MINISTRY OF SCIENCE AND HIGHER EDUCATION
OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
"NORTH CAUCASUS FEDERAL UNIVERSITY"

# METHODOLOGICAL INSTRUCTIONS

for completing laboratory work
in the discipline "Database tuning" for master's students of the direction 09.04.03 "Applied informatics" focus (profile) "Management knowledge"

Stavropol 2026

CONTENT

# INTRODUCTION

The aim of this course is to develop master's degree students understanding of the main trends in the development of information systems related to changes in the conditions in the field of application, development of professional skills in the field of forecasting, modeling and creation of information processes in a specific subject area. This course should form in masters the knowledge and skills necessary for the correct selection of tools for creating information systems, determining a suitable data model, organizing requests to stored data and other aspects on which the effectiveness of the systems being developed largely depends. During the course, the master should form an idea of
promising information technologies for the creation, analysis and maintenance of professionally oriented information systems.

In the course of achieving the goal, the following tasks are solved:

- development of logical and algorithmic thinking;
- study of the principles of operation of software and hardware tools and organization of data in information systems using databases;
- mastering work with modern DBMS;
- development of the ability to independently solve processing problems text and non-text information in the database;
- acquiring skills in problem algorithmization, programming on algorithmic language, debugging and executing tasks on a personal computer;
- study of the prospects for the development of information technologies in information systems in the subject area;
- study of information resource markets and their features use.

LABORATORY WORK #1Index structures Subject -

Index structures

The purpose of the work isMaster index structures. The

competencies being formed or their parts -OPK-5

Theoretical part.

Indexes allow you to find information in huge databases as efficiently as possible.

SQL Server 2008 supports two basic types of indexes: clustered and nonclustered. Both types of indexes are implemented as a balanced tree (B-tree), with the leaf level at the bottom of the structure. The difference between the two types of indexes is that a clustered index provides physical ordering of data on disk. A clustered index is sparse - pointers in the B-tree leaves point to the data page.

A nonclustered index is dense and contains only the columns included in the index key. In dense indexes, pointers in the B-tree leaves point to rows of real data. If a table does not have a clustered index defined, it is called a heap or unsorted table. In the latter case, the table is physically organized (sorted) in the order in which new records are added, unlike tables with clustered indexes, which are ordered by the values of the sort key. It can be said that a table can be represented in one of two forms, as a heap or as a clustered index [1].

Clustered Indexes

Clustered indexes can be created based on one or more table columns - such an index is called an index key and has a number of limitations:

- The index cannot span more than 16 columns; the
- maximum size of an index key is 900 bytes.

The columns of a clustered index are called the clustering key. A clustered index has a special effect on SQL Server because it forces it to order the data in the table according to the clustering key. Because a table can only be ordered one way, you can only define one clustered index on it.

Clustered indexes specify the sort order of the data in a table. However, clustered indexes do not provide a physical sort order. A clustered index does not result in physical ordering of the data on disk,

because this would result in a large number of disk I/O operations when splitting pages. It only ensures that the indexed page chain is logically ordered, allowing SQL Server to traverse the page chain directly when searching for data. As SQL Server traverses the indexed page chain, the data rows are read in clustering key order [2].

Non-clustered index

A nonclustered index does not impose any restrictions on the ordering of records in the table, so you can create many nonclustered indexes on a single table, but these indexes have the same restrictions as clustered indexes:

- The index cannot span more than 16 columns; the
- maximum size of an index key is 900 bytes.

The leaf level of a nonclustered index contains a pointer to the data. If the table has a clustered index, the leaf level of the nonclustered index points to the clustering key. If there is no clustered index, the leaf level pages point to the data rows in the table [2].

The general syntax for creating a relational index is:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX index_name
    ON <object> (column [ASC | DESC] [, … n]) [INCLUDE
    (column_name [, … n] )]
    [WHERE <filter_predicate> ]
    [WITH (<relational_index_parameters> [, ... n] )]
    [ON {partition_schema_name ( column_name ) | file_group_name | default }]

    [FILESTREAM_ON { filestream_group_name |
section_schema_name | "NULL"}][ ; ] [3]
```

Composite index

A composite index can be created based on several fields. In this case, the restrictions described earlier apply. If the index is built on fixed-size fields, the sum of the lengths of these fields must not exceed these 900 bytes; if the index is built on variable-length fields, the sum of the maximum field sizes can exceed 900 bytes, but the sum value for each record cannot exceed 900 bytes. For example, a table has two variable-length fields of 500 bytes each. SQL Server allows you to create a composite key based on these two fields if there are no records whose sum of lengths for both fields exceeds 900 bytes. It is worth noting that a composite index for (Column1, Column2) is different from (Column2, Column1), as well as from indexes created for these two fields separately.

Index fragmentation

Operating system files typically become fragmented over time due to repeated write operations. Indexes can also become fragmented, but index fragmentation is different from file fragmentation.

When an index is created, all index key values are stored in order on index pages. When a row is deleted from a table, SQL Server must delete the corresponding entry in the index, which creates "holes" in the index page. SQL Server does not reclaim the freed space because the cost of detecting and reusing "holes" in the index is too high. If a value in the base table changes, SQL Server moves the entry with the pointer to another location, which creates another "hole". When index pages become full and page splits are required, index fragmentation occurs again. Over time, table indexes that have data changes become fragmented [2].

To control the degree of index fragmentation, a parameter called the fill factor is commonly used. You can also use the ALTER INDEX statement to eliminate fragmentation. The fill factor is an index parameter that specifies the percentage of free space that is reserved on each leaf-level page when an index is created or rebuilt. The reserved space allows additional values to be placed in the future, thus reducing the number of page splits. The fill factor is measured in whole percentages, for example, a value of 75 means that each leaf-level page created must contain 25% free space to accommodate future values [1].

Defragmenting indexes

Because SQL Server does not return space to the system, it is necessary to periodically free up empty space in an index to maintain the performance gain that created the index in the first place. To defragment indexes, use the ALTER INDEX statement [2].

```
ALTER INDEX { index_name |
      ALL} ON <object>
      { REBUILD
            [ [PARTITION = ALL]
               [ WITH ( <rebuild_index_option> [ ,...n ] ) ] | [ PARTITION
               =partition_number
```

```
                    [ WITH ( <single_partition_rebuild_index_option>
                        [ ,...n ] )
                ]
            ]
        ]
    | DISABLE
    | REORGANIZE
        [ PARTITION =partition_number ]
        [ WITH ( LOB_COMPACTION = { ON | OFF } ) ] | SET
  ( <set_index_option> [ ,...n ] )
    }
[ ; ] [3]
```

When defragmenting indexes, you can select the REBUILD or REORGANIZE options.

The first parameter rebuilds all index levels and fills the pages according to the fill factor parameter. When rebuilding a clustered index, only the clustered index is rebuilt, but if you specify the ALL parameter, both the clustered index and all nonclustered indexes on the table are rebuilt. Rebuilding an index updates the entire B-tree structure, so unless the ONLINE parameter is specified, the table is locked until the rebuild is complete [2]. For example, to rebuild the IX_BillID index on the BillItem table, you would execute the following query:

```
ALTER INDEX IX_BillID
ON BillItem
REBUILD
```

The REORGANIZE option eliminates defragmentation at the leaf level only. Intermediate-level pages and the root page are not defragmented. The REORGANIZE operation is always performed online, so it does not cause a long-term table lock. [2] For example, to reorganize the IX_BillID index on the BillItem table, you would run the following query:

```
ALTER INDEX IX_BillID
ON BillItem
REORGANIZE
```

Working with Indexes in MS SQL Server Management Studio

To see what indexes have been created, open the Index tab.

Bill tables in the Object Explorer panel. Full path to the tab: Databases - EducationDatabase - Tables - [table name] - Indexes is shown in Figure 1.1. According to the figure, one clustered index PK_Bill has been created for this table.

Check for clustered indexes on all tables in your database yourself.

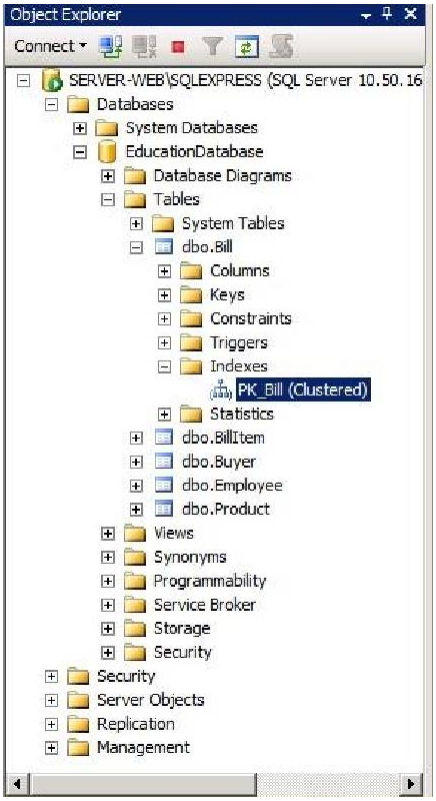



Figure 1.1 – Object Explorer, Indexes tab expanded

Let's create an additional index on the BillItem table's foreign key field BillID. There are two ways to create an index:

Executing the CREATE INDEX query. Let's create a query in a new tab by clicking the New Query button on the standard toolbar. The toolbar is shown in Figure 1.2.


Figure 1.2 – Toolbar

After opening a new tab, we will execute the query shown in Figure 1.3. In order to execute the query, you need to click the Execute button on the toolbar (Figure 1.2), or press the F5 key on the keyboard.

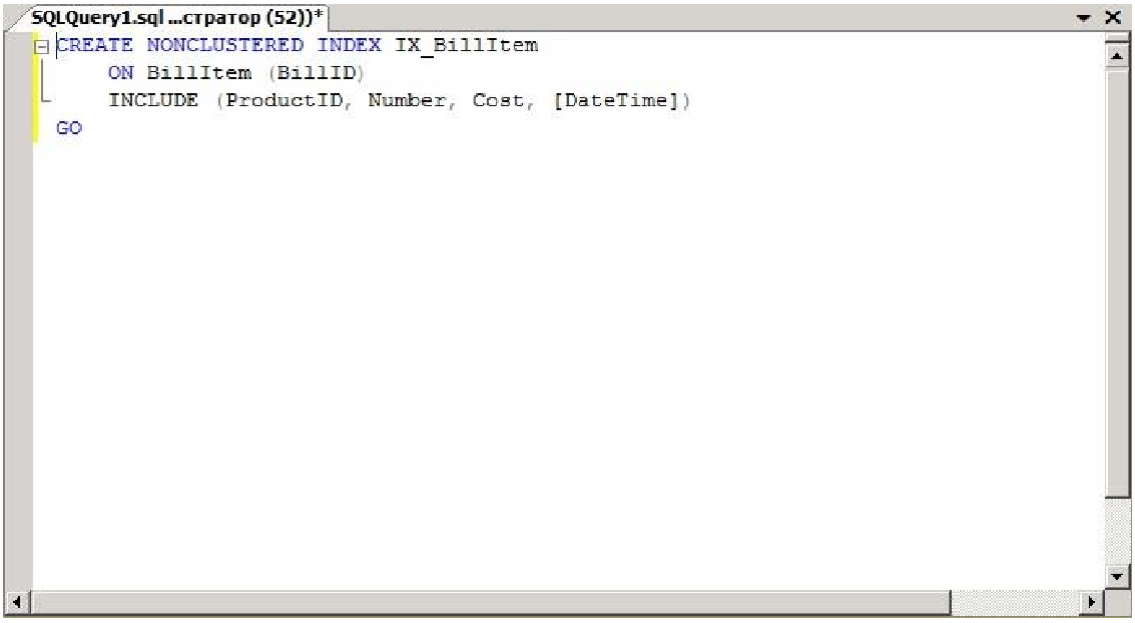



Figure 1.3 – CREATE INDEX query

Using the Microsoft SQL Server Management Studio graphical interface. In the context menu, on the Indexes tab, select New Index, as shown in Figure 1.4.

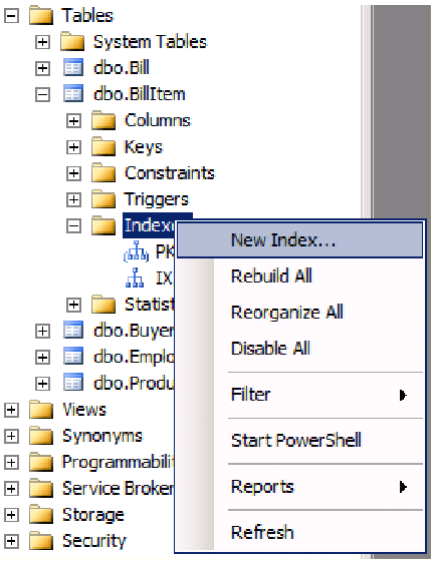



Figure 1.4 – Context menu of the Indexes tab

In the window that opens, you must specify the index name, sorting attributes, and index type (clustered, non-clustered, or primary XML index). If the table already has a clustered index, when you try to create a new clustered index, the system will issue a warning about the possibility of deleting the existing index and creating a new one. When you create a clustered index, all non-clustered indexes are rebuilt.

In addition, in the index creation window, you can specify the support flag for unique values in indexed fields. The presence of such an index will prevent duplicate values from being added to indexed fields.

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Check for indexes on key fields of the table. If necessary create clustered indexes. To create a new index, use the CREATE INDEX command, or in Microsoft SQL Management Studio, under Tables/table_name/Indexes, use the New Index… command.
2. Create non-clustered indexes on the foreign key fields of the database tables data. Explain why such indexes are needed?
3. Create non-clustered indexes on the information fields: Name and Date in all tables of the database. Explain why such indexes are needed?
4. For the cluster index and the index on the Date field of the check records table Get information about advanced index properties. Explain the meaning of the information provided in the Fragmentation section of the Properties page. Explain how the index tree depth, number of leaves, and fragmentation factor are calculated.
5. Rebuild the clustered index on the BillItem table using the command ALTER INDEX or by using the Rebuild command in the index context menu.

6. Prepare material for inclusion in the course report presentation "Database tuning".

Test questions:

1. Create non-clustered indexes on the foreign key fields of the database tables data. Explain why such indexes are needed?

2. Create non-clustered indexes on the information fields: Name and Date in all tables in the database. Explain why such indexes are needed?

3. For the cluster index and the index on the Date field of the check records table, get information about advanced index properties. Explain the meaning of the information provided in the Fragmentation section of the Properties page. Explain how the index tree depth, number of leaves, and fragmentation factor are calculated.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

LABORATORY WORK #2.Connecting data tables

Subject -Connecting data tables

The purpose of the work ismaster joining data tables

The competencies being formed or their parts -OPK- 5

Theoretical part. JOIN operator

The JOIN operator implements a relational join operation based on a predicate rule. The rule itself can be simple or complex based on a composition of simple predicate rules. The most common and practically significant is the so-called equijoin, implemented by the binary operation of attribute equality "=". Equijoin is used to restore, without loss of information, data from tables that have undergone the normalization procedure and the application of the projection operation to them.

Other possible variants of simple predicates are implemented through the operations: "<", "<=", "<>", "=>", ">". In general, there is no restriction on the naming of the columns of the joined tables. Using the same names for the primary and foreign key columns in related tables makes the data management process more convenient and is a recommended practice. Although the actual relationships implemented by creating PRIMARY KEY and FOREIGN KEY constraints are not required, at least one column in each table must have the same value (the most common case, an equijoin) for the results to be meaningful [1].

If a column with the same name used in a query appears in more than one table, in the select list, or when listing columns in the WHERE clause and other clauses of the SELECT statement, you must specify the column name along with the table name. For example, the Name column appears in the Product, Employee, and Buyer tables. If you are writing a query that joins these tables and want to include the Name column in the select list or elsewhere in the query, you must specify the columns as Employee.Name, and so on. To avoid long code due to long schema and object names, you can use table name aliases. The following example shows how to use aliases.

```
SELECT B.BillID,
       B.U..Name AS BuyerName,
       E.Name AS EmployeeName
FROM Bill AS B JOIN Employee AS E ON B.EmployeeID=E.EmployeeID
               LEFT JOIN Buyer AS B.U. ON B.U..BuyerID=B.BuyerID
```

The query results in a selection of the buyer's and store employee's last names for each transaction. The buyer's last name is only displayed if it

was specified in the transaction (connectionLEFT JOIN). Otherwise, NULL is output.

When defining the condition of the JOIN operator, you must specify the tables to be joined, the type of join, and the join condition formed from the columns.

There are several types of connections.

Inner join

Inner joins return only those rows that satisfy the join condition. Although an inner join can be specified in either the FROM or WHERE clause, it is recommended to specify the JOIN operator in the FROM clause [1].

The following example returns the purchase date and receipt ID from the Bill table, the receipt record ID, the item ID, the quantity of items purchased, the receipt record date, and the cost from the BillItem table. In this example, aliases are used not only for the table names but also for the column names, because both tables have a DateTime and BillID column. Note that the other columns do not require fully qualified names, because their names are unique in both tables.

```
SELECTB.BillID,
        B.Date ASBillDT,
        BillItemID,
        ProductID, Number,
        Cost,
        BI.Date ASBillItemDT
FROMBillASBINNER JOINBillItemASBIONB.BillID=BI.BillID
```

Because an inner join is specified, rows from the Bill table that do not have a match in the BillItem table are not returned. To overcome this behavior, use the OUTER JOIN operator.

Outer join

An outer join can be used to return all rows from one table and data from rows in a second table that match rows in the first, or to return all rows from all tables specified in the JOIN clause. The word OUTER can be omitted from the syntax, but LEFT, RIGHT, or FULL must be specified [1]. Remember that using outer joins defines an unambiguous order in which tables are joined. If a similar result can be obtained using inner joins, it is recommended to avoid using outer joins.

You can use the LEFT and RIGHT keywords to specify the table from which all rows are returned. When you specify a LEFT OUTER JOIN, all rows from the table specified to the left of the JOIN keyword are returned. This table is called the outer table. You can achieve the same result with either LEFT or RIGHT by changing the order in which you specify the table names. For example, the following two queries return the same set of data:

```
SELECTB.BillID,
        B.Date,
        B.U..Name
FROMBillASBLEFT JOINBuyerASB.U.ONB.BuyerID=B.U..BuyerID

SELECTB.BillID,
        B.Date,
        B.U..Name
FROMBuyerASB.U.RIGHT JOINBillASBONB.U..BuyerID=B.BuyerID
```

The FULL OUTER JOIN operator displays every row from both tables JOIN clauses. This can be useful for finding unmatched rows when the tables do not implement referential integrity. When there are foreign key constraints on the join column, the FULL OUTER JOIN operator provides the same results as the LEFT OUTER JOIN operator on the table in which the foreign key is defined to the left of the JOIN keyword. This is because the foreign key column must have a corresponding row for the primary key column to be satisfied. Additionally, when foreign key constraints are specified, an OUTER JOIN operator defined on the table with the primary key specified as the outer table provides the same results as an INNER JOIN operator.

Cross joins

A cross join produces a result called the Cartesian product of the two tables. In the JOIN clause, each row from the first table is joined to each row from the second table. This type of join is used only in exceptional situations. The syntax for the CROSS JOIN operator is the same as for other types of joins, except that this join does not require you to specify the join conditions. To obtain a selection of all possible combinations of buyers and sellers, you would execute the following query: [2]

```
SELECTB.NameASBuyerName,
        E.NameASEmployeeName
    FROMBuyerASBCROSS JOINEmployeeASE
    Presentations
```

Some of the SELECT statements you create are used to solve a specific, one-time problem, while other SELECT statements are used over and over again in your production environment. Some queries that are used repeatedly in your production environment contain complex business logic and complex T-SQL code that you don't want to reproduce every time you need the query. SQL Server allows you to store a SELECT statement in the database using an object called a view[1].

A view is a simple SELECT statement with a given name that is stored in the database. The main advantage of a view is that once created, it acts like a table for all other SELECT statements you want to write [1].

The syntax for creating a view is as follows:

