Contents

[1 Purpose 3](#_Toc526502809)

[2 Requirements 3](#_Toc526502810)

[2.1 Required “Seed” Volumes 3](#_Toc526502811)

[2.2 Required Final Volumes 4](#_Toc526502812)

[3 Primary Tables 4](#_Toc526502813)

[4 Secondary Tables 4](#_Toc526502814)

[4.1 Example 5](#_Toc526502815)

[5 Table Classification 6](#_Toc526502816)

[6 Design Guidelines 8](#_Toc526502817)

[7 Types of Data Loads 9](#_Toc526502818)

[7.1 Initial Data 9](#_Toc526502819)

[7.2 Demo Data. 9](#_Toc526502820)

[7.3 Test Data 9](#_Toc526502821)

[7.3.1 Functional Test 9](#_Toc526502822)

[7.3.2 Performance Test Data 9](#_Toc526502823)

[7.4 Client Data 11](#_Toc526502824)

[8 Procedure for data import 11](#_Toc526502825)

[8.1 GetCertified data import 11](#_Toc526502826)

[8.1.1 Importing project into eclipse 11](#_Toc526502827)

[8.1.2 Import via Eclipse 15](#_Toc526502828)

[8.2 Loading data – setting up arguments and environment 18](#_Toc526502829)

[8.2.2 Import via batch script 23](#_Toc526502830)

[8.3 Classroom2020 data 25](#_Toc526502831)

[8.4 GetCertified data 25](#_Toc526502832)

[9 Business Rules 26](#_Toc526502833)

[9.1 Quiz Definition – Unique Quiz – Quiz Instance – Quiz Question - Question 26](#_Toc526502834)

[9.2 Question Types 27](#_Toc526502835)

[9.2.1 Single choice question 28](#_Toc526502836)

[9.2.2 Multiple choice question 28](#_Toc526502837)

[9.2.3 Essay 28](#_Toc526502838)

[9.2.4 Ordering 29](#_Toc526502839)

[9.2.5 Fill in the blanks 29](#_Toc526502840)

[9.2.6 True False 30](#_Toc526502841)

[9.2.7 Matching 30](#_Toc526502842)

[10 Verifying created data 30](#_Toc526502843)

[11 List of Excel files that can be loaded into GC & CL2020 31](#_Toc526502844)

# Purpose

Purpose of the test data is to have a comprehensive set of data for:

1. Unit and integration tests (to verify functionalities of the system)
2. Load and Stress testing (to determine capacity and limits of the system)
3. Application demos

Development and testing without good quality data is either impossible or extremely inefficient and. For that reason neither activity should be performed on any level other than design before procedure for data load is established.

# Requirements

1. Test data has to cover all functional scenarios
2. Volume of test data should be realistic, for example it should have a number of students, tests, etc. One of criteria of test data quality is resemblance of the realistic situation.
3. Procedure has to be
   1. automated
   2. simple to use
   3. repeatable
   4. documented
4. All code and data required to create test data should be checked into SVN
5. Data load program has to ensure that the target database is empty (or clean) before data load commences.

## Required “Seed” Volumes

Seed data is loaded from Excel spreadsheets and later used to generate large volume of additional data using various algorithms (example will be given below in this document).

| **#** | **Data** | **Minimal** | **Comment** |
| --- | --- | --- | --- |
|  | Users | 1,000 |  |
|  | Tests | 0 | Will be created manually (several Quiz Definitions) |
|  | Test Instances | 0 | Will be created using Quiz Generator using predefined Quiz Definitions |
|  | Questions | 10,000 |  |
|  | Question pools | 100 |  |
|  | Answers | 30,000 |  |

## Required Final Volumes

Final data is created by multiplying seed data. Multiplication should be done using SQL and/or PL/SQL (could be executed from Java or shell scripts).

| **#** | **Data** | **Minimal** | **Max** | **Comment** |
| --- | --- | --- | --- | --- |
|  | Users | 1,000 | 100,000,000 |  |
|  | Tests | 1,000 | 100,000 |  |
|  | Test Instances | 100,000 | 500,000,000 |  |
|  | Questions | 100,000 | 10,000,000 |  |
|  | Question pools | 1,000 | 100,000 |  |
|  | Answers | 600,000 | 10,000,000 |  |
|  | Paper based tests (PDF files) | 100,000 |  |  |

# Primary Tables

Data in Primary tables have to be loaded from spreadsheets. This data has to be realistic and high quality since it will be used as a “seed” for generating additional data in Primary tables and all data in Secondary tables.

It is desirable that as much data as possible is created by loading from spreadsheets. Ideally we should try to move as many tables to this group.

# Secondary Tables

Values in Secondary tables could be generated using values from Primary tables and certain algorithm.

For example:

Primary tables USER and CLASS are loaded from the spreadsheets.

Data in Secondary table “USER\_CLASS” could be generated by using data in Primary tables. The simplest scenario is: “link all users with all classes”.

Various techniques such as random code generator could be used to enroll certain users in certain classes so test data is more realistic.

Typically we use PL/SQL to generated data in Secondary tables.

One such program is:

\youtestme\db\scripts\test\_data\_load\pkg\_test\_data\_load.sql

## Example

Data in Secondary table “QUIZ\_INSTANCE” could be generated using insert statement:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* QUIZ\_INSTANCE

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

insert into quiz\_instance (quiz\_instance\_id, unique\_quiz\_id, user\_id\_tested,

quiz\_instance\_status, activation\_datetime,

enabled\_from, enabled\_to, course\_id, class\_id,

resolve\_type)

select s\_quiz\_instance.nextval

,uq.unique\_quiz\_id

,us.user\_id

,decode(mod(uq.unique\_quiz\_id, 6)

,0 , 'ATN'

,1 , 'ATR'

,2 , 'EXP'

,3 , 'NAT'

,4 , 'QSS'

,5 , 'QDL'

,'NAT' )

,to\_date('2014-01-23', 'YYYY-MM-DD')

,to\_date('2014-01-16', 'YYYY-MM-DD')

,to\_date('2017-12-31', 'YYYY-MM-DD')

,1 -- course\_id

,1 -- class\_id

,qd.resolve\_type

from unique\_quiz uq

,quiz\_definition qd

,users us

where uq.quiz\_definition\_id = qd.quiz\_definition\_id

and mod(us.user\_id, 30) + mod(uq.unique\_quiz\_id, 30) between 8 and 16

and (mod(us.user\_id, 4) + mod(uq.unique\_quiz\_id, 5)) between 1 and 5

and us.user\_id <= 100 -- affect only first 100 users

;

Insert Statement Coding Guidelines:

1. Query has to join and use key information from parent tables where foreign key is mandatory (mandatory parent).
2. It is desirable to join and use data from tables where parent is optional.
3. In this query we are generating data based on pre-defined business rules. For more information see paragraph “Business Rules”.

