|  |
| --- |
|  |
| Trinity DB Testing Report  Contains test plans and cases, justifications and more. |
| |  |  |  | | --- | --- | --- | | Author: K.P.I. Shenesh Perera | 12/18/18 | [Course title] | |

Contents

[Introduction 3](#_Toc1528059)

[Passive query benchmarking 4](#_Toc1528060)

[Test Case 0: Benchmarking 4](#_Toc1528061)

[Data schema validation testing 4](#_Toc1528062)

[Test Case 1: Create 4](#_Toc1528063)

[Test Case 2: Insert Data 5](#_Toc1528064)

[Constraint testing 6](#_Toc1528065)

[Test Case 3: Primary Keys 6](#_Toc1528066)

[Test Case 4: Foreign Keys 6](#_Toc1528067)

[The Testing Suite complete query 7](#_Toc1528068)

[Conclusion 8](#_Toc1528069)

**Witness Statement by <Name>**

# Introduction

The database solution has been prominently inspected, designed, developed and evaluated thoroughly so far, now a several unit tests will be performed with SQL queries to see the data schema is validating data properly, constraints work as expected and operating the database is simple enough.

Therefore, the database trinityHS will be tested.

Following is the Entity Relationship Diagram in order to ensure that the model of the database is visualized in this same document thereby improving the ease of use.

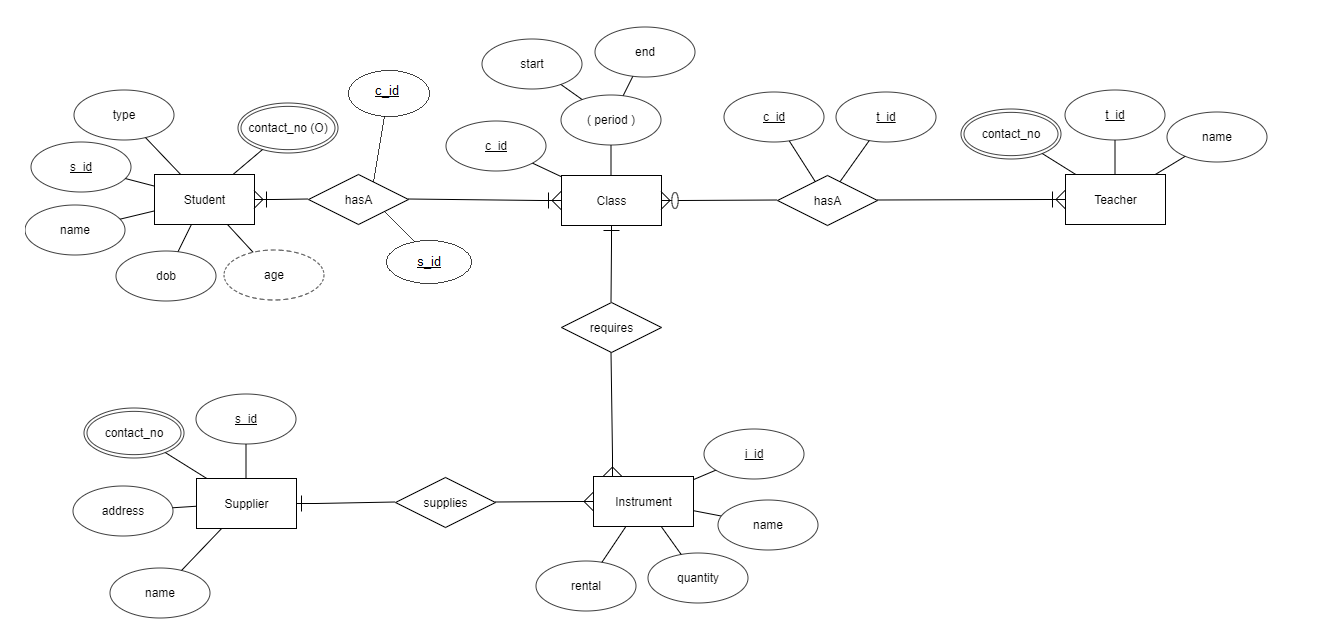


Figure 1.0, Shenesh Perera, 19/12/2018

*Please use Microsoft SQL Server Management Studio v11.0 query builder in order to perform the following test cases.*

# Passive query benchmarking

## Test Case 0: Benchmarking

This is done in order to assure the database solution does not lag behind and performs up to quality standard in order to make sure each of the following test cases are bench marked please run this query:

SET STATISTICS TIME ON

By running this command, during the evaluation of the below test cases the time taken to complete each test is shown.

# Data schema validation testing

## Test Case 1: Create

A table called “testx1” will be created within the database solution, this will be created with 3 columns called tst\_id of integer type, tst\_name of varchar (50) type, tst\_date of datetime type and tst\_money of money type. tst\_id will be the primary key and will also be auto incremented starting from 10 incrementing by 2. This is done in order to test all the datatypes and auto-incrementing used within the database solution.

