1. using System;
2. using System.Collections;
3. using System.Collections.Generic;
4. using System.Data;
5. using System.Data.OleDb;
6. using System.Globalization;
7. using System.Linq;
8. using System.Text;
9. using System.Threading.Tasks; 10

11 namespace Nihulon2.Model.DbAccess 12 {

13 /\*

1. \* Class used for getting data from the database
2. \* Inherits from DbAccess that has connection to the DB and query methods
3. \* Realizes the singleton pattern 17 \*/

18 class DbConnector : DbAccess 19 {

1. private static string conString;
2. private static DbConnector instance; 22
3. // Flag that shows if there are time overlaps at the exams table at the DB
4. private bool \_foundOverlaps; 25

26 /\*

1. \* Event called when exams with time overlap have been found
2. \* or when all overlaps have been fixed.
3. \* Used by the view for showing and hiding the relevant controls for
4. \* fixing the overlaps

31 \*/

1. public delegate void OverlapsStateChangedHandler(bool state);
2. public event OverlapsStateChangedHandler onOverlapsStateChanged; 34
3. // Private constructor. Can't be called from other places
4. private DbConnector(string connectionstring) : base(connectionstring) 37 { }

38

39

40 #region Properties 41

1. // Return a pointer to the instance of itself
2. // create new instance if not exists
3. public static DbConnector Instance

45 {

46 get

47 {

48 if (instance == null)

49 {

50 instance = new DbConnector(conString); 51 }

52 return instance; 53 }

54 }

55

1. // keeps the string with connection attributes
2. public static string ConnectionString 58 {
3. get { return conString; }
4. set

61 {

1. // The value is the path to the file of data base
2. // can be changed at the config file
3. conString = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source="

|  |  |  |  |
| --- | --- | --- | --- |
|  | | + |  |
| 65 | value + ";Persist Security Info=False;"; | |
| 66 | } | |
| 67 | } | |
| 68 | // Flag that shows if there are exams with time overlaps at the | | DB |
| 69 | public bool foundOverlaps | |  |
| 70 | { | |  |
| 71 | get { return \_foundOverlaps; } | |  |
| 72 | set | |  |
| 73 | { | |  |
| 74 | // set value | |  |
| 75 | \_foundOverlaps = value; | |  |
| 76 | // call the event handler | |  |
| 77 | if (onOverlapsStateChanged != null) | |  |
| 78 | onOverlapsStateChanged(\_foundOverlaps); | |  |
| 79 | } | |  |
| 80 | } | |  |
| 81 | #endregion | |  |
| 82 |  | |  |
| 83 | #region Related items methods | |  |
| 84 |  | |  |
| 85 | //Takes an instance of related item and inserts it into the DB | |  |
| 86 | //into the right table according to the item's type | |  |
| 87 | public void insertRelatedItem(string itemName, string itemType) | |  |
| 88 | { | |  |
| 89 | // Prepare the query according to type of the related item | |  |
| 90 | string cmdStr = ""; | |  |
| 91 | if (itemType == "חטיבות") | |  |
| 92 | cmdStr = "INSERT INTO Divisions (division\_name) VALUES | |  |
|  | (@name)"; | |  |
| 93 |  | else if(itemType == "מגמות") | |
| 94 |  | cmdStr = "INSERT INTO Courses (course\_name) VALUES (@name)"; | |
| 95 |  | else if (itemType == "חדרים") | |
| 96 |  | cmdStr = "INSERT INTO Rooms (room\_number) VALUES (@name)"; | |
| 97 |  |  | |
| 98 |  | // If the type and name are appropriate, execute the query | |
| 99 |  | if(cmdStr != "" && !string.IsNullOrEmpty(itemName)) | |
| 100 |  | { | |
| 101 |  | using (OleDbCommand command = new OleDbCommand(cmdStr)) | |
| 102 |  | { | |
| 103 |  | command.Parameters.AddWithValue("@name", itemName); | |
| 104 |  | base.ExecuteSimpleQuery(command); | |
| 105 |  | } | |
| 106 |  | } | |
| 107 | } |  | |
| 108 |  |  | |
| 109 | // | Takes type of needed items (Division, Course, Room) | |