# Table Classification

| # | TABLE NAME | Primary or Secondary | Initial Load | Comment |
| --- | --- | --- | --- | --- |
| 1. | ADDRESS | P |  |  |
| 2. | ANSWER | P |  |  |
| 3. | CLASS | P |  |  |
| 4. | CLASS\_COURSE | S |  |  |
| 5. | CODE | P | Y |  |
| 6. | CODE\_TYPE | P | Y |  |
| 7. | COURSE | P |  |  |
| 8. | DOCUMENT | P |  |  |
| 9. | EMAIL\_SENT | S |  |  |
| 10. | EMAIL\_TEMPLATE | P |  |  |
| 11. | FUNCTION | P | Y |  |
| 12. | GRADING\_SYSTEM | P |  |  |
| 13. | GRADING\_SYSTEM\_LABEL | P |  |  |
| 14. | LANGUAGE | P | Y |  |
| 15. | ORGANIZATION | P |  |  |
| 16. | POOL | P |  |  |
| 17. | POOL\_QUESTION | S |  |  |
| 18. | QUESTION | P |  |  |
| 19. | QUIZ\_CATEGORY | P |  |  |
| 20. | QUIZ\_DEFINITION | P |  |  |
| 21. | QUIZ\_DEFINITION\_POOL | S |  |  |
| 22. | QUIZ\_DEFINITION\_QUESTION | S |  |  |
| 23. | QUIZ\_EMAIL\_ACTION | S |  |  |
| 24. | QUIZ\_INSTANCE | S |  |  |
| 25. | QUIZ\_QUESTION | S |  |  |
| 26. | QUIZ\_RESULT | S |  |  |
| 27. | QUIZ\_RESULT\_ANSWER | S |  |  |
| 28. | QUIZ\_TEMPLATE | P |  |  |
| 29. | QUIZ\_TEMPLATE\_POOL | S |  |  |
| 30. | REPORT\_DEFINITION | P |  |  |
| 31. | REPORT\_INSTANCE | S |  |  |
| 32. | ROLE | P | Y |  |
| 33. | ROLE\_FUNCTION | P | Y |  |
| 34. | ST\_ANSWER | N/A |  |  |
| 35. | ST\_COURSE | N/A |  |  |
| 36. | ST\_QUESTION | N/A |  |  |
| 37. | ST\_QUIZ | N/A |  |  |
| 38. | ST\_QUIZ\_CATEGORY | N/A |  |  |
| 39. | SURVEY\_DEFINITION | N/A |  |  |
| 40. | SURVEY\_INSTANCE | N/A |  |  |
| 41. | SURVEY\_QUESTION | N/A |  |  |
| 42. | SYSTEM\_PARAMETERS | P |  |  |
| 43. | TIME\_ZONE | P | Y |  |
| 44. | UNIQUE\_QUIZ | S |  |  |
| 45. | USERS | P | Y |  |
| 46. | USERS\_CLASS | S |  |  |
| 47. | USERS\_POOLS | S |  |  |
| 48. | USER\_ACTION\_LOG | S |  |  |
| 49. | USER\_COURSE | S |  |  |
| 50. | USER\_PREFERENCES | P |  |  |
| 51. | USER\_QUIZ | S |  |  |
| 52. | USER\_ROLE | P | Y |  |

# Design Guidelines

Trebalo bi prevesti na engleski ...

Ovako bi bi bilo najbolje da se uradi:

* Loader treba da postoji samo jedan.
* Prametar treba da postoji samo jedan (konfig/property  file kao sto je originalno napravljeno).
* Moze da ima vise Java wrapper-a, skripti i konfiguracionih fajlova. Oni omogucajaju da se izvrsi nekoliko stvari u logicnom redosledu. Na primer load SMTSA pitanja, pa onda load dodatnih pitanja da "obogate" podatke sa onim sto nema u SMTSA podacima pa na kraju multiplikaor podataka koji ce od recimo 10,000 pitanja napraviti 1Milion (konacan broj treba takodje da bude parametar u konfiguracionom file-u)

Na primer:

1. skripta "load\_smtsa.bat" bi zvala loader sa parametrom za SMTSA podatke
2. skripta "load\_smtsa\_perf\_data.bat" moze da zove Java wrapper koji zove loader za SMTSA  (isto kao pod #1) pa onda multiplikator podataka

Ovih "bat" skripti mozemo da imamo koliko hoces, takodje i konfiguracionih fileova. Cak ljudi koji ih cesto koriste mogu da stave svoje ime u ime fajla da se zna da su specifino njihovi (loaduju podatke koje njima trebaju u njihove baze). Na primer:

* "load\_smtsa\_perf\_danilo\_1.bat"
* "load\_smtsa\_perf\_milos\_3.bat"
* itd.

Slicno i za konfiguracione fajlove. Java loader treba da bude samo jedan osim ako ne postoji specifican razlog zasto mora dva ili vise (ja ne vidim trenutno ni jedan jer su Spreadheetovi standardnog formata). Jedini razlog bi mogao da bude nova verzija koja nije kompatibilna sa starim formatima Excel spreadsheetova.

U svakom slucaju nikako ne smemo da dupliciramo code, to ce samo dovesti do totalne konfuzije i na kraju kad vise niko ne bude mogao da pohvata gde je sta, pocecemo da pisemo sve ispocetka.

Primer kako ne treba raditi je to odvajanje srpskog od engleskog loader-a.

# Types of Data Loads

## Initial Data

This data is loaded into the empty database to be distributed to the clients

Initial data will always have only Primary tables meaning that data is never generated.

## Demo Data.

This data is used to perform demos on application

## Test Data

Test data is used for:

1. Functional tests
2. Load (Performance) tests

### Functional Test

Data used for functional testing. Demo data and functional test data could be same or very similar.

### Performance Test Data

1. Load Test data consists of large amount of test data used for performance testing.
2. Most of the data is generated. Number of records is some tables are in millions.

Scripts that generate large amount of data is located here:

\Development\youtestme\db\scripts\test\_data\_load\performance\_test\_data\

| **#** | **File name and location** | **Description** |
| --- | --- | --- |
|  | \youtestme \db\scripts\test\_data\_load\performance\_test\_data\run\_cr\_database.bat | Implements database model. All tables are empty after this script is executed. |
|  | \youtestme\batch\_source\scripts\data\_load\LoadPerformanceTestData.bat |  |
|  | \youtestme\db\scripts\test\_data\_load\performance\_test\_data\run\_create\_vlume\_data\_sp.bat |  |
|  | \youtestme\db\scripts\test\_data\_load\performance\_test\_data\run\_create\_volume\_data\_sp.bat |  |
|  | \youtestme\db\scripts\test\_data\_load\performance\_test\_data\execute\_sp.sql |  |
|  | \youtestme\db\dbmodel\packages\pkg\_perf\_test\_data.sql |  |

## Client Data

1. This is data provided by particular client.
2. It is always sources from the files or other databases.
3. It is never generated.
4. Programs for this load could be unique for every client.
5. Client data cannot be used for demos or testing outside client’s project unless approved by the client.
6. Import specific data types

# Procedure for data import

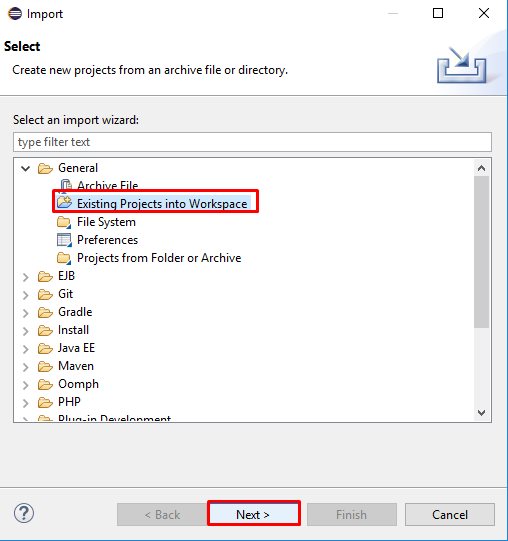
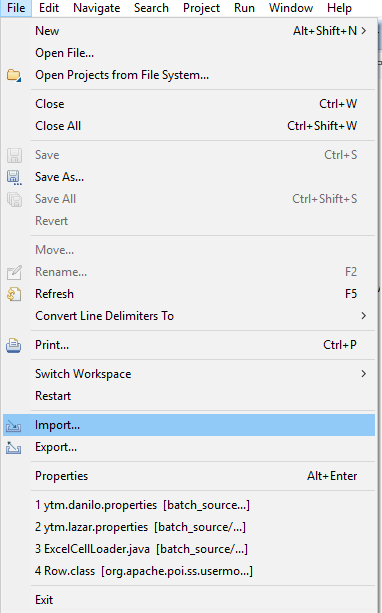
## GetCertified data import

