově vytvořené tabulky.



44 refresh

3.3 NAPLNĚNÍ TESTOVACÍMI DATY

Vytvoříme si skript, který naplní všechny tabulky smyšlenými daty.

například:

```
INSERT ALL
  INTO POZICE VALUES (1, 'Ředitel','Má na starosti
chod celého podniku.',300)
  INTO POZICE VALUES (2,'Manažer oddělení','Má na
starosti chod svého oddělení',250)
  INTO POZICE VALUES (3, 'Vedoucí','Má na starosti
tým pracovník?',150)
  INTO POZICE VALUES (4, 'Řadový pracovník','Plní
zadaný plán',100)
SELECT 1 FROM DUAL;
```

Tímto příkazem dokážeme vložit více záznamů v jediném příkazu.

3.4 ULOŽENÉ PROCEDURY (SP)

Pro názornou ukázku využití SP mějme tyto dvě procedury. Procedury mají za úkol operaci, která je známá jako "píchnutí v píchačce". Jedna slouží pro start činnosti, druhá pro její ukončení.

- SP_ZACATEK_PRACE nejprve ověří, zda pracovník již nemá nějakou neukončenou činnost. Pokud nemá, vytvoří záznam v tabulce DELA a informuje o tom v konzoli. Pokud má, vypíše hlášku, že pracovník má neukončenou činnost, kterou je nutno nejprve ukončit

```
CREATE OR REPLACE PROCEDURE SP_ZACATEK_PRACE(
c_zamestnance INTEGER,
c_cinnosti INTEGER
) AS
     c_exists number;
BEGIN
     SELECT count(*) into c_exists
     FROM DELA
     WHERE ZAMESTNANEC_C_ZAMESTNANCE =
     c_zamestnance AND DO IS null;
     IF c_exists <> 0 THEN
     DBMS_OUTPUT.PUT_LINE('Zaměstnanec již
     pracuje!');
     ELSE
```

```
     INSERT INTO DELA
     VALUES(c_zamestnance,c_cinnosti,sysdate,null);
    COMMIT;
     DBMS_OUTPUT.PUT_LINE('Zaměstnanec začíná
     pracovat.') ;
   END IF;
END SP_ZACATEK_PRACE;
/
```

- SP_KONEC_PRACE ověří, zda má pracovník skutečně neukončenou nějakou činnost. Pokud ne, informuje o tom v konzoli. Pokud ano, ukončí ji a informuje o ukončení v konzoli.

```
CREATE OR REPLACE PROCEDURE
SP_KONEC_PRACE(c_zamestnance
INTEGER) AS
     c_exists number;
BEGIN
     SELECT count(*) into c_exists FROM dela WHERE
     ZAMESTNANEC_C_ZAMESTNANCE = c_zamestnance AND
     DO IS null;
IF c_exists = 0 THEN
     DBMS_OUTPUT.PUT_LINE('Zamestnanec nema
     rozpracovanou činnost, která by se dala
     ukončit');
  ELSE
    UPDATE DELA
    SET DO = SYSDATE
    WHERE ZAMESTNANEC_C_ZAMESTNANCE = c_zamestnance
AND DO IS null;
    DBMS_OUTPUT.put_line('Činnost byla ukončena.');
  END IF;
END SP_KONEC_PRACE;
/
```

4 PŘÍKLADY POUŽITÍ JAZYKA SQL V RÁMCI NAVRŽENÉ DATABÁZE

Na níže uvedených příkladech jsou vidět některé možnosti použití jazyka v rámci vytvořené databáze. Těmito příkazy si zároveň ověříme správnost modelu.

```
SET SERVEROUTPUT ON;
```

Nastavení výstupu na konzoli.

```
INSERT INTO ZAMESTNANEC VALUES
(6,'Jan','Dobrota','192158071/0300', 4, 2);
```

Přidání nového zaměstnance.

```
DELETE FROM ZAMESTNANEC WHERE C_ZAMESTNANCE = 6;
```

Smazání nově vytvořeného zaměstnance.

```
DELETE FROM ZAMESTNANEC WHERE C_ZAMESTNANCE = 5;
```

Pokus o smazání zaměstnance, který končí chybou, neboť je vázán omezením cizího klíče s tabulkou DELA.

```
INSERT INTO CINNOST VALUES (5,'Projekt2','Projekt
dva se týká..');
```

Vytvoření nové činnosti.

```
DELETE FROM CINNOST WHERE C_CINNOSTI = 1;
```

Pokus o smazání činnosti, který končí chybou, neboť je vázána omezením cizího klíče s tabulkou ZAMESTNANEC.

```
INSERT INTO POZICE VALUES (5,'Pracovník úklidu',
'Zajišťuje čistotu a pořádek prostor.');
```