1. // Returns a list of needed items
2. public RelatedItem[] GetRelatedItemsByType(string itemType, bool showDisabled)

|  |  |  |
| --- | --- | --- |
| 112 | { |  |
| 113 |  | DataSet dSet = new DataSet(); |
| 114 |  | ArrayList items = new ArrayList(); |
| 115 |  | RelatedItem item; |
| 116 |  |  |
| 117 |  | // Build the command string according to the itemType |
| 118 |  | string cmdStr = getCommandStringByType(itemType); |
| 119 |  |  |
| 120 |  | if(showDisabled == false) // If we want only not disabled items |
| 121 |  | cmdStr += " WHERE disabled\_status = 0"; |
| 122 |  |  |
| 123 |  | // If the command string is initialized |
| 124 |  | // connect to DB and get data |
| 125 |  | if(cmdStr != "") |
| 126 |  | { |
| 127 |  | // Get data from DB |
| 128 |  | using (OleDbCommand command = new OleDbCommand(cmdStr)) |
| 129 |  | { |
| 130 |  | dSet = GetMultipleQuery(command); |
| 131 |  | } |
| 132 |  |  |
| 133 |  | // Get table from the data set |
| 134 |  | DataTable dt = new DataTable(); |
| 135 |  | try |
| 136 |  | { |
| 137 |  | dt = dSet.Tables[0]; |
| 138 |  | } |
| 139 |  | catch { } |
| 140 |  |  |
| 141 |  | // Get rows from the data table and fill the ArrayList with |
|  |  | items |
| 142 |  | foreach (DataRow row in dt.Rows) |
| 143 |  | { |
| 144 |  | // Create related item |
| 145 |  | item = new RelatedItem(); |
| 146 |  | // Fill the fields of the item |
| 147 |  | item.Name = row[0].ToString(); |
| 148 |  | item.IsDisabled = Convert.ToBoolean(row[1].ToString()); |
| 149 |  |  |
| 150 |  | items.Add(item); |
| 151 |  | } |
| 152 |  | return (RelatedItem[])items.ToArray(typeof(RelatedItem)); |
| 153 |  | } |
| 154 |  | return null; // If itemType is not valid |
| 155 | } |  |
| 156 |  |  |
| 157 | // | Change status of the related item to disabled / not disabled |
| 158 | public void changeStatusToRelatedItem(string name, string type) | |
| 159 | { | |
| 160 | name = name.Replace("(מבוטל)", ""); // remove the marker from the | |
|  | name | |
| 161 |  | |
| 162 | // Prepare the query according to the type of related item | |

1. // The query will invert the value of disabled\_status
2. string cmdStr = "";

165 if (type == "חטיבות")

166 cmdStr = "UPDATE Divisions SET disabled\_status = NOT disabled\_status WHERE division\_name = @name";

167 else if (type == "מגמות")

168 cmdStr = "UPDATE Courses SET disabled\_status = NOT disabled\_status WHERE course\_name = @name";

169 else if (type == "חדרים")

170 cmdStr = "UPDATE Rooms SET disabled\_status = NOT disabled\_status WHERE room\_number = @name";

171

172 if (cmdStr != "")

173 {

174 using (OleDbCommand command = new OleDbCommand(cmdStr))

175 {

176

1. command.Parameters.AddWithValue("@name", name);
2. base.ExecuteSimpleQuery(command);

179 }

180 }

181 }

182

1. // Get data from one of the tables of related items at the DB and return array with names
2. public string[] getRelatedItemsNamesByType(string relatedItemType)

185 {

1. DataSet dSet = new DataSet();
2. ArrayList names = new ArrayList();
3. string cmdStr = ""; 189

190 switch (relatedItemType)

191 {

192 case "חטיבות":

1. cmdStr = "SELECT division\_name from Divisions WHERE disabled\_status = 0";
2. break;

195 case "מגמות":

1. cmdStr = "SELECT course\_name from Courses WHERE disabled\_status = 0";
2. break;

198 case "חדרים":

1. cmdStr = "SELECT room\_number from Rooms WHERE disabled\_status = 0";
2. break;

