Project No. 2:
A database for transport management

Szymon Górski

1 Project Setup

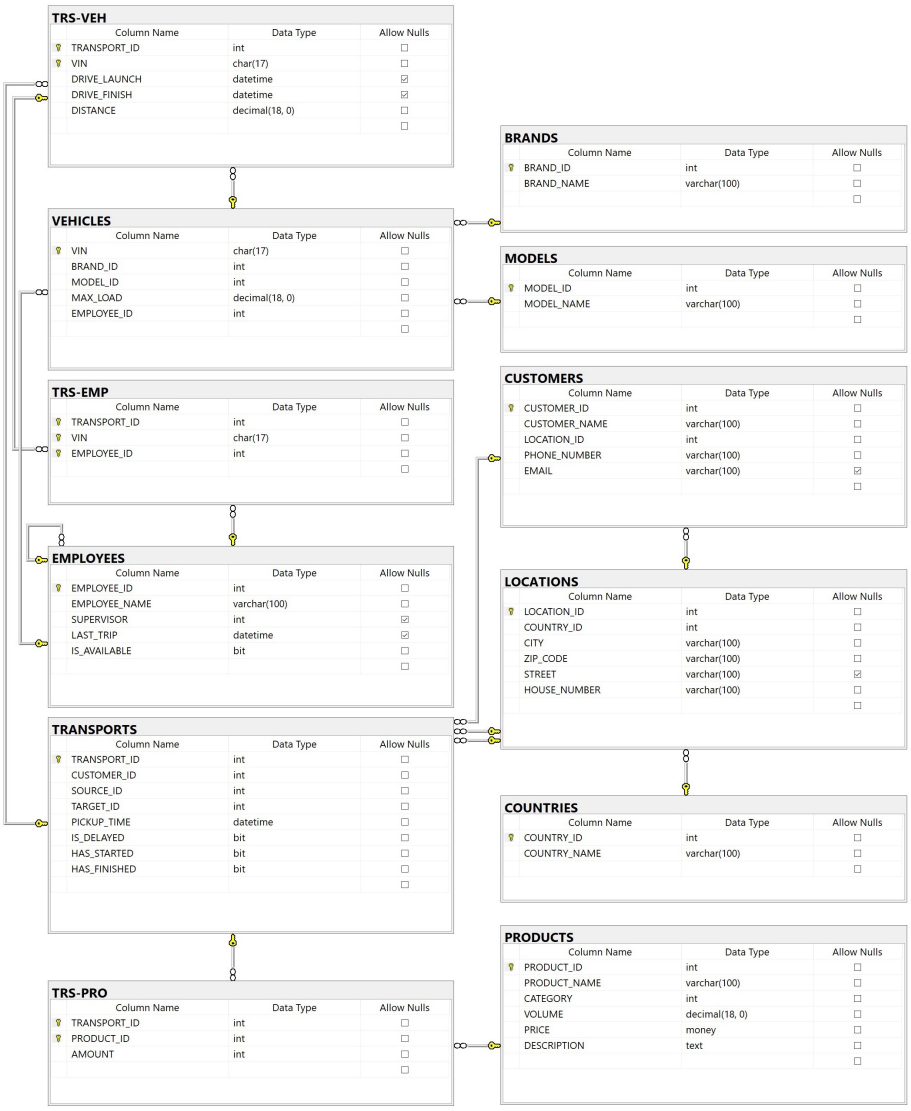
The project was done in Microsoft SQL Server Management Studio. The core table of the solution is `TRANSPORTS` table. In my solution there might be four statuses of a transport/order: pending, delayed and/or in progress, and delivered. Technically, no employee is necessary to deposit products for a delivery, therefore the transport procedure starts right after initialisation and time of this event is visible as the `PICKUP_TIME`. A database administrator can make employee(s) begin their task (`HAS_STARTED`) and the lowest value from table `TRS-VEH` (`TRANSPORTS-VEHICLES`) equals the time in which products will begin their journey. `HAS_FINISHED` also has to be checked manually, in case of adding a vehicle while an order is in progress. When drivers are unavailable, a stored procedure adds a status "delayed" represented by the `IS_DELAYED` flag.

