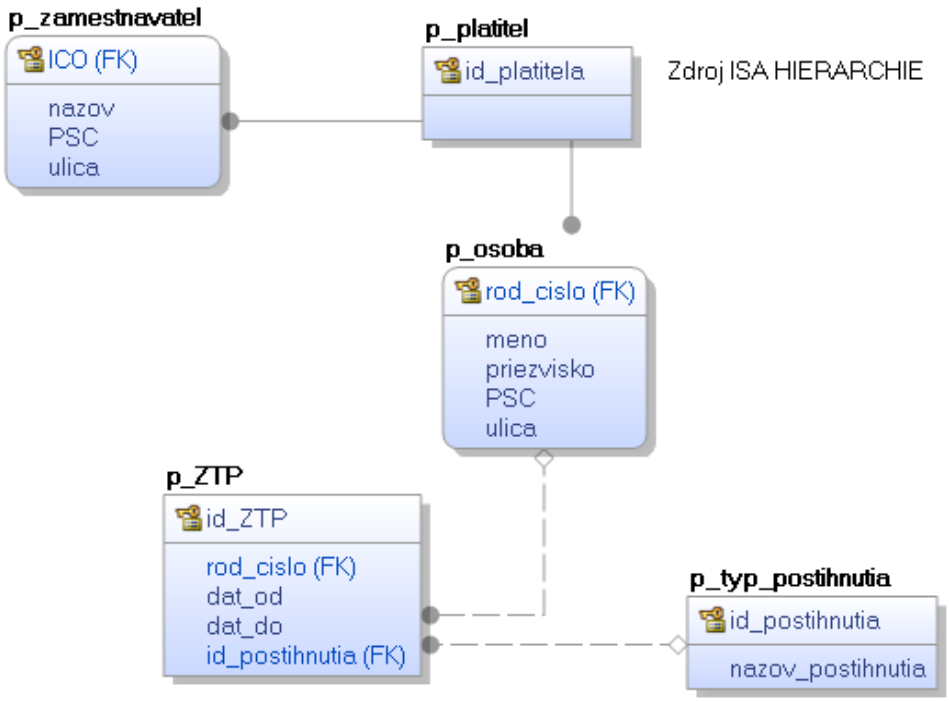
Kapitola 4

Modelovanie, DDL

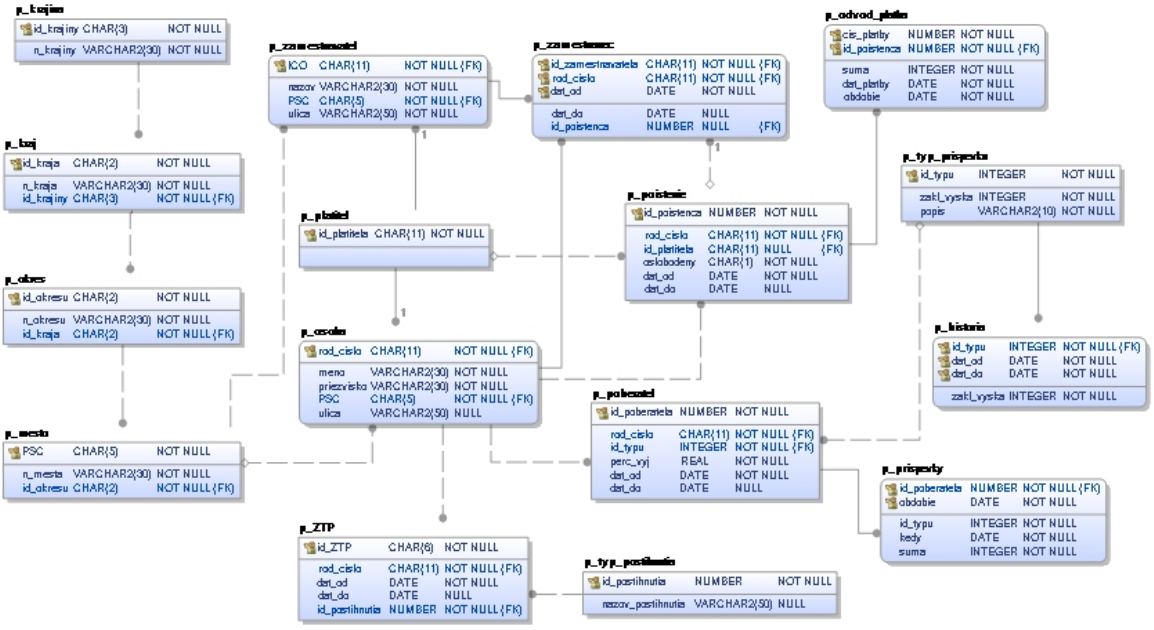
4.1 Zadanie cvičenia

1. Nakreslite ERA model podľa nasledovného vzoru.

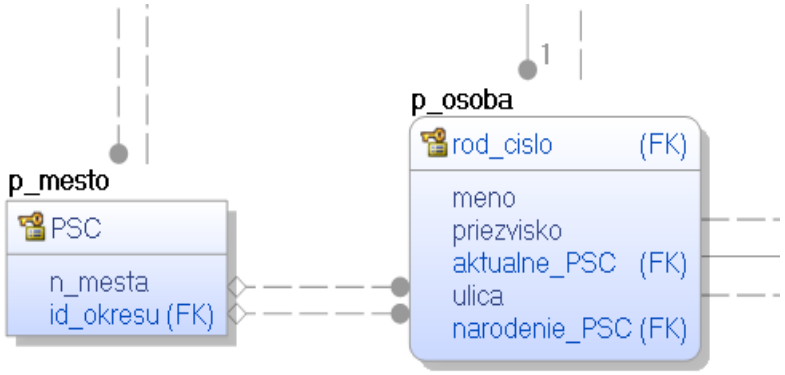


