Car park

Advanced Databases

Project

dr inż. Artur Wilczek

Group A2:

Tomasz Czarnecki

Krzysztof Kozieł

Hubert Olkiewicz

Table of contents:

1. Phase 1  
   1.1 Description  
   1.2 Structure (ERD diagram)  
   1.3 Transactions
2. Phase 2  
   2.1 Transaction time measurement  
   2.2 Data generator project screens  
   2.3 Problems
3. Phase 3
4. Phase 4  
   4.1 Transaction T1

4.2 Transaction T2

4.3 Transaction T3

4.4 Transaction T4

4.5 Transaction T5

1. Phase 5  
   5.1 Description of active rules

5.2 Description of experiments

6. Phase 6

7. Phase 7

7.1 Extensions

7.2 Operations

8. Phase 8

9. Phase 9

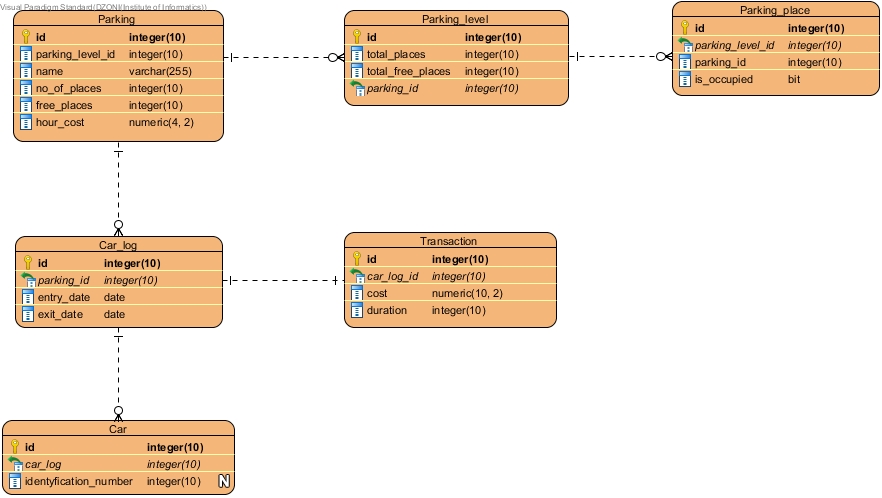
1. Phase 1  
   1.1 Description

Our topic is an example of car park which consists of many parkings with many different parking levels. Cars can enter, choose parking place on specific parking levels and also leave.

At the entrance, a car is registered in the system with timestamp and parking ID. Also system checks if the car exists in database - if not, it adds new record. The car can choose any of parking levels if there are free places. Each parking place has its own ultrasonic sensor to monitor, if place is occupied or not. At the exit, system updates car arrival record with timestamp.

Our system can also calculate each car arrival into duration time and total cost of park. That allows to make e.g. monthly statements.

