Курсовая работа

# ERM и PDM.





# Создание таблиц.

**CREATE TYPE** race\_type **AS ENUM** (  
 **'Totem'**,  
 **'Demon'**,  
 **'Mech'**,  
 **'Pirate'**,  
 **'Murloc'**,  
 **'Beast'**,  
 **'Dragon'**);  
  
**CREATE TYPE** outcome\_type **AS ENUM** (  
 **'Player1Win'**,  
 **'Tie'**,  
 **'Player2Win'**);  
  
**CREATE DOMAIN** natural\_int **AS INTEGER CHECK** (**VALUE** >= 0);  
**CREATE DOMAIN** qnt\_int **AS INTEGER CHECK** (**VALUE** > 0 **AND VALUE** <= 2);  
  
**CREATE TABLE IF NOT EXISTS Class** (  
 **Class\_id INTEGER PRIMARY KEY**,  
 **Class\_name TEXT UNIQUE NOT NULL**);  
  
**CREATE TABLE IF NOT EXISTS Set** (  
 **Set\_id INTEGER PRIMARY KEY**,  
 **Set\_name TEXT UNIQUE NOT NULL**);  
  
**CREATE TABLE IF NOT EXISTS** Mechanics (  
 **Mechanic\_id INTEGER PRIMARY KEY**,  
 **Mechanic\_name TEXT UNIQUE NOT NULL**);  
  
**CREATE TABLE IF NOT EXISTS** Players (  
 **Player\_id INTEGER PRIMARY KEY**,  
 **Player\_name TEXT UNIQUE NOT NULL**,  
 **Player\_country TEXT NOT NULL**,  
 **Player\_score** natural\_int **DEFAULT** 0  
);  
  
**CREATE TABLE IF NOT EXISTS** Cards (  
 **Card\_id INTEGER PRIMARY KEY**,  
 **Card\_name TEXT UNIQUE NOT NULL**,  
 **Card\_set\_id INTEGER NOT NULL REFERENCES Set** (**Set\_id**) **ON DELETE CASCADE**,  
 **Card\_description TEXT NOT NULL**,  
 **Card\_score** natural\_int **NOT NULL**,  
 **Class\_id INTEGER NOT NULL REFERENCES Class** (**Class\_id**) **ON DELETE CASCADE**);  
  
**CREATE TABLE IF NOT EXISTS** Minions (  
 **Card\_id INTEGER PRIMARY KEY REFERENCES** Cards **ON DELETE CASCADE**,  
 **race** race\_type  
);  
  
**CREATE TABLE IF NOT EXISTS** Spells (  
 **Card\_id INTEGER PRIMARY KEY REFERENCES** Cards **ON DELETE CASCADE**);  
  
**CREATE TABLE IF NOT EXISTS** Weapons (  
 **Card\_id INTEGER PRIMARY KEY REFERENCES** Cards **ON DELETE CASCADE**);  
  
**CREATE OR REPLACE VIEW** Weapon\_cards **AS  
 SELECT  
 card\_name**,  
 **card\_description**,  
 **set\_name**,  
 **card\_score  
 FROM** Weapons  
 **JOIN** Cards **ON** weapons.**card\_id** = cards.**card\_id  
 JOIN Class ON** cards.**class\_id** = **class**.**class\_id  
 JOIN set ON** cards.**card\_set\_id** = **set**.**set\_id**;  
  
**CREATE VIEW** Spell\_cards **AS  
 SELECT  
 card\_name**,  
 **card\_description**,  
 **class\_name**,  
 **set\_name**,  
 **card\_score  
 FROM** spells  
 **JOIN** Cards **ON** spells.**card\_id** = cards.**card\_id  
 JOIN Class ON** cards.**class\_id** = **class**.**class\_id  
 JOIN set ON** cards.**card\_set\_id** = **set**.**set\_id**;  
  