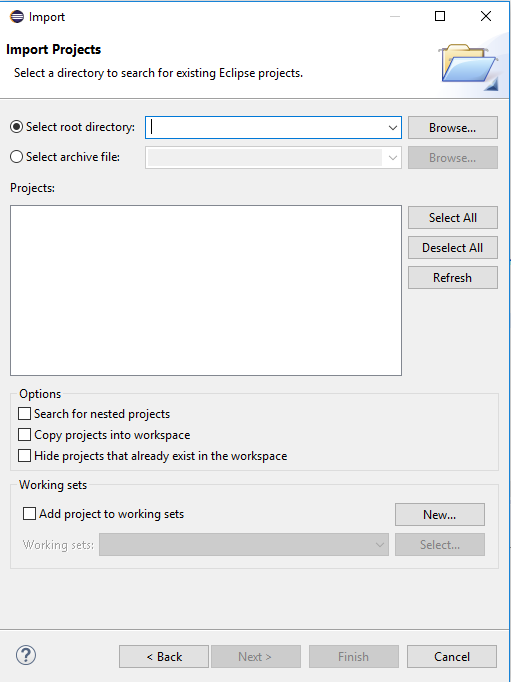
### Importing project into eclipse

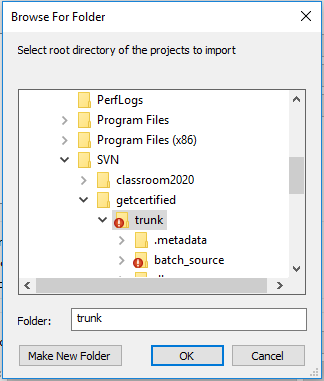
During that process Eclipse will automatically create "bin" directory with compiled java classes required to run loader program.

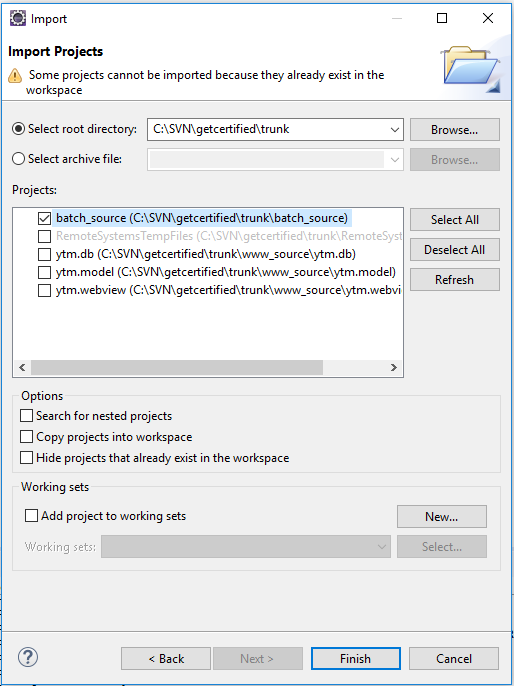
Before Importing, make sure that you updated trunk folder.

In order to import batch project, do the following:

* Open Eclipse , set workspace to: **\*getcertified\_repository\_path\*/getcertified/trunk**
* Import project – click File/Import/General/Existing Projects into Workspace, then click **Next**
* Click browse and set root folder to be the same as workspace folder, press **Ok**.



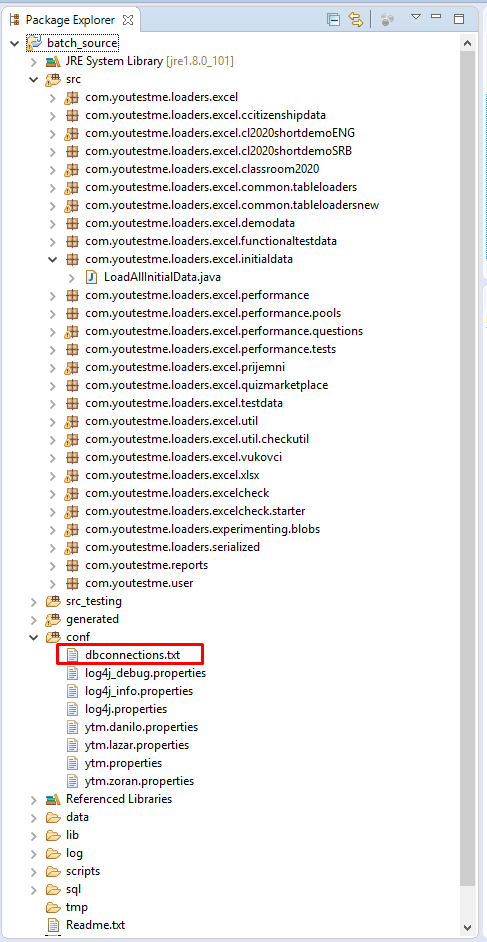


* All projects from trunk folder will appear, deselect all and check **batch\_source**. Click **Finish**.
* 