2. Všetkým atribútom určite **dátové typy** a **NULL/NOT NULL** constrainty.

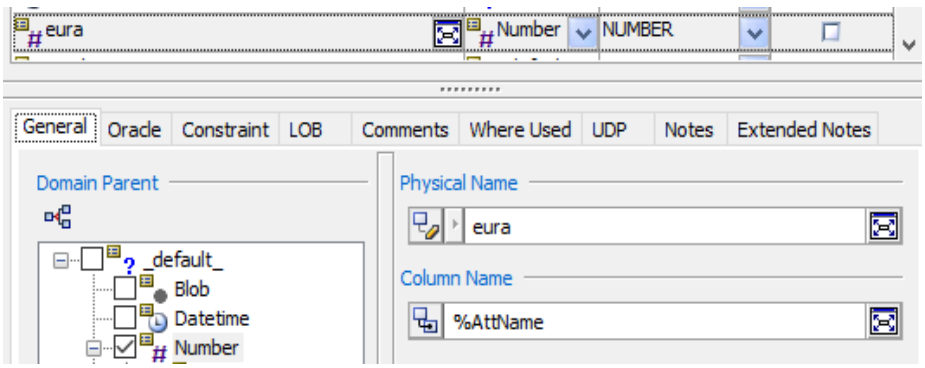
3. Spojte Vašu časť modelu s predpripravenou časťou (`soc_poist_cv4.dm2`). Dbajte na správnu definíciu vzťahu – **typ vzťahu, kardinalitu, členstvo vo vzťahu**.