201 }

202

203 if(cmdStr != "")

204 {

1. // Get data from DB
2. using (OleDbCommand command = new OleDbCommand(cmdStr))

207 {

208 dSet = GetMultipleQuery(command);

209 }

210

211 // Get table from the data set

|  |  |
| --- | --- |
| 212 | DataTable dt = new DataTable(); |
| 213 | try |
| 214 | { |
| 215 | dt = dSet.Tables[0]; |
| 216 | } |
| 217 | catch { } |
| 218 |  |
| 219 | // Get rows from the data table and fill the ArrayList with |

names

1. foreach (DataRow row in dt.Rows)
2. names.Add(row[0].ToString());

222 }

223 return (string[])names.ToArray(typeof(string));

224 }

225 #endregion 226

227 #region Exams methods 228

1. // Get exams according to the filters that are got as parameters
2. public List<Exam> getExams(string dateFromFilter, string dateToFilter, string divisionFilter,
3. string courseFilter, string roomFilter, bool showDisabledFilter, bool showNewFilter)

|  |  |  |
| --- | --- | --- |
| 232 | { |  |
| 233 |  | List<Exam> exams; |
| 234 |  | string cmdStr = "SELECT \* FROM Exams"; |
| 235 |  |  |
| 236 |  | // Build a complex WHERE condition according to the sent filters |

and add it to the query

1. cmdStr += bildWhereCondition(dateFromFilter, dateToFilter, divisionFilter, courseFilter,
2. roomFilter, showDisabledFilter, showNewFilter); 239
3. // If the command string is initialized
4. // connect to DB and get data
5. if (cmdStr != "")

243 {

1. exams = this.getListOfExamsFromDB(cmdStr);
2. return exams;

246 }

247 return null; // If initializing of the command string failed

248 }

249

1. // Load from the DB only the exams that have time overlaps
2. public List<Exam> getExamsWithOverlaps()

252 {

1. List<Exam> exams;
2. string cmdStr = "SELECT \* FROM Exams WHERE hasOverlap = -1"; 255
3. // If the command string is initialized
4. // connect to DB and get data
5. if (cmdStr != "")

259 {

1. exams = this.getListOfExamsFromDB(cmdStr);
2. return exams;

262 }

263 return null; // If initializing of the command string failed

264 }

265

1. // Adds a new exam to the DB
2. public void insertExam(Exam newExam)

268 {

269 string cmdStr = ""; 270

1. string date = convertDate(newExam.Date);
2. string creationDate = convertDate(newExam.dateOfCreation); 273
3. // Prepare the query
4. cmdStr = "INSERT INTO Exams " +
5. " ( exam\_date, supervisor\_name, division\_name, course\_name, group\_name, discipline\_name, room\_number, start\_time, ending\_time, canceled\_status, extratime\_status, date\_of\_creation ) " +
6. " VALUES(" + date + ", @supervisor, @division, @course, @group, @discipline, @room, @startTime, @endingTime, @isCanceled, @hasExtraTime, " + creationDate + ")";
7. // Set all parameters of the query and execute
8. using (OleDbCommand command = new OleDbCommand(cmdStr))

281 {

1. command.Parameters.AddWithValue("@supervisor", newExam.SupervisorName);
2. command.Parameters.AddWithValue("@division", newExam.division);
3. command.Parameters.AddWithValue("@course", newExam.course);
4. command.Parameters.AddWithValue("@group", newExam.GroupName);
5. command.Parameters.AddWithValue("@discipline", newExam.DisciplineName);
6. command.Parameters.AddWithValue("@room", newExam.room);
7. command.Parameters.AddWithValue("@startTime", newExam.StartTime);
8. command.Parameters.AddWithValue("@endingTime", newExam.EndingTime);
9. command.Parameters.AddWithValue("@isCanceled", (newExam.isCanceled ? 1 : 0));
10. command.Parameters.AddWithValue("@hasExtraTime", (newExam.hasExtraTime ? 1 : 0));
11. base.ExecuteSimpleQuery(command);

294 }

295 }

296

1. // Remove an exam from the DB by its ID
2. public void removeExam(int examId)