Přidání nové pracovní pozice.

```
DELETE FROM POZICE WHERE C_POZICE = 4;
```

Pokus o smazání pozice, který končí chybou, neboť je vázána omezením cizího klíče s tabulkou ZAMESTNANEC.

```
INSERT INTO ODDELENI VALUES (3,'Úklid','Zajišťuje
čistotu a pořádek prostor.');
```

Přidání nového oddělení.

```
DELETE FROM ODDELENI WHERE C_ODDELENI = 3;
```

Pokus o smazání oddělení, který končí chybou, neboť v tomto oddělení nám již pracují zaměstnanci.

```
EXEC SP_ZACATEK_PRACE(2,1);
```

Začátek činnosti zaměstnance s identifikátorem 2 na činnosti s identifikátorem 1.

```
EXEC SP_KONEC_PRACE(2);
```

Konec činnosti zaměstnance s identifikátorem 2.

```
DELETE FROM DELA
WHERE ZAMESTNANEC_C_ZAMESTNANCE IN (
  SELECT ZAMESTNANEC.C_ZAMESTNANCE
  FROM ZAMESTNANEC
  WHERE  ZAMESTNANEC.PRIJMENI LIKE 'Pan');
```

Smazání všech dat o činnostech pracovníka s příjmením Pan.

4.1 ROZSÁHLEJŠÍ DOTAZY

Počet (dle pracovní pozice a oddělení) zaměstnanců pracovníků, kteří pracovali v daný měsíc přesčas

Tabulka DELA obsahuje testovací data převážně pro měsíc leden. Funkčnost lze tedy otestovat vložením parametru 1. Vnitřní dotaz seskupuje odpracované hodiny do dne, pro případ, že by pracovník měl více než jednu činnost za den. Předpokládá se, že pracovník by měl mít za den 8,5 hodiny. Tato hodnota se zde odečítá. Dotaz tedy již zobrazuje přesčasové hodiny. Vnější dotaz sumarizuje hodiny po dnech do jedné hodnoty, a to sumy přesčasů za pracovníka a filtruje pouze pracovníky, u kterých je přesčas větší než 0. a konečně CTE které nám z předešlého výsledku vrací počet dle pracovní pozice a oddělení.

```
with Prescasy as
(SELECT NAZEV_ODDELENI,NAZEV_POZICE,
ZAMESTNANEC_C_ZAMESTNANCE,SUM(Hodin_Za_Den)
AS Suma_Prescasy
FROM
```

```
  (SELECT
  ODDELENI.NAZEV_ODDELENI,POZICE.NAZEV_POZICE,
  ZAMESTNANEC_C_ZAMESTNANCE,extract(day from od) as
  Den,SUM((DO - OD)* 24 )-8.5 AS Hodin_Za_Den
       FROM DELA, ZAMESTNANEC, ODDELENI,POZICE
       WHERE DELA.ZAMESTNANEC_C_ZAMESTNANCE =
  ZAMESTNANEC.C_ZAMESTNANCE AND
ODDELENI.C_ODDELENI=ZAMESTNANEC.ODDELENI_C_ODDELENI
AND
  POZICE.C_pozice = ZAMESTNANEC.Pozice_c_pozice AND
  CINNOST_C_CINNOSTI not in (2,3) AND extract(year
  from OD) = extract(year from sysdate) AND
  extract(month from OD) = &C_mesice
  group by ODDELENI.NAZEV_ODDELENI,
  POZICE.NAZEV_POZICE, ZAMESTNANEC_C_ZAMESTNANCE,
  extract(day  from od))
  group by NAZEV_ODDELENI, NAZEV_POZICE,
  ZAMESTNANEC_C_ZAMESTNANCE
  having SUM(Hodin_Za_Den) > 0
  )
  select
  Prescasy.NAZEV_ODDELENI,Prescasy.NAZEV_POZICE,count
  (Prescasy.ZAMESTNANEC_C_ZAMESTNANCE )as
  Poc_Zam_S_prescasem_v_mesici
  from Prescasy
  group by Prescasy.NAZEV_ODDELENI,
  Prescasy.NAZEV_POZICE
  ;
```

