# Database Basics MS SQL Exam – 27 June 2020

Submit your solutions in the SoftUni Judge system at [Judge](https://judge.softuni.org/Contests/2482/Databases-MSSQL-Server-Exam-27-Jun-2020)

# Washing Machine Service

*Your task is to implement Washing Machine Service database based on the specs given below. Database will include information about broken machines which will be distributed to mechanics through so called "jobs". Each mechanic may have several jobs and for each job he/she can require some parts. To let the mechanic get whats needed he/she has to make order for a particular part also specifying how many of it will be needed (specifying quantity).*

# Section 1. DDL

You have been given the E/R Diagram of the washing machine service:



Crate a database called **WMS**. You need to create **9 tables**:

* **Clients** – contains information about the customers that use the service;
* **Mechanics** – contains information about employees;
* **Jobs** – contains information about all machines that clients submitted for repairs;
* **Models** – list of all washing machine models that the servie operates with;
* **Orders** – contains information about orders for parts;
* **Parts** – list of all parts the service operates with;
* **OrderParts** – mapping table between **Orders** and **Parts** with additional **Quantity** field;
* **PartsNeeded** – mapping table between **Jobs** and **Parts** with additional **Quantity** field;
* **Vendors** – list of vendors that supply parts to the service.

Include the following fields in each table. Unless otherwise specified, **all fields are required**.

**NOTE: Please keep in mind that in case you have to work with a date, you have to use the exact same data type, described in the models tables. For example, data type Date means that you have to use Date, DateTime means that you have to use DateTime. If you don't use the correct type, the Judge system won't accept your submission as correct.**

**NOTE: Keep in mind that Judge doesn't accept "ALTER" statement and square brackets naming (when the names are not keywords).**

You have been tasked to create the tables in the database by the following models:

### Clients

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **ClientId** | 32-bit **Integer** | Primary table identificator, Identity |
| **FirstName** | **String** up to **50** symbols, ASCII |  |
| **LastName** | **String** up to **50** symbols, ASCII |  |
| **Phone** | **String** containing **12** symbols | String length is **exactly 12** chars long |

### Mechanics

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **MechanicId** | 32-bit **Integer** | Primary table identificator, Identity |
| **FirstName** | **String** up to **50** symbols, ASCII |  |
| **LastName** | **String** up to **50** symbols, ASCII |  |
| **Address** | **String** up to **255** symbols, ASCII |  |

### Jobs

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **JobId** | 32-bit **Integer** | Primary table identificator, Identity |
| **ModelId** | 32-bit **Integer** | Relationship with table **Models** |
| **Status** | **String** up to **11** symbols, ASCII | Allowed values: '**Pending**', '**In Progress**' and '**Finished**'; Default value is '**Pending**' |
| **ClientId** | 32-bit **Integer** | Relationship with table **Clients** |
| **MechanicId** | 32-bit **Integer** | Relationship with table **Mechanics**; Can be NULL |
| **IssueDate** | **Date** |  |
| **FinishDate** | **Date** | Can be NULL |

### Models

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **ModelId** | 32-bit **Integer** | Primary table identificator, Identity |
| **Name** | **String** up to **50** symbols, ASCII | Unique |

### Orders

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **OrderId** | 32-bit **Integer** | Primary table identificator, Identity |
| **JobId** | 32-bit **Integer** | Relationship with table **Jobs** |
| **IssueDate** | **Date** | Can be NULL |
| **Delivered** | **Boolean** | Default value is False |

### Parts

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **PartId** | 32-bit **Integer** | Primary table identificator, Identity |
| **SerialNumber** | **String** up to **50** symbols, ASCII | Unique |
| **Description** | **String** up to **255** symbols, ASCII | Can be NULL |
| **Price** | Monetary value up to **9999.99** | Cannot be zero or negative |
| **VendorId** | 32-bit **Integer** | Relationship with table **Vendors** |
| **StockQty** | 32-bit **Integer** | Cannot be negative; Default value is **0** |