1.2 Structure (ERD diagram)



*Car park ERD diagram*

1.3 Transactions

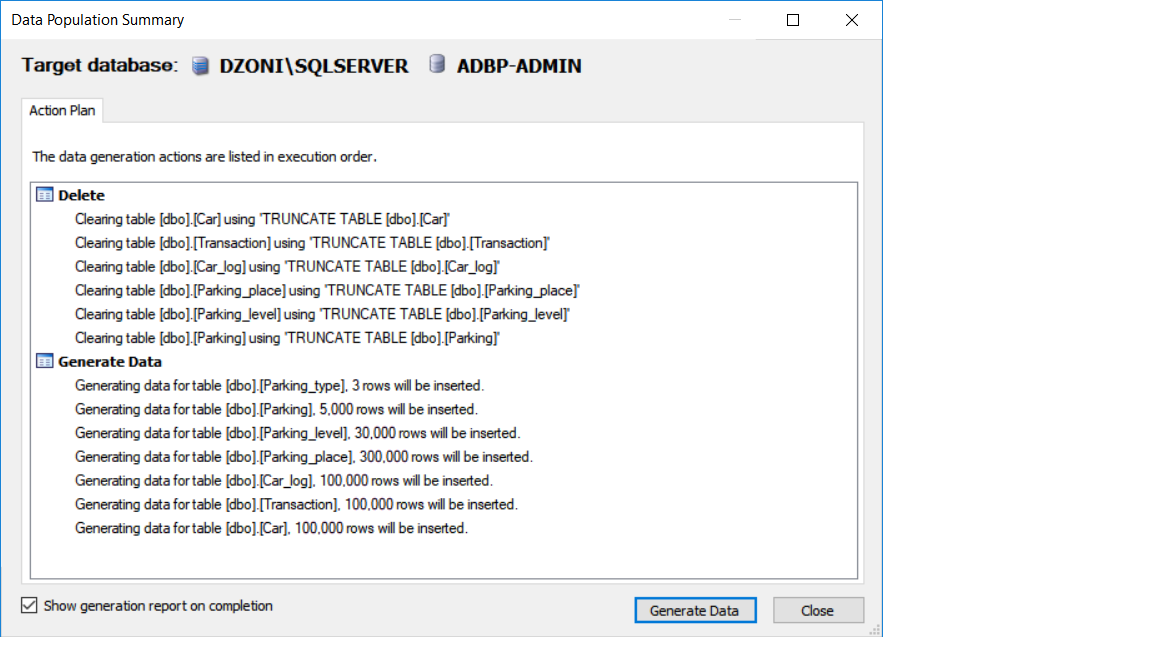
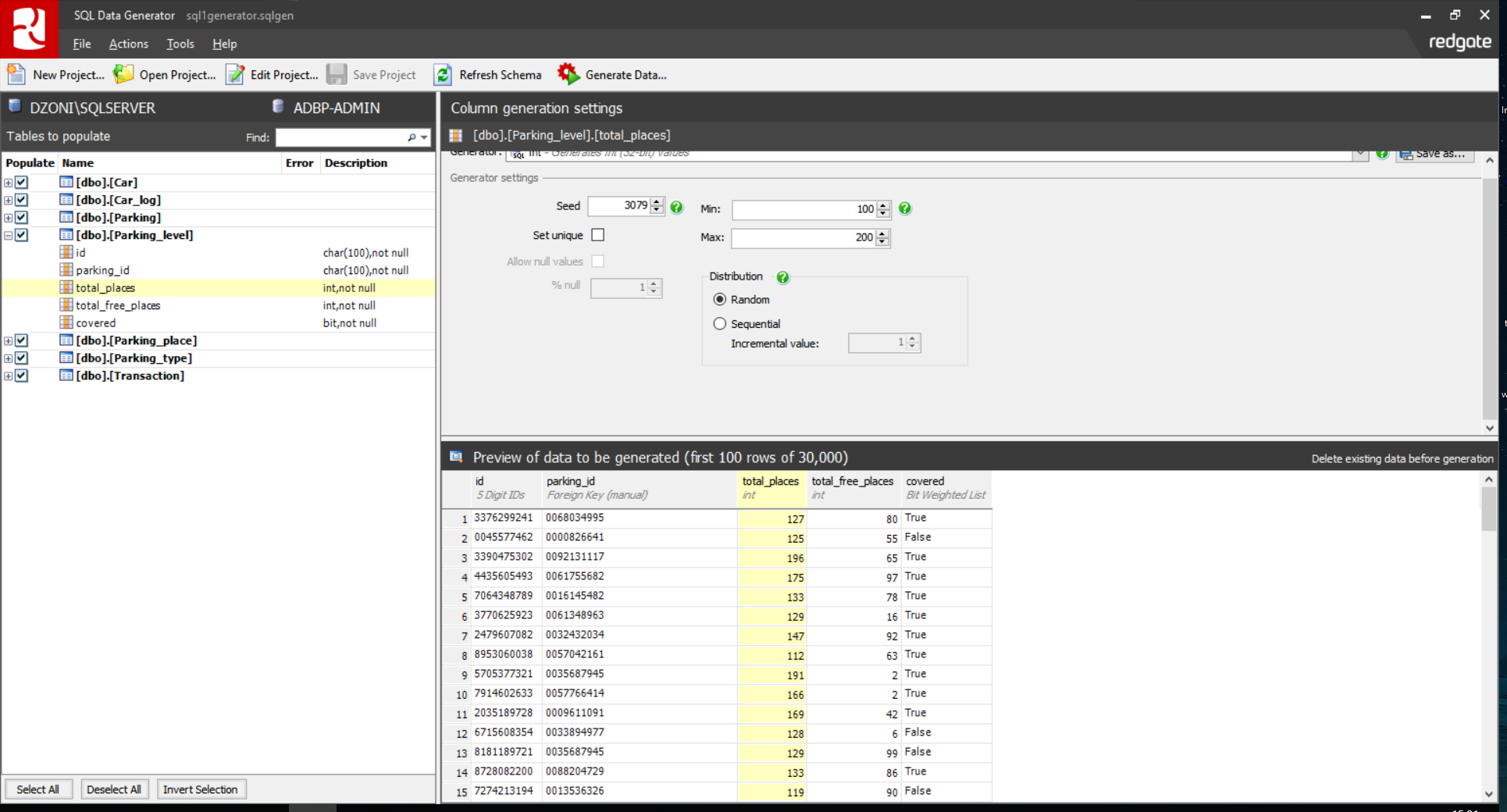
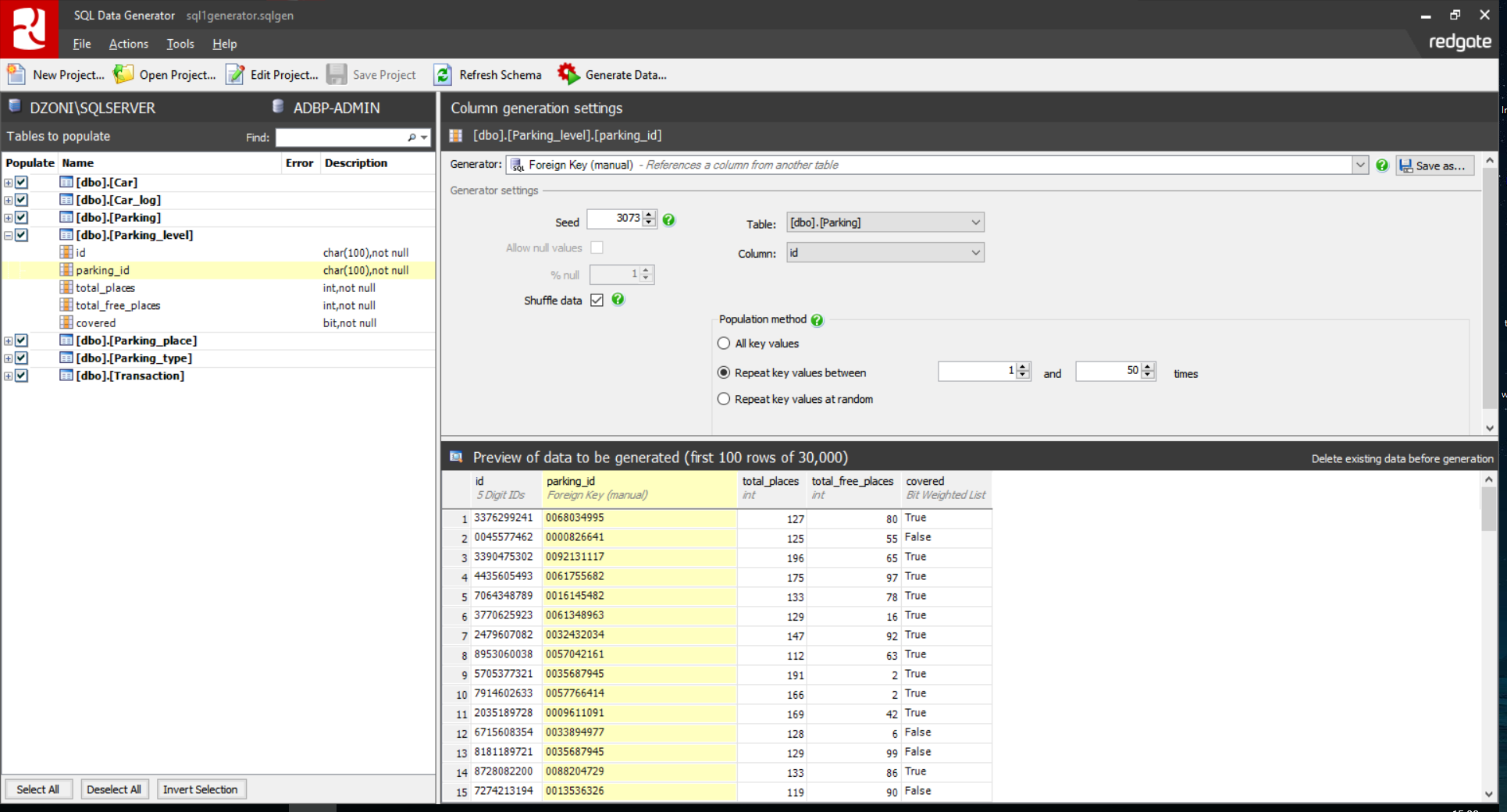
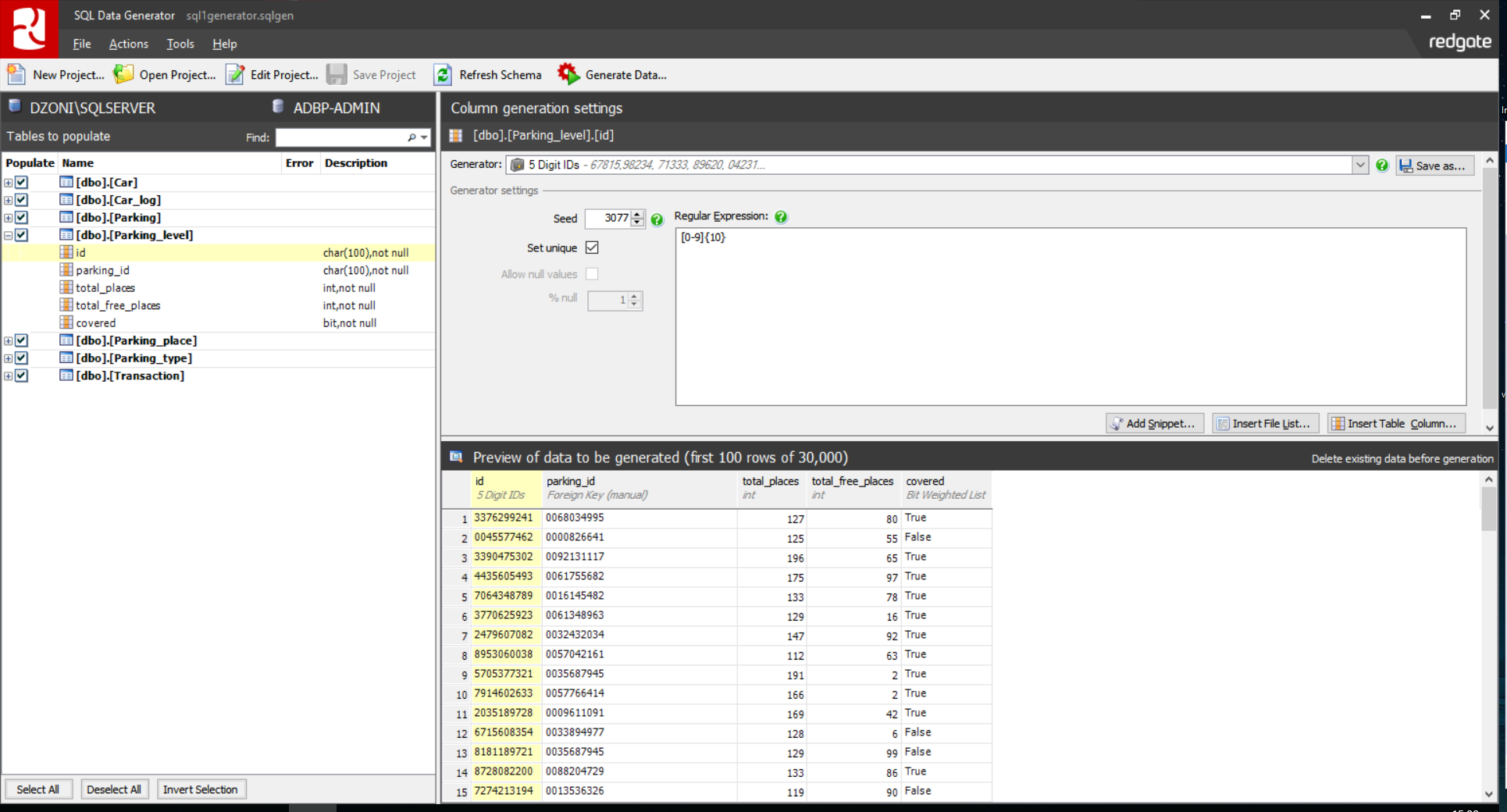
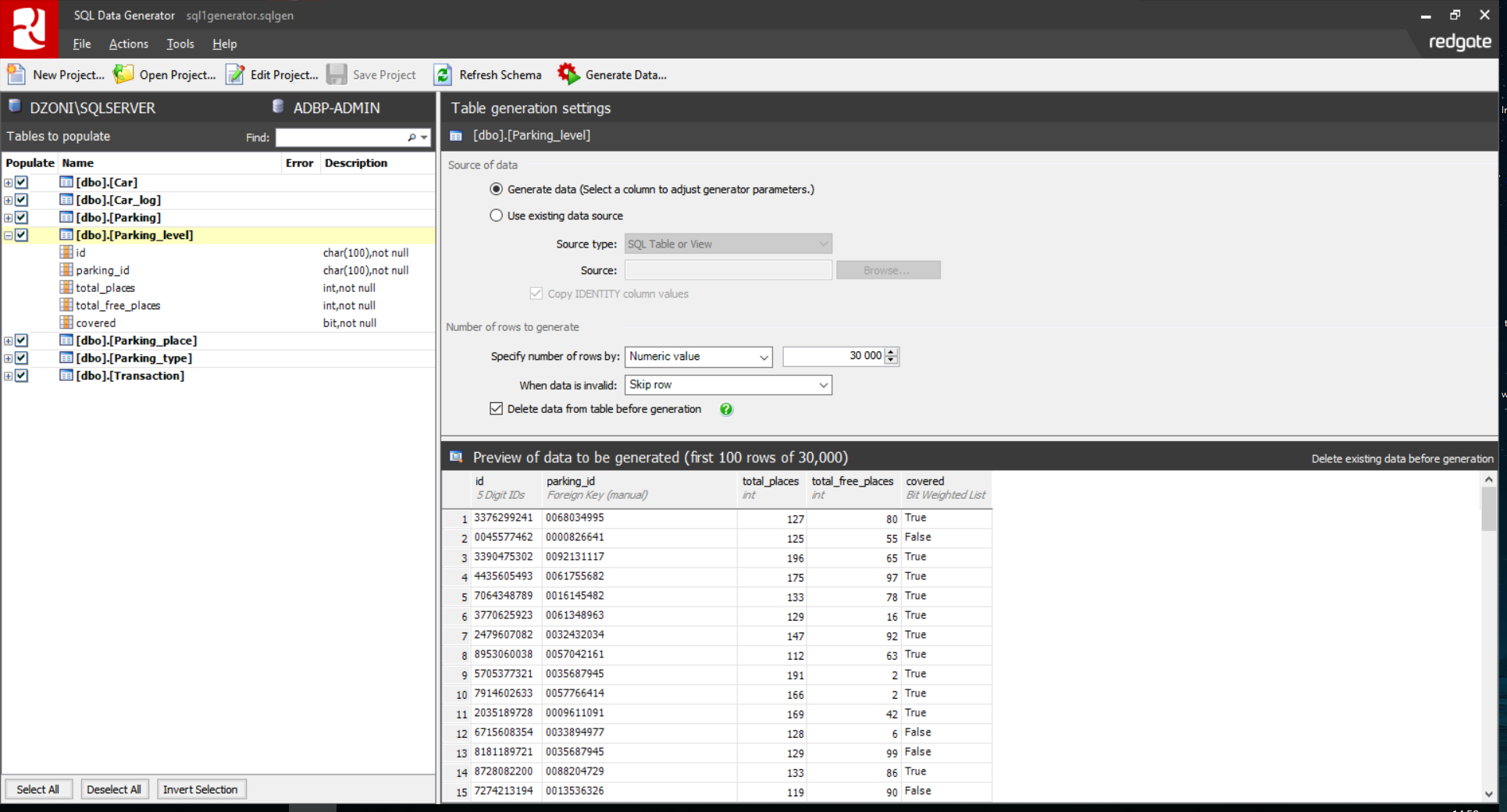
1. **Transaction calculation**- counts duration of the each car\_log, calculates cost of each car stop, updates attributes for each transaction and shows changed data to the user .
2. **Car\_log data presentation**- collects whole car\_log data and presents for it for the user
3. **Special places status assigning**  - Event finds parkings with not many free places, sets place statuses as special and shows changed places with parking level id and parking id
4. **Parking close (parking place)** - Situation when all cars must leave their parking places. Each parking place state is set as free (is\_occupied = false) and the amount of free places on the level and whole parking lot gets updated.
5. **Cars leaving** - Finds parkings with low attendance, deletes those parkings and every table with dependent data.

