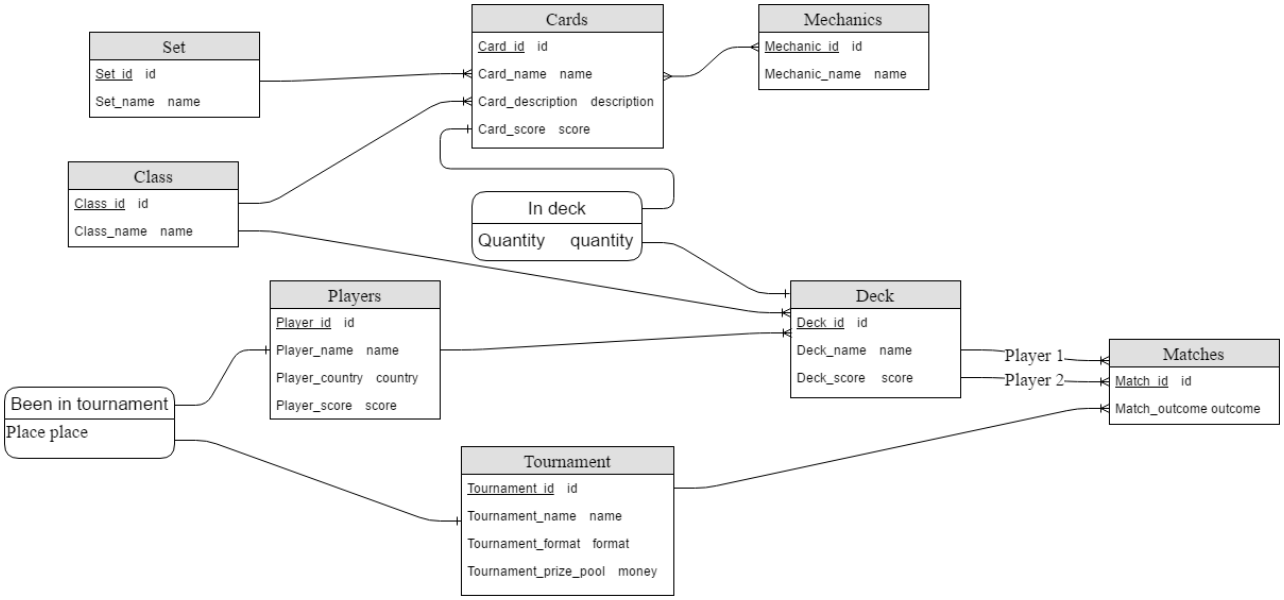
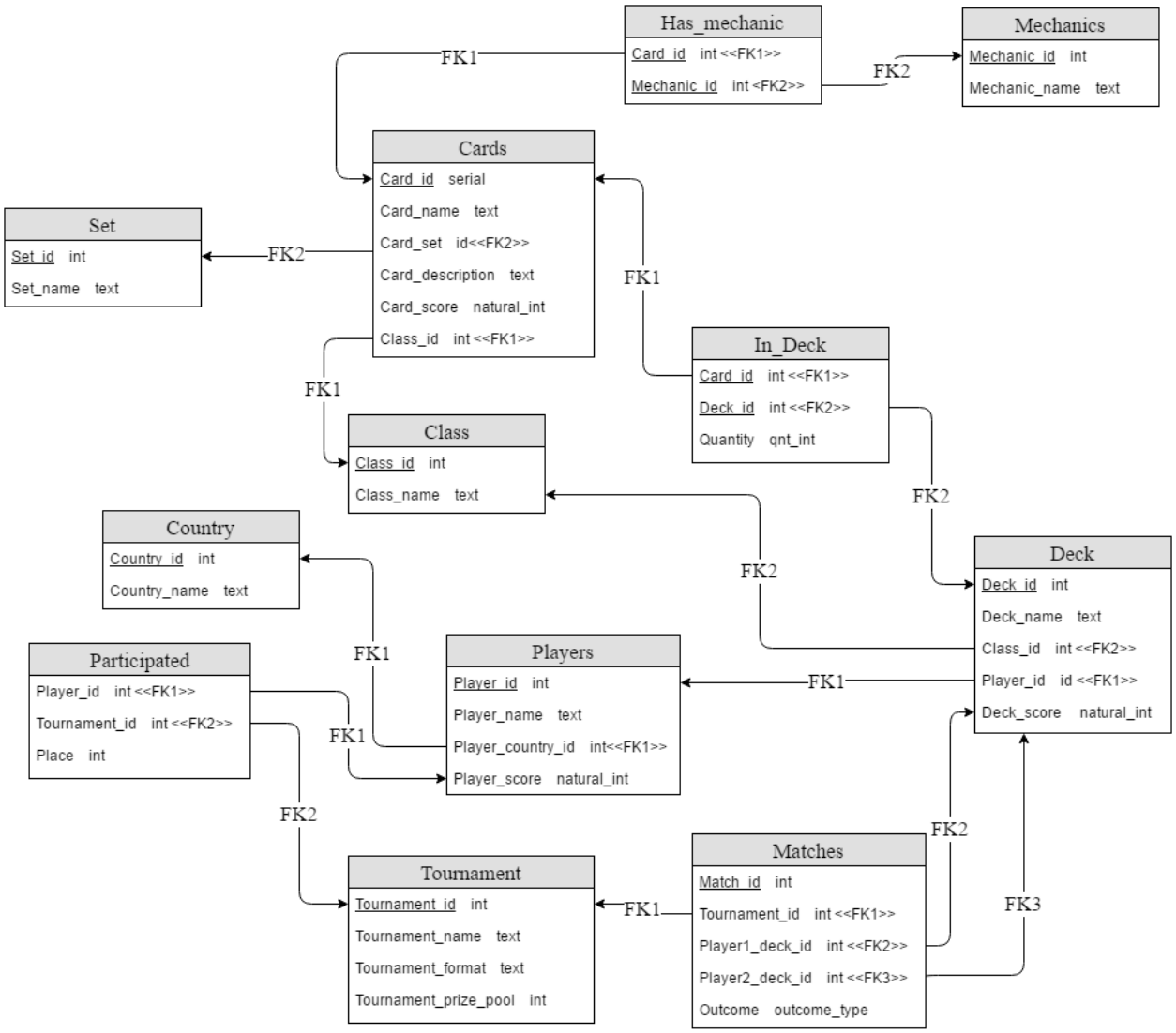
Курсовая работа

0. ERM и PDM.





1. 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
);
```

2. 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
```

3. 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';
```

4. Заполнение базы данными.
    a. Class.sql
    b. Sets.sql
    c. Cards.sql
    d. Mechanics.sql
    e. Minions.sql
    f. Spells.sql
    g. Weapons.sql
    h. Has_mechanic.sql
    i. Country.sql
    j. Players.sql
    k. Decks.sql
    l. In_deck.sql
    m. Tournaments.sql
    n. Participated.sql
    o. Matches.sql