**CREATE VIEW** Minion\_cards **AS  
 SELECT  
 card\_name**,  
 **card\_description**,  
 **class\_name**,  
 **race**,  
 **set\_name**,  
 **card\_score  
 FROM** Minions  
 **JOIN** Cards **ON** Minions.**card\_id** = cards.**card\_id  
 JOIN Class ON** cards.**class\_id** = **class**.**class\_id  
 JOIN set ON** cards.**card\_set\_id** = **set**.**set\_id**;  
  
**CREATE TABLE IF NOT EXISTS** Decks (  
 **Deck\_id INTEGER PRIMARY KEY**,  
 **Deck\_name TEXT NOT NULL**,  
 **Class\_id INTEGER NOT NULL REFERENCES Class** (**Class\_id**) **ON DELETE CASCADE**,  
 **Player\_id INTEGER NOT NULL REFERENCES** Players (**Player\_id**) **ON DELETE CASCADE**,  
 **Deck\_score** natural\_int **DEFAULT** 0,  
 **UNIQUE** (**Player\_id**, **Deck\_name**)  
);  
  
**CREATE TABLE IF NOT EXISTS** Has\_mechanic (  
 **Card\_id INTEGER NOT NULL REFERENCES** Cards (**Card\_id**) **ON DELETE CASCADE**,  
 **Mechanic\_id INTEGER NOT NULL REFERENCES** Mechanics (**Mechanic\_id**) **ON DELETE CASCADE**,  
 **PRIMARY KEY** (**Card\_id**, **Mechanic\_id**)  
);  
  
**CREATE TABLE IF NOT EXISTS** In\_deck (  
 **Card\_id INTEGER NOT NULL REFERENCES** Cards (**Card\_id**) **ON DELETE CASCADE**,  
 **Deck\_id INTEGER NOT NULL REFERENCES** Decks (**Deck\_id**) **ON DELETE CASCADE**,  
 **Quantity** qnt\_int **NOT NULL**,  
 **PRIMARY KEY** (**Card\_id**, **Deck\_id**)  
);  
  
**CREATE TABLE IF NOT EXISTS** Tournament (  
 **Tournament\_id INTEGER PRIMARY KEY**,  
 **Tournament\_name TEXT NOT NULL UNIQUE**,  
 **Tournament\_format TEXT NOT NULL**,  
 **Tournament\_prize\_pool** natural\_int **DEFAULT** 0  
);  
  
**CREATE TABLE IF NOT EXISTS** Participated (  
 **Player\_id INTEGER NOT NULL REFERENCES** Players (**Player\_id**) **ON DELETE CASCADE**,  
 **Tournament\_id INTEGER NOT NULL REFERENCES** Tournament (**Tournament\_id**) **ON DELETE CASCADE**,  
 **Place INTEGER NOT NULL**,  
 **PRIMARY KEY** (**Player\_id**, **Tournament\_id**)  
);  
  
**CREATE TABLE IF NOT EXISTS** Matches (  
 **Match\_id INTEGER PRIMARY KEY**,  
 **Tournament\_id INTEGER NOT NULL REFERENCES** Tournament (**Tournament\_id**) **ON DELETE CASCADE**,  
 **Player1\_deck\_id INTEGER NOT NULL REFERENCES** Decks (**Deck\_id**) **ON DELETE CASCADE**,  
 **Player2\_deck\_id INTEGER NOT NULL REFERENCES** Decks (**Deck\_id**) **ON DELETE CASCADE**,  
 **Outcome** outcome\_type **NOT NULL**);

# Создание индексов.

**CREATE INDEX** card\_name\_index  
 **ON** cards **USING BTREE** (**card\_name**); *-- hash***CREATE INDEX** card\_set\_index  
 **ON** cards **USING BTREE** (**card\_set\_id**); *-- hash***CREATE INDEX** minions\_race\_index  
 **ON** minions **USING BTREE** (**race**); *-- hash***CREATE INDEX** deck\_name\_index  
 **ON** decks **USING BTREE** (**deck\_name**); *-- hash***CREATE INDEX** deck\_class\_id\_index  
 **ON** decks **USING BTREE** (**class\_id**); *-- hash***CREATE INDEX** participated\_index  
 **ON** participated **USING BTREE** (**tournament\_id**);  
  