299 {

300 string cmdStr = ""; 301

1. // Prepare the query
2. cmdStr = "DELETE FROM Exams WHERE exam\_id = @id"; 304

305 if (cmdStr != "")

306 {

307 using (OleDbCommand command = new OleDbCommand(cmdStr))

308 {

1. command.Parameters.AddWithValue("@id", examId);
2. base.ExecuteSimpleQuery(command);

311 }

312 }

313 }

314

1. // Save the changed exam into the DB
2. public void updateExam(Exam changedExam)

317 {

1. string cmdStr = "";
2. string date = convertDate(changedExam.Date); 320
3. // Prepare the query
4. cmdStr = "UPDATE Exams " +
5. "SET exam\_date = " + date + ", supervisor\_name = @supervisor, division\_name = @division, course\_name = @course, group\_name

= @group, discipline\_name = @discipline, room\_number = @room, start\_time = @startTime, ending\_time = @endingTime, canceled\_status = @isCanceled, extratime\_status = @hasExtraTime, exam\_comments = @comments " +

1. "WHERE exam\_id = " + changedExam.Id; 325

326

1. // Set all parameters of the query and execute
2. using (OleDbCommand command = new OleDbCommand(cmdStr))

329 {

1. command.Parameters.AddWithValue("@supervisor", changedExam.SupervisorName);
2. command.Parameters.AddWithValue("@division", changedExam.division);
3. command.Parameters.AddWithValue("@course", changedExam.course);
4. command.Parameters.AddWithValue("@group", changedExam.GroupName);
5. command.Parameters.AddWithValue("@discipline", changedExam.DisciplineName);
6. command.Parameters.AddWithValue("@room", changedExam.room);
7. command.Parameters.AddWithValue("@startTime", changedExam.StartTime);
8. command.Parameters.AddWithValue("@endingTime", changedExam.EndingTime);
9. command.Parameters.AddWithValue("@isCanceled", (changedExam.isCanceled ? 1 : 0));
10. command.Parameters.AddWithValue("@hasExtraTime", (changedExam.hasExtraTime ? 1 : 0));
11. command.Parameters.AddWithValue("@comments", changedExam.Comments);
12. base.ExecuteSimpleQuery(command);

|  |  |  |
| --- | --- | --- |
| 343 |  | } |
| 344 | } |  |
| 345 |  |  |
| 346 | // | Takes an array of strings with id, supervisor name and comment |
| 347 | // | Inserts the name of supervisor and adds the comment to the exam at |

the DB

348 public void updateExamFromExcel(string[] examData)

349 {

1. int id;
2. // If the id is valid
3. if(Int32.TryParse(examData[0], out id))

353 {

1. string supervisor = examData[1];
2. string newComment = examData[2]; 356
3. // Prepare the query
4. string cmdStr = "UPDATE Exams " +
5. "SET supervisor\_name = @supervisor, exam\_comments = @comments " +
6. "WHERE exam\_id = " + id; 361
7. // Set all parameters of the query and execute
8. using (OleDbCommand command = new OleDbCommand(cmdStr))

364 {

1. command.Parameters.AddWithValue("@supervisor", supervisor);
2. command.Parameters.AddWithValue("@comments", newComment); 367

368 base.ExecuteSimpleQuery(command);

369 }

370 }

371 }

372

1. // Finds the exams with time overlaps and marks them at the DB
2. // by setting the flag hasOverlap
3. public void markExamsWithOverlap()

376 {

1. Exam[] examsWithSameRoom;
2. Exam[] examsWithSameSupervisor;
3. bool hasOverlapsWithRoom = false, hasOverlapsWithSupervisor = false;
4. // Clear all hasOverlap flags at the DB
5. this.clearAllOverlapFlags(); 383
6. // Get all exams with potential overlaps
7. // (The exams that have the same room or supervisor at the same day
8. examsWithSameRoom = this.getSameRoomExams();
9. examsWithSameSupervisor = this.getSameSupervisorExams(); 388
10. // Check the potential overlaps whether there are the real overlaps
11. // and mark all overlapped exams at the DB
12. if (examsWithSameRoom.Length > 1 && examsWithSameRoom != null)
13. hasOverlapsWithRoom = this.checkAndMarkRoomOverlaps (examsWithSameRoom);
14. if (examsWithSameSupervisor.Length > 1 && examsWithSameSupervisor != null)
15. hasOverlapsWithSupervisor = this.checkAndMarkSupervisorOverlaps