```
CREATE VIEW [schema_name.] view_name [ ( column [, ... n] )] [ WITH
<view_attribute> [, ... n]] AS
select_statement [WITH
CHECK OPTION] [;]
```

A SELECT statement defined for a view can reference tables, other views, and functions, but cannot do the following:

- contain a COMPUTE or COMPUTE BY clause;
- create permanent or temporary table using keyword

INTO;

- use the OPTION clause; reference a
- temporary table; reference a variable of
- any type;
- contain an ORDER BY clause if the TOP parameter is not specified.

A view can contain multiple SELECT statements as long as you use the UNION or UNION ALL operators.[3]

Stored procedures

Stored procedures represent an abstraction layer that shields the application from the underlying database structure. As the backbone of any SQL Server application, stored procedures allow you to change the structure of your database and manage performance without rewriting applications or distributing updates [2].

A stored procedure is one or more instructions that are given a name and stored in a database. Almost any T-SQL command can be

included in a stored procedure, making the procedures suitable for applications and administrative tasks. The only commands that cannot be used in stored procedures are:

- USE <database_name>;
- SET SHOWPLAN_TEXT;
- SET SHOWPLAN_ALL;
- SET PARSEONLY;
- SET SHOWPLAN_XML;
- CREATE AGGREGATE;
- CREATE RULE;
- CREATE DEFAULT;
- CREATE SCHEMA;
- CREATE FUNCTION or ALTER FUNCTION; CREATE
- TRIGGER or ALTER TRIGGER; CREATE
- PROCEDURE or ALTER PROCEDURE;
- CREATE VIEW or ALTER VIEW.

The first time a stored procedure is called, SQL Server generates compilation and execution plans that are stored in the query cache and reused by subsequent executions. Therefore, by using stored procedures, you can achieve a small performance gain because subsequent executions of the stored procedure eliminate the need to parse, compile, and generate a query plan. But the main purpose of a stored procedure is to create a secure layer and application interface for your databases that isolates the application from changes in the database structure.

The following is a general syntax for creating a stored procedure [2].

CREATE {PROC | PROCEDURE} [schema_name.] procedure_name [; number]

[ {@parameter [type_schema_name.]data_type}
[VARYING] [= default] [OUT | OUTPUT] [READONLY] ] [, ... n]

[ WITH <procedure_option> [, ... n]] [ FOR
REPLICATION]
AS {<sql_statement> [;][, ... n] | <method_specifier>} [;]

Contents of the work.

1. Create a query that will generate a selection of data on receipt records with including data on sold goods. Include the following fields in the selection: BillID, BillItemID,

ProductID, Product.Name, Number, Price, Cost, Bill.Date, BillItem.Date. Use an inner join of tables (INNER JOIN). Apply the filter condition according to your option.

2. Create a query that will generate a data sample with information about receipts, sold goods and buyers with the following fields: BillID, BillItemID, ProductID, BuyerID, Product.Name, Number, Price, Cost, Bill.Date, BillItem.Date, Buyer.Name, Photo. Use an outer left join from the Bill table to the Buyer table (LEFT JOIN).

Explain what the resulting query does? Why is an outer join used in this case, and not an inner one as in the previous example?

3. Create a query that will generate a data sample with information about receipts, sold goods, customers and employees, expanding the list of fields in the query from point 2. Use an outer right join from the Bill table to the Employee table (RIGHT JOIN).

Explain what this query does?

4. Create a query that will generate a selection with data on receipts and buyers with the following fields: BuyerID, BillID, Name, Date. Limit the data selection using a date filter in the receipts table (use a filter in the WHERE clause). Use an outer full join between the Buyer and Bill tables (FULL JOIN).

Explain what this query does?

5. Create a query that will generate a selection containing all possible pairs (employee, buyer) with the following fields: EmployeeID, BuyerID, Buyer.Name, Employee.Name, Employee.Post. Use a cross join of the Employee and Buyer tables (CROSS JOIN).

6. Save the query from step 1 as a view. In the Microsoft SQL environment Management Studio, in the Views section, use the New View… command. Name the view BillAndProduct. Check the result with the command: SELECT * FROM BillAndProduct.

7. Save the queries from points 2 and 3 in a similar way.

8. Save the query from point 4 as a stored procedure with one parameter. – date. To create the procedure, use the CREATE PROCEDURE command or use the New Stored Procedure command in the Programmability/Stored Procedures section. Name the stored procedure BillAndBuyer. Test the functionality of the procedure by calling it: exec BillAndProduct @date='2011-07-05'.

9. Save the query from step 5 in a stored procedure.

10. Prepare material for inclusion in the final presentation on the Basics course data: special course.

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Filter by a single product name. Use a single condition filtering "=". Select the product name yourself.

2. Filter by multiple product names. Use the in filter condition. Select the list of products yourself.

3. Filter by a single date. Use a single filter condition "=". Select the check date yourself.

4. Filter with range limitation by receipt dates. Use filtering by range between or together "<=", "=>". Choose the date range yourself.

5. Filter by half-open check date range. Use filter by the condition "<=" or "=>". Choose the date range yourself.

Test questions:

1. Filter by multiple product names. Use the in filter condition. Select the list of products yourself.

2. Filter by a single date. Use a single filter condition "=". Select the check date yourself.

Filter with range limitation by receipt dates. Use filtering by range between or together «<=», «=>». Choose the date range yourself.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

LABORATORY WORK #3.Table Join Algorithms Subject -Table Join Algorithms

The purpose of the work ismaster table joining algorithms

The competencies being formed or their parts -OPK-5

Theoretical part.

One of the most important aspects of query tuning is measuring or quantifying performance. When measuring performance, you need to know what to actually measure, i.e. what metrics to use. SQL Server has three main metrics to consider: query cost, page reads, and query execution time.

Cost of request

Query cost is an internal metric used by SQL Server that takes into account the CPU and I/O resources consumed by a query. In theory, the lower the query cost, the better its performance. The cost is not affected by issues such as resource contention or waiting for locks to be released. In most cases, query cost is a good measure of performance, but if a query uses special elements such as scalar user-defined functions or Common Language Runtime (CLR) programs, their costs are not taken into account, making the query cost lower than the actual cost. For this reason, the query cost is called the estimated query cost.

Page readings

The number of page reads represents the number of 8-KB data pages that the SQL Server memory engine accessed during query execution. You can retrieve this statistic using the SET STATISTICS IO ON command. This command produces information similar to the following in the Message tab of the query window:

Table 'Buyer'. Scan count 0, logical reads 526, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

Table 'Bill'. Scan count 1, logical reads 713, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

The total number of pages read in this example is 526 + 713, the sum of the values labeled logical reads. Logical reads are the number

pages read from memory. Logical reads represent the number of pages of data read from any index on the Bill table. The other metrics report how many logical reads were performed from the hard disk (physical and read-ahead reads), the number of index or heap scans performed in response to a query, and the number of reads required to retrieve Large Object (LOB) data. LOB data is stored out of row with the varchar(max), nvarchar(max), varbinary(max), text, ntext, image, and XML data types. Page reads do not take into account the CPU resources consumed during query execution. Therefore, page reads are usually not as appropriate a measure of performance as query cost.
These metrics suffer from the same problem as query cost with scalar user-defined functions and CLR routines, in that reads caused by these objects are not included in the STATISTICS IO output.

Request execution time

Query execution time is the most variable metric, and is affected by locks and resource contention on the server. Given this, it is especially important to always include query execution time in performance comparisons because this measure can help identify problems that other performance metrics miss. If you run SET STATISTICS TIME ON, SQL Server will return the execution time of all queries in milliseconds.

Connection algorithms

The SQL Server query optimizer usually automatically selects the best query execution plan. Therefore, hints are recommended only for experienced users and database administrators when absolutely necessary.

Without explicitly specifying the argument (LOOP | HASH | MERGE), the optimizer chooses what it thinks is the most optimal plan. But we can always influence it if we explicitly specify the hint [3].

Nested Loop Joins

A LOOP JOIN, also called a nested iteration, uses one table as the outer table (the top table in the graphical plan) and the other as the inner table (the bottom table). LOOP JOIN compares the outer table with the inner table row by row. In the loop, for each outer row, the inner table is scanned and matching rows are output.

In the simplest case, the entire table or index is scanned during the search (naive nested loops join). If the search uses an index, the search is called an index nested loops join. If the index is created as part of the query plan (and destroyed after the query is completed), it is called a temporary index nested loops join. The optimizer itself chooses one of these searches.

LOOP JOIN is particularly efficient when the outer table is relatively small and the inner table is much larger and indexes exist on it. Nested loop join plans are robust because they do not require much memory and disk space. For most queries, nested loop joins will be the most efficient and will provide predictable behavior as the sizes of the tables being joined change.

An example of a query using the nested loop join algorithm is shown below:

```
SELECT  B.BillID,
BU.Nam     AS BuyerName,
e              AS EmployeeName
E.Name
FROM Bill AS B JOIN Employee AS E ON B.EmployeeID = E.EmployeeID LEFT
        LOOP JOIN Buyer AS BU ON BU.BuyerID = B.BuyerID Fusion
        connections
```

MERGE JOIN requires both input data sets to be sorted by the merge columns, which are defined by the equality (ON) clauses of the join predicate. If we have a join predicate "B.EmployeeID = E.EmployeeID", then table B must be sorted by B.EmployeeID, and table E must be sorted by E.EmployeeID.

Because each input set is sorted, the Merge Join operator takes a row from each input set and compares them. For example, for INNER JOIN operations, rows are returned if they are equal. If they are not equal, the row with the lower value is discarded and another row is taken from that input set. This process is repeated until all rows have been processed.

MERGE JOIN can support many-to-many merging. In this case, each time two rows are joined, a copy of each row of the second input stream must be saved. This allows, if duplicate rows are subsequently found in the first input stream, the saved rows to be reproduced. On the other hand, if it is clear that the next row of the first input stream is not a duplicate, the saved rows can be discarded. Such rows are saved in a temporary table

tempdb database. The amount of disk space required for this depends on the number of duplicates in the second input stream.

A one-to-many MERGE JOIN will always be more efficient than a many-to-many merge because it does not require a temporary table. In order to use a one-to-many merge, the optimizer must be able to determine that one of the input streams consists of unique rows. Typically, this means that there is a unique index on the input stream, or there is an explicit operator in the query plan (such as sort on DISTINCT or group by) that ensures that the input rows are unique.

MERGE JOIN is a very fast operation, but it can be resource intensive if sorting operations are required. However, at large volumes, with indexes and pre-sorting, merge join is the fastest join algorithm available [4].

An example of a query using the merge join algorithm:

```
SELECT B.BillID,
BU.Name AS BuyerName, E.Name
AS EmployeeName
FROM Bill AS B JOIN Employee AS E ON B.EmployeeID = E.EmployeeID LEFT MERGE
JOIN Buyer AS BU ON BU.BuyerID = B.BuyerID
```

Hash join

HASH JOIN is more efficient when working with large data sets and even when the tables are not sorted by the columns by which the join is performed. HASH JOIN is parallelized and scales better than any other join and greatly benefits from high- performance data warehouses. Always adequately evaluate the parallelization capabilities, since quite often the costs of supporting the parallelization procedure are commensurate with the time saved by parallel execution of operations.

The join is done using hashing, computing the hash of records from the smaller table (Build table) and inserting them into the hash table, then the larger table (Probe table) is processed one record at a time, scanning the hash table for matches [4].

An example of a query using the merge join algorithm:

```
SELECT B.BillID,
BU.Name AS BuyerName,
E.Name AS EmployeeName
FROM Bill AS B JOIN Employee AS E ON B.EmployeeID = E.EmployeeID LEFT HASH
JOIN Buyer AS BU ON BU.BuyerID = B.BuyerID
```

It should be noted that, despite the high performance of merge joins and hash joins, they are very sensitive to available hardware resources and sizes of joined tables. In this sense, the most predictable and reliable behavior is demonstrated by joins using nested loops.

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Queries created in points 1,2 of the previous lab work
rework it so that the nested loop join, hash join, and merge join algorithms are used to join tables.

2. We estimate the execution time of each of the received requests.
3. Save the received queries in views, adding to the name of the original representation of the name of the algorithm used.
4. Prepare material for inclusion in the course report presentation "Database tuning".

Test questions:

1. We estimate the execution time of each of the received requests.
2. Save the received queries in views, adding to the name of the original one. representation of the name of the algorithm used.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

# LABORATORY WORK #4.Queries with subqueries

Subject -Queries with subqueries

The purpose of the work ismaster queries with

subqueries The competencies being formed or their

parts -OPK-5 Theoretical part. Nested queries

A nested query is a query that is placed within a SELECT statement. INSERT, UPDATE, or DELETE, or into another subquery. A subquery can be used anywhere expressions are allowed. In this example, a subquery is used as an expression for a column named MaxUnitPrice in a SELECT statement.

A nested query is also called an inner query or inner select statement, while a statement containing a nested query is called an outer query or outer select statement.

Many Transact-SQL statements that include subqueries can be written as joins. Other queries can only be implemented using subqueries. In Transact-SQL, there is usually no performance difference between a statement that includes a subquery and a semantically equivalent version without the subquery. However, in some cases where existence is tested, joins provide better performance. Otherwise, to eliminate duplicates, the subquery must be processed to obtain each result from the outer query. In such cases, the join method provides better results [1].

Unrelated nested queries

The main purpose of an unbound subquery is to allow you to write more dynamic code that does not require the user to know all the intermediate values that currently exist in the database. For example, if you want to get a list of employees who have made at least one transaction, you will need to create a subquery that will return the employee IDs displayed in the Bill table [3].

```
SELECT EmployeeID,
        Name,
        Post
    FROM Employee
WHERE EmployeeID IN (SELECT DISTINCT EmployeeID FROM Bill)
```

Subqueries with the EXISTS keyword

When a subquery is entered using the EXISTS keyword, it acts as an existence test. The WHERE clause of the outer query tests whether the rows returned by the subquery exist. The subquery does not return any data, but returns a TRUE or FALSE value.

A subquery created using the EXISTS keyword has the following syntax.

```
WHERE [NOT] EXISTS (subquery)
```

The query below searches for all employees who have not made a single sale.

```
SELECT EmployeeID,
        Name,
        Post
    FROM Employee AS E
WHERE EXISTS (SELECT * FROM Bill AS B WHERE B.EmployeeID =
E.EmployeeID)
```

To understand the results of this query, we need to look at each product name to see if the subquery returns at least one row. In other words, if the existence test returns TRUE.

Note that subqueries entered using the EXISTS keyword differ from other subqueries in the following ways.

The EXISTS keyword is not preceded by the name of a column, constant, or other expression.

The select list for a subquery entered using the EXISTS keyword almost always consists of asterisks (*). Since the check is only for the existence of rows that satisfy the conditions specified in the subquery, there is no need to display the names of the columns found.

The importance of the EXISTS keyword is that it is often impossible to find another solution without nested queries. Although some queries created with the EXISTS keyword cannot be expressed in any other way,

Many queries use IN or a comparison operator modified with ANY or ALL to achieve similar results.

For example, the previous query could be specified using the IN operator.
[3].

Subqueries with NOT EXISTS operator

The NOT EXISTS operator works the same as the EXISTS operator, except that the WHERE clause that uses this operator is executed if the subquery does not return any rows [3].

Table variables

A special data type that can be used to store a result set for later processing. The table type is primarily used to temporarily store the set of rows returned as the result set of a table-valued function.

Functions and variables can be declared as table variables. Table variables can be used in functions, stored procedures, and packages.

Queries that modify table variables do not generate parallel query plans. Modifying very large table variables or table variables in complex queries can degrade performance. In such cases, it may be appropriate to consider using temporary tables. Queries that read table variables without modifying them can be executed in parallel.

Using table variables provides the following benefits.

- Variabletable behaves like a local variable. It has exactly a specific scope of application. This is the function, stored procedure, or package in which it is declared.

- Inside this area the variabletable can be used as a regular table. It can be used anywhere a table or table expression is used in SELECT, INSERT, UPDATE, and DELETE statements. However, the table variable cannot be used in the following statement:

SELECT select_list INTO table_variable

table variables are automatically cleared at the end of the function, stored procedure, or package where they were defined.

- RestrictionsCHECK, DEFAULT values, and computed columns in table type declarations cannot call user-defined functions.

- When using variables in stored procedurestable has to resort to recompilations less frequently than when using temporary tables.

- Transactions using variablestable only continues during the process of updating the corresponding table variable. Therefore, table variables are less likely to be blocked and require fewer resources for maintaining logs.

You cannot create indexes or statistics on table variables. In some cases, you can improve performance by using temporary tables instead of table variables, which allow you to create indexes and maintain statistics.

You can create local and global temporary tables. Local temporary tables are visible only during the current session, while global temporary tables are visible in all sessions.

The name of a local temporary table must be preceded by a number sign prefix (# *table_name*), and the name of the global temporary table is a double number sign (##) *table_name*).

SQL statements can access a temporary table using the argument value specified in the CREATE TABLE statement.*table_name*.

If a local temporary table is created by a stored procedure or application that can be executed by multiple users at the same time, the Database Engine must be able to distinguish between tables created by different users. The Database Engine does this by internally appending a numeric suffix to the name of each local temporary table. The fully qualified name of a temporary table stored in a tablesysobjects databasestempdb, consists of the table name specified by the CREATE TABLE statement and a system-generated numeric suffix. To enable the addition of a suffix, the value of the parameter*table_name*, defined as the name of a local temporary table, must not exceed 116 characters.

Example of using a local temporary table:

```
CREATE TABLE#t(BuyerIDINT PRIMARY KEY)
INSERT INTO#t
select top1 BuyerIDFROMBuyer

SELECTB.BillID,
                B.Date,
                B.U..Name
FROM#tASTINNER
                JOINBuyerASB.U.ONT.BuyerID=B.U..BuyerID
                INNER
                JOINBillASBONB.U..BuyerID=B.BuyerID
```

DROP TABLE#t

The last query creates a temporary table and adds the ID of one customer to it. The main query fetches all purchases of the specified customer.

Temporary tables are automatically dropped when they go out of scope unless you explicitly drop them with the DROP TABLE statement.

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Select all checks for a given date that are not unique and there are other checks for this date. To form a query, use the EXISTS operator.

2. Select all customers who did not make a purchase on a specific date. date. Use NOT EXISTS operators to form a query.

3. Save the received requests.

4. Prepare material for inclusion in the course report presentation "Database tuning".

Test questions:

1. Create a temporary table (CREATE TABLE command) or variable type table (DECLARE TABLE command) with the same structure as the customer table, in

which add information about five random customers (use the INSERT operator). Select information about the 5 added customers and the dates of their purchases. Use a connection of a temporary table (variable of the table type) with the Bill receipts table. To form a query, use the IN operator.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

LABORATORY WORK #5.Query execution plans Subject -

Query execution plans

The purpose of the work ismaster query execution plans

The competencies being formed or their parts -OPK-5

Theoretical part.

SQL Server Management Studio is an interactive graphical tool that enables a database administrator or developer to create queries, execute multiple queries simultaneously, view results, analyze a query plan, and provide assistance to improve query performance. Execution plan options graphically display the data retrieval methods selected by the SQL Server Query Optimizer. The graphical display of the execution plan uses icons representing the execution of specific SQL Server statements and queries, instead of the tabular display provided by the SHOWPLAN_ALL or SHOWPLAN_TEXT options of the Transact-SQL SET statement or the XML representation provided by the SET SHOWPLAN_XML statement. The graphical display is very useful for understanding query performance metrics. SQL Server Management Studio shows which statistics are missing, thereby forcing query optimization through sampling quality estimates, and then allows you to easily create the missing statistics [3].

Using execution plan parameters

To view the execution plan for a query, open or enter a Transact-SQL script containing the queries you want to analyze in Management Studio's Query Editor. After you load the script in Management Studio's Query Editor, choose whether to display the estimated or actual execution plan by clicking the Display Estimated Execution Plan button or the Include Actual Execution Plan button on the Query Editor toolbar. The Query Editor toolbar is shown in Figure 5.1. Clicking the Display Estimated Execution Plan button analyzes the script and generates an estimated execution plan. Clicking the Include Actual Execution Plan button requires that you execute the script before the execution plan is generated. After the script has been analyzed and executed, click the Execution Plan tab to view a graphical representation of the plan generation results.


Figure 5.1 - Query Editor Toolbar.

To use the execution plan graphical features in Management Studio and the SHOWPLAN options of the Transact-SQL SET statement, users must have sufficient permissions to execute Transact-SQL statements and queries. Users must also have SHOWPLAN permission on all databases that contain objects referenced in the queries. For more information, see Showplan Security [3].

Reading the results of the execution plan

To view the execution plan, click the Execution Plan tab in the results pane. SQL Server Management Studio's graphical representation of the execution plan results is read from right to left and top to bottom. Each analyzed query in the batch is displayed along with its execution cost as a percentage of the total batch execution cost. Figure
5.2 shows the query execution plan:

```
SELECT B.BillID
      , BI.BillItemID
      , P.ProductID
      , E.EmployeeID          AS ProductName
      , P.Name
      , BI.Number
      , P.Price               AS DateBill
      , BI.Cost               AS DateBillItem
      , B.[Date]              AS EmployeeName
      , BI.[Date]             AS EmployeePost AS
      , E.Name                ProductPhoto
      , E.Post
      , P.Photo               AS E ON B.EmployeeID = E.EmployeeID AS BI
  FROM Bill AS B              ON B.BillID = BI.BillID
  INNER JOIN Employee         AS P ON BI.ProductID = P.ProductID
  INNER JOIN BillItem
  INNER JOIN Product
```

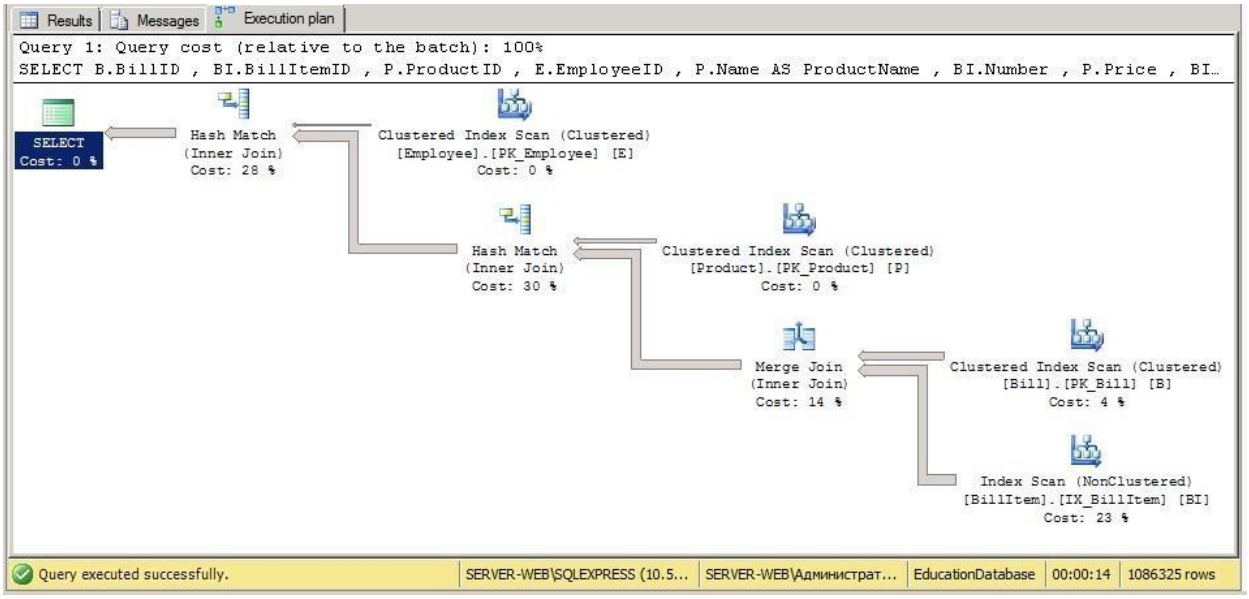



Figure 5.2 - Actual query execution plan.

The following descriptions provide guidance on interpreting the results of the execution plan graphical display in Management Studio:

1. Each node of the tree structure is represented as an icon,

indicating the logical and physical operator used to execute that part of the query or statement.

2. Each node is linked to its parent node. Child nodes of one

parent node are displayed in a single column. However, all nodes in a column do not necessarily share a common parent node. Rules with arrows at the end connect each node to its parent.

3. Operators are shown as symbols associated with a specific parent

node.

4. The width of the arrow is proportional to the number of lines. If there is data on actual number of rows, this data is used. Otherwise, the estimated number of rows is used.

5. If the request contains multiple instructions, multiple plans are shown

execution of the request.

6. Parts of tree structures are defined by the type of instruction being executed.

7. For parallel queries that involve multiple processors,

The Properties menu item for each node in the graphical execution plan displays information about the operating system threads used. To view the properties of a node, right-click the node and select Properties [2].

Graphical execution plan node tooltips

Each node displays information in a tooltip that appears when you hover over the node, as described in the following table. Not all nodes in the graphical execution plan have tooltips that are described here.

Table 5.1 - Main parameters of the graphical query execution plan node

| Element | Description |
|---|---|
| 1 | 2 |
| Physical operation (Physical Operation) | The operator used, such as Hash Join or Nested Loops. Physical operators displayed in red indicate that the query optimizer has issued a warning, such as a warning about insufficient column statistics or missing join predicates. This can cause the optimizer to select a query plan that is less efficient than expected. For more information about column statistics, seeUsing Statistics to Improve Query Performance.<br><br>If a graphical execution plan suggests creating or updating statistics or creating an index, the missing statistics or indexes can be immediately created or modified using the context menus in SQL Server Management Studio Object Explorer. For more information, see Index Instructions. |
| Logical operation (Logical Operation) | A logical operator that corresponds to a physical operator, such as the Inner Join operator. The name of the logical operator appears after the physical operator at the top of the tooltip. |
| Estimated size rows (Estimated Row Size) | Estimated size of the string obtained at the operator's output (in bytes). |
| Assumed cost of operations input/output (Estimated I/O Cost) | The estimated cost of performing I/O actions for this operation. This value should be as low as possible. |
| Assumed cost of process resource (Estimated CPU Cost) | Approximate CPU cost of all calculations for a given operation. |
| Assumed operator cost (Estimated Operator Cost) | The query optimizer cost of performing this operation. The cost of performing this operation as a percentage of the total cost of executing the query is shown in parentheses. Since the query engine chooses the most efficient operation to execute a query or statement, this value should be as low as possible. |
| Assumed subtree cost | The total cost to the query optimizer of performing this and all preceding operations in this subtree. |
| (Estimated Subtree Cost) | |

| | |
|---|---|
| Assumed number of lines (Estimated Number of Rows) | The number of lines output by the operator. |

Graphical execution plan icons

The icons displayed in the graphical execution plan in SQL Server Management Studio represent the operators used by SQL Server to execute statements. The list of the main operators is presented in Table 5.2 [3].

Table 5.2 - Basic operators of the graphical execution plan requests

| Icon | Operator | Description |
|---|---|---|
|  | Clustered Index Delete | The Clustered Index Delete operator deletes rows from the clustered index specified in the Argument column of the query execution plan.<br>The Clustered Index Delete operator is a physical operator. |
|  | Clustered Index Insert | The Clustered Index Insert operator of the Showplan statement inserts rows from its input into the clustered index specified in the Argument column.<br>The Clustered Index Insert operator is a physical operator. |
|  | Clustered Index Scan | The Clustered Index Scan operator scans the clustered index specified in the Argument column of the query execution plan.<br>Clustered Index Scan is a logical and physical operator. |
|  | Clustered Index Seek | The Clustered Index Seek operator uses index seeking capabilities to retrieve rows from a clustered index. The Argument column contains the name of the clustered index to use and the SEEK:() predicate.<br><br>Clustered Index Seek is a logical and physical operator. |
|  | Clustered Index Update | The Clustered Index Update operator updates the input rows of a clustered index specified in the Argument column. Clustered Index Update is a logical and physical operator. |
|  | Hash Match | The Hash Match operator builds a hash table by computing a hash value for each row of its input.<br><br>Hash Match is a physical operator. |
|  | Merge Join | The Merge Join operator performs inner join, left outer join, left semi join, left anti-semi join, right outer join, right semi join, right anti-semi join, and logical join operations.<br><br>The Merge Join operator is physical. |

| | | |
|---|---|---|
|  | Nested Loops | The Nested Loops operator performs the logical operations of inner join, left outer join, left semi-join, and anti-left semi-join. Nested Loops is a physical operator. |
|  | Non-clustered Index Delete | The NonclusteredIndex Delete operator deletes input rows from the nonclustered index specified in the Argument column. The Nonclustered Index Delete operator is a physical operator. |
|  | Non-clustered Index Insert | The Index Insert statement inserts rows from the input stream into the nonclustered index specified in the Argument column. In addition, the Argument column contains a SET:() predicate that specifies the value specified for each column. Index Insert is a physical operator. |
|  | Non-clustered Index Scan | Operator Index Scan receives All records nonclustered index specified in the Argument column. If the optional WHERE:() predicate appears in the Argument column, only those rows that satisfy the condition specified in that predicate are returned.<br><br>The Index Scan operator is a logical and physical operator. |
|  | Non-clustered Index Seek | The Index Seek operator uses index seeking capabilities to retrieve rows from a nonclustered index.<br><br>Index Seek is a logical and physical operator. |
|  | Non-clustered Index Update | The Nonclustered Index Update physical operator updates the rows specified in the input parameters in the nonclustered index specified in the Argument column. If the SET:() predicate is present, each updated column is assigned the specified value. Nonclustered Index Update is a physical operator. |
|  | Sort | The Sort operator sorts the input rows. The Argument column contains either a DISTINCT ORDER BY:() predicate if the operation removes duplicates, or an ORDER BY:() predicate followed by a comma-separated list of columns to sort. The columns are prefixed with the ASC value if they are sorted in ascending order, or with the DESC value if they are sorted in descending order. Sort is a logical and physical operator. |

Textual representation of the query execution plan

Setting the value of the SET SHOWPLAN_TEXT parameter is done at run or startup time, not at parse time.

When SET SHOWPLAN_TEXT ON is executed, SQL Server returns information for each Transact-SQL statement without executing it. When this option is set to ON, information is returned for the execution plans of all subsequent SQL Server statements until the option is set again.

OFF. For example, if a CREATE TABLE statement is executed while SET SHOWPLAN_TEXT is ON, SQL Server returns an error message on a subsequent SELECT statement that refers to the same table, informing the user that the specified table does not exist.
Thus, subsequent accesses to that table will generate errors. If SET SHOWPLAN_TEXT is OFF, SQL Server executes the statements without generating a report with execution plan information.

The SET SHOWPLAN_TEXT statement outputs data in a human-readable format intended for Microsoft Win32 platform command-line applications. The SET SHOWPLAN_ALL statement returns more detailed data intended for processing by programs that specifically target this output format.

The SET SHOWPLAN_TEXT and SET SHOWPLAN_ALL statements cannot be specified in a stored procedure. They can only be specified as statements in a batch.

The SET SHOWPLAN_TEXT statement returns data as a set of rows that forms a hierarchical tree that represents the sequence of steps performed by the SQL Server query processor as it executes each statement. Each statement reflected in the output contains one line of statement text, followed by several lines of execution details.
Figure 5.3 shows the text execution plan for the query discussed earlier [2].

| | StmtText |
|---|---|
| 1 | \|--Hash Match(Inner Join, HASH:([E].[EmployeeID])=([B].[EmployeeID])) |
| 2 | \|--Clustered Index Scan(OBJECT:([EducationDatabase].[dbo].[Employee].[PK_Employee] AS [E])) |
| 3 | \|--Hash Match(Inner Join, HASH:([P].[ProductID])=([BI].[ProductID])) |
| 4 | \|--Clustered Index Scan(OBJECT:([EducationDatabase].[dbo].[Product].[PK_Product] AS [P])) |
| 5 | \|--Merge Join(Inner Join, MERGE:([B].[BillID])=([BI].[BillID]), RESIDUAL:([EducationDatabase].[dbo].[Bill].[BillID] as [B].[BillID]=[EducationDatabase].[dbo].[BillItem].[BillID] as [BI].[BillID])) |
| 6 | \|--Clustered Index Scan(OBJECT:([EducationDatabase].[dbo].[Bill].[PK_Bill] AS [B]), ORDERED FORWARD) |
| 7 | \|--Index Scan(OBJECT:([EducationDatabase].[dbo].[BillItem].[IX_BillItem] AS [BI]), ORDERED FORWARD) |

Figure 5.3 – Textual query execution plan

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions for performing laboratory work coincide with those generally accepted for personal computer users. Do not independently

repair the personal computer, install and remove software; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety regulations when working with electrical equipment; do not touch electrical outlets with metal objects; the user's work area of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. For queries created in points 1 and 2 of Lab 2 and queries, created in point 1, lab 3, generate a graphical and textual plan for executing queries. Study the execution plans and explain the order of joining tables.

2. Prepare material for inclusion in the course report presentation "Database tuning".

Test questions:

1. Estimate the complexity of individual nodes in the query plan tree. Select the most labor-intensive operations, study them and explain why these operations are the most labor-intensive compared to the others.

2. Estimate the volume of data transferred between plan operators execution of the request.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

# LABORATORY WORK #6.Transactions

Subject -Transactions

The purpose of the work ismaster transactions

The competencies being formed or their parts -

OPK-5 Theoretical part

Since data processing is a critical function in online transaction processing (OLTP) databases, transactions play an important role in the process of managing and maintaining consistent data.

Transactions are often defined as a set of actions or operations that succeed or fail as a unit. Moreover, transactions can provide four main characteristics of data processing processes that access a database.

- Atomicity – if a transaction involves two or more chunks
data, either all are recorded, or none of them.

- Consistency – at the end of a transaction either there is a new adequate representation of the data, or the data is returned to its original state. Returning data to its original state is part of the rollback mechanism provided by SQL Server transactions.

- Isolation – during the execution of a transaction (before it is
committed or will be rolled back) data must be isolated and inaccessible to other transactions. The isolation level can be controlled in each transaction.

- Duration or durability – after the transaction is committed, the final
The state of the data is preserved even if the server crashes or is rebooted. This transactional property is ensured by checkpoints and the database recovery process that occurs when SQL Server starts.

The ACID (Atomicity, Consistency, Isolation, Durability) notation is used to denote these four properties.

By default, in SQL Server, each INSERT, UPDATE, and DELETE statement is a separate transaction that is automatically committed and does not provide a rollback option.

You can enable implicit transactions in your connection parameters, and the Database Engine will automatically start a transaction when you execute the following commands: ALTER TABLE, CREATE, DELETE, DENY, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE TABLE, or UPDATE.

A transaction is active until you manually issue a COMIT or ROLLBACK statement. You can enable implicit transactions by using the SET IMPLICIT_TRANSACTIONS ON statement through the Object Linking and Embedding Database (OLE DB) or Open Database Connectivity (ODBC) APIs on the ANSI page of the Query Options dialog box in SSMS, or by modifying the server properties to change the default behavior to enable implicit transactions on all connections unless they are explicitly set to OFF on a specific connection [1].

Defining Explicit Transactions

Explicit transactions are typically defined in stored procedures. An explicit transaction begins when the BEGIN TRANSACTION statement is executed. The transaction ends when the COMMIT TRANSACTION statement or the ROLLBACK TRANSACTION statement is entered. Once a transaction is committed, SQL Server ensures that data is written to the database, even in emergency situations. The ROLLBACK statement returns the data to the state before the transaction began. Although the ROLLBACK statement returns the data to its previous state, some components, such as initial values for identifying columns, are not restored [1].

Special ROLLBACK scenarios

When transactions are nested by having multiple BEGIN TRANSACTION statements in a session, the ROLLBACK statement rolls back the outermost nested transaction. This is true even if the outer transaction's ROLLBACK statement is preceded by COMMIT statements for transactions at lower nesting levels. In the following example, the data is rolled back to the transaction started at row 1, and the inserted row is completely deleted from the table.

```
1. BEGIN TRANSACTION
2. INSERT INTO Employee
3. VALUES (“New name”, “Position”);
4.         BEGIN TRANSACTION
5.         UPDATE Buyer
6.         SET Name = “Egorov Anatoly Vasilievich”
7.         WHERE BuyerID = 26;
8.         COMMIT TRANSACTION;
9. ROLLBACK;
```

If you want to roll back only part of a transaction, you can set savepoints using the SAVE TRANSACTION savepoint_name statement and then

reference the savepoint name in the ROLLBACK statement. By doing this, you tell the Database Engine to roll back changes only up to the point where you issued a SAVE TRANSACTION statement with the same name.

If you specify multiple savepoints with the same name, the ROLLBACK statement rolls back to the most recent ROLLBACK statement. If you want to roll back an entire transaction, enter a ROLLBACK TRANSACTION statement with or without a transaction name. Keep in mind that using ROLLBACK TRANSACTION statements without a name rolls back all nested transactions [1].

Setting insulation levels

The following transaction isolation levels can be set using the SET TRANSACTION ISOLATION LEVEL statement.

READ UNCOMMITTED – Allows statements to read rows updated by any transaction and not yet committed to the database. This isolation level minimizes the likelihood of conflicts, but allows dirty reads and non-repeatable reads.

READ COMMITTED – Allows statements in current connections and transactions to perform unrepeatable reads, but disallows dirty reads (data updated by a transaction opened in another connection). This is the default isolation level in SQL Server.

REPEATABLE READ – prevents transactions from reading uncommitted modified data (dirty reads) and ensures that shared locks are maintained until the current transaction completes.

SNAPSHOT – Requires ALLOW_SNAPSHOT_ISOLApTaIrOamNetteor be ONb.aTsehsis isoladtaiotan level creates a snapshot of the data as it is read into the transaction, but does not support data locking. Updates from another transaction may be made, but the current transaction does not see those changes reflected in subsequent reads of the source data. If the current transaction modifies the data, those modifications are visible only to the current transaction.

SERIALIZABLE – does not allow reading data that has been updated but not committed by other transactions. In addition, no other transaction can update data that has been read by the current transaction until the current transaction completes. The SERIALIZABLE isolation level protects against phantom reads, but creates the highest level of contention and blocking [1].

The syntax for determining the transaction isolation level is:

```
SET TRANSACTION ISOLATION
                LEVEL
    { READ UNCOMMITTED
     | READ COMMITTED
     | REPEATABLE READ
     | SNAPSHOT
     | SERIALIZABLE
     }
```

Once the SET TRANSACTION ISOLATION LEVEL statement is executed in a session, all transactions within that connection use the specified isolation level [1].

Functions IDENT_CURRENT, SCOPE_IDENTITY, @@IDENTITY

To find out the last added identifier (IDENTITY), the functions IDENT_CURRENT, SCOPE_IDENTITY and @@IDENTITY are used.

The IDENT_CURRENT function is almost identical to the SQL Server identity functions SCOPE_IDENTITY and @@IDENTITY. All three functions return the most recently created identity values. However, these functions differ in scope and session.

IDENT_CURRENT returns the last identity column value created for a specific table in any session and search scope.

@@IDENTITY returns the last identity value created for any table in the current session across all search scopes.

SCOPE_IDENTITY returns the last identity value created for any table in the current session within the current search scope.

The IDENT_CURRENT function returns NULL if called on an empty table or on a table without an identity column.

The statements and transactions that failed may change the current identity for the table and cause gaps in the identity column values. The identity is never rolled back, even though the transaction that attempted to insert the value into the table was not committed. For example, if an INSERT statement failed because of an IGNORE_DUP_KEY constraint violation, the current identity for the table is still incremented.

Be careful when using the IDENT_CURRENT function to predict the next identity value to be generated. The actual value generated may differ from that obtained using IDENT_CURRENT plus IDENTITY_SEED because other sessions may be performing insert operations [3].

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Add a field quantity to the Product table. Fill in the field arbitrary values from 0 to 100.

2. Create a stored procedure AddBill that will generate a check, and then return its identifier. The input parameters of the procedure are BuyerID and EmployeeID. All statements of this stored procedure must be executed in a transaction.

3. Make a selection of the check contents using query #1 from laboratory work #2.

4. Check the functionality of the written stored procedures. Try create a receipt and add 2 items with sufficient quantity in the store and 1 item with insufficient quantity. Make sure that the receipt contains only 2 sold items.

5. Prepare material for inclusion in the course presentation "Database tuning".

Test questions:

1. Create a stored procedure for adding a product to a receipt. Name it AddBillItem. Procedure input parameters: BillID, ProductID, Quantity. Set

transaction isolation level to SERIALIZABLE mode. After opening the transaction (BEGIN TRANSACTION), it is necessary to check the quantity of goods in the store. If there is not enough goods in the store, the procedure must complete the transaction (ROLLBACK TRANSACTION). Otherwise, the stored procedure reduces the available quantity of goods in the store and adds a purchase record to the receipt (records are added to the BillItem table). Close the transaction (COMMIT TRANSACTION). Explain the choice of transaction isolation level.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

# LABORATORY WORK #7.Working withXML. Part 1

Subject -Working withXML

The purpose of the work ismaster working withXML

The competencies being formed or their parts -OPK-

5 Theoretical part. Structure of XML document

An XML document consists of declarations, elements, comments, special characters, and directives.

XML is a tagged document markup language. In other words, any XML document is a set of elements, with the beginning and end of each element designated by special marks called tags.

An element consists of three parts: a start tag, content, and an end tag. A tag is text enclosed in angle brackets "<" and ">". An end tag has the same name as a start tag, but begins with a slash "/". An example of an XML element:

<buyer>Sergey Dovlatov</buyer>

Element names are case-sensitive, i.e. <author>, <Author>, and <AUTHOR> are names of different elements. A closing tag is always required. If a tag is empty, i.e. has no content and no closing tag, it has a special form:

<tag/>

Any element can have attributes containing additional information.

about the element. Attributes are always included in the element's start tag and look like this:

attribute_name="attribute_value"

An attribute must have a value, which must always be enclosed in single or double quotes. Attribute names are also case-sensitive. An example of an element that has an attribute:

<employee post="Seller">Sergey Dovlatov</employee > Elements

must either follow each other or be nested:

```
<books>
   <book isbn="5887821192">
```

```
        <title>Part of speech</title>
        <author>Brodsky, Joseph</author>
        <present/>
    </book>
    <book isbn="0345374827">
        <title>March of the Lonely</title>
        <author>Dovlatov, Sergey</author>
        <present/>
    </book>
</books>
```

Here the books element contains two nested book elements, which in turn have an isbn attribute and contain three consecutive elements: title, author, and present, the last of which is empty because it corresponds to a logical flag in this case.

From the above description, it is clear that the XML syntax resembles the HTML syntax (which is natural, since both are dialects of the same language SGML), but the requirements for the formatting of valid XML documents are higher. Another very important difference between XML and HTML is that the content of elements, i.e. everything contained between the start and end tags, is considered data. This means that XML does not ignore space characters and line breaks, as HTML does.

Any XML document consists of a prologue and a root element, for example:

```
<?xml version="1.0"?>
<books>
    <book isbn="0345374827">
        <title>March of the Lonely</title>
        <author>Dovlatov, Sergey</author>
        <present/>
    </book>
</books>
```

In this example, the prologue is reduced to a single directive (the first line of the document) indicating the XML version. It is followed by an XML element with a unique name, which contains all other elements and is called the root element. A directive (processing instruction) is an expression enclosed in special tags "<?" and "?>", which contains instructions to the program processing the XML document.

The XML standard reserves only one directive <?xml version="1.0"?>, indicating the version of the XML language to which the document corresponds (there is no second version of XML yet). In reality, this directive is somewhat richer and in its most general form looks like this:

<?xml version="1.0" encoding="ISO–8859–1" standalone="yes"?>

Here, the encoding attribute specifies the character encoding of the document. By default, XML documents are considered to be created in UTF-8 or UTF-16 format. The standalone attribute indicates whether the document contains external sections. The yes value means that there are no such sections, and the no value means that they do.

In general, the prolog may also contain document type declarations. XML

documents may contain comments that are ignored by the application processing the document. Comments are constructed according to the same rules as in HTML:

- start your comment with the characters "<!--",
- end your comment with "-->"
- Do not use "--" symbols inside comments.
- Example comments:

<!— this is a comment —> <!— and
here is another comment,
occupying more than one line —>

All element, attribute, and section names must begin with a Unicode letter and consist of letters, digits, dot (.), underscore (_), and hyphen (–) characters. The only restriction is that they must not begin with the letter combination xml in any case; such names are reserved for future extensions of the language. It is significant that the standard allows the use of not only English letters in names, but also any other, although existing XML processors are often limited by the encoding systems that their creators have built into them.

XML documents can be well-formed and valid.[5]

Well-formed XML documents

To be well-formed, a document must follow the syntax rules established for XML by the W3C in the XML 1.0 recommendation. Informally, "well-formed" means primarily that the document must contain one or more elements, and one of them, the root element, must contain all the other elements. In addition, each element must follow the nesting rules. For example, the following document would not be well-formed:

formed because the closing </Bill> tag occurs after the opening <Buyer> tag for the following element:

```
<?xml version=”1.0” encoding=”UTF-8”?> <
Document >
        <Bill>
                Check
                <Buyer>
                </Bill>
                Buyer
</Buyer>
</Document> [6]
```

Valid XML documents

Most browsers check XML documents to see if they are well-formed. Some browsers can also check whether a document is valid. An XML document is valid if it has a Document Type Declaration (DTD) or XML schema associated with it, and if the document conforms to that DTD or schema. That is, the DTD or schema specifies a set of rules for the internal integrity of the document itself, and if the browser can verify that the document conforms to those rules, it is valid [6].

XML document parsing function

The sp_xml_preparedocument procedure reads an input XML document, parses it using the MSXML parser (Msxmlsql.dll), and produces a parsed document ready for consumption. The parsed document is a tree representation of the various nodes in the XML document: elements, attributes, text, comments, and so on.

sp_xml_preparedocument returns a handle that can be used to access the newly created internal representation of the XML document. The handle is valid only for the duration of the session or until it is discarded by executing sp_xml_removedocument.

The parsed document is stored in an internal SQL Server cache. The MSXML parser uses one-eighth of the total memory available to SQL Server. To avoid out-of- memory issues, free the memory using sp_xml_removedocument.

For backward compatibility, sp_xml_preparedocument removes carriage return (char(13)) and line feed (char(10)) characters from attributes, even if these characters are significant.

Means syntactic analysis XML, called procedure sp_xml_preparedocument, allows you to parse internal DTDs and entity declarations. Because maliciously crafted DTDs and entity declarations can be used to perform denial of service (DoS) attacks, it is strongly recommended that you do not directly pass XML documents from questionable sources to sp_xml_preparedocument.

To prevent recursive entity expansion attacks, sp_xml_preparedocument limits the number of entities that can be placed in the root entity of a document to 10,000. This limitation does not apply to character or numeric entities. The limitation allows you to store documents with many entity references, but prevents a single entity from recursively expanding into a chain longer than 10,000 extensions.

The sp_xml_preparedocument procedure limits the number simultaneously number of elements that can be opened to 256.

The sp_xml_removedocument procedure removes the embedded representation of the XML document specified by the document handle and invalidates the document handle [3].

Converting an XML document to a table

To transform an XML document into a flat table, you must use the OPENXML statement.

OPENXML provides a rowset view of an XML document. Because OPENXML is a rowset provider, OPENXML can be used in Transact-SQL statements that can use rowset providers such as tables, views, or the OPENROWSET function.

The syntax is shown below:

OPENXML( idoc int [ in ] , rowpattern nvarchar [ in ] , [ flags byte [ in ] ] ) [ WITH ( SchemaDeclaration | TableName ) ] [3]

Parsing XML documents The input document has the following form:

```
<root>
    <buyer id="34" name="Sidorov Ivan Petrovich">
        <bill id= "17843" date="08/17/2011 cost="175.37"/>
                                                                14:47:19"
        <bill id= "17881" date="17.08.2011 15:12:58"
    cost="2908.4"/>
```

```
    </buyer>
</root>
```

In order to transform this document into a flat table containing the fields: BuyerID, Name, BillID, Date, Cost, we will execute the following query:

```
DECLARE @idoc int
DECLARE @doc XML = '<root><buyer id="34" name="Sidorov Ivan Petrovich"><bill id="17843" date="08/17/2011 14:47:19" cost="175.37"/><bill id="17881" date="08/17/2011 15:12:58" cost="2908.4"/></buyer></root>'

EXEC sp_xml_preparedocument @idoc OUTPUT, @doc SELECT
*
FROM            OPENXML (@idoc, '/root/buyer/bill', 2)
                  WITH (BuyerID int '../@id', Name nvarchar(100)
              '../@name', BillID int '@id',

              [Date] Date '@date',
              Cost Money '@cost')
EXEC sp_xml_removedocument @idoc;
```

The result of executing this query is shown in Figure 7.1.

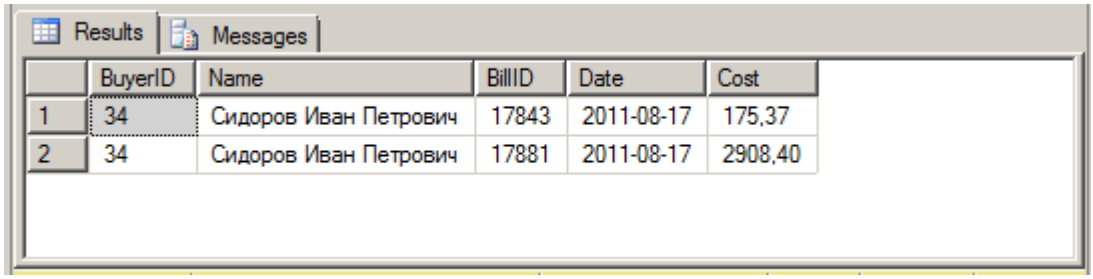



Figure 7.1 – Result of parsing an XML document

Using INSERT…SELECT Statements

Using the INSERT…SELECT statement allows you to add rows to an existing table based on data retrieved from another table. The following statement selects rows from the Employee table and adds them to the Buyer table. Because the number and data type of columns in the result set must match the target table, concatenation is used to obtain the correct number of columns.

```
INSERT INTO Buyer
SELECT Name FROM Employee
```

Equipment and materials

To perform the lab work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit

(x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Create a new stored procedure AddBillXML. The input parameter is

This stored procedure will produce an XML document, an example of which is shown below.

```
<root>
        <bill billId="1473">
                <product productId="1" quantity="5"/> <product
                productId="23" quantity="1"/> <product
                productId="7" quantity="34"/> </bill>

</root>
```

2. Using sp_xml_preparedocument, sp_xml_removedocument and OPENXML parse the XML document, selecting the fields productId and quantity for the receipt record. Create a flat temporary table with the contents of the XML document.

3. Prepare material for inclusion in the course report presentation "Database tuning".

Test questions:

1. After parsing the XML document and forming a flat table, open transaction to generate a new receipt. Create a new receipt (use the AddBill stored procedure from worksheet #6). Add to the created receipt using the INSERT…SELECT statement only those products that are in sufficient quantity. Using the UPDATE statement, update the Quantity field of the Product table, subtracting the products that were purchased. Choose the transaction isolation level yourself and comment on your choice.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

LABORATORY WORK #8.Working withXML. Part 2

Subject -Working withXML

The purpose of the work ismaster working withXML The

competencies being formed or their parts -OPK-5

Theoretical part. FOR XML clause

A SELECT query returns results as a set of rows. If necessary, you can get the results of a SQL query in XML format. To do this, you must specify the FOR XML clause in the query. The FOR XML clause can be used in top-level queries and in subqueries. The top-level FOR XML clause can be used only in the SELECT statement. In subqueries, the FOR XML clause can be used in INSERT, UPDATE, and DELETE statements. It can also be used in assignment statements.

The FOR XML clause can specify one of the following modes:

- RAW
- AUTO
- EXPLICIT
- PATH

In RAW mode, a single <row> element is created for each row in the rowset returned by the SELECT statement.

```
- - Simple XML selection from one table SELECT EmployeeID, [Name]
FROM dbo.Employee FOR XML RAW
```

You can rename the <row> element, add a root element to the selection, and redefine the structure.

```
- - Selecting data from two related tables with the root specified -
- - element and renaming <row> SELECT e1.
[Name], b1.Date, b1.BillID FROM dbo.Employee
e1
INNER JOIN dbo.Bill b1 ON b1.EmployeeID=e1.EmployeeID FOR XML
RAW('Empl'), ROOT('Employers')

- - Column values are formatted as nested nodes when fetching from a single table

SELECT ProdID, EmployeeID, Qty FROM Orders FOR XML
RAW('Ord'), ROOT('Orders'), ELEMENTS

- - Child table column values are output as nested elements

SELECT e1.[Name], e1.Post, b1.BillID, b1.Date FROM
dbo.Employee e1
```

```
INNER JOIN dbo.Bill b1(NOLOCK) ON b1.EmployeeID=e1.EmployeeID WHERE
b1.Date<'01/01/2011'
FOR XML RAW('Empl'), ROOT('Employers'), ELEMENTS
```

The FOR XML AUTO mode allows you to create the simplest XML fragment with nested elements. XML nesting is created heuristically, depending on the method of defining the SELECT instruction. Control over the form of the created XML structure is minimal. Element names and nesting levels are generated based on naming and relationships between objects in the database.

```
- - When selecting from one table using FOR XML AUTO
- - Each node, unlike the FOR XML RAW mode, has a default
- - table name <Bill>
SELECT * FROM Bill FOR XML AUTO

- - Automatic generation of nesting levels when selecting from
- - two related tables
SELECT Employee.[Name], Employee.Post, Bill.BillID, Bill.Date FROM Employee

INNER JOIN Bill ON Bill.EmployeeID=Employee.EmployeeID
FOR XML AUTO--If you enable the ELEMENTS option, the column values
- - are formed as nested subelements
```

To form simple nested selections, the TYPE option is used. This the option allows you to group data within one element and form them in accordance with the rules of the FOR XML AUTO mode

```
- - This query returns ProdID and Qty for each OrderID as nested elements of the
<Orders> node.
SELECT Bill.BillID, Bill.Date,
        (SELECT BillItem.Number, BillItem.Cost FROM BillItem WHERE
          BillItem.BillID=Bill.BillID FOR XML AUTO, TYPE) FROM Bill

WHERE Bill.Date<'01/02/2010' AND Bill.EmployeeID=1 FOR XML
AUTO, TYPE
```

EXPLICIT mode provides more control over the shape of the XML structure. The XML structure can have mixed attributes and elements. This requires a special format for the result rowset that is created as a result of the query. The format of this rowset is then matched against the shape of the XML structure. The advantages of EXPLICIT mode include the ability to have mixed attributes and elements, the ability to create wrappers and nested compound properties, the ability to create space-separated values (for example, the BillID attribute can contain a list of order ID values), and mixed content.

In order for the query processing logic in EXPLICIT mode to produce the desired XML, the following two metadata columns are required:

- the first column should provide the tag number of the current element (integer type), with the column nameTagThe request must specify a unique tag number for each element that will be created from the rowset;
- the second column should specify the tag number for the parent element, and this the column must have a nameParent.

Thus, the columnsTagAndParentprovide information about the hierarchy. Let's look at an example of creating an XML structure using EXPLICIT mode.

```
SELECT DISTINCT
                1                       AS Tag,
                NULL                    AS Parent,
                e1.EmployeeID           AS  [Employee!1!EmpID],
                e1.[Name]               AS    [Employee!1!Name],
                NULL                    AS [Bill!2!Date],
                NULL                    AS   [Product!3!Name],
                NULL                    AS [Product!3!Cost]
                NULL
FROM dbo.Employee e1
INNER JOIN dbo.Bill b1(NOLOCK) ON b1.EmployeeID=e1.EmployeeID
        AND b1.Date<'02.01.2010'
INNER JOIN dbo.BillItem bi1 ON bi1.BillID=b1.BillID
INNER JOIN dbo.Product p1(NOLOCK) ON p1.ProductID=bi1.ProductID UNION ALL

SELECT DISTINCT
                2                       AS Tag,
                1                       AS Parent,
                e1.EmployeeID,
                e1.[Name],
                b1.Date,
                NULL,
                NULL
FROM dbo.Employee e1
INNER JOIN dbo.Bill b1(NOLOCK) ON b1.EmployeeID=e1.EmployeeID
        AND b1.Date<'02.01.2010'
INNER JOIN dbo.BillItem bi1 ON bi1.BillID=b1.BillID
INNER JOIN dbo.Product p1(NOLOCK) ON p1.ProductID=bi1.ProductID UNION ALL

SELEC           3                       AS Tag,
T               2                       AS Parent,
                e1.EmployeeID,
                e1.[Name],
                b1.Date,
                p1.[Name],
                bi1.Cost
FROM dbo.Employee e1
INNER JOIN dbo.Bill b1(NOLOCK) ON b1.EmployeeID=e1.EmployeeID
        AND b1.Date<'02.01.2010'
```

```
INNER JOIN dbo.BillItem bi1 ON bi1.BillID=b1.BillID
INNER JOIN dbo.Product p1(NOLOCK) ON p1.ProductID=bi1.ProductID ORDER BY
[Employee!1!EmpID], [Bill!2!Date], [Product!3!Cost] FOR XML EXPLICIT
```

This query creates an XML structure with three levels of hierarchy. For each level of hierarchy, a separate query is written, and all of them are combined together using UNION ALL.

The first query retrieves values for the <Employee> element and its attributes. The query assigns the fieldTagmeaning1 for the <Employee> element, and the fieldParent NULL value because it is a top-level element.

The first query also specifies names for the columns in the result set of rows. These names form three column groups. The group with the valueTag, equal to 1, in the column name specifies <Employee> as the element, and EmpID and Name as attributes. The second group of columns with Tag value equal to 2 in the column name specifies <Bill> as the element and Date as the attribute. And the third group respectively specifies the
<Product> element with Name and Cost attributes.

The second query retrieves the values for the <Bill> element. ColumnTagis assigned value 2 for the <Bill> element, and the columnParent- meaning1, which results in
<Employee> being used as the parent.

The third query retrieves the values for the <Product> element. ColumnTag the value 3 is assigned to the <Product> element, and the columnParent- meaning3, which results in <Bill> being used as the parent.

As a result, the values in the meta columnsTagAndParent, the information specified in the column names, and the correct row order produce the desired XML when using EXPLICIT mode.

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair your personal computer, install or remove software on your own.

provision; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety regulations when working with electrical equipment; do not touch electrical outlets with metal objects; the user's work area of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Prepare material for inclusion in the course report presentation "Database tuning".

Test questions:

1. Write a request that returns the check number, total amount on the check, and full name employee, check date; information about each product, name, price, quantity, cost. Return the result as XML, presented in the previous lab.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

LABORATORY WORK #9.Hierarchically organized data Subject - Hierarchically organized data

The purpose of the work ismaster hierarchical organized data The competencies being formed or their parts -OPK-5

Theoretical part. Approaches to the organization of hierarchical structures

Any sufficiently complex natural or artificial is characterized by a hierarchsiycsatlem structure - a multi-level form of organization of objects with the correspondence of lower-level objects to one or more upper-level objects. Examples of such structures can be found in completely different subject areas: classification of goods, contractors, product assembly, job hierarchy, administrative-territorial division, genealogical tree, and finally, simply a tree of enumeration of options or a class tree.

In this regard, software developers increasingly have to work with hierarchically organized data. In general, everything comes down to modeling a multi-level relationship "master-subordinate", "ancestor-descendant", "general-specific", i.e. a graph without cycles is modeled.

There are 3 main approaches to organizing hierarchical data:

- parent-child;
- nested set method; materialized
- path model.

Parent-Child

Parent-child method or adjacency list model – is one of the first methods developed for storing hierarchies in relational databases.

This is an intuitive way of organizing a hierarchical structure, which is a reflexive relationship - a closure of the database table relationship on itself (Figure 1).

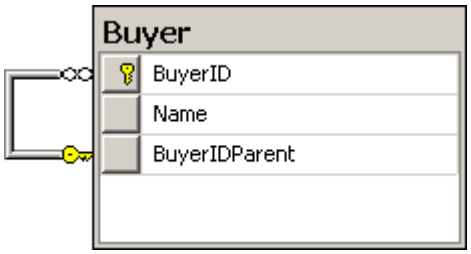



Figure 1. Example of reflexive connection

As is known from theory, a graph can be represented as a matrix, where at the intersection of the i-th row and the j-th column there is 1 if there is a connection (edge, arc) between the nodes (vertices) of the graph with numbers i and j, or 0 otherwise. Such an abstraction is called an adjacency matrix.

The adjacency matrix can also be represented as a list (set) of pairs with numbers (identifiers, codes) of vertices according to the principle: if there is a pair, there is a connection; if there is no pair, there is no connection.

Root nodes are distinguished from other pairs by an empty (NULL) reference to the ancestor, in

the example given this is the "BuyerIDParent" field.

Retrieving immediate parents or descendants of this hierarchical data organization is a simple SQL query, while retrieving a subtree at a given node would require a recursive query.

```
- - Selecting immediate descendants of a node SELECT * FROM
dbo.Buyer where BuyerIDParent=1
- - Select all descendants of some node
WITH BuyerChild(BuyerID, BuyerIDParent, [Name], [Level]) AS (

        SELECT BuyerID, BuyerIDParent, [Name], 1 FROM
        dbo.Buyer
        WHERE
        BuyerID=1 UNION
        ALL
        SELECT dbo.Buyer.BuyerID, dbo.Buyer.BuyerIDParent, dbo.Buyer. [Name],
        [Level]+1
        FROM dbo.Buyer INNER JOIN BuyerChild ON
        BuyerChild.BuyerID=dbo.Buyer.BuyerIDParent
)
SELECT * FROM BuyerChild ORDER BY [Level], BuyerIDParent
```

The advantages and disadvantages of the adjacent node list model are given in Table 1.

Table 1. Advantages and disadvantages of the parent-child approach

| Characteristic | Adjacency List Model peaks |
|---|---|
| Simplicity of structure (number of tables/links/ minimum number of fields) | 1/1/3 |
| Forward selection of children of a node | Yes |
| Direct subtree selection (all descendants of a node) | No, recursion |
| Forward path selection from a node to the root (all ancestors of the node) | No, recursion |
| The order of nodes when sorting | No |
| Quick insertion of new nodes | Yes |
| Fast subtree traversal | Yes |
| Quick subtree deletion | Yes, recursion |
| Storage redundancy | No |
| Number of tree levels | Unlimited |
| Additional integrity support (other than referential) | Not needed |

Nested Set Model

The nested-set model of trees was proposed by Joe Celko and is a method of storing the graph traversal path in prefix order (Figure 2).

The square in the figure represents a node, the number in its left corner is the ordinal number of the route stage when entering the node, and the number on the right is the ordinal number of the exit. With the prefix traversal order, the root of the graph, the vertex "Fedorov Yegor Ivanovich", is visited first, and then the prefix traversal of the subtrees with the roots "Kravtsov Evgeny Nikolaevich", ..., "Petko Tatyana Filipovna" is performed in the specified sequence.

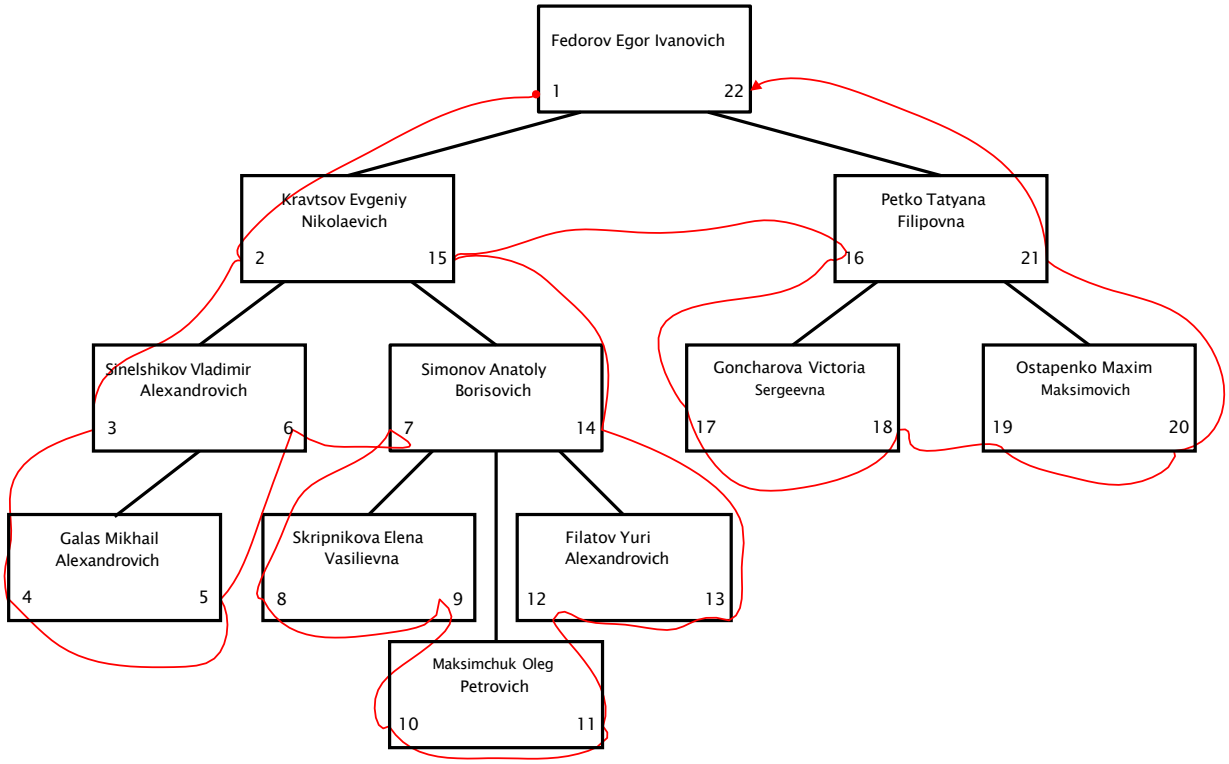



Figure 2. Example of a prefix bypass route

As you can see, the numbers of descendants are always located in the interval between the corresponding numbers of the ancestor, however distant. By storing such an order of tree traversal in the database (Figure 3), when selecting descendants/ancestors of a certain node, recursion can be avoided.

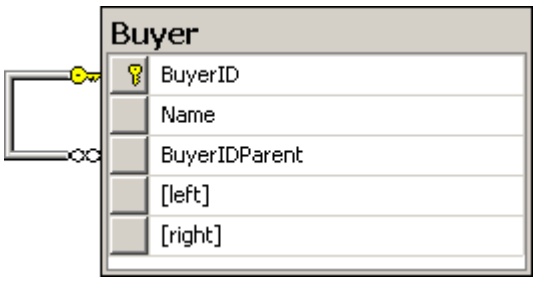



Figure 3. Example of a table with fields for storing prefix order of traversal