**NOTE: As tst\_id is auto-incremented, during insertions it is not required to specify data for this column.**

The query that must be run:

CREATE TABLE testx1 (

tst\_id int IDENTITY(10, 2) PRIMARY KEY,

tst\_name varchar(50),

tst\_date datetime,

tst\_money money

)

Status: Pass / Fail

Comment:

## Test Case 2: Insert Data

In order to ensure that the schema of the table “testx1” accepts only data of the declared datatypes, both data of the correct and wrong data types will be inserted in order to test each. The first round will insert data of the correct data types, then on the data types will be changed in order to see if the database rejects them. The choice of inserting incorrect data is to ensure that the schema functions properly, the inserted data doesn’t have to be complex, significant or have any real sense as such.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Query** | **Data** | | | **Status** | |
| tst\_name | tst\_date | tst\_money | Expected | Received |
| INSERT INTO testx1 VALUES ('abc', '20181018 00:00:00 AM', 2000.00) | ‘abc’ | ‘20181018 00:00:00 AM’ | 2000.00 | Pass |  |
| INSERT INTO testx1 VALUES (34, '20191018 00:00:00 AM', 3000.00) | 34 | ‘20191018 00:00:00 AM’ | 3000.00 | Fail |  |
| INSERT INTO testx1 VALUES ('efg', 20191018 00:00:00 AM, '3000.00') | ‘efg’ | 20191018 00:00:00 AM | ‘3000.00’ | Fail |  |

Status: Pass/ Fail

Comment:

# Constraint testing

## Test Case 3: Primary Keys

Primary keys essentially have 2 constraints that have to be tested, UNIQUE and NOT NULL. A primary key field must be completely different than the other values within its column and can never be empty. In order to check this and to see if the primary key attribute has been properly applied on “tst\_id”, 2 queries will be performed. The 1st query will attempt to insert data with an already existing ID (100) while the next will try to update ID of 100 to null.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Query** | **Data** | | | **Status** | |
| tst\_name | tst\_date | tst\_money | Expected | Received |
| INSERT INTO testx1 VALUES (100, 'abc', '20181018', 2000.00) | ‘abc’ | ‘20181018 00:00:00 AM’ | 2000.00 | Pass |  |
| INSERT INTO testx1 VALUES (100, 'abc', '20181018', 2000.00) | 34 | ‘20191018 00:00:00 AM’ | 3000.00 | Fail |  |
| UPDATE testx1 SET tst\_id = null WHERE tst\_id = 100 |  |  |  | Fail |  |

Status: Pass / Fail

Comment:

## Test Case 4: Foreign Keys

Foreign keys are primary keys from other tables. In order to perform this test case, a new table called “testx2” will be created with columns “tst2\_id” with type int and “tst\_id” with type int where tst2\_id is the primary key and tst\_id is the foreign key referencing tst\_id from table testx1. In order to ensure foreign key relationship has been established, after inserting data to testx2 I will attempt to delete the tst\_id of that is being referenced by the inserted data to testx2 and also I will attempt to insert and update a tst\_id value that does not exist in testx1 to testx2.

CREATE TABLE testx2 (

tst2\_id int PRIMARY KEY,

tst\_id int FOREIGN KEY REFERENCES testx1(tst\_id)

)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Query** | **Data** | | | **Status** | |
|  | tst2\_id | tst\_id | Expected | Received |
| INSERT INTO testx2 VALUES (1, 15) |  | 1 | 15 | Pass |  |
| INSERT INTO testx2 VALUES (1, 13) |  | 1 | 13 | Fail |  |
| DELETE FROM testx1 WHERE tst\_id = 15 |  |  |  | Fail |  |
| UPDATE testx1 SET tst\_id = null WHERE tst\_id = 100 |  |  |  | Fail |  |

# The Testing Suite complete query

CREATE DATABASE test --create only if you wish to test separately

USE test --use only if you wish to test separately

--USE trinityHS recommended to test with trinityHS

SET STATISTICS TIME ON --Run to benchmark each command

--Test Case 1

CREATE TABLE testx1 (

tst\_id int PRIMARY KEY IDENTITY(10, 5),

tst\_name varchar(50),

tst\_date datetime,

tst\_money money

)

SELECT \* FROM testx1 -- Run to view

--Test Case 2

INSERT INTO testx1 VALUES ('abc', '20181018 00:00:00 AM', 2000.00)

INSERT INTO testx1 VALUES (34, '20191018 00:00:00 AM', 3000.00)

INSERT INTO testx1 VALUES ('efg', 20191018 00:00:00 AM, '3000.00')

UPDATE testx1 SET tst\_id = '5' WHERE tst\_id = 10

SELECT \* FROM testx1 -- Run to view

--Test Case 3

INSERT INTO testx1 VALUES (100, 'abc', '20181018', 2000.00)

INSERT INTO testx1 VALUES (100, 'abc', '20181018', 2000.00)

UPDATE testx1 SET tst\_id = null WHERE tst\_id = 100

SELECT \* FROM testx1 -- Run to view

--Test Case 4

CREATE TABLE testx2 (

tst2\_id int PRIMARY KEY,

tst\_id int FOREIGN KEY REFERENCES testx1(tst\_id)

)

INSERT INTO testx2 VALUES (1, 15)

INSERT INTO testx2 VALUES (1, 13)

UPDATE testx2 SET tst\_id = 12 WHERE tst2\_id = 1

DELETE FROM testx1 WHERE tst\_id = 15

SELECT \* FROM testx2 -- Run to view

# Conclusion

By performing these 5 test cases, all user requirements and system requirements have been thoroughly evaluated all primary, secondary and tertiary system requirements and user requirements have been tested.

The reason why 2 separate tables have been used for the testing purposes, is to avoid pollution of the database solution. By mimicking the contents of all the tables that satisfy the user and system requirements in these 2 separate tables testx1 and testx2, all test cases can be performed without corrupting the database solution, which I believe is very important.

I have performed all these test cases and verified them, the witness statement stated above done by <Name> will adhere to my statement.

This database solution is ready for deployment and is perfectly tested to see if it meets system and user requirements, these tests will prove that it has and the witness statement will confirm these tests.