### OrderParts

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **OrderId** | 32-bit **Integer** | Relationship with table **Orders**;  Primary table identificator |
| **PartId** | 32-bit **Integer** | Relationship with table Parts;  Primary table identificator |
| **Quantity** | 32-bit **Integer** | Cannot be zero or negative; Default value is **1** |

### PartsNeeded

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **JobId** | 32-bit **Integer** | Relationship with table **Jobs**;  Primary table identificator |
| **PartId** | 32-bit **Integer** | Relationship with table **Parts**;  Primary table identificator |
| **Quantity** | 32-bit **Integer** | Cannot be zero or negative; Default value is **1** |

### Vendors

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints** |
| **VendorId** | 32-bit **Integer** | Primary table identificator, Identity |
| **Name** | **String** up to **50** symbols, ASCII | Unique |

## Database design

Submit all of your **CREATE** statements to Judge. **Do not include** database creation statements.

**Look for hints in the details of your submission!**

# Section 2. DML

**Before you start you have to import "Data.sql". If you have created the structure correctly the data should be successfully inserted.**

In this section, you have to do some data manipulations:

## Insert

Let's **insert** some sample data into the database. Write a query to add the following records into the corresponding tables. All Ids should be auto-generated. Replace names that relate to other tables with the appropriate ID (look them up manually, there is no need to perform table joins).

**Clients**

|  |  |  |
| --- | --- | --- |
| **First Name** | **Last Name** | **Phone** |
| Teri | Ennaco | 570-889-5187 |
| Merlyn | Lawler | 201-588-7810 |
| Georgene | Montezuma | 925-615-5185 |
| Jettie | Mconnell | 908-802-3564 |
| Lemuel | Latzke | 631-748-6479 |
| Melodie | Knipp | 805-690-1682 |
| Candida | Corbley | 908-275-8357 |

**Parts**

|  |  |  |  |
| --- | --- | --- | --- |
| **Serial Number** | **Description** | **Price** | **VendorId** |
| WP8182119 | Door Boot Seal | 117.86 | 2 |
| W10780048 | Suspension Rod | 42.81 | 1 |
| W10841140 | Silicone Adhesive | 6.77 | 4 |
| WPY055980 | High Temperature Adhesive | 13.94 | 3 |

## Update

Assign all **Pending** jobs to the mechanic **Ryan Harnos** (look up his ID manually, there is no need to use table joins) and change their **status** to '**In Progress**'.

## Delete

Cancel Order with ID **19** – delete the order from the database and all associated entries from the mapping table.

# Section 3. Querying

**You need to start with a fresh dataset, so run the "Data.sql" script again. It includes a section that will delete all records and replace them with the starting set, so you don’t need to drop your database.**

## Mechanic Assignments

Select all **mechanics** with their **jobs**. Include job **status** and **issue date**. Order by mechanic Id, issue date, job Id (all ascending).

### Required columns

* **Mechanic Full Name**
* **Job Status**
* **Job Issue Date**

### Example

|  |  |  |
| --- | --- | --- |
| **Mechanic** | **Status** | **IssueDate** |
| Joni Breland | Finished | 2017-01-12 |
| Joni Breland | Finished | 2017-01-17 |
| Joni Breland | Finished | 2017-01-24 |
| … | … | … |

## Current Clients

Select the names of all **clients** with active **jobs** (not **Finished**). Include the **status** of the job and **how many days** it’s been since it was submitted. Assume the **current date** is 24 April 2017. Order results by time length (descending) and by client ID (ascending).

### Required columns

* **Client Full Name**
* **Days going** – how many days have passed since the issuing
* **Status**

### Example

|  |  |  |
| --- | --- | --- |
| **Client** | **Days going** | **Status** |
| Gertude Witten | 18 | In Progress |
| Brittni Gillaspie | 14 | In Progress |
| Levi Munis | 12 | In Progress |
| … | … | … |

## Mechanic Performance

Select all **mechanics** and the **average time** they take to finish their assigned **jobs**. Calculate the average as an integer. Order results by mechanic ID (ascending).