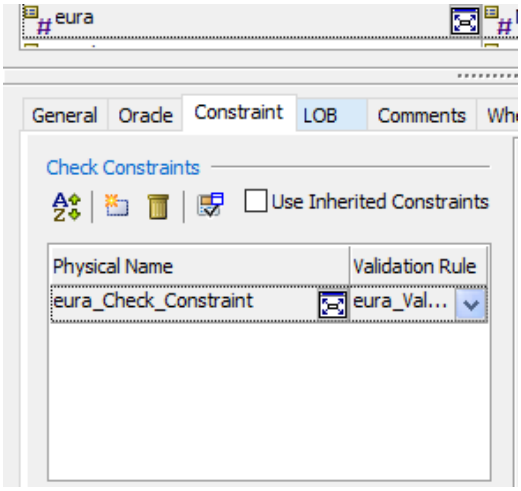


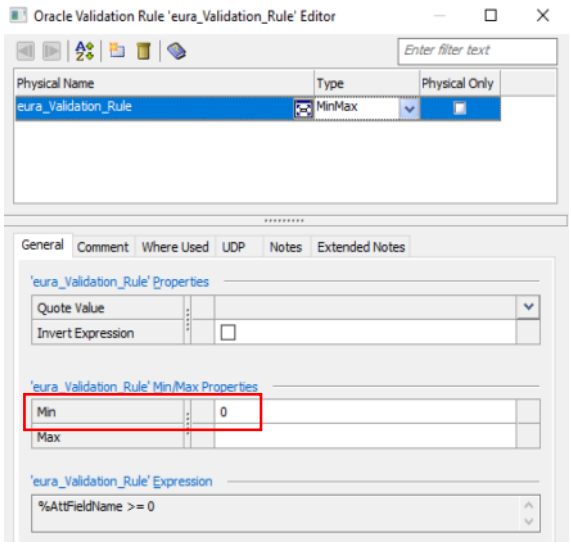
4. Vytvorte rekurzívne vzťahy medzi osobami vyjadrujúce rodokmeň (otec, matka).
5. Vytvorte vzťah medzi tabuľkami osoba a mesto vyjadrujúci mesto, v ktorom sa osoba narodila. Nezabudnite daný atribút premenovať.



6. Definujte nasledovné domény:
   (a) **eura** - suma musí byť reálne kladné číslo







(b) **bool** - vymenovaná množina ('a'/'A' – ano, 'n'/'N' – nie)

(c) **percenta** - reálne číslo z intervalu <0,1 ; 100 >

7. Priraďte doménu `eura` všetkým sumám za odvody, príspevky (`suma`, `zakl_vyska`).
8. Priraďte doménu `percenta` atribútu `perc_vyj` v tabuľke poberatel a doménu `bool` atribútu `oslobodeny` (tab. `poistenie`).
9. Zobrazte dátový diagram tak, aby obsahoval entity, vzťahy (definujte popis vzťahov), atribúty, označenie FK, dátové typy a NULL / NOT NULL constrainty.
10. Vygenerujte SQL skripty – tabuľky a vzťahy.

> ***Dôležité***
> Skript spúšťajte na serveri len v prípade, že bol vygenerovaný bez chýb !!!

!

11. Vygenerovaný SQL skript uložte do súboru `script_soc_poist.sql` a preneste na server pomocou `WINSCP`. Zároveň si dôkladne preštudujte vygenerovaný kód. Definícia a priradenie vlastnej domény bude vyzerať nasledovne:

```
CREATE TABLE p_odvod_platba
(
        cis_platby              NUMBER NOT NULL ,
        suma                    NUMBER NOT NULL  CHECK (suma >= 0),
        dat_platby              DATE NOT NULL ,
        obdobie                 DATE NOT NULL ,
        id_poistenca            NUMBER NOT NULL
);

ALTER TABLE p_odvod_platba
        ADD  PRIMARY KEY (cis_platby,id_poistenca);
```

12. Spustite ho v SQLplus príkazom

    ```
    SQL> start <nazov_suboru>
    ```

13. Vypíšte si zoznam tabuliek, ktorých názov začína na `'p_'`. Koľko ich máte? (Upravte nasledujúci príkaz)

    ```
    SQL> select table_name from tabs;
    ```

14. Zrušte všetky tabuľky sociálnej poisťovne v správnom poradí.

    ```
    SQL> drop table <nazov_tabulky>;
    ```

4.2 Úlohy na opakovanie (Select)

1. Napíšte nasledovné podmienky, ktoré by ste použili napr. vo WHERE príkazu select, ak `datum_od` a `datum_do` sú stĺpce s dátovým typom `DATE`. Tieto podmienky musia fungovať nech je akýkoľvek aktuálny dátum. Nezabudnite na roky:

   a) datum_od je minulý rok
   b) datum_od je budúci rok
   c) datum_od je minulý mesiac
   d) datum_od je budúci mesiac
   e) datum_od je o dva týždne
   f) datum_od a datum_do sú v jednom kalendárnom mesiaci
   g) medzi dátumami datum_od a datum_do neubehlo viac ako 3 roky. (s presnosťou na dni)
   h) medzi dátumami datum_od a datum_do neubehlo viac ako 12 hodín.