**CREATE INDEX** has\_mechanic\_card\_id\_index  
 **ON** has\_mechanic **USING BTREE** (**card\_id**);  
  
**CREATE INDEX** heroes\_class\_index  
 **ON class USING BTREE** (**class\_name**); *-- hash***CREATE INDEX** in\_deck\_card\_id\_index  
 **ON** in\_deck **USING BTREE** (**card\_id**);  
  
**CREATE INDEX** players\_country\_index  
 **ON** players **USING BTREE** (**player\_country**); *-- hash*

1. Создание функций и триггеров.

**CREATE OR REPLACE FUNCTION** *add\_card\_into\_deck*()  
 **RETURNS TRIGGER AS $$  
DECLARE  
 \_player\_id INTEGER;  
 \_class\_id INTEGER;  
 \_card\_class\_id INTEGER;  
 \_card\_score INTEGER;  
 \_add\_score INTEGER;  
BEGIN  
 SELECT  
 player\_id,  
 class\_id  
 FROM decks  
 WHERE deck\_id =** *NEW***.deck\_id  
 INTO \_player\_id, \_class\_id;  
  
 SELECT card\_score  
 FROM cards  
 WHERE card\_id =** *NEW***.card\_id  
 INTO \_card\_score;  
  
 IF (***TG\_OP* **= 'INSERT')  
 THEN  
 SELECT class\_id  
 FROM cards  
 WHERE card\_id =** *NEW***.card\_id  
 INTO \_card\_class\_id;  
 IF (\_card\_class\_id <> \_class\_id AND \_card\_class\_id <> 3)  
 THEN  
 RAISE EXCEPTION E'Illegal class card for this deck:%,%', \_class\_id,** *NEW***.card\_id;  
 END IF;  
  
 IF (***get\_cards\_in\_deck***(***NEW***.deck\_id) +** *NEW***.quantity > 30)  
 THEN  
 RAISE EXCEPTION E'Illegal number of cards in this deck:%',** *NEW***.deck\_id;  
 END IF;  
  
 \_add\_score =** *NEW***.quantity \* \_card\_score;  
 END IF;  
  
 IF (***TG\_OP* **= 'UPDATE')  
 THEN  
 IF (***get\_cards\_in\_deck***(***NEW***.deck\_id) + (***NEW***.quantity -** *OLD***.quantity) > 30)  
 THEN  
 RAISE EXCEPTION E'Illegal number of cards in this deck:%',** *NEW***.deck\_id;  
 END IF;  
 \_add\_score = (***NEW***.quantity -** *OLD***.quantity) \* \_card\_score;  
 END IF;  
  
 UPDATE decks  
 SET deck\_score = deck\_score + \_add\_score  
 WHERE deck\_id =** *NEW***.deck\_id;  
  
 RETURN** *NEW***;  
END;  
$$ LANGUAGE 'plpgsql'**;

**CREATE OR REPLACE FUNCTION** *get\_cards\_in\_deck*(\_deck\_id **INTEGER**)  
 **RETURNS INTEGER AS $$  
DECLARE  
 \_count INTEGER;  
 \_card\_id INTEGER;  
 \_quantity INTEGER;  
BEGIN  
 \_count = 0;  
 FOR \_card\_id, \_quantity IN (SELECT  
 card\_id,  
 quantity  
 FROM in\_deck  
 WHERE deck\_id = \_deck\_id) LOOP  
 \_count = \_count + \_quantity;  
 END LOOP;  
 RETURN \_count;  
END;  
$$ LANGUAGE 'plpgsql'**;  
  
**DROP TRIGGER IF EXISTS** card\_in\_deck  
**ON** in\_deck;  
**CREATE TRIGGER** card\_in\_deck  
**BEFORE INSERT OR UPDATE  
 ON** in\_deck  
**FOR EACH ROW  
EXECUTE PROCEDURE** *add\_card\_into\_deck*();