```
- - Select all descendants of some node
SELECT b2.* FROM dbo.Buyer b1 INNER JOIN dbo.Buyer b2 ON b2.[left]
BETWEEN b1.[left] AND b1.[right]
WHERE b1.BuyerID=1
```

An obvious complication of the nested set model is the recalculation of the traversal order when adding new nodes or moving existing ones.

To calculate the values*left*AndThe following algorithm is used for the right vertices:

1. The traversal starts from the root node. The value*left*the root is assigned1. To the counter*p*is assigned2.

2. If there are no direct descendants for the current node or among the descendants there are none for whom the meaning*left*not assigned, then the value*right* this vertex is equal*p.p-p*-1. Go to 5.

3. The direct descendant of the current node for which the value*left*it hasn't happened yet assigned, appointed*left-p.p-p*-1.

4. The current node becomes the direct descendant to which the assignment was made. meaning*left*. Transition to2.

5. If there is a parent for the current vertex, then the current vertex becomes this parent. Go to 2. Otherwise, end of the algorithm.

The advantages and disadvantages of the multiple tree model are given in Table 2.

Table 2. Advantages and disadvantages of the multiple tree model

| Characteristic | Adjacency List Model peaks |
|---|---|
| Simplicity of structure (number of tables/links/ minimum number of fields) | 1/0/4 |
| Forward selection of children of a node | Yes |
| Direct subtree selection (all descendants of a node) | Yes |
| Forward path selection from a node to the root (all ancestors of the node) | Yes |
| The order of nodes when sorting | Yes |
| Quick insertion of new nodes | No |
| Fast subtree traversal | No |
| Quick subtree deletion | Yes |
| Storage redundancy | No |
| Number of tree levels | Unlimited |

| | |
|---|---|
| Additional integrity support (other than referential) | Needed, complex |

Materialized Path Model

The materialized path model is a relatively new direction, which uses a special key that is associated with each vertex of the tree and contains information about all the vertices along the path (a sequence of vertices, where each vertex is connected to a neighboring arc) from the root of the tree to the vertex itself (Figure 4).

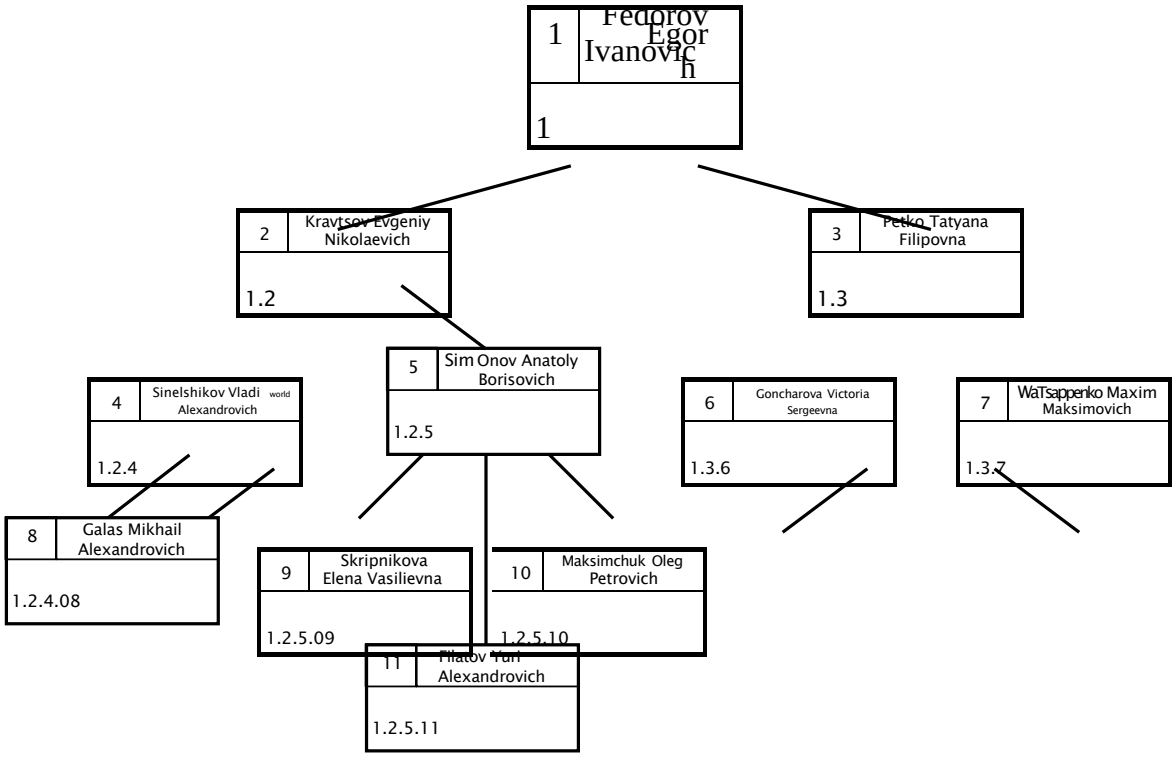



Figure 4. Example of a prefix bypass route

The square in the figure represents a node, the number in the upper left corner is the node identifier, and in the lower left corner is the materialized path, which contains a list of numbers of all values that need to be passed, from the root of the hierarchy to the given value. Each hierarchy level is separated by the symbol "."

This approach is the most visual in terms of codification of elements: each node receives an intuitively understandable meaning, the code itself and its parts carry a semantic load.

A query to retrieve all descendants of a node using the materialized path model would look like this:

- - Select all descendants of some node

```
SELECT b2.* FROM dbo.Buyer b1 INNER JOIN dbo.Buyer b2 ON b2.treeKey
BETWEEN b1.treeKey AND b1.treeKey+'/'
WHERE b1.BuyerID=1
```

In the above query, the "treeKey" field stores the materialized path. The advantages and disadvantages of the multiple materialized path model are summarized in Table 3.

Table 3. Advantages and disadvantages of the materialized path model

| Characteristic | Adjacency List Model peaks |
|---|---|
| Simplicity of structure (number of tables/links/ minimum number of fields) | 1/0/2 |
| Forward selection of children of a node | Yes |
| Direct subtree selection (all descendants of a node) | Yes |
| Forward path selection from a node to the root (all ancestors of the node) | Yes |
| The order of nodes when sorting | Yes |
| Quick insertion of new nodes | Yes |
| Fast subtree traversal | No |
| Quick subtree deletion | Yes |
| Storage redundancy | No |
| Number of tree levels | Limited |
| Additional integrity support (other than referential) | It's necessary, not complicated |

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Add a manager column to the employees table (foreign key to the code employee). Add data to the table so that all existing employees are

are subordinate to 2 new managers (each seller to only one of his own managers), and the 2 new managers, in turn, are subordinate to the store director.

2. Add 2 new columns to the table (right key, left key). Write query for key calculation and key input using Joe Selko's method.

3. Add a new column (materialized path) to table 1. Write request to calculate and update a materialized path.

4. Write a query to select all subordinates (at all levels) of a certain employee for each of the three ways of representing the hierarchy.

Test questions:

1. Prepare material for inclusion in the course report presentation "Database tuning".

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

# LABORATORY WORK #10.Partitioning tables

Subject -Partitioning tables

The purpose of the work ismaster table partitioning The

competencies being formed or their parts -OPK-5

Theoretical part.

Partitioning of tables (horizontal partitioning of tables) and indexes is used to improve performance, scalability, and manageability in very large databases (VLDB). VLDB is the name given to databases whose total size is measured in hundreds of gigabytes, and possibly terabytes or more. A large table is one whose performance or maintenance time exceeds acceptable limits. A table can also be considered large if the actions of one user significantly affect another user, or if database maintenance operations affect the capabilities of other users.

On multiprocessor systems, partitioning large tables results in improved performance through parallelism. Large-scale operations on extremely large data sets (millions of rows) can benefit from parallel processing of independent subsets of the data. A simple example of the performance improvements that partitioning can provide is aggregation. Instead of grouping data into one large table, SQL Server can group it into multiple partitions independently, and then combine the aggregations.

In SQL Server, related tables (for example, Bill and BillItem) that are partitioned by the same partition key (in this case, BillId) and the same partition function are called aligned. If the SQL Server optimizer detects that two partitioned and aligned tables are being joined, it may choose to join the data that resides in the same partitions first and then merge the results. This allows SQL Server to use multiprocessor systems more efficiently.

Partitioning makes large tables and indexes more manageable by allowing you to quickly and efficiently access and manage subsets of data while maintaining the integrity of the entire collection. When using partitioning, loading data from an OLTP system into an OLAP system takes significantly less time than using non-partitioned tables and indexes.

Partitioned tables and indexes divide data into blocks (partitions), which can be distributed across multiple filegroups in the database. The data is partitioned horizontally, so that groups of rows are mapped to individual partitions. A table or index is treated as a single logical entity when queries or updates are performed on the data. All partitions of an index or table must reside in the same database.

Partitioned tables and indexes support all the features and capabilities associated with designing and querying standard tables and indexes, including constraints, defaults, identity and timestamp values, and triggers.

Before partitioning a table or index, you must plan to create the following database objects:

- partitioning functions;
- partitioning schemes.

The partitioning function determines how rows in a table or index are distributed across partitions based on the values of certain columns, called partition columns.

The partition scheme maps each of the partitions defined by the partition function to a filegroup.

When planning a partitioning function, two conditions should be taken into account: the columns whose values determine how the table is partitioned, i.e. the so- called partitioning columns, and the range of partitioning column values for each of the partitions. This range determines the number of partitions into which the table is divided. In MS SQL Server, tables can have no more than 1000 partitions. For example, the partitioning function for the Bill table can be split by sales month by the Bill.Date column.

When performing frequent queries to equivalently join two or more partitioned tables, their partitioned columns must match the columns on which the join is performed. Additionally, the tables or their indexes must be ordered. This means that they either use a common named partitioning function or different ones that produce the same result. That is:

1. The specified tables are partitioned by the same number of parameters, having the same data type.
2. The tables listed have the same number of sections.

3. In the specified tables, the partitions have the same boundary values. When these conditions are met, the SQL Server query optimizer performs the join operation much faster, since in this case the partitions themselves can be joined. If the query performs a join of two unordered tables, or tables that are not partitioned for joining, the presence of partitioning in the table can slow down the query instead of speeding it up.

To improve I/O performance, you may want to combine partitions into file groups located on different physical disks. When sorting data for I/O operations, SQL Server first sorts the data into partitions. With this scheme of operation, SQL Server can access only one disk at a time, which can lead to a decrease in performance. To improve performance, it is recommended to distribute the data files of different partitions across multiple hard disks, while setting up RAID. This way, despite sorting the data into partitions, SQL Server can simultaneously access all hard disks of each partition. This configuration can be applied both to partitions within a single file group and to partitions located in multiple file groups.

Partitioning algorithm

To create a partitioned table or index, you need to perform the following steps:

1. Create a partition function to specify how the table or the index where this function is used may be partitioned.
2. Create a partition scheme to specify the placement of partitions by the function. partitioning for file groups.
3. Create a table or index using a partition scheme. The partition function specifies how the table or index is partitioned. The partition function transforms a domain into a set of partitions. To create a partition function, you specify the number of partitions, the partitioning column, and the range of values for the partitioning column for each partition. Note that you can specify only one partitioning column. All data types that are valid as index columns can be used as partitioning columns, except timestamp. The following data types cannot be specified: ntext, text, image, xml, varchar(max), nvarchar(max), and varbinary(max). Additionally, you cannot specify columns that contain user-defined types or Microsoft .NET Framework common language runtime (CLR) alias data types.

The partition function has the following syntax:

```
CREATE PARTITION FUNCTION partition_function_name
( input_parameter_type )
AS RANGE [ LEFT | RIGHT ]
FOR VALUES ( [ boundary_value [ ,...n ] ] ) [ ; ]
```

Where,

input_parameter_type - the data type of the column used for sectioning;

boundary_value – boundary values for each section;

LEFT | RIGHT – indicates which area of the value range belongs to argument*boundary_value*[,*...n*] (to the left or right).

A partition scheme maps the partitions resulting from the partitioning function to a specific set of filegroups.

When creating a partition scheme, you specify the filegroups to which the table partitions are mapped, based on the arguments to the partition function. You must specify enough filegroups to accommodate the number of partitions. You can specify that all partitions be mapped to different filegroups, that some partitions be mapped to a single filegroup, or that all partitions be mapped to a single filegroup. You can also specify additional, "unassigned" filegroups in case you plan to add more partitions later. In this case, SQL Server marks one of the filegroups with the NEXT USED property. This means that the filegroup will contain the next partition added.

Only one partitioning function can be used for a partitioning scheme. However, a partitioning function can participate in two or more partitioning schemes.

The function to create a partition scheme has the syntax:

```
CREATE PARTITION SCHEME partition_scheme_name
AS PARTITION partition_function_name
[ ALL ] TO ( { file_group_name | [ PRIMARY ] } [ ,...n ] ) [ ; ]
```

Where

partition_function_name – the name of the partition function used sectioning scheme;

ALL – specifies that all sections are associated with the filegroup, defined by the argument*file_group_name*, or with the primary filegroup, if indicated [PRIMARY];

file_group_name – specifies the names of file groups containing sections, specified by the argumentpartition_function_name. If specified[PRIMARY], the section is stored in the primary filegroup.

Creating partitioned tables

The EducationDatabase training database accumulates sales data; the current (last processed) sales year is 2011. Let's partition the Bill table. We'll select the product sale date Date as the partitioning attribute. Each section will contain sales data for a specific month in 2011, except for the last section, which will contain sales data for the last month of 2011 and later. Additionally, one section will contain sales made before 2011.

Table partitioning order:

1. For the Bill table, in the context menu, select Storage -> Create Partition. The Partition Wizard window will open.
2. In the first step of the wizard, select the table column on the basis of which partitioning will be performed. Let's select the Date column (Figure 1).

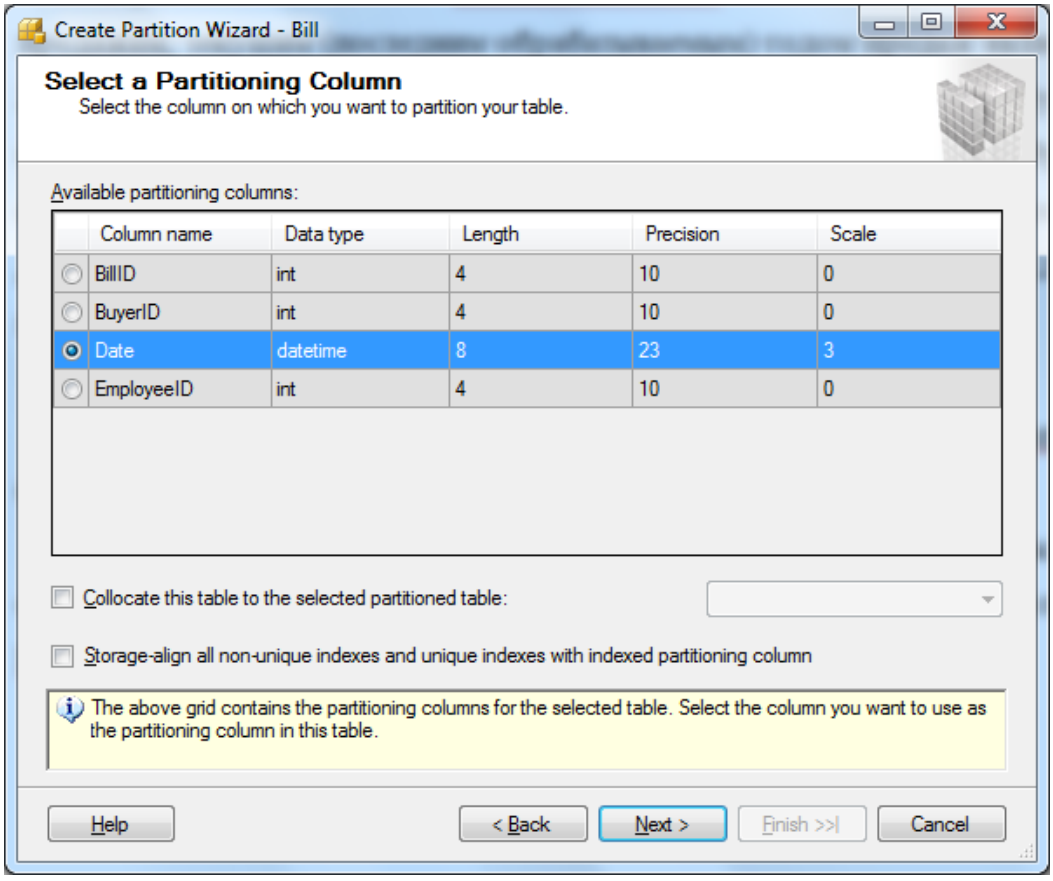


Figure 1 – Selecting a table partition column

3. In the Select a Partition Function step, specify the name of the partition function. (Figure 2).

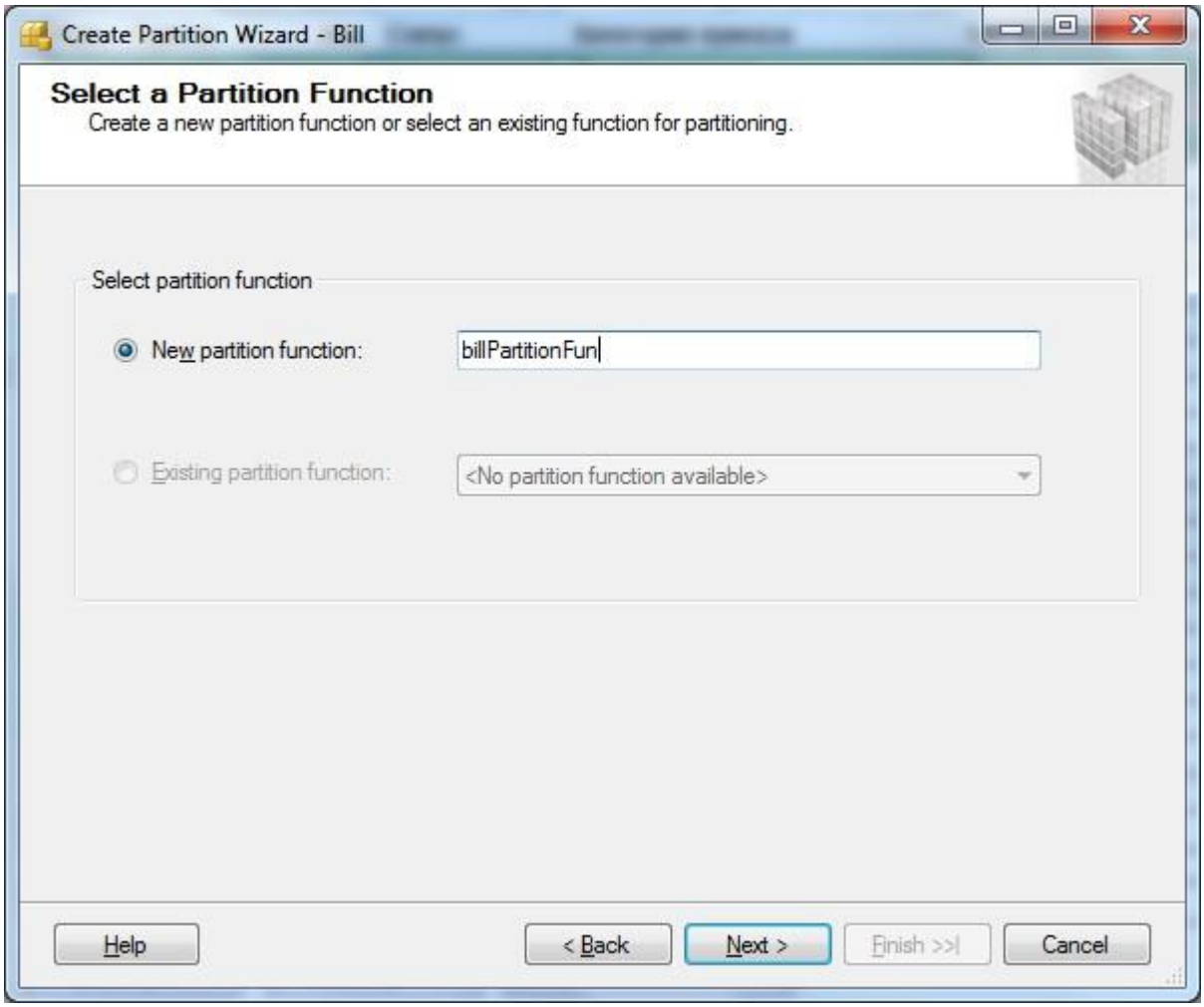



Figure 2 – Creating a partition function

4. In the Select a Partition Scheme step, specify the partition scheme name. (Figure 3).

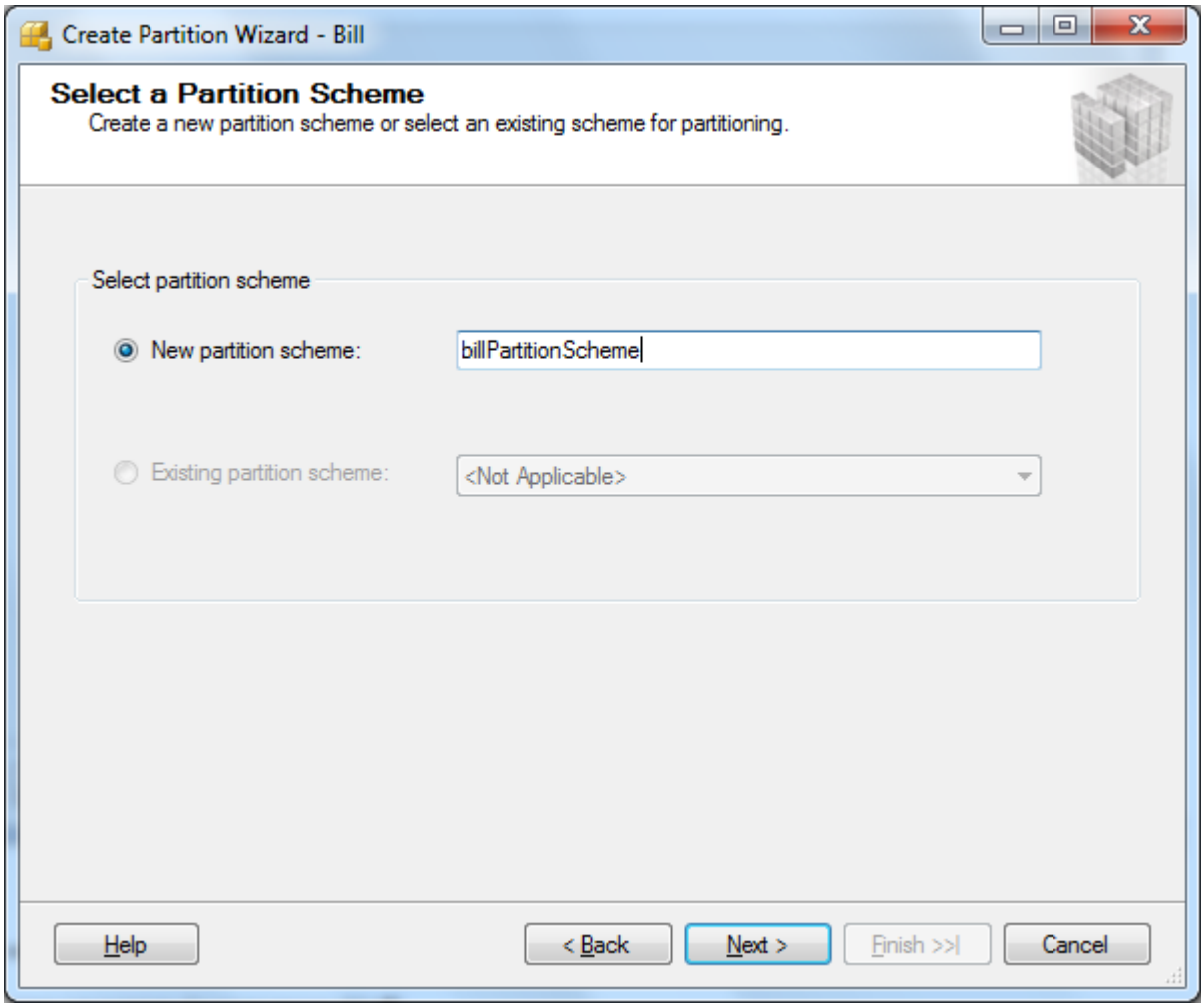


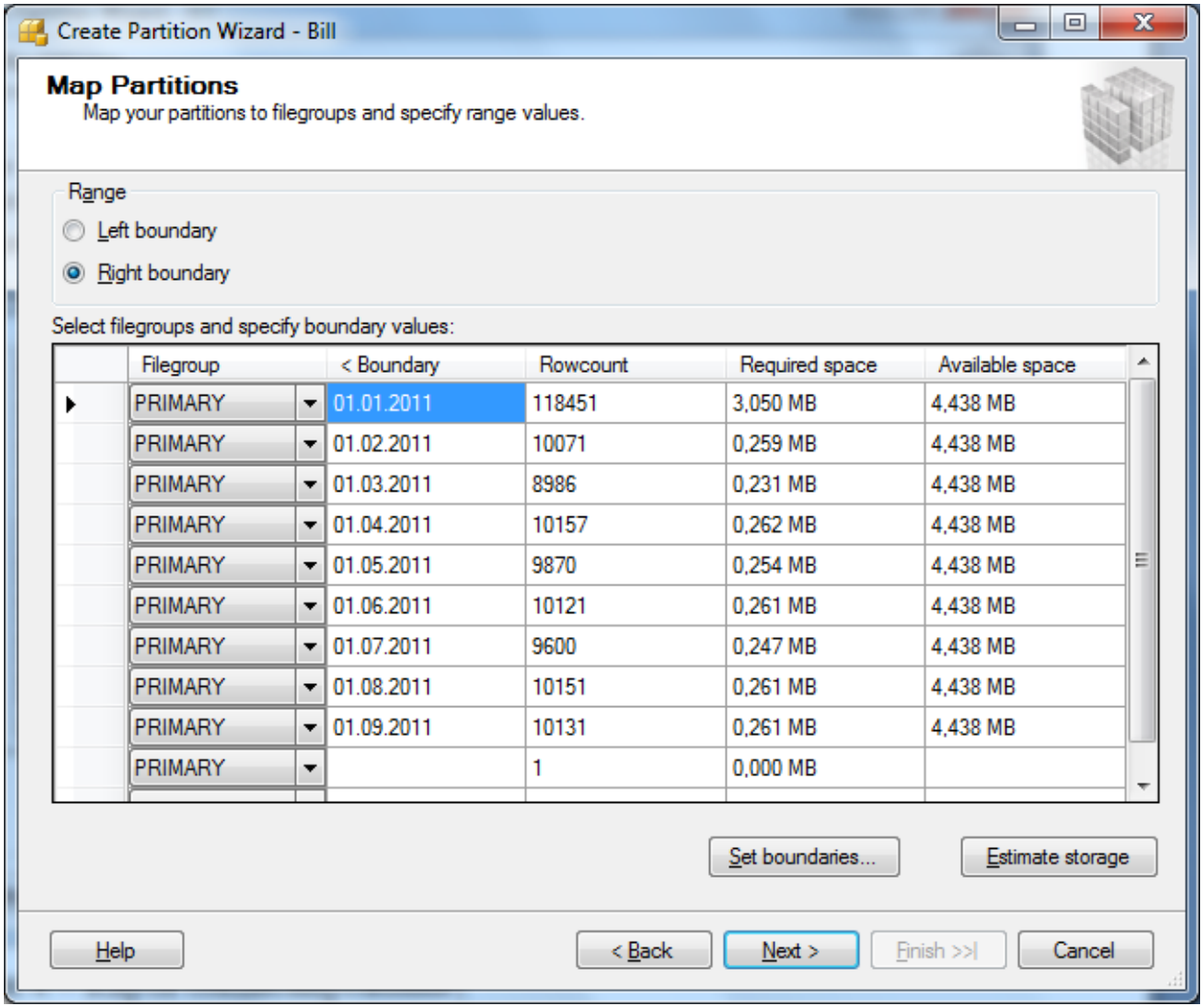

Figure 3 – Creating a partitioning scheme

5. In the Map Partitions step, you need to define the partition boundaries and link them to file groups. First, you need to decide on the type of partitioning intervals. If the Left Boundary option is selected, intervals of the type (previous value, current value] will be generated. If the Right Boundary option is selected, intervals of the type [previous value, current value] will be generated. Each section can be assigned its own file group, or several sections (possibly all) will be associated with one file group. Let's select the PRIMARY file group for all sections. It should be taken into account that significant gains in performance and throughput can be achieved by placing different sections in different file groups on different physical disks. In this case, it is possible to provide, for certain queries, parallel access to disk arrays. The right boundaries of the section ranges are specified in the Boundary column, and it is necessary to specify the boundaries of each section. Automatic calculation of boundaries is possible

ranges using the Set Boundaries… button. In the latter case, the DBMS will automatically determine the minimum and maximum values in the partitioning column and suggest the section size (by month, by quarter, etc.). The user must independently determine the required upper and lower partitioning boundaries (we are interested in 2011) and decide on the section size (we will choose by month). The Estimate Storage button is used to calculate the characteristics of the sections: the number of records, the required amount of memory, the amount of memory occupied. As a result, the partitioning map will look like the one shown in Figure 4.

Figure 4 – Partitioning map

6. In the Select an Output Option step, select the Create Script option, open the script




in new query editing window Script to New Query Window (Figure 5). Other possible options are: Run Immediately, Schedule.

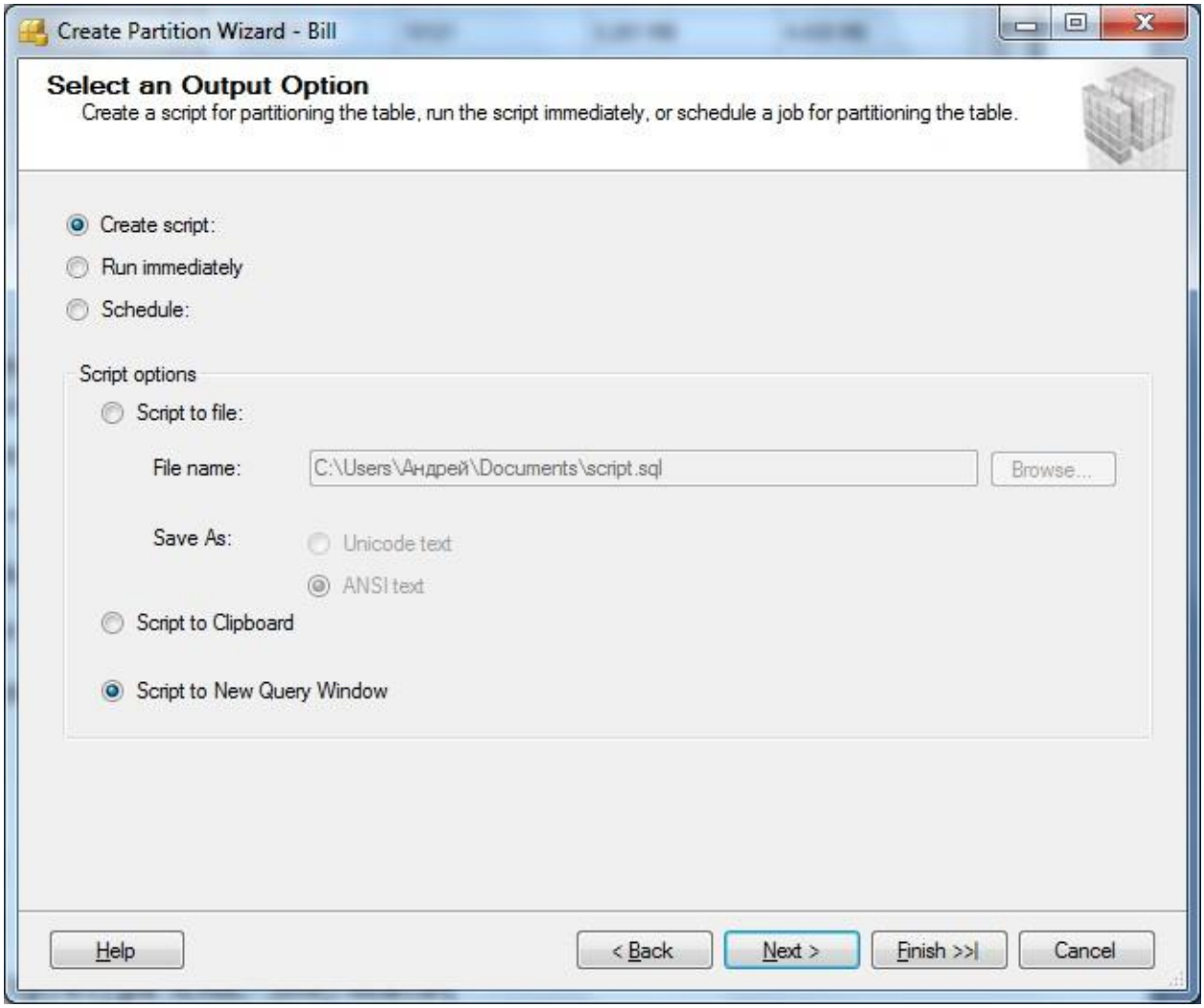



Figure 5 – Unloading options

7. In the final step Review Summary, check the summary information about partitioning and click Finish.

Execute the generated script:

```
USE [EducationDatabase]
GO
BEGIN TRANSACTION

CREATE PARTITION FUNCTION[billPartitionFun](datetime) AS
RANGE RIGHTFOR VALUES
        (N'2011-01-01T00:00:00',N'2011-02-01T00:00:00',
        N'2011-03-01T00:00:00',N'2011-04-01T00:00:00',
        N'2011-05-01T00:00:00',N'2011-06-01T00:00:00',
        N'2011-07-01T00:00:00',N'2011-08-01T00:00:00',
        N'2011-09-01T00:00:00')

CREATE PARTITIONSCHEME [billPartitionScheme]AS PARTITION
[billPartitionFun]TO([PRIMARY],[PRIMARY],[PRIMARY], [PRIMARY],[PRIMARY],
[PRIMARY],[PRIMARY],[PRIMARY], [PRIMARY],[PRIMARY])
```

```
CREATE CLUSTERED INDEX
[ClusteredIndex_on_billPartitionScheme_634648357407858521] [dbo].[Bill]  ON
([Date])WITH(SORT_IN_TEMPDB=OFF,IGNORE_DUP_KEY=OFF, DROP_EXISTING
=OFF,ONLINE=OFF)ON [billPartitionScheme]([Date])

COMMIT TRANSACTION
```

Please note that the DBMS creates a new clustered index on the Date column to replace the clustered index on the primary key-counter BillId, which implies reorganization of the data in sorting order by purchase dates and dividing the sorted data into sections.

Let's look at how the plans for executing queries to select data from the Bill table have changed.

Let's execute the query:

```
select*frombill
where datebetween'05.02.2011'and'20.02.2011'
```

In this case, all the resulting records are located in one section associated with the month "February 2011". The search for records is performed in one section (Figure 6) using a clustered index on the date field (Figure 7).

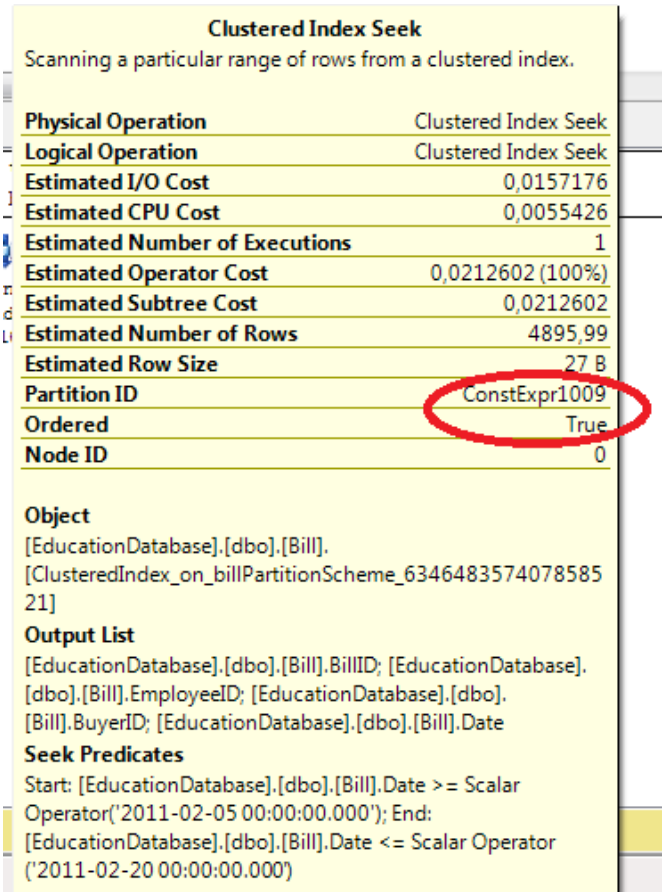



Figure 6 – Description of the search operation by cluster index

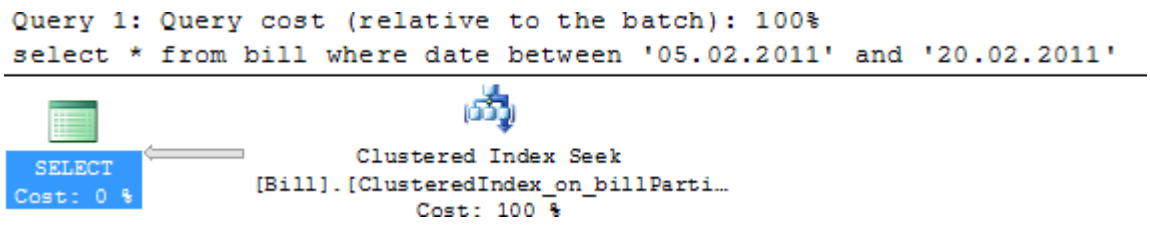



Figure 7 – Query execution plan

Let's execute the following

query: select*frombill

where datebetween'05.02.2011'and'20.03.2011'

Now the resulting records are located in two partitions associated with the months "February 2011" and "March 2011". The query plan has changed due to the appearance of an additional operation of reading the internal constant table with partition numbers (Figure 8). The search operation by the clustered index is performed twice according to the number of partitions participating in the query (Figure 9). The new query execution plan is shown in Figure 10.

Figure 8 – Operation of scanning the internal constant table

**Constant Scan**

Scan an internal table of constants.

| | |
|---|---|
| **Physical Operation** | Constant Scan |
| **Logical Operation** | Constant Scan |
| **Estimated I/O Cost** | 0 |
| **Estimated CPU Cost** | 0,0000022 |
| **Estimated Number of Executions** | 1 |
| **Estimated Operator Cost** | 0,0000022 (0%) |
| **Estimated Subtree Cost** | 0,0000022 |
| **Estimated Number of Rows** | 2 |
| **Estimated Row Size** | 11 B |
| **Node ID** | 1 |

**Values**

(Scalar Operator((3))); (Scalar Operator((4)))

**Output List**

PtnIds1006

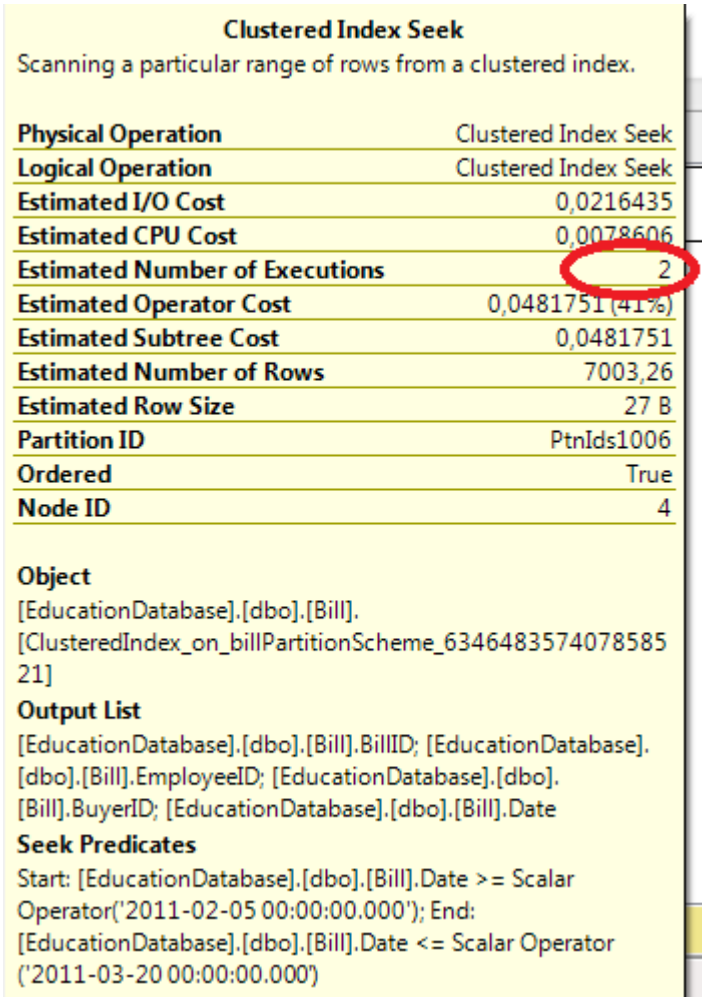



Figure 9 – Clustered Index Search Operation

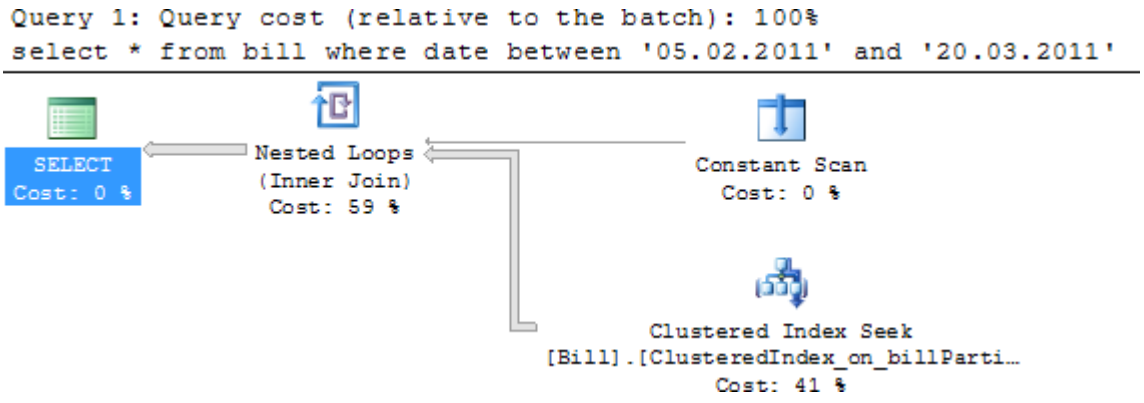



Figure 10 – Query execution plan two

Presented examples clearly demonstrate advantages data partitioning – the data access interface does not change (the same SQL query syntax is used), but the query optimizer changes the data access method in a way that is transparent to the developer and the application, with the ability to execute individual operators in parallel (in the latter case, searching by cluster index).

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Write 2 queries to join the Bill and BillItem tables in the first case with filtering Bill.date between '02/05/2011' and '02/20/2011', in the second case with filtering Bill.date between '02/05/2011' and '03/20/2011'. Study the query execution plans, explain why they are different. Save the query execution plans.
2. Partition the BillItem table using the scenario described in this lab. Use BillItem.Date as the partition key. Explain whether the partitioned tables Bill and BillItem are aligned?
3. Carry out the requests from point 1 and study the plans for their implementation. Explain that changed in the query execution plans.

Test questions:

1. Prepare material for inclusion in the course report presentation "Database tuning".

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., rev. - Moscow: Academy, 2012. - 314, [1] p. : ill. ; 22. - (Higher

professional education. Computer science and Grif: Computer Science. Bachelor's degree) - Additional UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL:

With.

http://biblioclub.ru/index.php?page=book&id=231519

# LABORATORY WORK #11.Principles of construction of statistical and dynamic model.

Subject -Principles of constructing a statistical and dynamic model.

The purpose of the work ismaster the principles of constructing statistical and dynamic models.

The competencies being formed or their parts -OPK-

5 Theoretical part.

There are two main types of information model:static and dynamic.

Static information model

A static model describes the permissible states of a system. Static models describe the types of objects in a system, their properties and relationships. Describing objects, they define, of course, their names.Reach agreement on naming everything objects means winning half the battle, which is why XML information models are sometimes called vocabularies.

When constructing a static information model, the following steps must be completed:

Step 1: Identify, name, and define concepts. Step 2: Organize
concepts into a class hierarchy.

Step 3: Identify Relationships, Multiplicity, and Constraints

Step 4: Adding properties to specify the details of the values associated with objects

Step 1. Naming concepts

First, we need to make a list of concepts related to the system. Sometimes they even suggest describing the system on paper and highlighting all the nouns. In any case, this stage, as a rule, does not present any difficulties.

Next, and this sometimes takes more time, you need to describe the types of objects. The description of the term must be precise so that there is no disagreement about the essence of the definition. The value of modeling is that it prevents potential sources of misunderstanding.

When you're done, you'll probably have a long list of object types, some of which will have long names. You want to choose names that people in the business can understand and interpret correctly, because they won't always check the written definitions.

In addition to naming object types, it is also worth determining how individual instances can be identified. Perhaps there is code in this type of business that you should know or write. It is important to keep in mind that problems are often found in business coding schemes.

At the end of this step, we will have a list of object types with names and definitions that have been agreed upon.

Stage 2: Taxonomy

Taxonomy is a term used in biology to refer to a classification system; in information modeling it is also called a type hierarchy (sometimes called a*ontology).* Having listed and named the object types, they must be organized into a hierarchical classification scheme. Often these hierarchical relationships arise already at the stage of defining the object types.

The key here is the is phrase. By writing a sentence like “A is a kind of B” or “Every A is a B,” you have defined the subtype relationships in your taxonomy.

Sometimes these actions are called a test.*is a” (“is”, “represents”).* Be careful though, since this construct is also used to describe the relationship between a specific instance and its type, it is safer to write this test in the form*“is a kind of“.*

Identifying subtypes can be useful when designing a document, but more importantly, it helps you better understand the definitions of object types.

If you've done object-oriented programming, you understand how to define type hierarchies. But programmers often think of object classes primarily in terms of modules of functionality within a system, rather than the concepts they represent in the outside world. Verbs, rather than nouns, are then used to denote object types - which is wrong..

So, step 2 comes down to organizing the object types into a type hierarchy. Step

3: Finding Connections

Once the objects have been named, in static information modeling the relationships that exist between them must be defined. The relationships (called associations in UML) can be shown simply by stating them as simple sentences or they can be shown graphically in the form of a diagram. There are many notations for diagrams that describe the relationships between objects, and everyone can choose the one that suits them best. Diagrams should be made as

simple and intuitive, focusing on key messages and leaving the details to easier to maintain text documents.

There is some information you need to know about each connection:

1. The multiplicity of a connection shows how many objects of each type are accepted in her participation.

a. One-to-many relationships: one chapter contains many paragraphs, one A person buys a lot of tourist trips.

b. Many-to-many relationships are also common: one author can write several books, but a book can also have several authors.

c. One-to-one relationships.

2. When modeling information for the final XML representation a particularly important type of connection isinclusion links. The multiplicity of these relationships is always one-to-many and one-to-one. Although there is no hard and fast rule about which objects form containment relationships, you can sometimes use the rules of ordinary language: a chapter contains paragraphs, a resort contains hotels, and a hotel contains visitors. The UML defines two forms of containment relationships. The first is aggregation, a relatively loose association of objects that allows a group of objects to be considered as a whole over time (for example, a tour group, where the same people may belong to different groups at different times). The second form is composition..This is a more strict form; separate parts the whole cannot exist independently of it (for example, rooms in a hotel cannot exist independently of the hotel).

Finding appropriate names for relationships can be difficult. When writing relationships, it is useful to use full phrases like "a hotel is located in a resort" or "a person is the author of a book". We do not need to use these names in XML markup tags; they only appear in the documentation for the system.

So, at the end of step 3, we have defined the relationships that exist between the types of objects in our model.

Step 4. Description of properties

Object types and relationships form the skeleton of a static information model; properties can be compared to the flesh on bones. Properties are simple values associated with objects. A person can be defined as having a height, weight, nationality, and occupation; a hotel has a certain number of rooms, floors, and price category.

We should not include relationships in the list of properties of the object again: the location of the hotel is not a property of the object if we have already modeled it as a relationship with the resort.

The main thing we need to know about properties is their data type. Do they have a fixed range of values, are they numeric, what units are they expressed in? Is the property mandatory and does it have a default value?

At the end of stage 4, we completed the formation of a static information model: we received a complete description of the types of objects in the system, their relationships with each other and their properties.

Dynamic information model

Dynamic models describe what happens to information: examples of such models are workflow diagrams, data flows, and object life cycles. Dynamic models consist of statements like: “The pathology department will send the test results to the consultant responsible for the patient.” Dynamic models describe the process of information exchange: data is sent from one place to another for a specific purpose.

To build a dynamic model, you can use various software systems (for example, workflow diagrams and data flow diagrams can be built using the BPWin program).

There are several approaches to dynamic modeling:

- Workflow models
- Data flow models
- Object models
- Life cycles of objects
- Use cases
- Object interaction diagrams

Workflow models

Workflow models focus on the role of people and organizations in getting the job done, with information storage and processing playing a secondary role. A process model describes, for example, what will happen to a traveler if he or she has an accident while on vacation. It identifies the actions that the local representative at the resort, the agent in the country where the resort is located, and the head office are responsible for. As a result, it becomes clear who should be responsible for arranging medical care, transporting the traveler home, and informing relatives.

It may describe various forms that are filled in and sent between participants, and not involve computer systems at all. A process model typically focuses on the roles, responsibilities, and tasks of each actor in the system, while a workflow model deals with the documents that are transferred between actors. Data Flow Models

Data Flow Models

Data flow models are very similar to the previous type, but here the focus is on information systems rather than business systems. This model describes *data warehouses*(data stores), where information is permanently located (this could be a database on a computer or just an office with folders),*processors*,manipulating this data, and*data streams*,transmitting data from one processor to another.

It makes heavy use of the static information model: the latter describes what concepts like traveler and hotel mean, but says nothing about where the information is stored. In contrast, the data flow model makes it clear that information about a travel experience resides in a purchase database until the trip is completed and all bills are paid, at which point a summary of that information is passed to the marketing information system, and the rest is archived.

Object models

Object models contain both dynamic and static components. The dynamic or behavioral part of an object definition focuses on what each object can do or has done, representing a set of operations or methods that describe its actions.

Object life cycles

Object life cycles (called object lifelines in UML) also focus on individual objects, but take a more holistic approach. They describe what happens to an object during its life: how it is created, what events happen to it, how it reacts to those events, and what conditions ultimately lead to its destruction.

Object lifecycles are very useful for testing the completeness of a model. There is often a tendency to focus on only some events at the expense of others. Until we determine how each object enters and leaves the system, we will not have a complete understanding.

Use cases

Use cases analyze the performance of specific user tasks (e.g., a person who purchased a vacation package cancels their order). A use case resembles a process model, but generally focuses on the activity of one specific user.

Use cases can be useful both at the stage of modeling business activity and in describing the internal behavior of information systems. One danger is mixing the two levels. It is better not to do this, since they are of interest to different audiences.

When presented as a use case, a user interface dialog describes what information the user exchanges with the system, rather than how it is presented on the screen. This naturally leads to an XML implementation in which the information content is separated from the presentation details.

Object interaction diagrams

Object interaction diagrams allow you to analyze the exchange of messages between objects at a finer level of detail than a data flow model.

Object interaction diagrams are invaluable when it comes to describing interactions between different systems. They allow us to determine what information is contained in which messages. Because these messages are written in XML, object interaction diagrams give us the context we need to begin designing the XML structure of each individual message.

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a malfunction of the personal computer, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace

The personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Study the basic principles of constructing a static and dynamic model.
2. Construct a statistical and dynamic model for a given option.
3. Prepare a report including the statement of the problem, static and dynamic models.
4. Defend laboratory work. Test questions:

1. Provide a definition of a static information model.
2. What stages must be completed when constructing a static information system? models.
3. What is the “concept identification” stage when constructing a static information model.
4. What is the stage of “organizing concepts into a class hierarchy” when construction of a static information model.
5. What is the “connection definition” stage when constructing a static information model.
6. What is the “property description” stage when constructing a static information model.
7. Provide a definition of a dynamic information model.
8. What are workflow models?
9. What are data flow models?
10. What are object models?
11. What do life cycles of objects describe?
12. What use cases describe.
13. What do object interaction diagrams describe?

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

LABORATORY WORK #12.Language constructionsXML.

Subject -Language constructionsXML.

The purpose of the work ismaster the language

structuresXML. The competencies being formed or their

parts -OPK-5 Theoretical part. XML syntax

Any XML document consists of the following parts:

· Optional prologue.

· Document body.

· An optional epilogue following the element tree. Let's look at each part in more detail.

Prologue

The prologue consists of several parts:

1. an optional XML Declaration that is enclosed between symbols <?...?>. The announcement contains:

· xml mark and version number (*version)*specificationsXML;

· indication of character encoding (*encoding),* in which the document is written (according to default*encoding=”UTF-8”*);

· parameters*standalone*which can take the values “*yes”*or*“no”*(By default*standalone=”yes”).*Meaning "“yes” indicates that the document contains all required element declarations, and “no” indicates that external DTD definitions are needed.

All this together might look like this:

<?xml version="1.0" encoding-"windows-1251" standalone="yes"?>.

It is important to note that in the XML declaration only the attribute*version*is mandatory, all other attributes can be omitted and therefore take default values. It is also important to remember that all these attributes should be specified only in the order given above.

2. comments.

3. processing commands.

4. empty space symbols.

an optional Document Type Declaration (DTD) that is enclosed between <! DOCTYPE...> symbols and may span multiple lines.

This section declares the tags used in the document, or provides a link to a file containing such declarations.

The document type declaration may also be followed by comments, processing commands, and white space characters.

Since all these parts are optional, the prologue can be omitted.

Document body

The document body consists of one or more elements. In a well-formed XML document, the elements form a simple hierarchical tree, which necessarily includes *root element(*root element) in which all other elements of the document are nested. Element names must be unique within the document. The name of the root element is considered the name of the entire document and is specified in the second part of the prologue after the word Doctype.

An element begins with an opening tag, then comes the optional content of the element, after which a closing tag is written (unlike HTML, the presence of a closing tag is mandatory, an exception are elements without content, the so-called empty elements, which can be written in a shortened form:

<element_name/>). The following may be used as the element's content:

1. other elements
2. symbolic data
3. symbol references

*In order to insert a symbol into the text of a document, which, for example, is not present in the keyboard layout or may be incorrectly interpreted by the analyzer, use symbol references. A symbol reference must begin with an ampersand and end with a semicolon.*

*Character references are written in the following form:*

*&#character_code_in_Unicode;.*

*The symbol code can also be written in hexadecimal form. In this case, the symbol "x" is placed before it:*

*&#xHexadecimal_code_of_character;.*

4. entity references

*Entity references allow any string constants to be included in the content of elements or the value of attributes. Entity references, like character references, begin with an ampersand, followed by the entity name, and end with a semicolon:*

*&entity_name;.*

*Entity references instruct the parser to substitute a string of characters previously specified in the document type definition.*

5. comments

*If you need to insert a comment into the text of a document or make some fragment "invisible" to the analyzer program, then it is formatted as follows:*

*<!--…comment text…-->*

6. sectionsCDATA

*The CDATA section is used to define a region of the document that the analyzer will treat as plain text when parsing, ignoring any instructions or special characters.*
*The analyzer does not break the CDATA section into elements, but considers it simply a set of characters. Unlike comments, the contents of this section are not ignored, but are passed unchanged to the analyzer program output, so that they can be used in an application.*

*A CDATA section begins with the line <![CDATA[ followed by the contents of the section. The section ends with two closing square brackets and a less-than sign:*

*<![CDATA[section contents]]>*

7. processing instructions.

*Processing instructions contain directions to the XML document parser.*
*Processing instructions are enclosed between the <? and ?> symbols.*

*Immediately after the initial question mark, the name of the software module to which the instruction is intended is written. Then, separated by a space, comes the instruction itself, passed to the software module. The instruction itself is a regular string that should not contain the set of characters "?>", which denotes the end of the instruction. An example of a processing instruction is the XML declaration line:*

*<?xml version="1.0" encoding="windows-1251"?>*

*This instruction is intended for the program that processes the XML document. The instruction passes it the version number and the encoding in which the document is written.*

Start tags or empty element tags in XML may contain attributes, which are name=value pairs. Only one instance of an attribute name is allowed in a single start tag..Attributes can contain references to objects, references to symbols, text symbols. Unlike HTML, in XML attribute values must be enclosed in apostrophes (') or quotation marks (“).

Thus, an attribute can be written in one of two formats: attribute_name=”attribute_value” attribute_name= 'attribute_value'.

Description of document structure using DTD

XML 1.0 introduces a special DOCTYPE declaration to associate a DTD declaration with a document instance. The DOCTYPE declaration contains the DOCTYPE keyword, followed by the name of the root element of the document, and then a construct with content declarations. You can write the external subset of declarations in a separate DTD file, include the internal subset in the body of the DOCTYPE declaration, or do both. In the latter case (mixing internal and external DTDs), internal DTDs can specify new declarations or overwrite those contained in external ones (by definition of the XML specification, parsers read the internal subset first, and therefore declarations contained therein have priority).

The internal markup declaration block of the DOCTYPE tag consists of a left square bracket, a list of declarations, and a right square bracket:

<! DOCTYPE root_element_name […internal subset declarations go here ... ]>

For external DTDs, the DOCTYPE declaration consists of the usual keyword and the name of the root element, followed by another keyword, SYSTEM or PUBLIC, indicating the source of the external DTD definition, and then the locale of that definition.

The content allowed in an XML document is defined by four types of markup declaration in the DTD. The following table shows the keywords associated with these declarations and their meanings. The first two types relate to the information we expect to find in the XML document—elements and attributes.

The last two types are used for support. They make life especially easy for the developer of an XML entity dictionary. They typically consist of content that is used so frequently in a DTD or document that it justifies a special declaration. The use of this declaration resembles the include statement in C/C++, where a name is used as a substitute for the content. Notations describe content that is not written in XML. They are used to declare a particular class of data and associate it with an external program.

DTD construct Meaning ELEMENT Declaration of the XML element type ATTLIST Declaration of the attributes that can be assigned to specific element types, as well as the permitted values for those attributes

ENTITY Declaration of reusable content NOTATION Formatting declaration for external content that should not be parsed

(e.g. binary data) and for external applications that process the content

Let's take a closer look at the first two types of markup declaration in DTD. Each element must be described in an XML document. An element declaration begins with the symbols <!ELEMENT, followed by the element name and its contents, separated by a space. The declaration ends with the "greater than" symbol. Elements are divided into four groups based on their contents.

1. Empty element - can have attributes, but does not contain text or generated elements elements. Declared as follows: the element name is followed by the keyword EMPTY.

Example: <!ELEMENT element_name EMPTY>

2. The element contains only child elements, but no text. Declared as follows: after the element name, all nested elements are listed in brackets, separated by commas. Moreover, nested elements must follow in the XML document in the order in which they are listed in the declaration.

Example: <ELEMENT element_name (elem_1,elem_2)>

3. The element contains not only the generated elements, but also the text. It is declared as follows: after the element name, the keyword is indicated in brackets
# PCDATA, after which, separated by commas, as in the previous case, all nested elements (if any) are listed.

Example: <ELEMENT element_name (#PCDATA, elem_1,elem_2)>

<ELEMENT element_name (#PCDATA)>

4. An element that is open to any content. It is declared as follows:
The keyword ANY is specified after the element name.

Example: ELEMENT element_name ANY>

Sometimes only one of several nested elements or lists (a list of elements specified in brackets) may appear. In this case, their names are listed separated by a vertical bar ( | ). For example: <!ELEMENT element_name (elem_1,(elem_2|elem_3))> - the element element_name must contain the element elem_1, and then either elem_2 or elem_3. The elements appear in this order.

If a nested element can be written in the declared element several times, then this must be indicated using an asterisk, plus, or question mark.

? — an element or list may occur zero or one time;

* — an element or list may occur zero or more times;

+ — an element or list may occur one or more times.

After the declaration of an element, its attributes are declared. All attributes of one element are declared at once, as a single list. The list begins with the symbols <!ATTLIST, followed by the name of the element to which the attributes belong, separated by a space. Then comes the name of the attribute, its type or a list of values that it can take (all values are listed through a vertical bar, in brackets), a sign of the mandatory presence of the attribute in the element, or a default value (this value will be used if the attribute is not explicitly written in the XML document).

The attribute type is written using one of the keywords: 1.

CDATA – a string of characters.

2. ID is a unique identifier that uniquely identifies an element,

in which this attribute occurs; the values of such an attribute must not be repeated in the document.

They play the same role as primary keys in database tables.

3. IDREF — an identifier containing one of the values of attributes of the id type,

used as a reference to other elements.

4. IDREFS - an identifier containing a set of values of attributes of the id type, listed through spaces; also used as a reference to several elements at once.

5. ENTITY — the name of an entity declared in the same object that is not checked by the analyzer.

DTD description.

6. ENTITIES — names of unverified entities.

NMTOKEN is a word containing only characters used in names. Attributes of this type can contain names of other elements or attributes, for example, to refer to them.

7. NMTOKENS - words listed separated by spaces.

8. NOTATION — designation deciphered in the DTD description. The mandatory attribute is written using keywords: 1. #REQUIRED — the attribute must be written in the element;

2. #IMPLIED - the attribute is optional and has no default value;

3. #FIXED - the attribute has only one value, which is written right here,

separated by a space.

Example: <!ATTLIST city type (city | town | village) ”city”> <!

ATTLIST pre xml:lang NMTOKEN "ru_RU">

<!ATTLIST pre xml:space (default | preserve) "preserve">

Description of the document schema in XSD language

There are different ways to link an XML document with its XSD schema: 1. submit files with the schema to the parser input.

2. set files with the scheme as a property of the analyzer.

3. specify directly in the XML document. Let's consider method 3 in more detail.

If the elements of the document do not belong to any namespace and are written with a prefix, then the attribute is written in the root element of the document

wit
hou
t

noNamespaceSchemaLocation, which specifies the location of the schema file in the form of a URI: < root_element_name xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance xsi:noNamespaceSchemaLocation="file_name.xsd">

In this case, the schema should not have a target namespace.

If the elements of the document belong to a certain namespace, then the schemaLocation attribute is used, in which the namespace and the location of the file with the schema describing this namespace are listed, separated by a space.

The elements that will make up the XML document are declared in the schema by the element component:

<xsd:element_name="element name" type="element type" minOccurs="Hafewest number of occurrences of the element in the document" maxOccurs="Maximum number of occurrences" />

The default value of the optional minOccurs and maxOccurs attributes is 1, which means that if these attributes are absent, the element must appear exactly once in the XML document.

Specifying the element type in the type attribute is convenient if it is a built-in simple type or a predefined type. Then only the type name can be written in the type attribute.

If the element type is defined here, then it is better to define the element type:

<xsd:element name=”element name”>

Defining the element type

</xsd:element>

Declaring an element attribute is also easy:

<xsd:attribute name="attribute name" type="attribute type" use="mandatory !

_____attribute" default="default value" />

The optional use attribute takes three values:

- optional — the attribute being described is optional (this is the default value);
- required — the described attribute is mandatory;

- prohibited — the attribute being described is not applicable. This value is useful when defining a subtype to override some attributes of the base type.

The definition of the attribute type, which must be a simple type, can be moved to the content of the attribute element:

<xsd:attribute name="attribute name">

Attribute type

</xsd:attribute>

Definition of simple types

A simple type in XML schemas is defined by the simpleType schema component, which has the form <xsd:simpleType name="type name”>Type definition</xsd:simpleType»

Defining new types of simple elements

In addition to the built-in types, new types of simple elements can be defined in XML schemas. They are introduced as

1. restriction of a built-in or previously defined simple type,
2. list of simple types
3. union of simple types.

*Simple type contraction*is defined by the componentrestriction, in which the base attribute specifies the simple type being narrowed, and the content specifies the restrictions that specify the simple type being defined.

Tags that define constraints are called facets. Here is a list of them:

1. <maxExclusive> — the largest value that is no longer included in the defined type
;
2. <maxInclusive> — the largest value of the defined type;
3. <minExclusive> — the smallest value that is no longer part of the defined type;
4. <minInclusive> — the smallest value of the defined type;
5. <totalDigits> — the total number of digits in the defined numeric type — contraction of the decimal type;
6. <fractionDigits> — the number of digits in the fractional part of the number;
7. <length> — length of values of the defined type;
8. <maxLength> — the largest length of values of the defined type;
9. <minLength> — the smallest length of values of the defined type;
10. <enumeration> — one of the enumerated values;
11. <pattern> — regular expression;

12. <whiteSpace> — is used when narrowing the string type and defines the method conversion of whitespace characters '\n', '\r', '\t'. The value attribute of this tag takes one of three values:

- preserve — do not remove whitespace characters;
- replace — replace space characters with spaces;
- collapse - after replacing space characters with spaces, remove the initial and trailing spaces, and from several consecutive spaces leave only one.

The following attributes, called fundamental facets, can be written in facet tags:

1. ordered— specifies the ordering of the defined type, takes one of three values:
   - false—type is unordered;
   - partial—the type is partially ordered;
   - total — the type is completely ordered;
2. bounded - specifies whether the type is bounded or unbounded with the value true or false;
3. cardinality - specifies the finiteness or infinity of a type with the value finite or countably infinite;
4. numeric - shows whether this type is numeric or not, with a value of true or false. *List* is defined by the componentlist, in which the itemType attribute specifies the type of elements of the defined list. The type of list elements can also be defined in the contents of the list element.

When defining a list, you can use the <length>, <minLength>, <maxLength>, <enumeration>, <pattern> facets.

*Simple union type*is defined by the componentunion, in which the memberTypes attribute can be used to specify the names of the types being united. For example:

<xsd: unionTypes="xsd: string xsd; integer lisstOfInteger" />

Another way is to write in the contents of the union component the definitions of the simple types included in the union.

Defining Complex Types

An element type is called complex if the element contains other elements and/or if the element's opening tag contains attributes.

A complex type is defined by the complexType component, which has the form:

<xsd:complexType name="type name" >

Type definition

</xsd:complexType>

The optional name attribute specifies the name of the type, and the content of the complexType component describes the elements that make up the complex type and/or the attributes of the opening tag.

The definition of a complex type can be divided into three groups:

*1.*defining the type of an empty element (an element that does not contain a body, but containing only attributes in the opening tag).

*Each attribute is declared by a single attribute component, for example:*

*<xsd:complexType name="imageType">*

*<xsd:attribute name="href" type="xsd:anyURI" /> </*

*xsd:complexType>*

2. definition of the type of an element with a simple body (an element containing a body simple type and attributes in the opening tag).

*This type differs from the simple type only by the presence of attributes and is defined by the simpleContent component. The body of this component must contain either the restriction component or the extension component. The base attribute specifies the type (simple) of the body of the element being described.*

*The extension component specifies the attributes of the opening tag of the element being described.*

*In addition to attributes, the restriction component describes the simple content type of the element and/or facets that restrict the type specified by the base attribute.*

3. defining the type of an element that contains nested elements.

*If the values of the defined complex type are elements containing nested elements, then before listing their descriptions, you must select a model group of nested elements. The point is that the nested elements that make up the defined type can appear either in a certain order or in an arbitrary order, and you can also select only one of the listed elements. This is called a model group of elements. It is determined by one of three components: sequence, all, or choice.*

*The sequence component is used when the listed elements must be written in a document in a specific order.*

*If you write the xsd:all component instead of the xsd:sequence component, the elements can be listed in any order. The choice component is used when you need to select one of several elements.*

4. Defining the type of an element with a complex body

*When defining a complex type, you can use an already defined, basic, complex type, expanding it with additional elements, or, conversely, deleting some elements from it. To do this, you need to use the complexContent component. In this component, as in the simpleContent component, either the extension component is written if you need to expand the base type, or the restriction component is written if you need to narrow it. The base type is specified by the base attribute, just like when writing the simpleContent component, but now it must be a complex type, not a simple one. When narrowing the base type with the restriction component, you must list the elements that will remain after narrowing.*

Transforming documents using XSLT

The XSLT transformation language is one of the implementations of XML. All XML elements declared in the XSLT language belong to the namespace http:// www.w3.org/1999/XSL/Transform. They are usually written with the prefix xsl. If this prefix is adopted, the root element of the XSLT document—the stylesheet—will be called xsl:stylesheet, which has one mandatory attribute version, indicating the version of the language.

The stylesheet can be written to a file with the xsl extension. A link to the stylesheet can be placed in the XML document as one of the processing instructions, namely the xml- stylesheet instruction. An example of such a link is given below.