(examsWithSameSupervisor);

1. // If found any time overlaps, set the flag "foundOverlaps"
2. if (hasOverlapsWithRoom || hasOverlapsWithSupervisor)
3. this.foundOverlaps = true;
4. else
5. this.foundOverlaps = false;

401 }

402

403 #endregion 404

405 #region Private methods for internal use 406

407 // Returns a command string with SELECT from one of the tables

408 // of related items (Division, Course, Room) according to the itemType

409 private string getCommandStringByType(string itemType)

410 {

411 string cmdStr;

412 switch (itemType)

413 {

414 case "חטיבות":

415 cmdStr = "SELECT \* FROM [Divisions]";

416 break;

417 case "מגמות":

418 cmdStr = "SELECT \* FROM [Courses]";

419 break;

420 case "חדרים":

1. cmdStr = "SELECT \* FROM [Rooms]";
2. break;
3. default:
4. cmdStr = "";
5. break;

426 }

427 return cmdStr;

428 }

429

430

431 // Build WHERE condition according to the parameters

432 private string bildWhereCondition(string dateFromFilter, string dateToFilter, string divisionFilter, string courseFilter, string roomFilter, bool showDisabledFilter, bool showNewFilter)

|  |  |  |
| --- | --- | --- |
| 433 | { |  |
| 434 |  | string str = " WHERE "; |
| 435 |  |  |
| 436 |  | // Flag that shows if there is any condition |
| 437 |  | // if no conditions, the method returns empty string |
| 438 |  | bool whereIsSet = false; |
| 439 |  |  |
| 440 |  | if(dateFromFilter != "") // if the period of time filter is set |
| 441 |  | { |

442 // Convert dates to the correct form for the query (ex. 12.07.2019 -> #7/12/2019#)

443 dateFromFilter = convertDate(dateFromFilter);

444 dateToFilter = convertDate(dateToFilter); 445

446 // Add the condition to the WHERE and set the flag that WHERE

|  |  |  |
| --- | --- | --- |
|  |  | is set |
| 447 | str += "exam\_date >= " + dateFromFilter + " AND exam\_date <= " | |
|  | + dateToFilter; | |
| 448 | whereIsSet = true; | |
| 449 | } | |
| 450 | if(divisionFilter != "הכול") // if the division filter is set | |
| 451 | { | |
| 452 | if (whereIsSet) // if there is at least one condition before, | |
|  | add "AND" to expression | |
| 453 |  | str += " AND "; |
| 454 |  | str += " division\_name = \"" + divisionFilter + "\""; |
| 455 |  | whereIsSet = true; |
| 456 | } |  |
| 457 | if | (courseFilter != "הכול") // if the course filter is set |
| 458 | { |  |
| 459 |  | if (whereIsSet) // if there is at least one condition before, |
|  |  | add "AND" to expression |
| 460 |  | str += " AND "; |
| 461 |  | str += " course\_name = \"" + courseFilter + "\""; |
| 462 |  | whereIsSet = true; |
| 463 | } |  |
| 464 | if | (roomFilter != "הכול") // if the room filter is set |
| 465 | { |  |
| 466 |  | if (whereIsSet) // if there is at least one condition before, |
|  |  | add "AND" to expression |
| 467 | str += " AND "; | |
| 468 | str += " room\_number = \"" + roomFilter + "\""; | |
| 469 | whereIsSet = true; | |
| 470 | } | |
| 471 | if(showDisabledFilter == false) // If disabled exams are going to | |

be shown

472 {

473 if (whereIsSet) // if there is at least one condition before, add "AND" to expression

474 str += " AND ";

475 str += " canceled\_status = 0";

476 whereIsSet = true;

