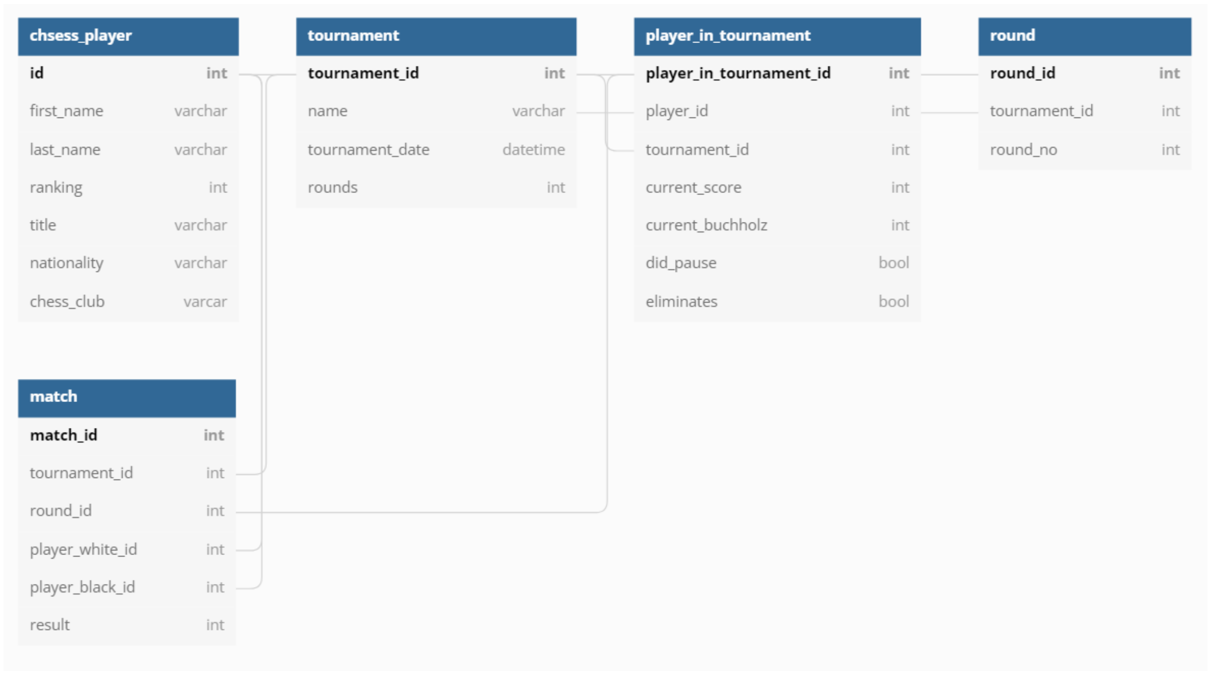
Chess_Database - Miniprojekt

Michał Godek i Jakub Kościelniak

Projekt realizowany przy użyciu

Postgresq

Python



chsess_player	
id	int
first_name	varchar
last_name	varchar
ranking	int
title	varchar
nationality	varchar
chess_club	varcar

tournament	
tournament_id	int
name	varchar
tournament_date	datetime
rounds	int

player_in_tournament	
player_in_tournament_id	int
player_id	int
tournament_id	int
current_score	int
current_buchholz	int
did_pause	bool
eliminates	bool

round	
round_id	int
tournament_id	int
round_no	int

match	
match_id	int
tournament_id	int
round_id	int
player_white_id	int
player_black_id	int
result	int

Schemat bazy

Połączenie się z bazą

Do połączenia się z bazą potrzebny jest moduł pgcog2

```
def __init__(self):
    params = self.get_config()
    self.__connection_pool = psycopg2.pool.ThreadedConnectionPool(1, 10,
**params)
    self.__lock = threading.Lock()
```

params to parametry odczytywane z pliku zawiera nazwę hosta, bazy, użytkownika i hasło