**CREATE OR REPLACE FUNCTION** *add\_match\_into\_matches*()  
 **RETURNS TRIGGER AS $$  
DECLARE  
 \_player\_id\_1 INTEGER;  
 \_player\_1\_deck\_score INTEGER;  
 \_player\_1\_score\_diff INTEGER;  
  
 \_player\_id\_2 INTEGER;  
 \_player\_2\_deck\_score INTEGER;  
 \_player\_2\_score\_diff INTEGER;  
  
 \_prize\_pool\_factor INTEGER;  
  
 \_deck\_diff\_1 INTEGER;  
 \_deck\_diff\_2 INTEGER;  
BEGIN  
 IF (***get\_cards\_in\_deck***(***NEW***.player1\_deck\_id) <> 30 OR** *get\_cards\_in\_deck***(***NEW***.player2\_deck\_id) <> 30)  
 THEN  
 RAISE EXCEPTION E'Submitted unfinished decks:%,%',** *NEW***.player1\_deck\_id,** *NEW***.player2\_deck\_id;  
 END IF;  
 SELECT  
 player\_id,  
 deck\_score  
 FROM decks  
 WHERE deck\_id =** *NEW***.Player1\_deck\_id  
 INTO \_player\_id\_1, \_player\_1\_deck\_score;  
 SELECT  
 player\_id,  
 deck\_score  
 FROM decks  
 WHERE deck\_id =** *NEW***.Player2\_deck\_id  
 INTO \_player\_id\_2, \_player\_2\_deck\_score;  
  
 IF (\_player\_id\_1 = \_player\_id\_2)  
 THEN  
 RAISE EXCEPTION E'Player can\'t play with himself:%', \_player\_id\_1;  
 END IF;  
 SELECT tournament\_prize\_pool  
 FROM tournament  
 WHERE tournament\_id =** *NEW***.tournament\_id  
 INTO \_prize\_pool\_factor;**

**\_deck\_diff\_1 = (\_prize\_pool\_factor / 200.0) \* (\_player\_1\_deck\_score / \_player\_2\_deck\_score);  
 \_deck\_diff\_2 = (\_prize\_pool\_factor / 200.0) \* (\_player\_2\_deck\_score / \_player\_1\_deck\_score);  
 IF (***NEW***.outcome = 'Player1Win')  
 THEN  
 \_player\_1\_score\_diff = \_deck\_diff\_1;  
 \_player\_2\_score\_diff = -\_deck\_diff\_2;  
 ELSEIF (***NEW***.outcome = 'Tie')  
 THEN  
 \_player\_1\_score\_diff = 0;  
 \_player\_2\_score\_diff = 0;  
 ELSEIF (***NEW***.outcome = 'Player2Win')  
 THEN  
 \_player\_1\_score\_diff = -\_deck\_diff\_1;  
 \_player\_2\_score\_diff = \_deck\_diff\_2;  
 END IF;  
  
 UPDATE players  
 SET player\_score = player\_score + \_player\_1\_score\_diff  
 WHERE player\_id = \_player\_id\_1;  
 UPDATE players  
 SET player\_score = player\_score + \_player\_2\_score\_diff  
 WHERE player\_id = \_player\_id\_2;  
 RETURN** *NEW***;  
END;  
$$ LANGUAGE 'plpgsql'**;  
  
**DROP TRIGGER IF EXISTS** match\_add  
**ON** matches;  
**CREATE TRIGGER** match\_add  
**BEFORE INSERT  
 ON** matches  
**FOR EACH ROW  
EXECUTE PROCEDURE** *add\_match\_into\_matches*();  
  