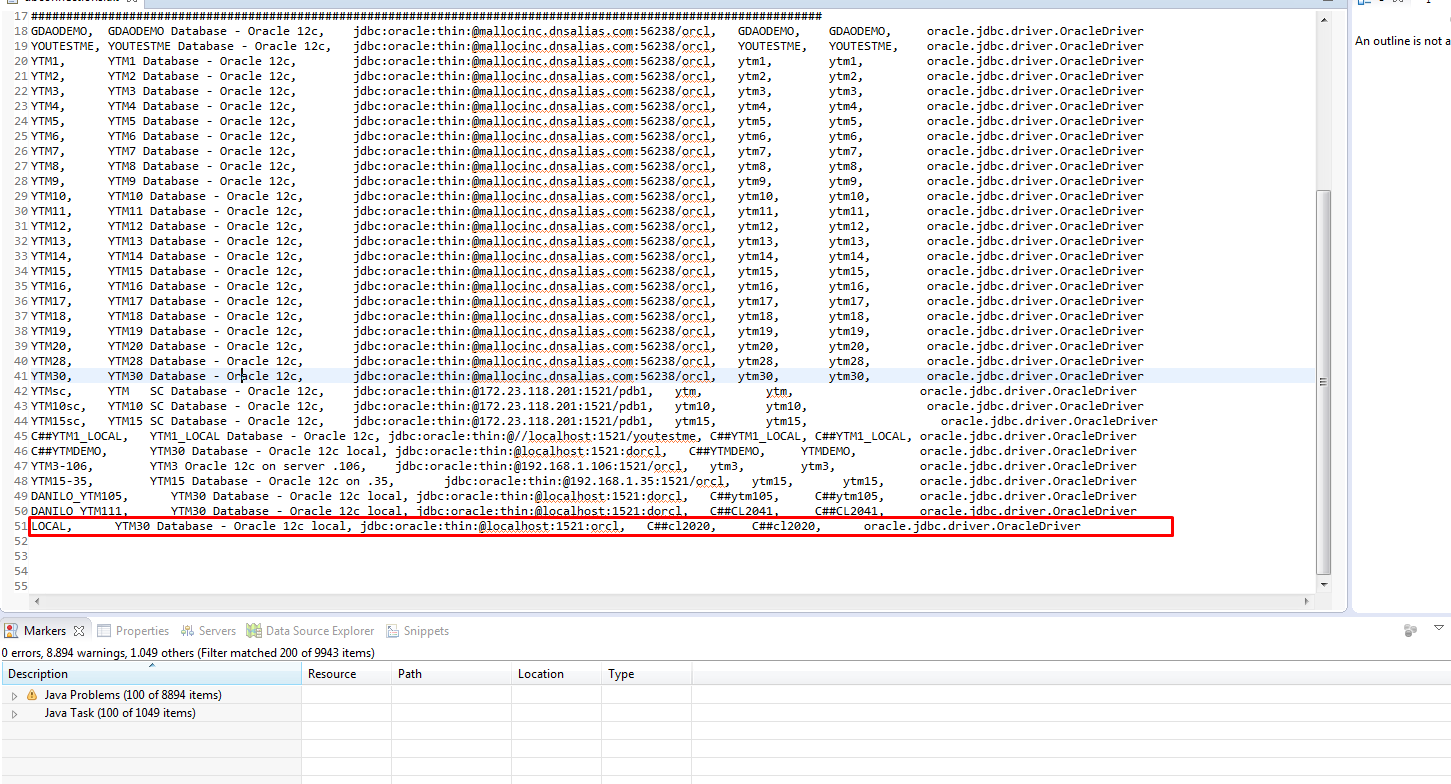
### Import via Eclipse

#### dbconnections.txt

Expand batch source project (click on folder on top left screen) and open conf/dbconnections.txt.



You can check available connections here. Also you can add your connection string. Connection string represents one line and this is an example of it:



This is the explanation of the fields:

* **LOCAL**, represents a database name that will be used in properties file (see 1.2.2). Note: You may put any name, just be sure that it is unique inside dbconnections.txt
* **YTM30 Database - Oracle 12c local,** represents a database description
* **jdbc:oracle:thin:@localhost:1521:orcl –** this is Oracle URL. Insert your database information in the given format - jdbc:oracle:thin:@address:port**:sid** or jdbc:oracle:thin:@address:port**/service\_name**

NOTE – put **/** before Service Name

Note – Oracle URL always starts with **jdbc:oracle:thin:@**, you put parameters after **@**

* **C##gc** – represents DB username
* **C##gc** – reporesents DB password
* **oracle.jdbc.driver.OracleDriver**, represents an oracle driver for communication between Java and Database and should just be copied since ojdbc is used.

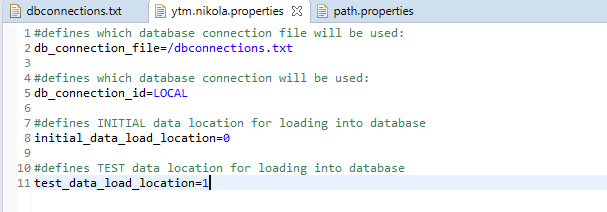
Make sure that you have comma “,” separator after each field.

#### Configuration file

Once connection string is set inside dbconnections.txt, create your own properties file by copying ytm.danilo.properties and rename it to ytm.yourname.properties (inside conf folder)

After that, set up db\_connection\_id = database name

(In example from the previous page, since I have put LOCAL as database name – this should be db\_connection\_id = LOCAL)



In addition to that, **initial\_data\_location** has to be set, as well as **test\_data\_location** (if necessary). Values to be put here are numbers that are defined in **conf/path.properties.**

*Example 1 – We want to load INITIAL data:*

Initial data location is **number 0**

In order to load initial data, we should put only initial data location and ignore test\_data\_location. Value of test\_data\_location is irrelevant so it can stay as it is.

*Example 2 – We want to load GetCertified data:*

initial\_data\_load\_location=0

test\_data\_load\_location=1

*Example 3 – We want to load SMATSA data:*

initial\_data\_load\_location=0

test\_data\_load\_location=1

## Loading data – setting up arguments and environment

After setting up our configuration file, it is time to load data. In order to do so, we have to run data loader.

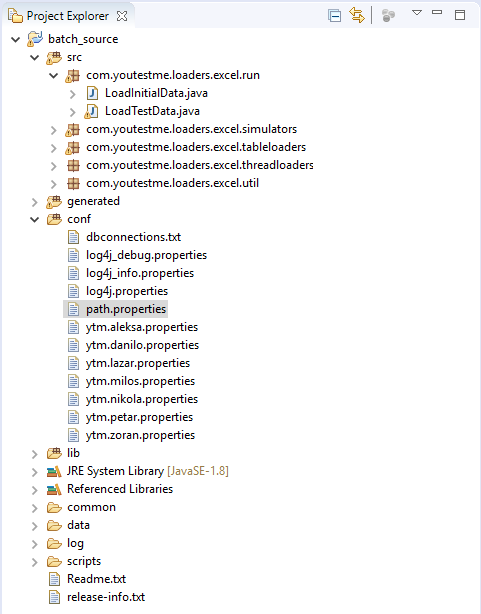
Currently there are 3 different data loaders:

* **Initial data**– Essential data that is required for application to work.

(src\com\youtestme\loaders\excel\run\LoadInitialData.java)

* **Test data** – Test set of data used for demonstration or application testing.

(src\com\youtestme\loaders\excel\run\LoadTestData.java)

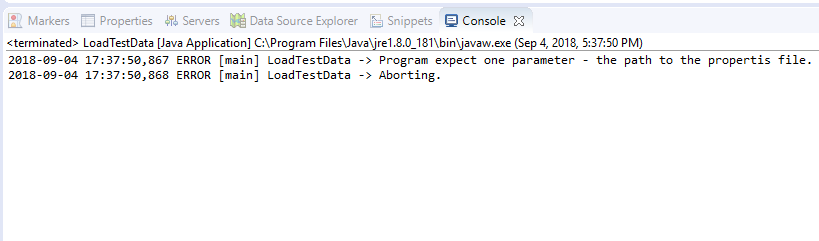


This example shows you how to run Test data.

In order to run loader, configuration has to be set for each class that has to be run.