Operacje sql w bazie

Żeby były możliwe potrzeba zainijcować kursor

```
cur = conn.cursor()
```

Wykonywanie poleceń sql

```
cur.execute(command)
```

Zamknięcie kursora i skommitowanie

```
cur.close()
conn.commit()
```

Tworzenie bazy

```
table_player = """
CREATE TABLE IF NOT EXISTS chess_player (
    player_id SERIAL PRIMARY KEY,
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    ranking INT NOT NULL,
    title VARCHAR(10),
    nationality VARCHAR(3) NOT NULL,
    chess_club VARCHAR(50)
    );
"""

table_tournament = """
CREATE TABLE IF NOT EXISTS tournament (
    tournament_id SERIAL PRIMARY KEY,
    name VARCHAR(20) NOT NULL,
    tournament_date DATE,
    rounds INT NOT NULL
    );
"""
```

```
table_player_in_tournament = """
CREATE TABLE IF NOT EXISTS player_in_tournament (
    player_in_tournament_id SERIAL PRIMARY KEY,
    player_id INT NOT NULL,
    tournament_id INT NOT NULL,
    current_score INT NOT NULL,
    current_buchholz INT NOT NULL,
    did_pause BOOL NOT NULL,
    eliminated BOOL NOT NULL,
    FOREIGN KEY (player_id) REFERENCES chess_player(player_id) ON UPDATE
CASCADE ON DELETE CASCADE,
    FOREIGN KEY (tournament_id) REFERENCES tournament(tournament_id) ON
UPDATE CASCADE ON DELETE CASCADE,
    UNIQUE (player_id, tournament_id)
    );
"""

round = """
CREATE TABLE IF NOT EXISTS round (
    round_id SERIAL PRIMARY KEY,
    tournament_id INT NOT NULL,
    round_no INT NOT NULL,
    FOREIGN KEY (tournament_id) REFERENCES tournament(tournament_id) ON
UPDATE CASCADE ON DELETE CASCADE
);
"""

table_match = """
CREATE TABLE IF NOT EXISTS match (
    match_id SERIAL PRIMARY KEY,
    tournament_id INT NOT NULL,
    round_id INT NOT NULL,
    player_white_id INT NOT NULL,
    player_black_id INT NOT NULL,
    result INT,
    FOREIGN KEY (tournament_id) REFERENCES tournament(tournament_id) ON
UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (player_white_id) REFERENCES chess_player(player_id) ON
UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (player_black_id) REFERENCES chess_player(player_id) ON
UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY (round_id) REFERENCES round(round_id) ON UPDATE CASCADE ON
DELETE CASCADE
    );
"""
```

Operacje w bazie

```
def insert_match(self, match):
    """Inserts match into database, returns generated match_id"""
    conn = None
    match_id = None
    try:
        conn = self.__connection_provider.get_connection()
        sql = """
        INSERT INTO match(tournament_id, round_id, player_white_id,
player_black_id, result)
        VALUES (%s, %s, %s, %s, %s) RETURNING match_id;
        """
        cur = conn.cursor()
        cur.execute(sql, [match.tournament.tournament_id,
match.round.round_id, match.player_white.player_id,
match.player_black.player_id,
                    match.get_result()])
        match_id = cur.fetchone()[0]
        set_object(match, match_id)
        conn.commit()
        cur.close()
    except psycopg2.DatabaseError as error:
        print(error)
    finally:
        if conn is not None:
            self.__connection_provider.free_connection(conn)
    return match_id
```

przykładowe operacje w bazie insert_match

```
def get_all_tournaments(self):
    conn = None
    tournaments = []
    try:
        conn = self.__connection_provider.get_connection()
        sql = """
        SELECT tournament_id
        FROM tournament;
        """
        cur = conn.cursor()
        cur.execute(sql)

        for row in cur:
            tournaments.append(self.get_tournament_by_id(row[0]))

        conn.commit()
        cur.close()
    except psycopg2.DatabaseError as error:
        print(error)
    finally:
        if conn is not None:
            self.__connection_provider.free_connection(conn)
    return tournaments
```