2. Vypíšte menný zoznam osôb (spolu s `id_poistenca`), za ktoré boli zaplatené odvody v minulom roku. (potlačte duplicity)
3. Vypíšte predchádzajúci výpis, ale tak, že duplicity ani nevyrobíte. (Nápoveda - použite vnorený select).
4. Vypíšte výplatnú listinu príspevkov, ktoré majú byť vyplatené tento mesiac.
5. Vypíšte počet osôb, ktoré sú samoplatci (osoba je platiteľom sama sebe).

4.3 Jazyk DDL

4.3.1 Tabuľka

4.3.1.1 Základná syntax - tabuľka

1. Vytvorenie tabuľky

```
CREATE TABLE [schema.]nazov_tabulky
  ( { nazov_stlpca datatype [DEFAULT expr] { [column_constraint] }[...]
      |
      table_constraint
    }[...]
  )
```

2. Úprava štruktúry tabuľky

```
ALTER TABLE [schema.]nazov_tabulky
{ ADD (   {
           nazov_stlpca datatype [DEFAULT expr] { [column_constraint] }[...]
           |
           table_constraint
           } [, ...]
       )
  |
  MODIFY
  |
  DROP { COLUMN nazov_stlpca
         |
         CONSTRAINT nazov_constraintu
       }
}
```

3. Zrušenie tabuľky (schéma tabuľky spolu s dátami)

```
DROP TABLE [schema.]nazov_tabulky;
```

4.3.1.2 Základná syntax - Obmedzenia v tabuľke

1. Stĺpcové obmedzenie

```
column_constraint ::=
  [CONSTRAINT nazov_constraintu]
  {
     [NOT] NULL
   |
    {UNIQUE | PRIMARY KEY}
   |
    REFERENCES [nazov_schemy.] nazov_tabulky [ (  nazov_stlpca  ) ]
     [ON DELETE CASCADE ]
   |
    CHECK (condition)
  }
```

2. Tabuľkové obmedzenie

```
table_constraint ::=
  [CONSTRAINT nazov_constraintu]
  {
    { UNIQUE | PRIMARY KEY} ( { nazov_stlpca } [,...] )
     |
     FOREIGN KEY ( { nazov_stlpca } [,...] ) REFERENCES
       [nazov_schemy.]nazov_tabulky
          [ ( { nazov_stlpca } [,...] ) ] [ON DELETE CASCADE ]
     |
     CHECK (condition)
  }
```

4.3.1.3 Príklady

1. Vytvorenie tabuľky bez primárneho kľúča

```
CREATE TABLE os_udaje
 (
  rod_cislo CHAR(10) NOT NULL,
  meno VARCHAR2(15) NOT NULL,
  priezvisko VARCHAR2(15) NOT NULL,
  ulica VARCHAR2(20),
  obec VARCHAR2(20),
  psc CHAR(5) NOT NULL,
  okres VARCHAR2(20),
  st_prisl CHAR(2) NOT NULL
);
```

2. Vytvorenie tabuľky **s jednoduchým primárnym kľúčom**

   1. možnosť

```
CREATE TABLE os_udaje
(
  rod_cislo CHAR(10) NOT NULL PRIMARY KEY,
  ...
);
```

   2. možnosť

```
CREATE TABLE os_udaje
(
  rod_cislo CHAR(10) NOT NULL,
  ...,
  PRIMARY KEY (rod_cislo)
);
```

   3. možnosť

```
CREATE TABLE os_udaje
(
  rod_cislo CHAR(10) NOT NULL,
  ...
);

ALTER TABLE os_udaje
  ADD ( PRIMARY KEY (rod_cislo) );
```

3. Vytvorenie tabuľky **s kompozitným primárnym kľúčom**

   1. možnosť

```
CREATE TABLE st_odbory
(
  c_st_odboru  SMALLINT NOT NULL,
  c_specializacie SMALLINT NOT NULL,
  ...,
  PRIMARY KEY (c_st_odboru, c_specializacie)
);
```