None of values which can be obtained from different tables repeat in any other table. This means that some queries need to be performed to obtain all information about a transport from other tables requested in the task.

Although the following ER diagram is the ultimate form of the database (including modifications necessary for a store procedure), initial values from Part 2 are inserted manually and are selected in such a way that they match all conditions given by both task and stored procedures (i.e. volume of orders, driver(s) assigned, time of delivery and number of drivers). These conditions also remain fulfilled after `UPDATE`s from Part 2. The solution consists of the following files:

```
TaskNo2_Initialisation.sql
TaskNo2_SelectNo1.sql
TaskNo2_SelectNo2.sql
TaskNo2_SelectNo3.sql
TaskNo2_SelectNo4.sql
TaskNo2_SelectNo5.sql
TaskNo2_UpdateNo1.sql
TaskNo2_UpdateNo2.sql
TaskNo2_UpdateNo3.sql
TaskNo2_StoredProcedure.sql
```

2 ER diagram



The diagram was generated in Microsoft SQL Server Management Studio. It is also included as a separate file in the solution folder.

Explanation of ambiguous columns in the solution:

LOCATIONS:

- only COUNTRIES were listed in a separate table – it would be inefficient to store even a full list of cities, towns and villages in the world (7-digit number of records),
- the rest of address parts (CITY, ZIP_CODE, STREET, HOUSE_NUMBER) are defined as strings, because of different formatting of these values in different countries,
- STREET can be null as some villages with a single street do not specify such a division,
- for simplicity of a solution and in order not to repeat data among tables, both addresses for transports (SOURCE_ID, TARGET_ID) and customers locations (LOCATION_ID) are stored in this single table,

PRODUCTS:

- CATEGORY must be of value 1, 2, 3 or 4,
- VOLUME and PRICE must be positive values,

EMPLOYEES:

- each employee has his or her own supervisor represented by his EMPLOYEE_ID, yet CEO has no supervisor, therefore a NULL value will represent his or her case,
- LAST_TRIP stores a date after which an employee will be able to perform a transport – this column is necessary for stored procedures,
- IS_AVAILABLE represents if an employee is able to drive and is of a bit type, as MS SQL has no Boolean type; by default has equal to 0, this means "no delay", 1 means "delay"; other true-false statements are represented in the same way,

TRS-PRO (TRANSPORT-PRODUCTS):

- AMOUNT must be a positive integer.

3 Initialisation

Database initialisation is performed with TaskNo2_Initialisation.sql file. It is responsible for a database creation, tables creation and initial data insertion.

4 Indexes

Indexes used by me are non-clustered, as their main purpose is to speed up queries and the physical order of data do not matter. Here is the list of all indexes by their detailed usage:

- indexes with `UNIQUE` constraint, for values that need to be distinctive therefore should not repeat:

  ```
  I_COUNTRY_NAME
  I_BRAND_NAME
  I_MODEL_NAME
  I_PRODUCT_NAME
  I_PHONE_NUMBER
  ```

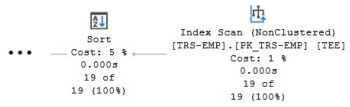
- indexes of foreign keys to speed up queries:

  ```
  I_SUPERVISOR
  I_COUNTRY_ID
  I_BRAND_ID
  I_MODEL_ID
  I_EMPLOYEE_ID_VEHICLES
  I_LOCATION_ID
  I_CUSTOMER_ID
  I_SOURCE_ID
  I_TARGET_ID
  I_TRANSPORT_ID_TRS-VEH
  I_VIN_TRS-VEH
  I_TRANSPORT_ID_TRS-PRO
  I_PRODUCT_ID
  I_TRANSPORT_ID_TRS-EMP
  I_EMPLOYEE_ID_TRS-EMP
  ```

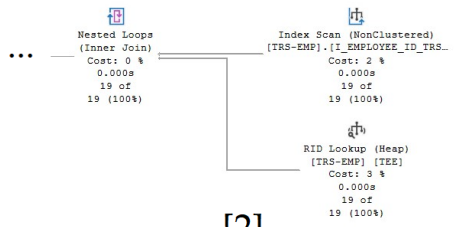
- indexes of rarely-edited but potentially often-read columns:

  ```
  I_EMPLOYEE_NAME
  I_VOLUME
  I_PRICE
  I_CUSTOMER_NAME
  I_DISTANCE
  I_AMOUNT
  ```