477 }

478 if (showNewFilter == true) // If only new exams are going to be shown

479 {

1. if (whereIsSet) // if there is at least one condition before, add "AND" to expression
2. str += " AND ";
3. string date = convertDate(DateTime.Today.ToString ("dd.MM.yyyy"));
4. str += " date\_of\_creation = " + date;
5. whereIsSet = true;

485 }

486

1. // If there is no WHERE conditions, return an empty string
2. if (whereIsSet)
3. return str;
4. else
5. return "";

|  |  |
| --- | --- |
| 492 | } |
| 493 |  |
| 494 | // Convert a date to the correct form for the query (12.07.2019 -> |

#7/12/2019#)

495 private string convertDate(string date)

496 {

497

498 string[] partsOfDate = date.Split('.');

499 string newDate = "#" + partsOfDate[1] + "/" + partsOfDate[0] + "/"

+ partsOfDate[2] + "#";

500

501 return newDate;

502 }

503

504 // Get the exams with the same room at the same day from the DB

505 private Exam[] getSameRoomExams()

506 {

507 List<Exam> exams; 508

1. string cmdStr = "SELECT \* FROM Exams INNER JOIN " +
2. "(SELECT exam\_date, room\_number, COUNT(\*) AS occurrences FROM Exams " +
3. "WHERE canceled\_status = 0 GROUP BY exam\_date, room\_number HAVING COUNT(\*) > 1) AS t1 " +
4. "ON(t1.exam\_date = Exams.exam\_date) AND (t1.room\_number = Exams.room\_number) " +
5. "ORDER BY Exams.room\_number, Exams.exam\_date"; 514

515 // Get data from DB

516 exams = this.getListOfExamsFromDB(cmdStr);

517 return exams.ToArray();

518 }

519 // Get the exams with the same supervisor at the same day from the DB

|  |  |
| --- | --- |
| 520 | private Exam[] getSameSupervisorExams() |
| 521 | { |
| 522 | List<Exam> exams; |
| 523 |  |
| 524 | string cmdStr = "SELECT \* FROM Exams INNER JOIN " + |
| 525 | "(SELECT exam\_date, supervisor\_name, COUNT(\*) AS |

occurrences FROM Exams " +

526 "WHERE (canceled\_status = 0) AND(supervisor\_name

<> \"\") " +

527 "GROUP BY exam\_date, supervisor\_name HAVING COUNT (\*) > 1) AS t1 " +

528 "ON(t1.exam\_date = Exams.exam\_date) AND (t1.supervisor\_name = Exams.supervisor\_name) " +

529 "ORDER BY Exams.supervisor\_name, Exams.exam\_date"; 530

531 // Get data from DB

532 exams = this.getListOfExamsFromDB(cmdStr);

533 return exams.ToArray();

534 }

535

536 // Sets the flag "hasOverlap" to 0 for all exams

537 private void clearAllOverlapFlags()

538 {

539 // Set command string

540 string cmdStr = "UPDATE Exams SET hasOverlap = 0 " +

541 "WHERE hasOverlap <> 0";

542

543 // Execute

544 using (OleDbCommand command = new OleDbCommand(cmdStr))

545 {

546 base.ExecuteSimpleQuery(command);

547 }

548 }

549

1. // Takes the array of exams that has the same room at the same day
2. // and checks whether there are time overlaps by using the formula:
3. // if aStart < bEnd AND bStart < aEnd => there is an overlap between a and b
4. // If found, marks the couple of overlapped exams at the DB
5. private bool checkAndMarkRoomOverlaps(Exam[] exams)

555 {

556 bool hasOverlaps = false; 557

558 // when the flag "sameRoom" is false, no need to check

559 // the current exam with others because all exams sorted by rooms

560 // makes the loop faster

561 bool sameRoom; 562

563 for (int i = 0; i < exams.Length - 1; i++)

564 {

565 sameRoom = true;

566 for(int j = i+1; j < exams.Length && sameRoom; j++)

567 {

568 // if the exams have the same room or supervisor, and the date

569 if (exams[i].Date == exams[j].Date && exams[i].room == exams[j].room)

570 {

571 // Check the couple of exams and if thy are overlapped, mark them at the DB

572 // and set the flag

573 if (this.checkAndMarkTwoExamsIfOverlapped(exams[i], exams[j]) && hasOverlaps == false)

|  |  |  |  |
| --- | --- | --- | --- |
| 574 |  |  | hasOverlaps = true; |
| 575 |  |  | } |
| 576 |  |  | else |
| 577 |  |  | sameRoom = false; |
| 578 |  | } |  |
| 579 |  | } |  |
| 580 |  | return | hasOverlaps; |
| 581 | } |  |  |

582 // Takes the array of exams that has the same supervisor at the same day

583 // and checks whether there are time overlaps.