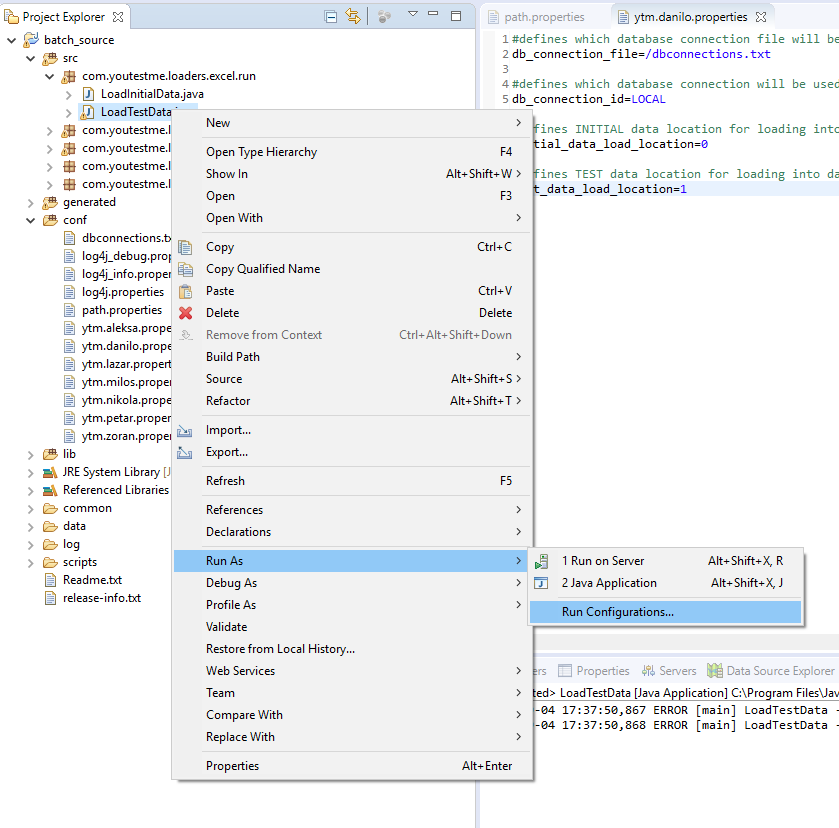
To do so do the following:

1. First run LoadTestData.java from the above list and this would appear on the console:

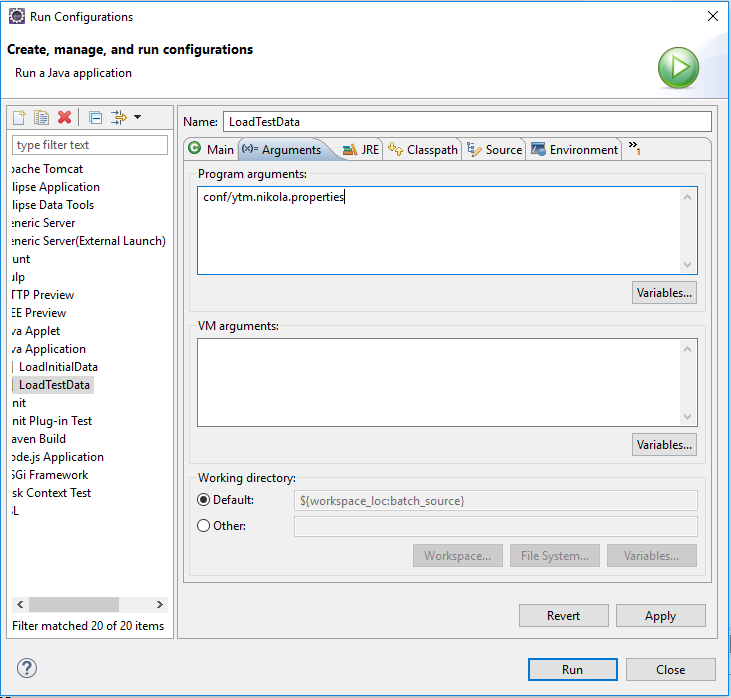


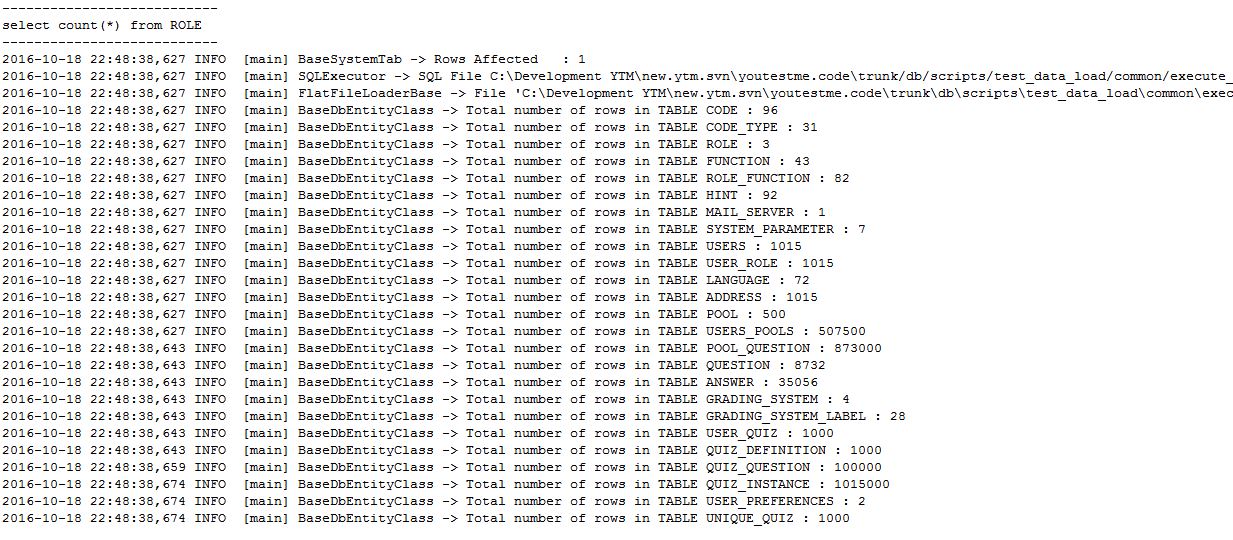
With this, we make sure that we “focused” LoadTestData.java inside configurations, since eclipse struggles with it and does not focus it if we right click before this step. Note: This step has to be done for each loader if we want each loader to work – first run loader, then go to step 2.