Even while running the following queries, changes of execution can be notice in execution plans:



[1]



[2]

Where [1] is a query using non-indexed tables, and [2] with indexes. To optimise execution even further, I applied non-clustered structure to all composite primary keys. These are:

```
[PK_TRS-VEH] ([TRANSPORT_ID], [VIN])
[PK_TRS-PRO] ([TRANSPORT_ID], [PRODUCT_ID])
[PK_TRS-EMP] ([TRANSPORT_ID], [VIN], [EMPLOYEE_ID])
```

5 `SELECT` queries

Each query is numbered to match the subtask from Part 4.

Query No. 1:
Row 5 yields:
`2021-05-13 Poland 2200`
which is a sum of orders with `TRANSPORT_ID` 5 and 6.

Query No. 2:
Inner `SELECT` allows to iterate over customers:
`WHERE C.CUSTOMER_ID = CC.CUSTOMER_ID`
and get TOP 15 values for each.

Query No. 3:
Inner `SELECT` and `LEFT JOIN` allows to get all employees regardless of earlier joins. It returns a sum of all seconds of all drivers divided by a number of all employees.

Query No. 4:
Inner `SELECT` again allows to iterate over customers:
`WHERE C.CUSTOMER_ID = CC.CUSTOMER_ID`
and returns an `ID` of a transport, for which a maximum number of drivers were assigned for a given client.

Query No. 5:
The first `WHERE` statement filters out all orders for which source and destination countries are the same (it is possible as `LOCATIONS` table was joined twice). The second returns all transports with the unwanted object.

6 `UPDATE` queries

For all updates I used `ISOLATION LEVEL READ COMMITTED`, as no important data reading was done.

Query No. 1:
This query is a cosmetic change only.

Query No. 2:
This query changes status of a few employees to unavailable.

Query No. 3:
This query inserts a new pending transport to a database, then it reads the ID of a new row and assigns vehicles, products and drivers to this order.

7 A stored procedure

The procedure `CHECK_TRANSACTION` runs two concurrent cursors, one (`AVAILABLE_EMPLOYEES`) to list employees that can replace these assigned, and the other one (`UNAVAILABLE_EMPLOYEES`) for those who are assigned but cannot drive for any reason.

First, the procedure checks if a transport with a given ID exists and has not finished yet (no need to assign anything for such an order). Then, if the list of unavailable employees is non-empty it assigns next available employee from the second list. If there are enough employees to do all the replacements, the transport is marked as "not-delayed" and a transaction is committed. When there are not enough employees available, a transaction is rolled back and in the next transaction the transport is marked as "delayed".

To show possibilities of this stored procedure, I used updates No. 2 and 3. When applied to a newly created database (with 10 orders), a new transport gets index No. 11. Without executing queries mentioned above, the procedure executed for ID equal to 11 (`EXECUTE CHECK_TRANSACTION 11;`) returns:

```
Transport No. 11
Transaction not found.
```

After running update No. 2 and 3, a new transport with an employee with `EMPLOYEE_ID = 5` appears (a part of `TRS-EMP` table):

19	10	1GCGK13U85F930679	6
20	11	3GCEC14X06G175122	5
21	11	WAUFGAFC2EN138014	4

However, this employee is unavailable, therefore after running a procedure:

```
Transport No. 11
Changes committed successfully.
```

this value changes to 3, as employee with this ID is available at the moment:

19	10	1GCGK13U85F930679	6
20	11	3GCEC14X06G175122	3
21	11	WAUFGAFC2EN138014	4

After running this procedure once again it returns:

```
Transport No. 11
Transport has no unavailable drivers assigned. Nothing to be
    modified.
```

Running the procedure for a finished transport (e.g. `EXECUTE CHECK_TRANSACTION 6;`) results in:

```
Transport No. 6
Transport has already been finished. Nothing to be modified.
```