584 // If found, marks the overlapped exams at the DB

585 private bool checkAndMarkSupervisorOverlaps(Exam[] exams)

586 {

587 bool hasOverlaps = false; 588

589 // when the flag "sameSupervisor" is false, no need to check

590 // the current exam with others because all exams sorted by supervisor

|  |  |
| --- | --- |
| 591 | // makes the loop faster |
| 592 | bool sameSupervisor; |
| 593 |  |
| 594 | for (int i = 0; i < exams.Length - 1; i++) |
| 595 | { |
| 596 | sameSupervisor = true; |
| 597 | for (int j = i + 1; j < exams.Length && sameSupervisor; j++) |
| 598 | { |
| 599 | // if the exams have the same room or supervisor, and the |
| 600 | date  if (exams[i].Date == exams[j].Date && exams |
| 601 | [i].SupervisorName == exams[j].SupervisorName)  { |
| 602 | // Check the couple of exams and if thy are |

overlapped, mark them at the DB

603 // and set the flag

604 if (this.checkAndMarkTwoExamsIfOverlapped(exams[i], exams[j]) && hasOverlaps == false)

605 hasOverlaps = true;

606 }

607 else

608 sameSupervisor = false; 609 }

610 }

611 return hasOverlaps;

612 }

613

614 // Set the flag "hasOverlap" at the DB for the couple of overlapped exams

615 private void markOverlap(int id1, int id2) 616 {

617 string cmdStr = "UPDATE Exams SET hasOverlap = -1 " + 618 "WHERE exam\_id = @id1 OR exam\_id = @id2"; 619

620 // Set all parameters of the query and execute

621 using (OleDbCommand command = new OleDbCommand(cmdStr)) 622 {

623 command.Parameters.AddWithValue("@id1", id1);

624 command.Parameters.AddWithValue("@id2", id2);

625 base.ExecuteSimpleQuery(command);

626 }

627 }

628

629

630 /\*

631 \* Fills an exam with data from the data row 632 \* and returns the exam

633 \*/

634 private Exam getExamFromDataRow(DataRow row) 635 {

636 Exam exam = new Exam(); 637

638 // Get time and date

|  |  |  |
| --- | --- | --- |
| 639 |  | DateTime dateT; |
| 640 |  | // Get the date of the exam |
| 641 |  | try |
| 642 |  | { |
| 643 |  | dateT = Convert.ToDateTime(row[1].ToString()); |
| 644 |  | exam.Date = dateT.ToString("dd.MM.yyyy"); |
| 645 |  | } |
| 646 |  | catch { } |
| 647 |  | // Get the time of the start |
| 648 |  | try |
| 649 |  | { |
| 650 |  | dateT = Convert.ToDateTime(row[8].ToString()); |
| 651 |  | exam.StartTime = dateT.ToShortTimeString(); |
| 652 |  | } |
| 653 |  | catch { } |
| 654 |  | // Get the time of the end |
| 655 |  | try |
| 656 |  | { |
| 657 |  | dateT = Convert.ToDateTime(row[9].ToString()); |
| 658 |  | exam.EndingTime = dateT.ToShortTimeString(); |
| 659 |  | } |
| 660 |  | catch { } |
| 661 |  | // Get the date of creation |
| 662 |  | try |
| 663 |  | { |
| 664 |  | dateT = Convert.ToDateTime(row[13].ToString()); |
| 665 |  | exam.dateOfCreation = dateT.ToShortDateString(); |
| 666 |  | } |
| 667 |  | catch { } |
| 668 |  |  |
| 669 |  | // Fill all fields of the exam with values |
| 670 |  | exam.Id = (int)row[0]; |
| 671 |  | exam.SupervisorName = row[2].ToString(); |
| 672 |  | exam.division = row[3].ToString(); |
| 673 |  | exam.course = row[4].ToString(); |
| 674 |  | exam.GroupName = row[5].ToString(); |
| 675 |  | exam.DisciplineName = row[6].ToString(); |
| 676 |  | exam.room = row[7].ToString(); |
| 677 |  | exam.isCanceled = Convert.ToBoolean(row[10].ToString()); |
| 678 |  | exam.hasExtraTime = Convert.ToBoolean(row[11].ToString()); |
| 679 |  | exam.Comments = row[12].ToString(); |
| 680 |  | exam.hasOverlap = Convert.ToBoolean(row[14].ToString()); |
| 681 |  |  |
| 682 |  | return exam; |
| 683 | } |  |
| 684 |  |  |
| 685 | /\* |  |
| 686 | \* | Gets exams according to the command string, |
| 687 | \* | fills each exam with data, builds a list with exams |
| 688 | \* | end returns the list |
| 689 | \*/ |  |
| 690 | private List<Exam> getListOfExamsFromDB(string cmdStr) | |
| 691 | { | |
| 692 | List<Exam> exams = new List<Exam>(); | |
| 693 | DataSet dSet = new DataSet(); | |
| 694 | Exam exam; | |