*--get all deck names from player***CREATE OR REPLACE FUNCTION** *get\_all\_players\_decks*(\_player\_name **TEXT**)  
 **RETURNS TABLE**(**deck\_id INTEGER**, **deck\_name TEXT**, **deck\_score** natural\_int) **AS $$  
SELECT  
 deck\_id,  
 deck\_name,  
 deck\_score  
FROM decks  
 NATURAL JOIN players  
WHERE player\_name = \_player\_name;  
$$ LANGUAGE 'sql'**;

*-- Player name to player\_id***CREATE OR REPLACE FUNCTION** *get\_player\_id*(\_player\_name **TEXT**)  
 **RETURNS INTEGER AS $$  
DECLARE  
 id INTEGER;  
BEGIN  
 SELECT player\_id  
 FROM players  
 WHERE player\_name = \_player\_name  
 INTO id;  
 RETURN id;  
END;  
$$ LANGUAGE 'plpgsql'**;  
  
*--get all cards in deck***CREATE OR REPLACE FUNCTION** *get\_all\_deck\_cards*(\_player\_name **TEXT**, \_deck\_name **TEXT**)  
 **RETURNS TABLE**(**quantity** qnt\_int, **card\_id INTEGER**, **card\_name TEXT**, **description TEXT**, **set TEXT**) **AS $$  
SELECT  
 in\_deck.quantity,  
 cards.card\_id,  
 cards.card\_name,  
 cards.card\_description,  
 set.set\_name  
FROM cards  
 JOIN in\_deck ON cards.card\_id = in\_deck.card\_id  
 JOIN decks ON in\_deck.deck\_id = decks.deck\_id  
 JOIN set ON cards.card\_set\_id = set.set\_id  
WHERE decks.player\_id =** *get\_player\_id***(\_player\_name) AND decks.deck\_name = \_deck\_name;  
$$ LANGUAGE 'sql'**;  
  
**CREATE OR REPLACE FUNCTION** *get\_rankings\_list*()  
 **RETURNS TABLE**(**player\_id INTEGER**, **player\_name TEXT**, **player\_score** natural\_int) **AS $$  
SELECT  
 players.player\_id,  
 players.player\_name,  
 players.player\_score  
FROM players  
ORDER BY players.player\_score DESC;  
$$ LANGUAGE 'sql'**;  
  
  
**CREATE TYPE** achievement\_type **AS** (  
 tournament\_name **TEXT**,  
 place **INTEGER**,  
 prize\_pool **INTEGER**);

*-- Player name to achievements***CREATE OR REPLACE FUNCTION** *get\_player\_achievements*(\_player\_name **TEXT**)  
 **RETURNS SETOF** achievement\_type **AS $$  
DECLARE  
 \_result achievement\_type;  
 \_player\_id INTEGER;  
 \_tournament\_id INTEGER;  
 \_tournament\_name TEXT;  
 \_place INTEGER;  
 \_prize\_pool INTEGER;  
BEGIN  
 \_player\_id =** *get\_player\_id***(\_player\_name);  
 FOR \_tournament\_id, \_place IN (SELECT  
 tournament\_id,  
 place  
 FROM participated  
 WHERE player\_id = \_player\_id) LOOP  
 \_tournament\_name = (SELECT tournament\_name  
 FROM tournament  
 WHERE tournament\_id = \_tournament\_id);  
 \_prize\_pool = (SELECT tournament\_prize\_pool  
 FROM tournament  
 WHERE \_tournament\_id = tournament\_id);  
 \_result.tournament\_name = \_tournament\_name;  
 \_result.place = \_place;  
 \_result.prize\_pool = \_prize\_pool;  
 RETURN NEXT \_result;  
 END LOOP;  
END;  
$$ LANGUAGE 'plpgsql'**;

1. Заполнение базы данными.
   1. Class.sql
   2. Sets.sql
   3. Cards.sql
   4. Mechanics.sql
   5. Minions.sql
   6. Spells.sql
   7. Weapons.sql
   8. Has\_mechanic.sql
   9. Country.sql
   10. Players.sql
   11. Decks.sql
   12. In\_deck.sql
   13. Tournaments.sql
   14. Participated.sql
   15. Matches.sql