2. Phase 2

2.1 Transaction time measurement

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Transaction No.** | **Run1**  **(ms)** | **Run 2**  **(ms)** | **Run 3**  **(ms)** | **Run 4**  **(ms)** | **Run 5**  **(ms)** |
| **1** | 9033 | 9126 | 9083 | 9583 | 9286 |
| **2** | 4703 | 4820 | 4730 | 4866 | 4763 |
| **3** | 2586 | 2766 | 2580 | 2840 | 2806 |
| **4** | 2986 | 2933 | 2866 | 3140 | 3290 |
| **5** | 4426 | 4160 | 4096 | 3950 | 4106 |
| **Total** | **28103** | **28420** | **27933** | **28610** | **28556** |

2.2 Data generator project screens



2.3 Problems

At the beginning we had some problems with proper time measurement. We wanted to create a console application program which connects to our DB and executes stored procedures with transactions. We realized that time measurement is wrong and different than in database management program. We found solution to measure execution time in miliseconds, in SQL Management Studio. Finally, we decided to use time measurement in that way.

Our second problem was data volume. We had to increase the amount of data several times so that the execution time took seconds. We also tried to change some database engine properties, but that didn’t increase the execution time too much.

3. Phase 3

Propositions of performance improvements:

* Primary keys on tables ID column and relationship constraints (foreign keys)
* Each table ID column datatype from Char to Integer
* Non-clustered indexes (On table foreign ID column)
* Clustered indexes (On table primary ID column)
* Hash indexes
* Unique indexes

4. Phase 4

T0 - execution time without improvements.

Ix - execution time with improvement. ‘x’ stands for type.

4.1 Transaction T1

Non-clustered and clustered indexes:

* I1 - Non-Clustered - on parking\_level table, column total\_places and also on parking\_table, column no\_of\_places, and car\_log table, column entry\_date  
  Clustered - only parking\_level, parking, car\_log IDs columns.
* I2 - Clustered indexes on parking\_id on car\_log table, parking ID column on parking table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Index type** | **T0 (ms)** | **I1 (ms)** | **I2 (ms)** |
| **Clustered** | 9222 | 9100 | 8541 |
| **Non-Clustered** | 9222 | 9020 | --- |

Conclusions:

**I1** - Average execution time on T1 without any improvements - 9222 ms. Execution plan has shown that no indexes were used.

**I2** - We have achieved a bit better execution time based on execution plan analysis. parking\_id column on car\_log table was sorted so we have added clustered index which does it.