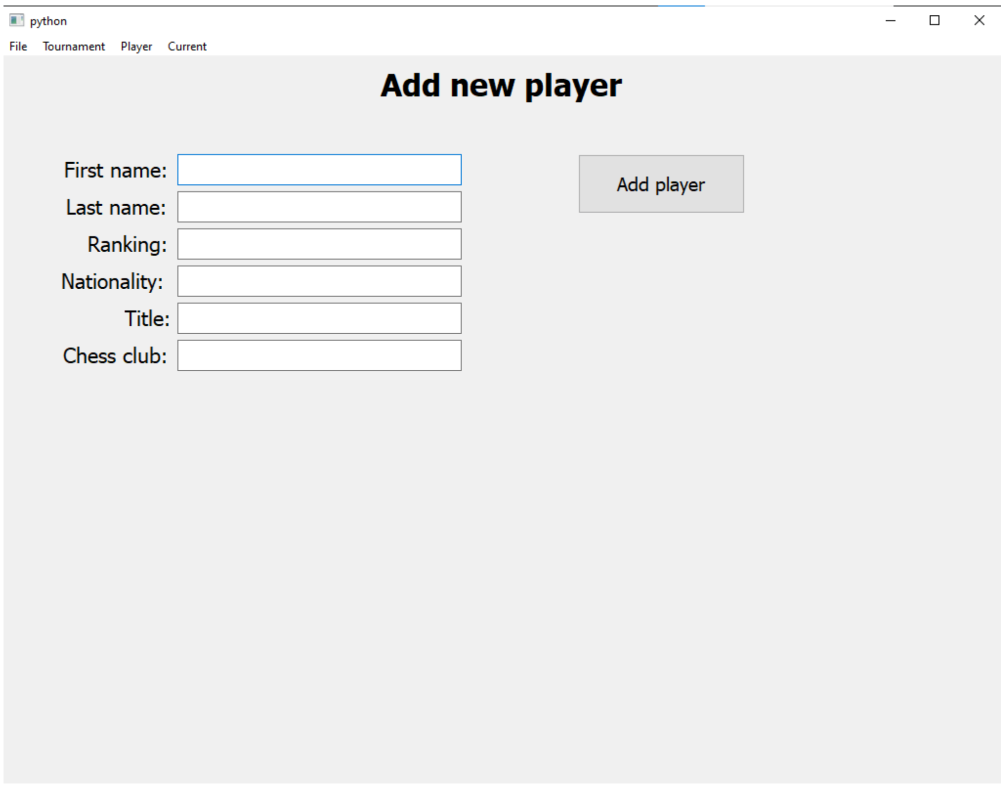
widoki w aplikacji

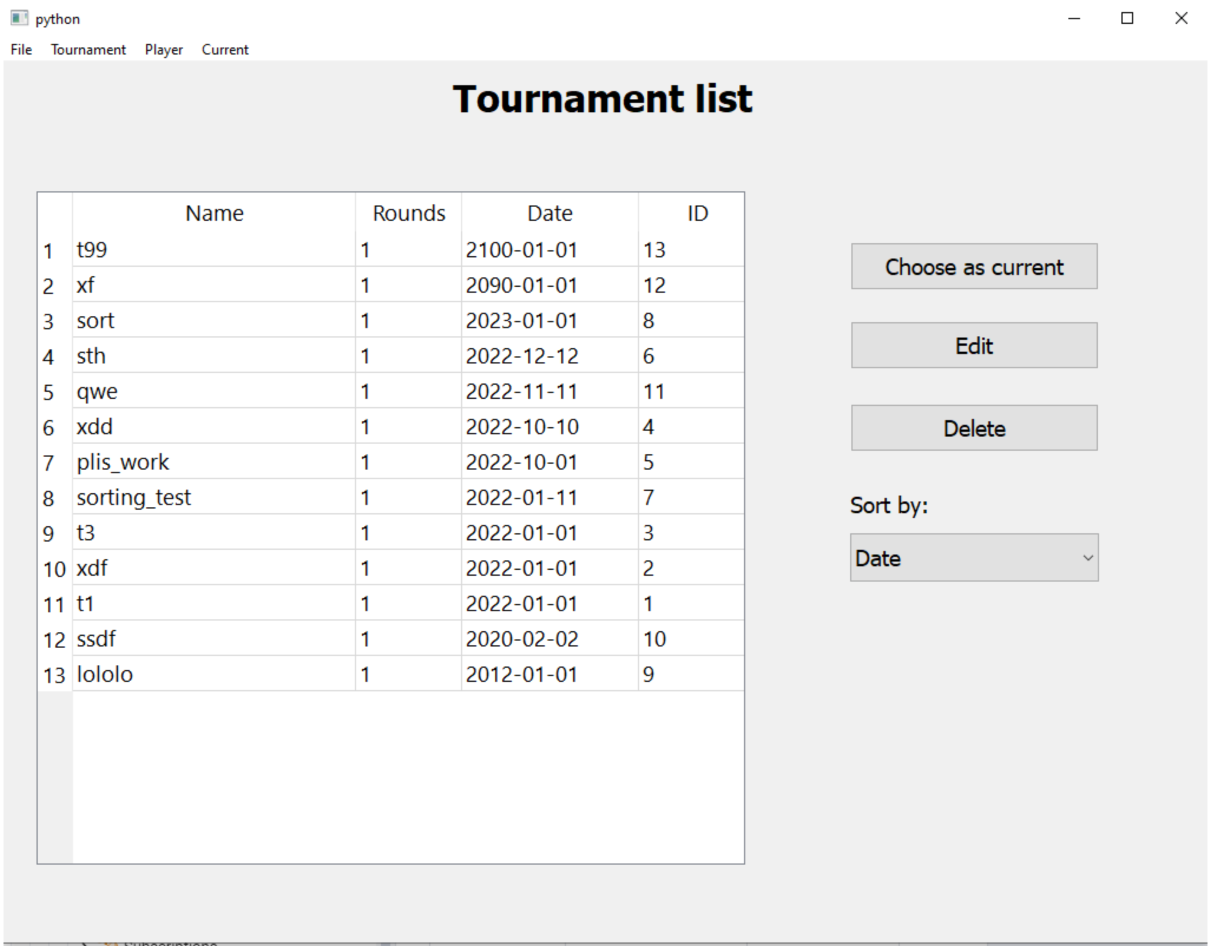
```
class CreateTournamentWidget(qtw.QWidget):
    tournament_added = qtc.pyqtSignal(str, str, int, list)

    def __init__(self, parent=None):
        super(CreateTournamentWidget, self).__init__(parent)
        self.parent = parent
        self.ui = Ui_CreateTournament()
        self.ui.setupUi(self)

        self.ui.add_player_table.setColumnCount(3)
        self.ui.add_player_table.setHorizontalHeaderLabels(('Name',
'Ranking', 'ID'))
        self.ui.add_player_table.setColumnWidth(0, 160)
        self.ui.add_player_table.setColumnWidth(1, 90)
        self.ui.add_player_table.setColumnWidth(2, 70)

self.ui.add_player_table.setHorizontalScrollBarPolicy(qtc.Qt.ScrollBarAlway
sOff)
        self.ui.delete_player_table.setColumnCount(3)
        self.ui.delete_player_table.setHorizontalHeaderLabels(('Name',
'Ranking', 'ID'))
        self.ui.delete_player_table.setColumnWidth(0, 160)
        self.ui.delete_player_table.setColumnWidth(1, 90)
        self.ui.delete_player_table.setColumnWidth(2, 70)

self.ui.delete_player_table.setHorizontalScrollBarPolicy(qtc.Qt.ScrollBarAl
waysOff)

        self.player_dao = PlayerDAO()
        self.tournamentDAO = TournamentDAO()
        self.fill_add_player_table(self.player_dao.get_all_players())

        self.list_tournament = ListTournamentsWidget


self.ui.create_tournament_button.clicked.connect(self.add_tournament)

self.ui.add_player_button.clicked.connect(self.add_player_to_tournament)

self.ui.delete_player_button.clicked.connect(self.delete_player_from_tourna
ment)
```