1. možnosť

```
CREATE TABLE st_odbory
(
  c_st_odboru      SMALLINT   NOT NULL,
  c_specializacie SMALLINT   NOT NULL,
  ...
);

ALTER TABLE st_odbory
  ADD ( PRIMARY KEY (c_st_odboru, c_specializacie) );
```

4. **Nesprávne pokusy** o vytvorenie kompozitného primárneho kľúča

   (a) NIE JE MOŽNÉ – Syntax ERROR – lebo PK môže byť len jeden

   ```
   CREATE TABLE TAB1
   ( pk1  INTEGER  NOT NULL  PRIMARY KEY ,
     pk2  INTEGER  NOT NULL  PRIMARY KEY );
   ```

   (b) JE MOŽNÉ

   ```
   CREATE TABLE TAB1
   ( pk1  INTEGER  NOT NULL  UNIQUE KEY ,
     pk2  INTEGER  NOT NULL  UNIQUE KEY );
   ```

   ale výsledok znamená, že relácia `TAB1` má dvoch kandidátov primárneho kľúča:

   - **KPK1:** `pk1`
   - **KPK2:** `pk2`

   ale **NIE** kompozitný PK zložený z atribútov `pk1, pk2`

5. Vytvorenie **cudzieho kľúča**

   1. možnosť – pozor na poradie tabuľka `os_udaje` aj spolu s primárnym kľúčom už musí existovať

   ```
   CREATE TABLE student
   (
     os_cislo  INTEGER NOT NULL PRIMARY KEY,
     rod_cislo  CHAR(10) NOT NULL,
     ...,
     FOREIGN KEY (rod_cislo) REFERENCES os_udaje (rod_cislo)
   );
   ```

   2. možnosť – využíva sa aby nebolo potrebné dbať na poradie tabuliek, stačí len príkazy spúšťať v poradí – CREATE TABLE, ALTER TABLE ADD PRIMARY KEY, a potom ALTER TABLE ADD FOREIGN KEY:

   ```
   ALTER TABLE student
     ADD ( FOREIGN KEY (rod_cislo) REFERENCES os_udaje (rod_cislo) );
   ```

4.3.1.4 Premenovanie

- **Premenovanie stĺpca**

```
ALTER TABLE table_name
  RENAME COLUMN old_name TO new_name;
```

- **Premenovanie tabuľky**

```
ALTER TABLE table_name
  RENAME TO new_table_name;
```

4.3.1.5 Zmena dátového typu stĺpca

Je možné uskutočniť len rozširujúce zmeny (napr. z pôvodného CHAR(2) na CHAR(6))

```
ALTER TABLE table_name
  MODIFY column_name new_data_type;
```

4.3.1.6 Zmena NULL/NOT NULL

Tieto úpravy samozrejme musia dovoliť dáta v tabuľke.

a)
```
ALTER TABLE table_name
    MODIFY column_name NOT NULL;
```

b)
```
ALTER TABLE table_name
    MODIFY column_name  NULL;
```

4.3.1.7 CHECK podmienka

1. Definícia pri stĺpci:

```
CREATE TABLE tab1
(
   id             integer   primary key,
   kladne_cislo   integer   CHECK ( kladne_cislo > 0 ),
   stav           char(1)   NOT NULL CHECK ( stav IN ('a','n') )
);
```

2. Definícia ako tabuľkové obmedzenie:

```
CREATE TABLE tab1
(
   id             integer   primary key,
   kladne_cislo   integer   CHECK ( kladne_cislo > 0 ),
   stav           char(1)   NOT NULL CHECK ( stav IN ('a','n') )
);
```

3. Doplenie pomocou ALTER TABLE:

```
ALTER TABLE tab1
    ADD CHECK ( kladne_cislo > 0 )
;
```

4.3.1.8 DEFAULT hodnota

1.
```
CREATE TABLE tab1
(
   id             integer   primary key,
   stav           char(1)   DEFAULT 'S'  CHECK( stlpec in ('S','T') ) NOT NULL
);
```

```
ALTER TABLE tab1
  MODIFY stav DEFAULT 'S';
```