1. Now right click on LoadTestData.java and go to **Run As/Run Configuration**.



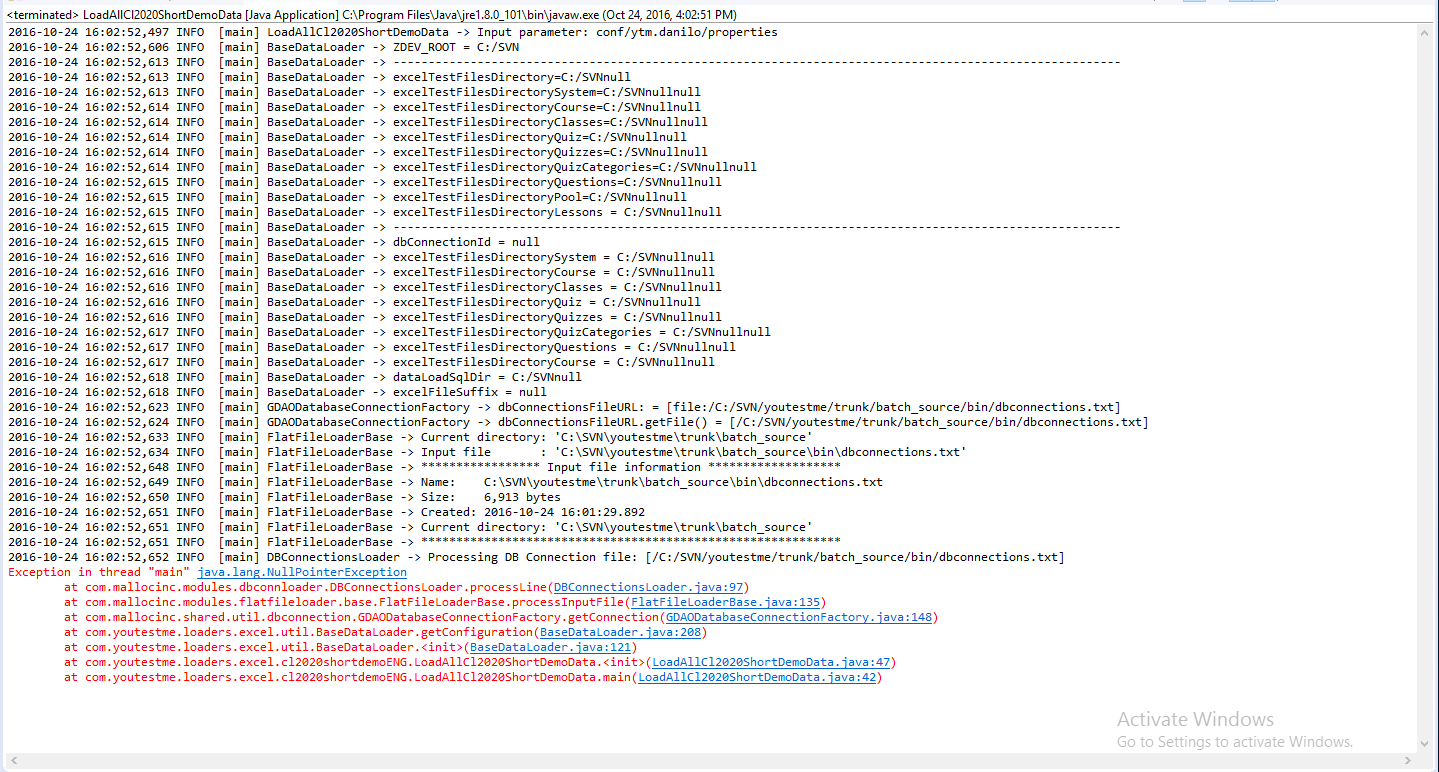
1. Go to **Arguments** tab and set the path to your properties file as an argument



1. Finally, click **run** and this should run loader and data should be successfully inserted into your database. **This is the console output after successful load.** This output gives information about the amount of records inside each table.

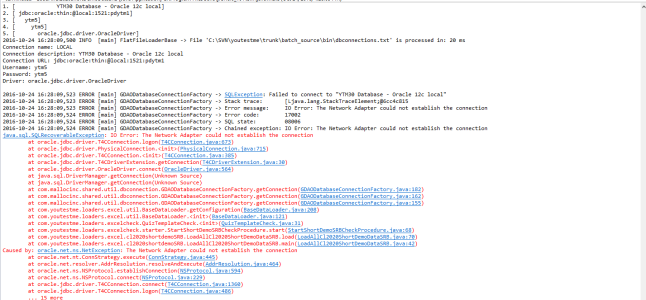
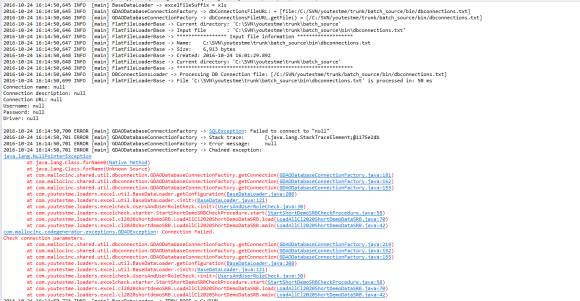
#### Troubleshooting

1. System cannot find path to excel. Console shows null path to excel files and java returns nullpointer exception.



**Answer**: Make sure that you set arguments properly (Arguments tab in configuration).

1. System cannot create connection to database.



**Answer:** There are two reasons why that may happen:

* Wrong connection string inside dbconnections.txt – make sure that you put connection string into requested format
* Wrong database name inside db\_connection\_id inside properties file

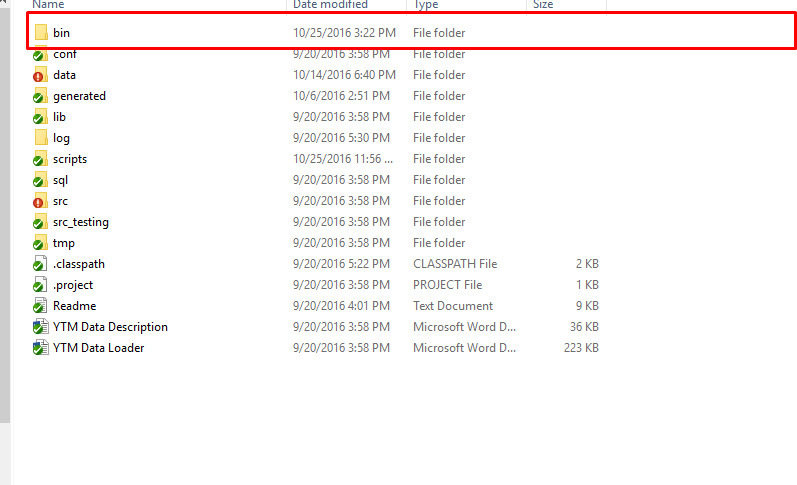
(no such name inside dbconnections.txt)

1. Unknown SID

**Answer:** Use **/** before service name of JDBC string in dbconnections.txt

### Import via batch script

Before you proceed to import this way, make sure that you imported up to date project via Eclipse (see 8.1.1) and that there is **up to date bin folder** inside batch\_source directory.

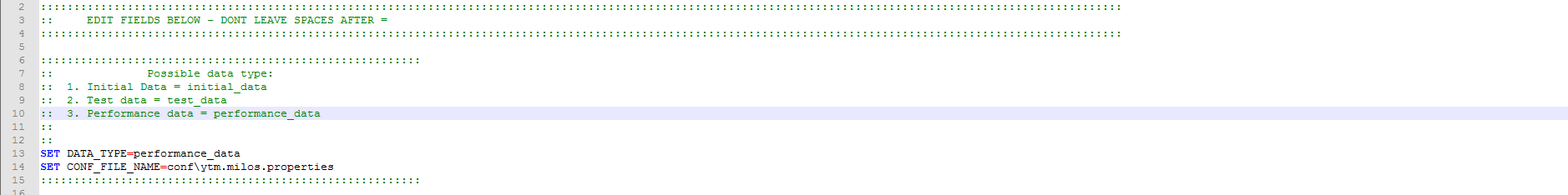


Go to:

C:\SVN\youtestme\trunk\batch\_source\scripts\

and open file "**load\_data.bat"** with Notepad (preferably [Notepad++](https://notepad-plus-plus.org/download/v7.1.html)).

The following fields have to be edited before running this script:



* 1. **DATA\_TYPE** – For classroom2020, there are two data types:
     + **initial\_data** –to load initial data
     + **test\_data** – to load test data
     + **performance\_data** – to load performance data
  2. **CONF\_FILE\_NAME** – path to your properties file (see the previous chapter for more info about properties file).

**Note:**

* **DON’T** put spaces between = and value
  + This is **good** example: DATA\_TYPE=initial\_data
  + This is **bad** example: DATA\_TYPE= initial\_data

After you have set variables, save script and then run it via **Command Prompt,** in order to do so – do the following:

1. Open Command Prompt (if you can’t find it –see [here](http://www.digitalcitizen.life/7-ways-launch-command-prompt-windows-7-windows-8))
2. If SVN repository is on C: (or wherever Windows is installed), then go to parent folder of **load\_data** file, for example:

**cd C:\SVN\youtestme\trunk\batch\_source\scripts\new\_data\_load**

Then from there, write **load\_data.bat,** press enter and batch script will load data.