### Required columns

* **Mechanic Full Name**
* **Average Days** – average number of days the machanic took to finish the job

### Example

|  |  |
| --- | --- |
| **Mechanic** | **Average Days** |
| Joni Breland | 9 |
| Malcolm Tromblay | 10 |
| Ryan Harnos | 5 |
| … | … |

## Available Mechanics

Select all **mechanics** without active **jobs** (**include mechanics which don't have any job assigned or all of their jobs are finished**). Order by ID (ascending).

### Required columns

* Mechanic Full Name

### Example

|  |
| --- |
| **Available** |
| Joni Breland |
| Ryan Harnos |
| … |

## Past Expenses

Select all **finished jobs** and the total cost of all **parts** that were **ordered** for them. Sort by total cost of parts ordered (descending) and by job ID (ascending).

### Required columns

* **Job ID**
* **Total** Parts Cost

### Example

|  |  |
| --- | --- |
| **JobId** | **Total** |
| 17 | 173.60 |
| 12 | 140.50 |
| 1 | 91.86 |
| … | … |

## Missing Parts

List all **parts** that are **needed** for active **jobs** (not **Finished**) without sufficient quantity **in stock** and in pending **orders** (the sum of parts in stock and parts ordered is less than the required quantity). Order them by part ID (ascending).

### Required columns

* **Part ID**
* **Description**
* **Required** – number of parts required for active jobs
* **In Stock** – how many of the part are currently in stock
* **Ordered** – how many of the parts are expected to be delivered (associated with order that is not Delivered)

### Example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **PartId** | **Description** | **Required** | **In Stock** | **Ordered** |
| 12 | Shock Dampener | 2 | 1 | 0 |
| 14 | Door Handle | 1 | 0 | 0 |
| 17 | Lid Switch Assembly | 1 | 0 | 0 |

# Section 4. Programmability

## Place Order

Your task is to create a user defined procedure (**usp\_PlaceOrder**) which accepts **job ID**, part **serial number** and **quantity** and creates an **order** with the specified **parameters**. If an order **already exists** for the given job **that and the order is not issued** (order's issue date is NULL), add the new product to it. If the part is already listed in the order, add the quantity to the existing one.

When a new order is created, set it's **IssueDate** to NULL.

Limitations:

* An order **cannot** be placed for a job that is **Finished**; error message ID 50011 "**This job is not active!**"
* The quantity **cannot** be zero or negative; error message ID 50012 "**Part quantity must be more than zero!**"
* The job with given ID **must exist** in the database; error message ID 50013 "**Job not found!**"
* The part with given serial number **must exist** in the database ID 50014 "**Part not found!**"

If any of the requirements aren't met, rollback any changes to the database you've made and **throw an** **exception** with the **appropriate message** and **state 1**.

### Parameters

* **JobId**
* **Part Serial Number**
* **Quantity**

### Example usage

|  |
| --- |
| **Query** |
| DECLARE @err\_msg AS NVARCHAR(MAX);  BEGIN TRY  EXEC usp\_PlaceOrder 1, 'ZeroQuantity', 0  END TRY  BEGIN CATCH  SET @err\_msg = ERROR\_MESSAGE();  SELECT @err\_msg  END CATCH |
| Response |
| Part quantity must be more than zero! |

## Cost Of Order

Create a **user defined function** (udf\_GetCost) that receives a **job's ID** and returns the total **cost** of all **parts** that were **ordered** for it. Return 0 if there are **no orders**.

### Parameters

* **JobId**

### Example usage

|  |  |
| --- | --- |
| **Query** | |
| SELECT dbo.udf\_GetCost(1) | |
| Id | Result |
| 1 | 91.86 |
| 3 | 40.97 |
| … | … |