Výsledek nad testovacími daty

	NAZEV_ODDELENI	NAZEV_POZICE	POC_ZAM_S_PRESCASEM_V_MESICI
1	Vedení	Vedoucí	1

45 výsledek dotazu 1 nad testovacími daty

Množství odpracovaných hodin za den v daném měsíci za pracovníka

Tento dotaz je prakticky pod-dotazem v předchozím příkladu. S tím rozdílem, že se zde neodečítá povinný počet hodin za den (-8.5) a na konci je přidáno řazení pro větší přehlednost výstupu. Dotaz tedy vrací počet odpracovaných hodin za pracovníka po dnech ve měsíci zadaném parametrem, a to v letošním roce. Dotaz zobrazuje informaci o názvu oddělení, ve kterém pracovník pracuje a pracovní pozici.

```
  SELECT
  ODDELENI.NAZEV_ODDELENI,POZICE.NAZEV_POZICE,ZAMESTN
  ANEC.PRIJMENI, ZAMESTNANEC.JMENO,extract(day from
  od) as Den,SUM((DO - OD)* 24 ) AS Hodin_Za_Den
       FROM DELA, ZAMESTNANEC, ODDELENI,POZICE
       WHERE DELA.ZAMESTNANEC_C_ZAMESTNANCE =
       ZAMESTNANEC.C_ZAMESTNANCE AND
ODDELENI.C_ODDELENI=ZAMESTNANEC.ODDELENI_C_ODDELENI
AND
  POZICE.C_pozice = ZAMESTNANEC.Pozice_c_pozice AND
  CINNOST_C_CINNOSTI not in (2,3) AND
  extract(year from OD) = extract(year from sysdate)
  AND extract(month from OD) = &C_mesice
  group by ODDELENI.NAZEV_ODDELENI,
  POZICE.NAZEV_POZICE,ZAMESTNANEC.PRIJMENI,
  ZAMESTNANEC.JMENO, extract(day  from od)
  ORDER BY
  ODDELENI.NAZEV_ODDELENI,POZICE.NAZEV_POZICE,ZAMESTN
  ANEC.PRIJMENI, ZAMESTNANEC.JMENO,extract(day from
  od);
```

Výsledek nad testovacími daty

	NAZEV_ODDELENI	NAZEV_POZICE	PRIJMENI	JMENO	DEN	HODIN_ZA_DEN
1	Vedení	Vedoucí	Bílek	Jan	4	9,5
2	Vedení	Vedoucí	Bílek	Jan	5	9,5
3	Vedení	Vedoucí	Bílek	Jan	6	8,5
4	Vedení	Vedoucí	Bílek	Jan	7	8,5
5	Vedení	Vedoucí	Bílek	Jan	8	8,5
6	Vedení	Vedoucí	Bílek	Jan	11	8,5
7	Vedení	Vedoucí	Bílek	Jan	12	8,5
8	Vedení	Vedoucí	Bílek	Jan	13	8,5
9	Vedení	Vedoucí	Bílek	Jan	14	8,5
10	Vedení	Vedoucí	Bílek	Jan	15	8,5
11	Vedení	Vedoucí	Bílek	Jan	18	8,5
12	Vedení	Vedoucí	Bílek	Jan	19	10,5
13	Vedení	Vedoucí	Bílek	Jan	20	11,5
14	Vedení	Vedoucí	Bílek	Jan	21	12
15	Vedení	Vedoucí	Bílek	Jan	22	11,5
16	Vedení	Vedoucí	Bílek	Jan	25	11,5
17	Vedení	Vedoucí	Bílek	Jan	26	10,5
18	Vedení	Vedoucí	Bílek	Jan	27	9,5
19	Vedení	Vedoucí	Bílek	Jan	28	10,5
20	Vedení	Vedoucí	Bílek	Jan	29	7,5
21	Výroba	Řadový pracovník	Kachl	Eduard	11	8,5
22	Výroba	Řadový pracovník	Kachl	Eduard	12	8,5
23	Výroba	Řadový pracovník	Kachl	Eduard	13	8,5
24	Výroba	Řadový pracovník	Kachl	Eduard	14	8,5
25	Výroba	Řadový pracovník	Kachl	Eduard	15	8,5
26	Výroba	Řadový pracovník	Kachl	Eduard	18	8,5
27	Výroba	Řadový pracovník	Kachl	Eduard	19	8,5
28	Výroba	Řadový pracovník	Kachl	Eduard	20	8,5
29	Výroba	Řadový pracovník	Kachl	Eduard	21	8,5
30	Výroba	Řadový pracovník	Kachl	Eduard	22	8,5
31	Výroba	Řadový pracovník	Kachl	Eduard	25	8,5
32	Výroba	Řadový pracovník	Kachl	Eduard	26	8,5
33	Výroba	Řadový pracovník	Kachl	Eduard	27	8,5
34	Výroba	Řadový pracovník	Kachl	Eduard	28	8,5
35	Výroba	Řadový pracovník	Test	Petr	4	8,5
36	Výroba	Řadový pracovník	Test	Petr	5	8,5
37	Výroba	Řadový pracovník	Test	Petr	6	8,5
38	Výroba	Řadový pracovník	Test	Petr	7	8,5
39	Výroba	Řadový pracovník	Test	Petr	8	8,5
40	Výroba	Řadový pracovník	Test	Petr	11	8,5
41	Výroba	Řadový pracovník	Test	Petr	12	8,5
42	Výroba	Řadový pracovník	Test	Petr	13	8,5
43	Výroba	Řadový pracovník	Test	Petr	14	8,5
44	Výroba	Řadový pracovník	Test	Petr	15	8,5
45	Výroba	Řadový pracovník	Test	Petr	18	8,5
46	Výroba	Řadový pracovník	Test	Petr	19	8,5
47	Výroba	Řadový pracovník	Test	Petr	20	8,5
48	Výroba	Řadový pracovník	Test	Petr	21	8,5
49	Výroba	Řadový pracovník	Test	Petr	22	8,5
50	Výroba	Řadový pracovník	Test	Petr	25	8,5