4.3.2 DDL a vzťahy

4.3.2.1 DDL a kardinalita vzťahu

- **1:1**

```
ALTER TABLE FK_tabulka
  ADD FOREIGN KEY (FK_stlpce) REFERENCES PK_tabulka;

ALTER TABLE FK_tabulka
  ADD UNIQUE (FK_stlpce);
```

- **1:N**

```
ALTER TABLE FK_tabulka
  ADD FOREIGN KEY (FK_stlpce) REFERENCES PK_tabulka;
```

4.3.2.2 DDL a povinnosť členstva vo vzťahu

- povinné členstvo

```
CREATE TABLE FK_tabulka
  (
     ...
     FK_stlpec1       dat_typ    NOT NULL,
     FK_stlpec2       dat_typ    NOT NULL,
     ....);

ALTER TABLE FK_tabulka
  ADD FOREIGN KEY (FK_stlpce) REFERENCES PK_tabulka;
```

4.3.2.3 DDL a typ vzťahu

1. **Identifikačný vzťah** - FK sa stáva súčasťou PK v tabuľke `fk_tabulka`

```
CREATE TABLE fk_tabulka
(
    pk_stlpec1   dat_typ   NOT NULL,
    pk_stlpec2  dat_typ   NOT NULL,
   ....,
    PRIMARY KEY ( pk_stlpec1, pk_stlpec2 )
);


CREATE TABLE fk_tabulka
(
    pk_stlpec    dat_typ   NOT NULL,
    PFK_stlpec1  dat_typ   NOT NULL,
    PFK_stlpec2  dat_typ   NOT NULL,
    ...

    PRIMARY KEY ( pk_stlpec,  PFK_stlpec1, PFK_stlpec2 ),
    FOREIGN KEY ( PFK_stlpec1, PFK_stlpec2 ) REFERENCES pk_tabulka
);
```

2. **Neidentifikačný vzťah** - klasický FK, ktorý sa nestáva súčasťou PK

```
CREATE TABLE fk_tabulka
(
    pk_stlpec    dat_typ   NOT NULL,
    FK_stlpec1   dat_typ   ,
    FK_stlpec2   dat_typ   ,
    ...

    PRIMARY KEY ( pk_stlpec ),
    FOREIGN KEY ( PFK_stlpec1, PFK_stlpec2 ) REFERENCES pk_tabulka
);
```

4.3.3 Index

1. Vytvorenie indexu

```
CREATE [UNIQUE] [CLUSTER] INDEX [schema.]nazov_indexu
  ON [schema.]nazov_tabulky (nazov_stlpca [ASC | DESC],...)
```

2. Znovu vytvorenie indexu.

```
ALTER INDEX [schema.]nazov_indexu REBUILD;
```

3. Zrušenie indexu.

```
DROP INDEX [schema.]nazov_indexu;
```

4.3.3.1 Príklady

- Vytvorenie **duplikátneho** indexu

```
CREATE INDEX ind_zp_oc ON  zap_predmety (os_cislo);
```

- Vytvorenie **unikátneho** indexu

```
CREATE UNIQUE INDEX ind_ou ON  os_udaje (rod_cislo);
```

```
CREATE UNIQUE INDEX ind_zp ON
zap_predmety(os_cislo, skrok, cis_predmet);
```

- Index s určením **smeru triedenia**:

```
CREATE INDEX ind_vysl ON vysledok
  ( pocet_bodov DESC, id_cloveka ASC );
```

```
alebo
```

```
CREATE INDEX ind_vysl ON vysledok
  ( pocet_bodov DESC, id_cloveka );
```

- **Zrušenie** indexu

```
DROP INDEX ind_zp;
```

4.3.4 Funkcionálne indexy

```
CREATE [UNIQUE] [CLUSTER] INDEX meno_indexu
ON meno_tabulky (funkcia(parametre) [ASC | DESC],...)
```

Tento druh indexu namiesto indexovania priamo stĺpcov, umožní indexovať výsledok funkcie (štandardnej, alebo aj užívateľom definovanej), pričom parametrami môžu byť stĺpce, alebo konštanty. Dôvodom je optimalizácia SQL dotazov.

```
CREATE INDEX idx_meno
    ON os_udaje ( upper(priezvisko));
```