1. If SVN repository is on different drive than Windows, then redirect it before using command from above. For example if SVN is on E:\ drive, and Windows is on C:\ - type **E:** and press enter. Then you can use commands from step 2. For example if SVN repository path is E:\SVN, then use:

**cd E:\SVN\youtestme\trunk\batch\_source\scripts\new\_data\_load**

## Classroom2020 data

|  |  |
| --- | --- |
| Name | Location |
| Initial (essential) data | youtestme\trunk\batch\_source\data\initial\_data |
| English data | youtestme\trunk\batch\_source\data\cl2020\_short\_demo\_batch |
| Serbian data | youtestme\trunk\batch\_source\data\cl2020\_short\_demo\_batch\_srpski |
| Performance data | youtestme\trunk\batch\_source\data\performance\_test\_data |
| SMATSA data | youtestme\trunk\batch\_source\data\smatsa\_data |

## GetCertified data

|  |  |
| --- | --- |
| Name | Location |
| Initial (essential) data | getcertified\trunk\batch\_source\data\initial\_data |
| Get Certified data | getcertified\trunk\batch\_source\data\get\_certified\_data |

# Business Rules

## Quiz Definition – Unique Quiz – Quiz Instance – Quiz Question - Question

Constraints for proper test data, to look like data loaded

Abbreviations used:

|  |  |
| --- | --- |
| **Code** | **Description** |
| QD | Quiz Definition |
| UQ | Unique Quiz |
| QI | Quiz Instance |
| QQ | Quiz question |
| Q | Question |

A lot of information will seem redundant, but it is important for quiz wizard, and templating.

This is the list of constraints that are important for Quiz in phase of creation and Quiz that is sent to students (quiz requests):

1. QD.RESOLVE\_TYPE == QI.RESOLVE\_TYPE (what goes for QD goes for QI too)
2. In case of quiz containing at least one FILL IN THE BLANKS or ESSAY question RESOLVE\_TYPE should be QRN, otherwise QRY.
3. QD.NUM\_OF\_UNIQUE\_TESTS must be the same like actual number of unique tests joined. (It is redundant, but it is necessary for templating)
4. If (QD.CREATE\_TYPE == “MAN”) => QD.NUM\_OF\_UNIQUE\_TESTS must be 1.
5. If (QD.DURATION\_TYPE == “DQZ”) => QD.DEFAULT\_DURATION must not be NULL
6. If (QD.DURATION\_TYPE == “DQS”) => QD.DEFAULT\_DURATION should be sum of QUIZ\_QUESTION.DURATION (but it will work without it too)
7. If (QD.DURATION\_TYPE == “DQS”) =>QD.LOOK\_TYPE must be “OPP”
8. QD.DEFAULT\_ENABLED\_FROM == QI.ENABLED\_FROM (what goes for QD goes for QI too)
9. QD.DEFAULT\_ENABLED\_TO == QI.ENABLED\_TO
10. If (QD.QUIZ\_DEFINITION\_STATUS == “QWA”) => QI.QUIZ\_INSTANCE\_STATUS must be “NAT”;
11. **If (QD.QUIZ\_DEFINITION\_STATUS == “QWA”) =>  
    QD.ENABLED\_TO must be null;  
    QD.ENABLED\_FROM must be null,   
    QI.INSTANCE\_STARTED must be null;   
    QI.INSTANCE\_FINISHED must be null;  
    QI.ACTIVATION\_DATETITIME must be null or new Timestamp (0);  
     (Quiz is not scheduled yet.)**
12. **If (QD.QUIZ\_DEFINITION\_STATUS == “QAC”) =>  
    QD.ENABLED\_TO must NOT be null;  
    QD.ENABLED\_FROM must NOT null;  
    QD.ENABLED\_TO >QD.ENABLED\_FROM;   
    QI.ACTIVATION\_DATETITIME must be timestamp of changing QD.QUIZ\_DEFINITION\_STATUS from QWA to QAC;   
    QI.ENABLED\_FROM, QI.ENABLED\_TO must be the same as the ones in QD.  
    (Quiz is scheduled and students can take when the current time is between enabled from and enabled to.)**
13. If (QD.QUIZ\_DEFINITION\_STATUS == “QAC”&& QI.QUIZ\_STARTED == null) => QI.QUIZ\_INSTANCE\_STATUS should be “NAT” in order to be displayed as quiz request.
14. If (QD.QUIZ\_DEFINITION\_STATUS == “QAC” || QD.QUIZ\_DEFINITION\_STATUS == “SUS”) => QD.GRADING\_SYSTEM\_ID, QD.ENABLED\_TO, QD.ENABLED\_FROM must not be NULL and ENABLED\_FROM<ENABLED\_TO.
15. QD.POINTS\_EASY, QD.POINTS\_MEDIUM, QD.POINTS\_HARD, QD.DURATION\_EASY, QD.DURATION\_MEDIUM, QD.DURATION\_HARD is relevant only if QD.CREATE\_TYPE == “GEN”
16. Don’t use “IQT” for QD.REPORT\_TYPE. Not implemented in this phase.

Constraints important for attempted quiz instances (not for quiz requests):

1. If (QI.QUIZ\_STARTED != null && QD.REPORT\_TYPE ==”OPA”) => QI.QUIZ\_INSTANCE\_STATUS must be “ATN”
2. If (QI.QUIZ\_STARTED != null && QD.REPORT\_TYPE ==”IMM”) => QI.QUIZ\_INSTANCE\_STATUS must be “ATR”
3. User can be assigned to quiz only in one quiz instance.
4. Q.QUESTION\_TYPE, Q.DIFFICULTY\_CODE must not be null.
5. Q.POINTS, Q.PENALTY, Q.DURATION, Q.FREQUENCY\_FACTOR should not be null, because it can’t be null if question has been created using app interface. But it should work ether way.
6. QQ.QUESTION\_ORDINAL\_NUMBER must be different for every question in quiz.
7. QQ.FREQUENCY\_FACTOR must not be null if QD.CREATE\_TYPE == “GEN” (generated quiz)
8. If (QD.DURATION\_TYPE == “DQS”) => QQ.DURATION must not be null
9. QQ.PENALTY, QQ.POINTS must not be null.
10. If Q.QUESTION\_TYPE is from {SNC, MLC, TFC, FBL, MCH, ORD} it needs to have answers.
11. If Q.QUESTION\_TYPE == “ESY” it should not have any answer.
12. If Q.QUESTION\_TYPE == “SNC” it should have exactly one correct answer
13. If Q.QUESTION\_TYPE == “MLC” it should have at least one correct answer  
    Lazar will add more specific details.
14. If (QD.QUIZ\_DEFINITION\_STATUS == “QAC” || QD.QUIZ\_DEFINITION\_STATUS == “QSU”) => QD.GRADING\_SYSTEM\_ID must not be null.