4.2 Transaction T2

Non-clustered and clustered indexes:

* I1 - Non-Clustered - all tables IDs columns.  
  Clustered - all tables IDs columns

|  |  |  |
| --- | --- | --- |
| **Index type** | **T0 (ms)** | **I1 (ms)** |
| **Clustered** | 4712 | 4420 |
| **Non-Clustered** | 4712 | 4576 |

Conclusions:

**I1** - Clustered - minor time improvement. Indexes were used but mainly joins take time in statement execution. Non-Clustered -no indexes were used.

4.3 Transaction T3

Non-clustered and clustered indexes:

* **I1** - Non-clustered -on parking, parking\_place, parking\_level IDs columns and on parking\_level\_id column on parking\_place table.   
  Clustered indexes were same with exception to parking\_level\_id column on parking\_place table.
* **I2** - Mixed - I1(Clustered version) + non-clustered index on column isOccupied on parking\_place table.

|  |  |  |  |
| --- | --- | --- | --- |
| **Index type** | **T0 (ms)** | **I1 (ms)** | **I2 (ms)** |
| **Clustered** | 2743 | 870 | 950 |
| **Non-Clustered** | 2743 | 912 | --- |

Conclusions:

**I1** - Execution time vastly improved. Execution plan has shown indexes were used.

**I2** - Additional (non-clustered) index wasn’t used hence no time improvement to I1 time.

4.4 Transaction T4

Non-clustered and clustered indexes:

* I1 - Non-clustered - parking, parking\_level, parking\_place, parking\_type IDs columns.  
  Clustered - same as I1
* I2 - Non-clustered indexes on total\_free\_places and total\_places column on parking\_level table

|  |  |  |  |
| --- | --- | --- | --- |
| **Index type** | **T0 (ms)** | **I1 (ms)** | **I2 (ms)** |
| **Clustered** | 4312 | 4723 | --- |
| **Non-Clustered** | 4312 | 2854 | 4210 |

Conclusions:

**I1** - Clustered indexes were used but time wasn’t improved, and even got a bit worse.

**I2** - Indexes weren’t used despite of fact that indexed columns are marked in WHERE statement

4.5 Transaction T5

Non-clustered and clustered indexes:

* **I1** - Non-clustered - all existing tables IDs and foreign keys columns.  
  Clustered - all existing tables IDs, and non-clustered indexes on foreign keys columns
* **I2** - Non-clustered - table parking, columns no\_of\_places and total\_places

|  |  |  |  |
| --- | --- | --- | --- |
| **Index type** | **T0 (ms)** | **I1 (ms)** | **I2 (ms)** |
| **Clustered** | 4123 | 17373 | --- |
| **Non-Clustered** | 4123 | 13803 | 4100 |

Conclusions:

**I1** - Non-clustered as well as clustered indexes were used but time got significantly worse. Execution plan shown that indexes were deleted (T5 is select>delete operation) hence worsen time than without indexes.

**I2** - Indexes weren’t used.

5. Phase 5

5.1 Description of active rules

**Rule 1: Check if car can enter parking**

Entering the car parking (Car\_log table) with specific parking id, checks if free places are greater than 0. If not, car cannot choose that parking.

**Rule 2: Decrease number of free places**

If specific place is occupied on parking level (Parking\_place table), it fires a trigger to decrease number of free parking places in two places (Parking and Parking\_level tables).

**Rule 3: Calculate car stop cost**

Parking is paid from 9 to 18. Trigger is fired on every new transaction insert and it calculates total amount to pay according to defined rule (Transaction table).

**Rule 4: Clear old data**

When you log into database, a trigger fires and checks if there are records in Car\_log table which are older than month. If so, trigger deletes that records and also its transactions records (Car\_log and Transaction table).

**Rule 5: Car arrival loop**

If specific type of car arrives to the parking, it fires a trigger which cause arrival of another car, until maximum car amount of certain type is reached (e.g 10) (Car\_log table).

**Rule 6: Free short stop check constraint**

For car\_log which duration would be less than 5, transaction can’t be inserted.

5.2 Description of experiment

**Experiment 1:**

Looping trigger. Based on active rule n.5, on approach of certain car (eg. police car) tigger will call(insert) another exactly same car. We can control loop using rules’ constraint concerning maximum car amount of same type.

**Experiment 2:**

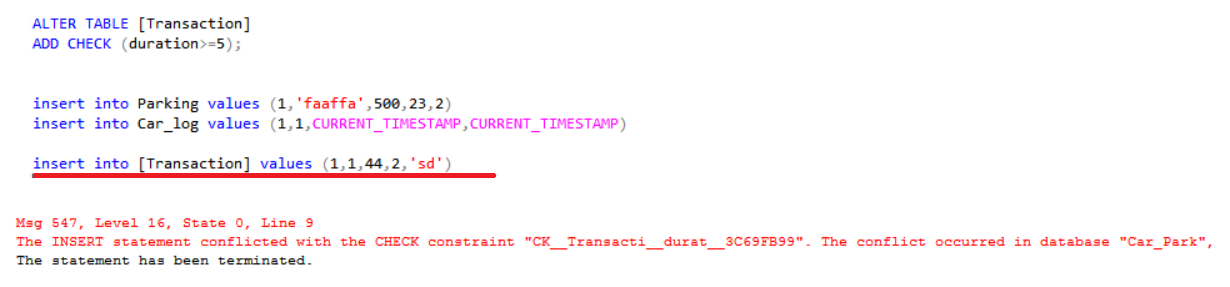
Detecting order of triggers. Two triggers will be duplicated in order to be able to check and manipulate order of firing triggers. Log table will be created to store firing order of triggers.