46 výsledek dotazu 2 nad testovacími daty

ZÁVĚR

Databáze je termín, který mají lidé v povědomí, ale málokdo zná, pravou podstatu tohoto slova. Na databázových systémech jsou dnes založeny všechny aplikace, které si zakládají na efektivním zpracování informací. Využití lze nalézt v bankovnictví, zdravotnictví, energetice a vůbec všech odvětvích lidské činnosti. Potřebu uchovávat informace má člověk už od pradávna.

Při tvorbě jsem využil poznatků ze svých studijních a pracovních zkušeností. Tvorbou jsem také rozšířil své znalosti o mnoho dalšího. V teoretické části jsem poskytl základní přehled pojmů z oblasti relačních databází, a některé možnosti programovacího jazyka databáze. V praktické části Jsem poskytl návod, jak vytvořit datový model a uložit ho do vybraného databázového systému, a to včetně příkladů praktického využití jazyka SQL. Věřím, že tato práce může sloužit jako materiál pro výuku databázových systémů.

RESUMÉ

V první části práce jsem uvedl hlavní pojmy z relačních databází, popsal jednotlivé databázové objekty, spolu s jejich vztahy. Dále jsem popsal možnosti dotazovacího jazyka SQL spolu s prostředky zajištění integrity dat. Popsal jsem zde i odstraňování redudance v datech. Ve druhé části práce lze nalézt průvodce instalací a konfigurací databázového systému, spolu s popisem jednotlivých nástrojů. Na praktickém příkladu jsou zde představeny hlavní možnosti SQL Developeru spolu s PL/SQL. Ve třetí části je krok po kroku vysvětleno vytvoření datového modelu a jeho uložení do systému Oracle. Na praktickém příkladu zde nalezneme i využití uložených procedur. V poslední části práce ukazuji praktické příklady použití jazyka SQL na vytvořené databázi v několika jednoduchých SQL příkazech a pro názornost i ve dvou rozsáhlých SQL dotazech.

SUMMARY

In the first part of this thesis i introduced main terms from relation databases with various database objects and relationships between them. Then i described redundancy and its removing. The second part of this thesis is a manual how to install and configure the database system with description of particular tools. Here i show possibilities of SQL Developer software and PL/SQL with practical examples. In the third part is a step by step guide to creating data model as well as applying to Oracle system. Here you can also find practical examples of stored procedures. Last part shows practical examples of SQL language with some simple SQL commands and two comprehensive SQL queries.

SEZNAM LITERATURY

Complete Ranking, 2016. DB-Engines. [online]. [cit. 2016-04-05]. Dostupné z: http://db-engines.com/en/ranking

Database Documentation, 2005. Oracle Help Center. [online]. [cit. 2016-04-05]. Dostupné z: docs.oracle.com/en/database/

HERNANDEZ, Michael J. Návrh databází, 2006. 1. vyd. Praha: Grada. Profesionál. ISBN 80-247-0900-7.

LACKO, Ľuboslav , 2003. SQL: hotová řešení. Vyd. 1. Brno: Computer Press. K okamžitému použití (Computer Press). ISBN 80-7226-975-5.

PAVLOVSKÁ, Helena, 2011. Vztahy mezi objekty. Krokodýlovy databáze. [online]. [cit. 2016-04-06]. Dostupné z: http://krokodata.vse.cz/DM/Vztahy

Steven Feuerstein, 2016. Practically Perfect PL/SQL. Youtube. [online]. [cit. 2016-04-06]. Dostupné z: https://www.youtube.com/channel/UCpJpLMRm452kVcie3RpINPw/feed

ŠIMŮNEK, Milan, 1999. SQL: kompletní kapesní průvodce. Vyd. 1. Praha: Grada. ISBN 80-7169-692-7.

SEZNAM OBRÁZKŮ

PŘÍLOHY NA PŘILOŽENÉM DIGITÁLNÍM DATOVÉM MÉDIU

- Text závěrečné práce ve formátech DOCX a PDF.
- Logický a relační model dat ve formátu DMD.
- Skript fyzického modelu ve formátu DDL.
- Skript pro naplnění testovacími daty.
- Sled testovacích příkazů,
- Dva rozšířené dotazy v jednom SQL souboru.