

Version: 03.01

Date: 11-Mar-2016

Developed by:

Verified by:

Endava SQL Discipline

SQL Test

# Revision History

|  |  |  |
| --- | --- | --- |
| **Revision** | **Date of revision** | **Description of modifications** |
| 03.01 | 11-Mar-2016 | The third version of the document. |

# Description

**About the test:**

|  |  |
| --- | --- |
|  |  |
| Applied Level | Intermediate |
| Number of tasks | 12 |
| Domain | Standard DB |
| Test type | PC |
| Test duration | 2 h |

**Evaluation info:**

|  |  |
| --- | --- |
|  |  |
| Evaluated person |  |
| Evaluator name |  |
| Date of evaluation |  |
| Evaluation result  (passed/failed) |  |

# Test Tasks

## Precondition:

Use the below credentials:

|  |  |
| --- | --- |
| **Server Name** | MDCH-AMWCI-S01 |
|
| **DB** | AMInternship |
|
|

## Task 1

**Which are the different types of available Indexes in SQL Server?**

**Answer:**

|  |
| --- |
| 1. Clustered indexes 2. Non-Clustered indexes |

## Task 2

**How many Clustered and Non-Clustered Indexes can be created on a sinlge table?**

**Answer:**

|  |
| --- |
| Clustered index can be only one.  Non-clustered up to 999 |

## Task 3

**What is the difference between Clustered and Non-Clustered Indexes?**

**Answer:**

|  |
| --- |
| **Clustered**   * Clustered indexes sort and store the data rows in the table or view based on their key values. These are the columns included in the index definition. There can be only one clustered index per table, because the data rows themselves can be sorted in only one order. * The only time the data rows in a table are stored in sorted order is when the table contains a clustered index. When a table has a clustered index, the table is called a clustered table. If a table has no clustered index, its data rows are stored in an unordered structure called a heap.   **Nonclustered**   * Nonclustered indexes have a structure separate from the data rows. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value. * The pointer from an index row in a nonclustered index to a data row is called a row locator. The structure of the row locator depends on whether the data pages are stored in a heap or a clustered table. For a heap, a row locator is a pointer to the row. For a clustered table, the row locator is the clustered index key. |

## Task 4

**What are the SQL VIEWS?**

**Answer:**

|  |
| --- |
| A view is a virtual table whose contents are defined by a query. Like a table, a view consists of a set of named columns and rows of data. Unless indexed, a view does not exist as a stored set of data values in a database. The rows and columns of data come from tables referenced in the query defining the view and are produced dynamically when the view is referenced. |

## Task 5

**Which are the advantages and disadvantages of SQL VIEWS?**

**Answer:**

|  |
| --- |
| **Advantages of views**  **Security**  Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data  **Query Simplicity**  A view can draw data from several different tables and present it as a single table, turning multi-table queries into single-table queries against the view.  **Structural simplicity**  Views can give a user a "personalized" view of the database structure, presenting the database as a set of virtual tables that make sense for that user.  **Consistency** A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured, or renamed.  **Data Integrity**  If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets the specified integrity constraints.  **Logical data independence.**  View can make the application and database tables to a certain extent independent. If there is no view, the application must be based on a table. With the view, the program can be established in view of above, to view the program with a database table to be separated.  **Disadvantages:**  A view can be created only in the current database.  A view can have a maximum of 1,024 columns.  **Performance**  Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables. If the view is defined by a complex, multi-table query then simple queries on the views may take considerable time.  **Update restrictions**  When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying source tables. This is possible for simple views, but more complex views are often restricted to read-only. |

## Task 6

**Which is the difference between a View and Temporary table?**

**Answer:**

|  |
| --- |
| The main difference between temporary tables and views is that temporary tables are just the tables in tempdb, but views are just stored queries for existing data in existing tables. So, there is no need to populate the view, because the data is already here. But temporary table needs to be populated first, and population is the main preformance-concerned issue. |

## Task 7

**Which is the difference between a Variable table and a Temporary one?**

**Answer:**

|  |
| --- |
| * Temporary Tables are real tables so you can do things like CREATE INDEXes, etc. If you have large amounts of data for which accessing by index will be faster then temporary tables are a good option. * Table variables can have indexes by using PRIMARY KEY or UNIQUE constraints. (If you want a non-unique index just include the primary key column as the last column in the unique constraint. If you don't have a unique column, you can use an identity column.) [SQL 2014 has non-unique indexes too](http://stackoverflow.com/questions/886050/sql-server-creating-an-index-on-a-table-variable/17385085#17385085). * Table variables don't participate in transactions and SELECTs are implicitly with NOLOCK. The transaction behaviour can be very helpful, for instance if you want to ROLLBACK midway through a procedure then table variables populated during that transaction will still be populated! * Temp tables might result in stored procedures being recompiled, perhaps often. Table variables will not. * You can create a temp table using SELECT INTO, which can be quicker to write (good for ad-hoc querying) and may allow you to deal with changing datatypes over time, since you don't need to define your temp table structure upfront. * You can pass table variables back from functions, enabling you to encapsulate and reuse logic much easier (eg make a function to split a string into a table of values on some arbitrary delimiter). * Using Table Variables within user-defined functions enables those functions to be used more widely (see CREATE FUNCTION documentation for details). If you're writing a function you should use table variables over temp tables unless there's a compelling need otherwise. * Both table variables and temp tables are stored in tempdb. But table variables (since 2005) default to the collation of the current database versus temp tables which take the default collation of tempdb ([ref](https://technet.microsoft.com/en-us/library/ms188927.aspx)). This means you should be aware of collation issues if using temp tables and your db collation is different to tempdb's, causing problems if you want to compare data in the temp table with data in your database. * Global Temp Tables (##tmp) are another type of temp table available to all sessions and users. |

## Task 8

**Create a view which will display the person id, first name, last name, job id, street name, street number, city name and country name from US.**

**Query:**

|  |
| --- |
| create view Person\_From\_US  as  select p.id, p.first\_name, p.last\_name, p.job\_id, a.street, a.street\_num, c.name, cntr.name  from person p  inner join address a ON a.id=p.address\_id  inner join city c ON c.id=a.city\_id  inner join country cntr ON cntr.id=c.country\_id  where cntr.name='US' |

## Task 9

**Create an index on first name and last name columns from the Person table.**

**Query:**

|  |
| --- |
| create index index\_person\_on\_first\_name  on person (first\_name)  create index index\_person\_on\_last\_name  on person (last\_name) |

## Task 10

**After you have created the index, insert another record in the table with the information: first name – Anatol, last name – Agachi, gender, address „Kingsford 8” and the job titile „Environmental Tech” and notice the time of result set. Verify if the result set from task 19 is modified.**

**Query:**

|  |
| --- |
| insert into person values ('Anatol','Agachi','1988-01-02',2,6,16)  00:00:00.011  Last time of result was  00:00:00.031 |

## Task 11

**Update ONLY the last entered record with the information: first name – Boris, last name – Albu, gender, address „Kingsford 8” and the job titile „Environmental Specialist” and notice the time of result set with the created index on the table.**

**Query:**

|  |
| --- |
| select \* from person  update person  set  first\_name='Boris',  last\_name='Albu',  job\_id=26  where first\_name='Anatol' And last\_name='Agachi'  time is 00:00:00.020 |

## Task 12

**Delete ONLY the new 2 entered records and delete the created index on the table.**

**Query:**

|  |
| --- |
| delete from person where first\_name='Anatol'  delete from person where first\_name='Boris'  DROP INDEX index\_person\_on\_first\_name  ON Person  DROP INDEX index\_person\_on\_last\_name  ON Person |