6. Phase 6

**Experiment 1:**

Sql server prevents triggers looping. Experiment proves that trigger executing same trigger won’t execute multiple times.

**Rule 6: Free short stop check constraint**



7. Phase 7

7.1 Extensions

**Extension 1**

Parking table:

**Name :** Parking\_places\_in\_parking

**Description :** Parking levels with parking places and those types will be stored in CLOB data type field in Parking table.

**Affected relations :** Parking - Parking\_level; Parking\_level - Parking\_place; Parking\_place - Parking\_type;

**Extension 2**

Parking table:

**Name :** Extended\_carlog\_for\_parking

**Description :** Car log data extended by transaction and car tables data will be stored in XML data type field in Parking table.

**Affected relations :** Car\_log - Car; Car\_log - Transaction; Parking - Car\_log;

**Extension 3**

Car log table:

**Name:** Transaction\_extended\_by\_parking\_data

**Description:** Transaction data extended by parking table data will be stored as XML data type in car log table.

**Affected relations:** Transaction - Parking; Car\_log - Transaction;

7.2 Operations

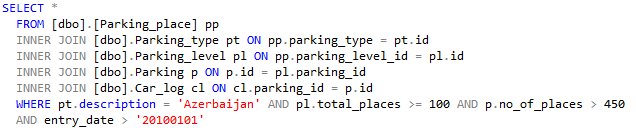
For each operation query we will use XPATH.

**Operation 1:**

**Name**: Total data select

**Description**: Searching total data for given parameters.

**Corresponding SQL statement:**

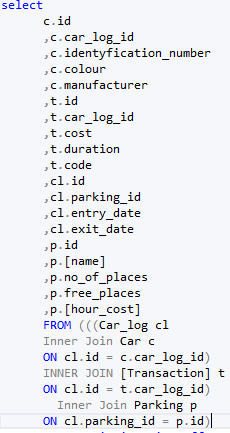


**Operation 2:**

**Name**: Extended\_carlog\_select

**Description**: Car log extended by parking and transaction data select.

**Corresponding SQL statement:**

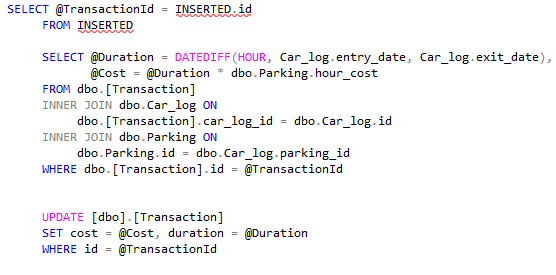


**Operation 3:**

**Name**: Transaction\_update

**Description**: Calculating cost of parking stop duration and those values update in Transaction table.

**Corresponding SQL statement:**

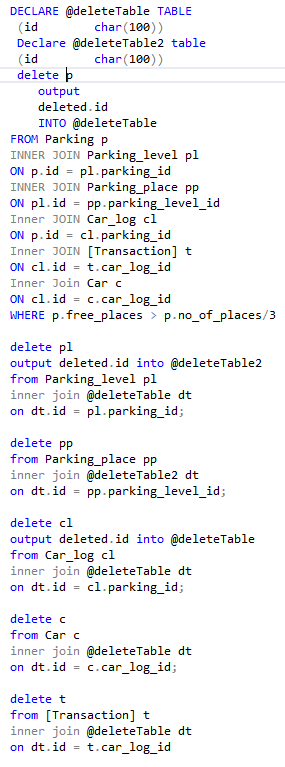
****

**Operation 4:**

**Name:** Not\_popular\_parkings\_delete

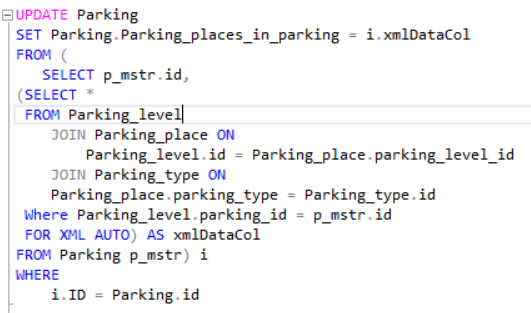
**Description:** Parkings with a lot of free places will be deleted with all related tables.