Graficzny interfejs aplikacji

```
def setupUi(self, EditPlayers):
    EditPlayers.setObjectName("EditPlayers")
    EditPlayers.resize(1024, 768)
    font = QtGui.QFont()
    font.setPointSize(14)
    EditPlayers.setFont(font)
    self.create_tournament_title = QtWidgets.QLabel(EditPlayers)
    self.create_tournament_title.setGeometry(QtCore.QRect(6, 6, 1011, 45))
    font = QtGui.QFont()
    font.setPointSize(24)
    font.setBold(True)
    font.setWeight(75)
    self.create_tournament_title.setFont(font)
    self.create_tournament_title.setAlignment(QtCore.Qt.AlignCenter)
    self.create_tournament_title.setObjectName("create_tournament_title")
    self.add_player_button = QtWidgets.QPushButton(EditPlayers)
    self.add_player_button.setGeometry(QtCore.QRect(90, 550, 341, 41))
    self.add_player_button.setObjectName("add_player_button")
    self.label_2 = QtWidgets.QLabel(EditPlayers)
    self.label_2.setGeometry(QtCore.QRect(690, 70, 131, 31))
    font = QtGui.QFont()
    font.setPointSize(16)
    self.label_2.setFont(font)
    self.label_2.setObjectName("label_2")
    self.delete_player_button = QtWidgets.QPushButton(EditPlayers)
    self.delete_player_button.setGeometry(QtCore.QRect(560, 550, 341, 41))
    self.delete_player_button.setObjectName("delete_player_button")
    self.save_button = QtWidgets.QPushButton(EditPlayers)
    self.save_button.setGeometry(QtCore.QRect(560, 640, 341, 51))
    self.save_button.setObjectName("save_button")
    self.label = QtWidgets.QLabel(EditPlayers)
    self.label.setGeometry(QtCore.QRect(180, 70, 156, 30))
    font = QtGui.QFont()
    font.setPointSize(16)
    self.label.setFont(font)
    self.label.setObjectName("label")
    self.not_in_tournament_table = QtWidgets.QTableWidget(EditPlayers)
    self.not_in_tournament_table.setGeometry(QtCore.QRect(90, 110, 341,
441))
    self.not_in_tournament_table.setObjectName("not_in_tournament_table")
    self.not_in_tournament_table.setColumnCount(0)
    self.not_in_tournament_table.setRowCount(0)
    self.in_tournament_table = QtWidgets.QTableWidget(EditPlayers)
    self.in_tournament_table.setGeometry(QtCore.QRect(560, 110, 341, 441))
    self.in_tournament_table.setObjectName("in_tournament_table")
    self.in_tournament_table.setColumnCount(0)
    self.in_tournament_table.setRowCount(0)

    self.retranslateUi(EditPlayers)
    QtCore.QMetaObject.connectSlotsByName(EditPlayers)
```