## Question Types

Question table (common fields for all question types):

1. QUESTION\_TEXT: Question text for specific question (html text from text editor)
2. QUESTION\_BLOB: Temporarily we do not use this field
3. PICTURE: We can have multiple pictures, and theirs urls is placed inside QUESTION\_TEST filed
4. DURATION: Duration of the question in seconds.
5. DIFFICULTY\_CODE: EASY = “DEA”, MEDIUM = “DME”, HARD = “DHA”
6. DIFFICULTY: We have only three parameters for difficulty, so we write default values for each one:
   1. EASY = 20, MEDIUM = 50, HARD = 80
7. POINTS: Number of points that person gets by answering this question correctly.
8. READ\_ONLY\_IND: If set to "Y" indicates that question is already assigned to someone and it cannot be edited any more.
9. FREQUENCY\_FACTOR: Factor that indicates how frequently this question should appear in quizzes.
   1. It is actually a probability that question will appear in the quiz. It can have values from 0 to 100.
10. PENALTY: Number of points that person looses for choosing or fiving wrong answer.
11. LARNING\_FEEDBACK: With this field we provide explanation for answer.

Answer table (common fields for all question types):

1. ANSWER\_TEXT: Answer text for specific answer (html text from text editor)
2. ANSWER\_ADDITIONAL\_INFO: This is additional information that can be used to explain the answer. It may be used for educational purposes. We do not use this right now.

### Single choice question

Question table:

* QUESTION\_TYPE: SINGLE\_CHOICE = "SNC"

Answer table:

* ANSWER\_ORDINAL\_NUMBER: Determines the order in which question will appear in the list under the question.
* CORRECT\_ANSWER\_IND: Indicates if this is the correct answer. Yes = ‘Y’, No = ‘N’.

### Multiple choice question

Question table:

* QUESTION\_TYPE: MULTI\_CHOICE = "MLC"

Answer table:

* ANSWER\_ORDINAL\_NUMBER: Determines the order in which question will appear in the list under the question.
* CORRECT\_ANSWER\_IND: Indicates if this is the correct answer. Yes = ‘Y’, No = ‘N’.

### Essay

Question table:

* QUESTION\_TYPE: ESSAY = "ESY"

Answer table: No data in this table for essay question type.

### Ordering

Question table:

* QUESTION\_TYPE: ORDERING = "ORD"

Answer table:

* ANSWER\_ORDINAL\_NUMBER: Represent the correct position for specific answer inside question.
* CORRECT\_ANSWER\_IND: null.

### Fill in the blanks

Question table:

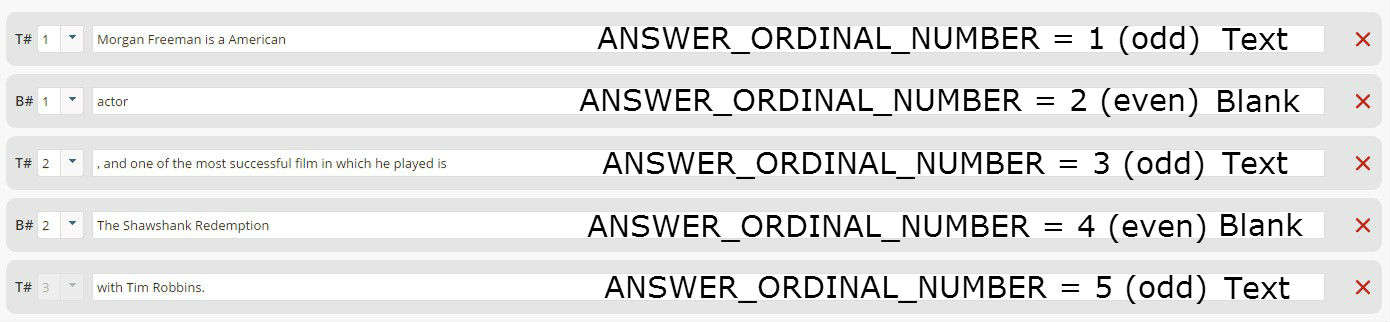
* QUESTION\_TYPE: FILL\_BLANKS = "FBL"

Answer table:

* ANSWER\_ORDINAL\_NUMBER: If in this field is placed odd number (1,3,5,…) then for this answer in ANSWER\_TEXT field is placed text that is not a blank, represent the body of question.

If in this filed is even number (2,4,6,…) then for this answer in ANSWER\_TEXT field is placed text which represent what is the correct text for blank line.

* CORRECT\_ANSWER\_IND: null.



Example: Morgan Freeman is American \_\_\_\_\_\_\_\_, and one of the most successful film in which he played is \_\_\_\_\_\_\_\_\_\_ with Tim Robbins.

### True False

Question table:

* QUESTION\_TYPE: TRUE\_FALSE = "TFC"

Answer table:

* ANSWER\_ORDINAL\_NUMBER: Determines the order in which question will appear in the list under the question.
* CORRECT\_ANSWER\_IND: Indicates if this is the correct answer. Yes = ‘Y’, No = ‘N’.

### Matching

Question table:

* QUESTION\_TYPE: MATCHING = "MCH"

Answer table:

* ANSWER\_ORDINAL\_NUMBER: Answers to the left side have ordinal number which is link to the right correct answer.Ordinal number of the answers which is placed on right side have been given by in the order of their addition.
* CORRECT\_ANSWER\_IND: Answers who are placed on the left side, they have CORRECT\_ANSWER\_IND = ‘N’ and other ones placed on the right side have CORRECT\_ANSWER\_IND = ‘Y’.

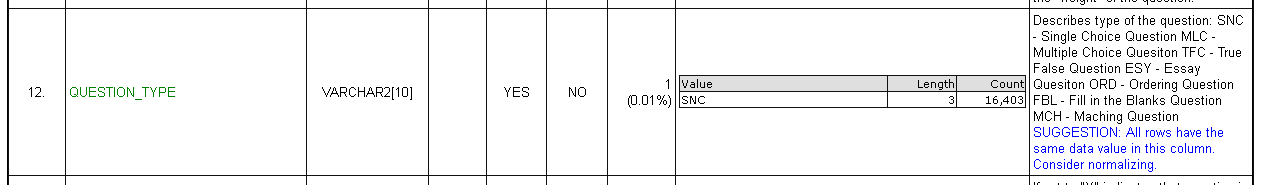


# Verifying created data

Some QA tips - or how you can quickly determine quality of your test environment:

The most important thing  is to have a sufficient data for complete coverage and the simplest way to quickly determine that is:

* all tables have to have some data (more is generally better)
* you have to have all codes, for example if you know that we have many question or user types and you see that there is only one in all records in the database you know that various scenarios cannot be tested (see picture below)
* similar logic can be applied to all codes and many other data. For example timestamps, names, locations, etc.
* empty tables also indicate that some functionalities cannot be tested. Traditionally the biggest problem are data missing in associative tables connect related tables with each other.
* report also gives you number of records in each table which is very useful as a starting point and to determine if you have sufficient data for any type of testing
* look at the "warnings" and "suggestions" in report. They usually indicate problem with data (but sometimes it may be ok).



# List of Excel files that can be loaded into GC & CL2020

1. getcertified\trunk\batch\_source\data\get\_certified\_data\excel\Quiz\Quizzes
2. getcertified\trunk\batch\_source\data\get\_certified\_data\excel\Pools
3. getcertified\trunk\batch\_source\data\get\_certified\_smatsa\_data\excel\Pools
4. getcertified\trunk\batch\_source\data\initial\_data\System

## Main GC data sets

Initial data - getcertified\trunk\batch\_source\data\initial\_data

All available data - getcertified\trunk\batch\_source\data\get\_certified\_data