|  |  |  |
| --- | --- | --- |
| 695 |  | // Get data from DB |
| 696 |  | using (OleDbCommand command = new OleDbCommand(cmdStr)) |
| 697 |  | { |
| 698 |  | dSet = GetMultipleQuery(command); |
| 699 |  | } |
| 700 |  |  |
| 701 |  | // Get table from the data set |
| 702 |  | DataTable dt = new DataTable(); |
| 703 |  | try |
| 704 |  | { |
| 705 |  | dt = dSet.Tables[0]; |
| 706 |  | } |
| 707 |  | catch { } |
| 708 |  |  |
| 709 |  | // Get rows from the data table and fill the ArrayList with items |
| 710 |  | foreach (DataRow row in dt.Rows) |
| 711 |  | { |
| 712 |  | // Create an exam |
| 713 |  | exam = new Exam(); |
| 714 |  | // Fill the exam with data from the row |
| 715 |  | exam = this.getExamFromDataRow(row); |
| 716 |  | // Add the new exam to the array |
| 717 |  | exams.Add(exam); |
| 718 |  | } |
| 719 |  | return exams; |
| 720 |  | } |
| 721 |  |  |
| 722 |  | // Takes two exams as parameters and |
| 723 |  | // checks whether there are time overlaps by using the formula: |
| 724 |  | // if aStart < bEnd AND bStart < aEnd => there is an overlap between a |
| 725 |  | and b  // If found overlap, mark the couple of exams at the DB |
| 726 |  | private bool checkAndMarkTwoExamsIfOverlapped(Exam examA, Exam examB) |
| 727 |  | { |
| 728 |  | DateTime aStart, aEnd, bStart, bEnd; |
| 729 |  |  |
| 730 |  | // Convert time from strings to DateTime for comparing |
| 731 |  | aStart = Convert.ToDateTime(examA.StartTime); |
| 732 |  | aEnd = Convert.ToDateTime(examA.EndingTime); |
| 733 |  | bStart = Convert.ToDateTime(examB.StartTime); |
| 734 |  | bEnd = Convert.ToDateTime(examB.EndingTime); |
| 735 |  |  |
| 736 |  | // Compare the time of the start and the end of the exams |
| 737 |  | if (aStart <= bEnd && bStart <= aEnd) |
| 738 |  | { |
| 739 |  | // If found overlap, mark the exams at the DB |
| 740 |  | this.markOverlap(examA.Id, examB.Id); |
| 741 |  | // Set the flag that overlap has been found |
| 742 |  | return true; |
| 743 |  | } |
| 744 |  | else |
| 745 |  | return false; |
| 746 |  | } |
| 747 |  |  |
| 748 |  | #endregion |
| 749 | } |  |

750 }

751