**Corresponding SQL Statement:**

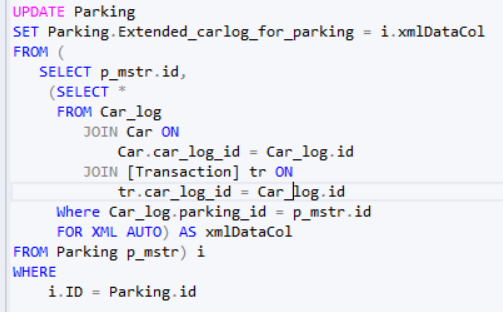
****

**8.1 Phase 8**

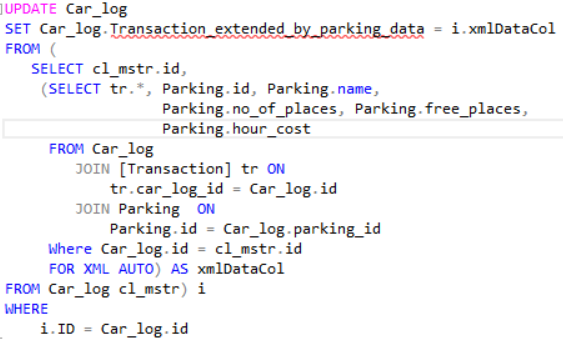
Extension 1 insert XML:



Extension 2 insert XML:



Extension 3 insert XML:

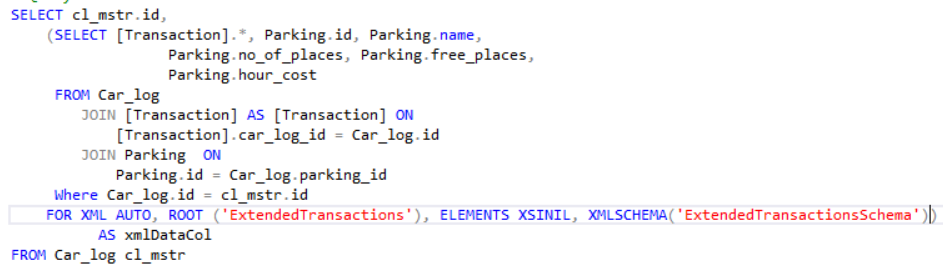


**9. Phase 9**

**XML Schema**

We have generated XSD of each database extension using XML AUTO with XMLSCHEMA clause.

Example:



To register the schema in database we have used command:

CREATE XML SCHEMA COLLECTION <nameOfSchemaCollection> AS <XSD\_structure>.

We had to change VARCHAR(MAX) column types of extensions to XML types. Otherwise, we could not assign schemas to our extensions.

**Generating XML**

We could genereate proper XML structures by using XML AUTO and XML PATH mode. With XML RAW we had problems with nesting more than one object in node. In XML EXPLICIT case it was not possible because we generate XML structure in subquery.

Error message:

*The FOR XML and FOR JSON clauses are invalid in views, inline functions, derived tables, and subqueries when they contain a set operator. To work around, wrap*

*the SELECT containing a set operator using derived table or common table expression or view and apply FOR XML or FOR JSON on top of it.*

**Validating XML**

We faced two problems with validation. First one, with datetime value - we had to convert it to smalldatetime value.

Error message:

*XML Validation: The canonical form of the value '2016-06-27T00:14:51.070' is not valid according to the specified type. This can result from the use of pattern facets on non-string types or range restrictions or enumerations on floating-point types.*

The second problem was with empty elements. We did not want to generate extension column in another extension element, so we had to add XSINIL to ELEMENT clause in FOR XML.

**Operations:**

1. Selecting extended parking places data for Parking

Execution time: **18833** **ms**

Execution time with index: **32253 ms**

Selecting semistructured data from Parking\_level node and Parking\_place nested node using XQuery and CROSS APPLY.

1. Selecting extended transactions data for Car\_log

Execution time: **9637 ms**

Execution time with index: **23227 ms**

Selecting semistructured data from Transaction node and Parking nested node using XQuery and CROSS APPLY.

1. Calculating cost and duration for Transactions

Execution process took at least 15 minutes, so we decided to limit querying data with SELECT TOP (200) and its execution time was **67627 ms**.

Execution time with index: **400 ms**

To achieve this operation we created temp table with IDs which we looped through. In loop, firstly we calculated cost and duration variables and then assign them in update query statement with XQuery. We also had to make another update query to set the second variable. With XQuery we could not assign two values in one update query.

1. Deleting parkings from Car\_log (nested nodes)

Execution time: **68640 ms**

Execution time with index: **430 ms**

Deleting Parking nested nodes for given parameters.

1. Deleting car logs from Parking (main nodes)

Execution time: **10056 ms**

Exectution time with index: **27350 ms**

Deleting entire xml objects for given parameters.