Aplikacja w działaniu



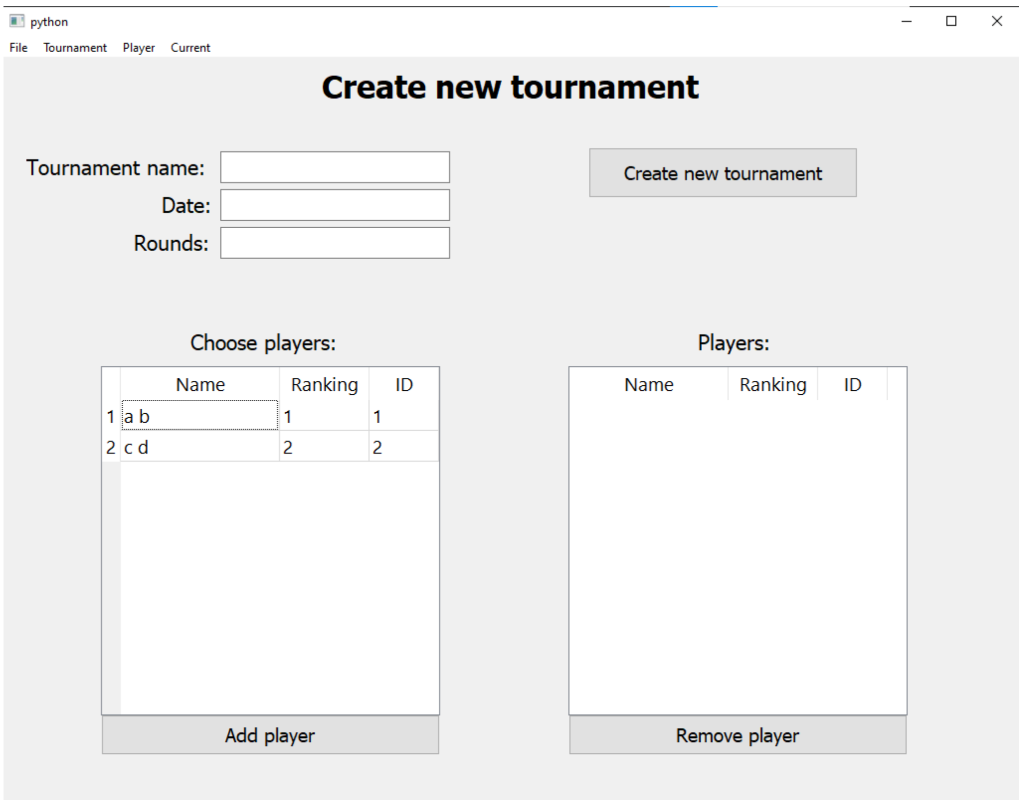


Tournament list

	Name	Rounds	Date	ID
1	t99	1	2100-01-01	13
2	xf	1	2090-01-01	12
3	sort	1	2023-01-01	8
4	sth	1	2022-12-12	6
5	qwe	1	2022-11-11	11
6	xdd	1	2022-10-10	4
7	plis_work	1	2022-10-01	5
8	sorting_test	1	2022-01-11	7
9	t3	1	2022-01-01	3
10	xdf	1	2022-01-01	2
11	t1	1	2022-01-01	1
12	ssdf	1	2020-02-02	10
13	lololo	1	2012-01-01	9

Sort by: Date

Lista turniejów możliwość sortowania po nazwie



	Name	Ranking	ID
1	a b	1	1
2	c d	2	2

Name	Ranking	ID

Utworzenie tunieju, dodanie graczy