```
<?xml version="1.0" encoding="windows-1251"
?> <? xml-stylesheet type="text/xsl"
href="simple.xsl"?>
<book>
<!-- contents -->
</book>
```

Upon "seeing" the xml-stylesheet processing instruction, the XML processor, if it is also an XSLT processor, will perform the transformation specified in the stylesheet.

The stylesheet can also be written not to a separate file, but directly to the converted XML document. For this purpose, the xsl:stylesheet element has an id identifier attribute, which can be referenced in the usual way from the xml-stylesheet processing instruction.

```
<?xml version="1.0" encoding="windows-1251"
?> <? xml-stylesheet type="text/xsl"
href="#simple"?>
<book>
<xsl:stylesheet version="1.0" id="simple"
```

xmlns:xsl="http://www.w3.org/1999/XSL/Transform"> <!-

- transformations -->

</xsl:stylesheet>

<!-- contents -->

</book>

Let's look at some elements of the XSLT language.

xsl:import declaration

The xsl:import element is written very simply:

<xsl:import href="URI address of the stylesheet" />

It can only be written directly in the root xsl:stylesheet element and only at the very beginning of the stylesheet. There can be several xsl:import elements.

The XSLT processor looks up the stylesheet at the address specified by the href attribute and substitutes it in place of the xsl:import element before transformation.

Some transformation rules from tables imported by xsl:import elements may conflict with rules imported from other tables or defined in the style sheet itself. In such cases, the rules that are written last are usually applied. Therefore, the order in which xsl:import elements are written in the style sheet is of great importance.

xsl:include declaration

The second element that includes external style sheets into the style sheet is the xsl:include element. It is written exactly like the xsl:import element and has the same effect:

<xsl:include href="URI address of stylesheet" />

It can be written not only directly in the root element xsl:stylesheet, but, unlike the xsl:import element, anywhere in the stylesheet. The second difference from the xsl:import element is that the order of writing xsl:include elements in the stylesheet does not matter.

xsl:variable declaration

The xsl:variable element defines the name of the object. It is written by the mandatory name attribute.

The object name must be a qualified XML name of the QName type. In addition, the select attribute of the variable can be used to specify the object itself, and the as attribute can be used to determine the type of the object. For example:

<xsl:variable name="var1" select="count(//person)" as="xs:integer" />

The name var1 will store the number of person elements. The object can be obtained from the contents of the xsl: variable element:

<xsl: variable name="var2">10</xsl:variable> or

created by the sequence constructor:

<xsl:variable name="var3">

<xsl:value-of select="count (//person) " />

<xsl:variable>

If the object is not obtained from the select attribute or the contents of an xsl:variable element, the name is bound to an empty string by default.

To get an object associated with a name defined by the xsl: variable element, you must put a dollar sign before the name: $varl, $var2. The scope of the name must be taken into account.

The scope of a name extends to the entire element in which it is defined, starting from where it is defined and including nested elements unless the same name is defined within them.

Names defined directly in the root xsl:stylesheet element are called global names, the rest are called local names.

Although the word "variable" is translated from English as "variable", the name created by the xsl:variable element is not the name of a variable, its value cannot be changed. It is only the name of some object, which is convenient to use in cases when the object needs to be used in many places in the style sheet, and its calculation is complex or cumbersome.

Declaration xsl:param

The xsl:param element is written either directly in the xsl:stylesheet element to specify a transformation parameter, or in the xsl:template element to specify a rule parameter, or in the xsl:function element as a function argument. It has one mandatory attribute, name, which defines the parameter name. In addition to it, there is often an optional attribute, select, in which an expression is written to obtain the parameter value:

<xsl:param name="pl" select="10 + 20" />

If the select attribute is absent, the value of the parameter is taken from the content of the element, which can be a constructor of a sequence of nodes and atomic values:

<xsl:param name="p2">10</xsl:param>

If both the select attribute and the element content are missing, the parameter is set to an empty string.

To get the value of a parameter, you need to write its name with a dollar sign:

&p1,&p2. For example:

<xsl:when test= "&p1=10">

The rules that determine the scope of parameters are the same as those for object names defined by the xsl:variable declaration.

Another optional attribute as contains the desired type to which the parameter value will be cast. Finally, the last attribute required, which takes the values yes or no (by default), indicates the mandatory nature of the parameter. If the parameter is mandatory, required="yes", then the xsl:param element must be empty and not contain the select attribute. In this case, it will receive a specific value when calling the function or the xsl:with-param element when calling the template.

xsl:with-param element

The xsl:with-param element refers to a parameter whose name is written in the mandatory name attribute. The optional select attribute can be used to specify an expression whose result will be the new value of the parameter:

<xsl:with-param name="pl" select="100 * 20" /> A new

value can also be specified in the element's content:

<xsl:with-param name="pl">100</xsl:with-param>

The xsl:with-param element is used only in xsl:apply-templates, xsl:apply-imports, xsl:call-template instructions.

xsl:value-of instruction

The xsl:value-of element evaluates the expression written in its mandatory select attribute and converts it to a string. For example, above we defined the object name var2. To get the value of the object var2, we would write:

<xsl:value-of select="$var2" />

If the expression evaluates to a sequence, the XSLT version 1.0 processor will select only the first element from the sequence, which is converted to a string.

xsl:if, xsl:for-each, xsl:choose control statements

The xsl:if element runs the sequence constructor contained in its body only if the expression written in the required and only test attribute is true:

<xsl:if test="$x > $y">

<xsl:value-of select="$x"•/> greater than <xsl:value-of

select="$y" /> </ xsl:if> '

The xsl:for-each element has an expression in its mandatory and only select attribute that results in a sequence. For each member of this sequence, the constructor contained in the body of the xsl:foreach element is executed. The result is a cycle that is executed as many times as there are elements in the sequence obtained as a result of evaluating the select attribute expression.

The xsl:choose element has no attributes, but its body contains one or more xsl:when elements and one optional xsl:otherwise element:

<xsl:choose>

<xsl when test="$day=l">Monday</xsl:when> <xsl

when test="$day=2">Tuesday</xsl:when> <xsl

when test="$day=3">Wednesday</xsl:when> <xsl

when test="$day=4">Thursday</xsl:when> <xsl

when test="$day=5">Friday</xsl:when> <xsl when

test="$day=6">Cy66ota</xsl:when> <xsl when

test="$day=7">Bocpece</xsl:when>

<xsl otherwise>Error determining day of week</xsl:otherwise> </

xsl:choose>

The xsl:when element has only one required parameter, test, which contains a logical expression, and a body, which contains a sequence constructor.

The xsl:otherwise element has no attributes, it only has a body containing a sequence constructor.

In the xsl:choose instruction, no more than one variant is ever executed. The xsl:when variants are examined in the order they are written in the xsl:choose instruction. As soon as a variant with a true value of the test expression is found, it is executed, and the result of this variant is the result of the entire instruction. The variants following it in order are not considered. If no variant matches, the result of the instruction is the result of the constructor written in the xsl:otherwise element. If there is no xsl:otherwise element in the instruction, the result is an empty string.

xsl:function declaration

The xsl:function element allows you to describe real user functions. The name of the function is written in the mandatory name attribute (the function arguments are specified by xsl:param elements, and the function body is a sequence constructor written in the contents of the xsl:function element. The result of the function will be a sequence created by the constructor. The function type can be specified by the optional as attribute.

The function name is a qualified name of type QName, and it must be written with a prefix.

All function arguments are positional, so all xsl:param elements must be written at the beginning of the function body, and the order in which they are written matters when calling the function. Function arguments cannot have default values, so xsl:param elements must be empty and must not have select attributes. For example:

<xsl:stylesheet version="2.0" xmlns:xsl="http://
www.w3.org/1999/XSL/Transform" xmlns:usr="http://
myexamples.com"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xsl: function name= " usr:sum " as="xsl:integer">
<xsl:param name="x" as="xsl:integer" />
<xsl:param name="y" as="xsl:integer" />
<xsl:value-of select="$x + $y" /> </
xsl:function>

You can call a function in any expression of the appropriate type by writing its name and arguments in brackets. For example:

<xsl:value-of select="10 + 2 * usr:sum(2, 3)" />

The function can be called recursively.

xsl:template declaration

The xsl:template element defines a template transformation rule. Its match attribute specifies a pattern for selecting nodes to be transformed, and its body contains a constructor for a sequence of nodes and atomic values that will be the result of transforming the nodes selected by the pattern.

<xsl: template match=" Sample" name=" Name"> Constructor</xsl:template> Each of the match and name attributes is optional, but at least one of them must be present.

The match attribute contains a pattern for selecting nodes to transform.

The name attribute specifies the name of the template. The template name is a regular XML qualified name of type QName. The template can be called by name using the xsl:call- template element, and if it does not contain the match attribute, such a call is mandatory, since the nodes to which it should be applied are unknown, and it will not be applied automatically. Very often, a named template contains parameters specified by xsl:param elements, and is called with different parameters just like a regular function.

The xsl:template element can have additional attributes mode, as, priority. The mode attribute specifies the processing mode.

The as attribute specifies the desired result type (the resulting sequence will be cast to this type).

The priority attribute assigns a priority to a rule, which will be taken into account when selecting rules applicable to a given node.

When calling a named xsl:call-template element template, the match, mode, and priority attributes are ignored.

If a named template does not have a match attribute, then it should not have mode and priority attributes either, they simply do not make any sense.

xsl:apply-templates instruction

The xsl:apply-templates element, most often written inside the xsl element: template, in its simplest form is empty:

<xsl:apply-templates/>

It specifies that all descendant nodes of the nodes selected by the parent xsl:template element are to be processed recursively.

The xsl:apply-templates element has optional attributes select and mode. The select attribute, whose value must be an expression yielding a sequence of nodes, limits processing to only the nodes specified in it.

The mode attribute selects a processing mode from the modes already defined in the xsl:template elements. A mode is any name of type QName, but two modes are predefined. There is the current mode, marked with the word #current, and a default mode, assumed if the mode attribute is absent or marked explicitly with the word # default.

The xsl:apply-templates element can contain xsl:sort and xsl:with-param elements.

xsl:for-each-group instruction

The xsl:for-each-group element selects a sequence of nodes and atomic values with its mandatory select attribute and groups its elements according to the attribute specified by the expression written in the group-by attribute. This attribute is called the group key. The group key is calculated anew for each element of the sequence. As a result, the original sequence selected by the select attribute is split into several sequences. In the following example, all name element nodes with the same value of the surname attribute are collected into one group:

<xsl:for-each-group select="name" group-by="@surname">

The expression that is the value of the group-by attribute can yield multiple group keys, and a single node can be in multiple groups at once. For example: <xsl:foreach-group select="name" group-by="(@second, @surname)">

The second way to split a sequence into groups is provided by the group- adjacent attribute. It collects into a group all consecutive elements of the sequence with the same key value. Such selection is possible if the group-adjacent attribute contains only one group key. The XSLT processor ensures that the value of the group-adjacent attribute is one and only one group key.

The third and fourth methods are applicable to sequences consisting only of nodes, without atomic values. These methods use the groupstartingwith attribute or the group-ending-with attribute. The value of these attributes may not be any expression, but only a pattern. All consecutive nodes are collected into one group, the first (last) of which satisfies the pattern. The remaining nodes in this group will not satisfy the pattern. If several consecutive elements of the sequence satisfy the pattern, they will be in different groups.

So, node groups are created by one of four attributes of the xsl:foreachgroup element: group-by, group-adjacent, group-starting-with, group-ending-with. Each xsl:for-each-group element must have one and only one of these attributes. The resulting groups exist and can be used only in the content of the xsl:for-each-group element. The body of the xsl:for-each-group element contains a sequence constructor that is executed once for each group.

For convenience of working with groups, the functions current-group() and current-group-key() have been introduced into the XSLT language. They can be used only in the body of the xsl:for-each- group element. Both functions have no arguments. The result of the current-group() function will be a sequence—the current group, and the result of the current-group-key() function will be the value of the key of the current group.

Sequence of transformations

Now that we have briefly described the basic elements of the XSLT language, we can understand how they perform transformations on an XML document.

An XSLT document is a stylesheet that consists of template transformation rules written by xsl:template elements. In each rule, the match attribute lists the sequence of source nodes that the rule is intended to transform, and the content of the xml:template element specifies the constructor of the sequence of transformed nodes. Each node of the source tree produces one or more nodes of the transformed tree.

The order in which the rules are written is not important.

because for each node in the source tree the entire style sheet is searched for a matching rule.

Applying the transformation rules

Typically, the viewing of the original XML document begins with its root node. A transformation rule is selected for it, the constructor of which explicitly or implicitly creates the root node of the transformed document. Then the constructor creates a sequence of nodes that will be descendants of the root node of the new document.

If there is no corresponding rule for a node in the document tree in the style sheet, then the default rule built into the XSLT processor is applied to it. The default rules are based on the node type.

For a root node and an element node, the default rule is to look at its descendants.

For an attribute node and a text node, the built-in default rule creates a text node containing their values. If the value of an attribute or text node is empty, the rule does nothing, or more precisely, creates an empty node sequence.

An empty node sequence is created for the comment node, processing instructions, and namespace node by default.

On the other hand, for a given source node, there may be several matching transformation rules in the stylesheet. One of them will always be the default rule. However, only one rule is ever applied to a node. It is selected as follows:

· firstly, from all imported rules the one that is written in is selected stylesheet last;

· secondly, the imported rule is discarded if there is a matching one rule in the style sheet itself;

· thirdly, the rule with the highest value of the priority attribute is selected. If If the priority attribute is missing, the rule is assigned a default priority, calculated based on the type of pattern written in the match attribute.

Creating Transformed Nodes

After the current node has been matched to a transformation rule, the constructor of the sequence of nodes and atomic values written in this rule is executed. The definition of the constructor is extremely general.

A sequence constructor is a sequence of style sheet nodes that share a common parent node, which when applied produces a sequence of nodes

and atomic values. The common parent node referred to in the definition is most often the xsl:template element node, although it can be xsl:variable, xsl:param, and many other XSLT elements.

There are four types of nodes that can appear in a sequence constructor:

· text nodes that will simply be copied into the constructed subsequence.

· XSLT instruction nodes,generating sequences of nodes and atoms values;

· external, non-XSLT instructions with names from a namespace other than XSLT namespaces. They also generate sequences of nodes and atomic values;

· elements from a namespace other than the XSLT namespace are not which are external instructions. They become nodes-elements of the constructed sequence.

The instruction nodes included in the constructor are executed and together with the other nodes of the constructor form one sequence of nodes and/or atomic values. All consecutive atomic values from the sequence created by the constructor are converted to strings and collected into one text node, in which they are written separated by a space. Then, in the resulting sequence of nodes, all consecutive text nodes are collected into one text node without any separators.

So, after applying each rule, we get a sequence of nodes, and after applying all the rules of the stylesheet to all selected nodes of the source tree, we get a set of such sequences. It remains to assemble one or more trees from this set of sequences, which will be the result of the transformation. They are created either explicitly by xsl:result- document elements, or implicitly by the initial transformation rule.

Very often the transformed tree is output to some device as an HTML, XHTML, XML document or simply as flat ASCII text. In the simplest case, serialization is performed, for which the XSLT language provides the xsl:output element.

Transforming Documents with DOM

The W3C DOM is a language- and platform-independent definition.

The W3C DOM model defines standard functionality for document navigation and manipulation of the content and structure of documents written in XML and HTML.

When using DOM to work with a text file in XML format, it parses the file, breaks it down into individual elements, attributes, comments, etc. It then creates a representation of the XML file in memory as a tree of nodes, in which each object in the document is treated as a node: elements, attributes, comments, processing commands, and even the plain text that makes up the attributes.

The developer can then access the document content using the node tree and make changes to it if necessary, for example to add a new element (to do this, simply create a new node and attach it as a child to the desired node).

The W3C DOM defines interfaces for the various DOM components to facilitate manipulation of the node tree, but no specific implementation is provided for these interfaces, and they can be implemented in any programming language.

The main types of interfaces that the object model supports are:

Document Interface

The Document node contains the following child nodes: Element (at most one), ProcessingInstruction (application command), Comment (comment), DocumentType (document type).

The document is a single entity. Methods for creating a document are not provided, this task is assigned to the user agent. The document object model does not provide a way to save the generated document, this task is assigned to the user agent. Most documents built with the help of a script disappear when the client browser window is closed due to the fact that client scripts used in browsers do not have access to the server's hard drive.

The value of the Name property of the Document interface is #document, and the value of the Value property of the Document interface is null.

A document can have only one Element node, namely the root node.

Node Interface

All DOM node types inherit a core set of properties from the Node interface.

NodeList Interface

The nodes of a NodeList can be accessed using the item method, which takes index as an argument. The length of the list can be determined using the length method.

The NodeList interface stores a record of a particular node's child nodes and their positions. It is the basis of the document tree structure in computer memory.

DOM in Internet Explorer 5 browsers

Internet Explorer 5 and above have built-in DOM libraries and XSL support. Many objects are available for client-side scripting to work with XML documents, the most important of which are the objectsXMLDOMDocument, XMLDOMNode, XMLDOMNodeList,XMLDOMParseErrorrepresenting the interface to access the entire document, its individual nodes and subtrees, providing the information necessary for debugging about the analyzer errors that have occurred, respectively.

The XMLDOMNode object implements the base DOM interfaceNode, is designed to manipulate a single node of the document tree. Its properties and methods allow you to obtain and change complete information about the current node
- its type (whether the current node is an element, comment, text, etc.), name, full name (with Namespace prefix), its contents, list of child elements, etc.

The XMLDOMDocument object represents the top level of the object hierarchy and contains methods for working with the document: loading it, analyzing it, creating elements, attributes, comments, etc. Many properties and methods of this object are also implemented in the Node class, since the document can be considered as a root node with subtrees nested within it.

The XMLDOMNodeList object is a list of nodes - subtrees and contains methods that can be used to organize the tree traversal procedure.

XMLDOMParserError object allows you to get all the necessary information about the error that occurred during document parsing. All properties of this object are read-only.

Creating an XML Document Using Microsoft Visual Studio .Net Microsoft Visual Studio .Net allows you to create XML documents and XSD schemas for them quite easily. In order to create an XML document, you need to execute the menu command in the Visual Studio .Net development environmentFile | New | File…in what appeared dialog boxNew Filein the setTemplatesselect iconXML FileAs a result, the following page will appear on the main workspace:

On this page you can type an XML document. Moreover, when writing an opening tag, the closing tag will appear automatically. Fill in

XML document with data can be done using the Data section. To do this, find the “Data” button at the bottom of the window. As a result, the page will look something like this:

The data entered in this way will be added to the XML document.

In order to write an XSD schema for the created XML document, you need to right-click on the workspace and select from the drop-down listCreate Schema.The result will be:

The type of elements and attributes can be selected from the drop-down

list. Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Study the XML language constructs, methods of describing the document structure using DTD, a description of the document schema in the XSD language. Learn how to process XML documents.

2. Write a statistical and dynamic model of XML document.

3. Describe the structure of a document using DTD.

4. Describe the document schema in XSD language

5. Transform the document using XSLT

6. Transform the document using DOM

7. Prepare a report including the statement of the problem, XML, DTD, XSD texts, XSLT documents, demonstration of the programs.

8. Defend laboratory work. Test questions:

1. What are markup languages. Stylistic, structural and semantic markings.
2. History of the development of markup languages: SGML, HTML, XML.
3. What is XML. How XML came into being. XML as data
4. XML document structure. Correctly formatted and valid documents.
5. XML parser. Event-driven parsers. Tree-like analyzers.
6. Why do we need a document type definition (DTD). General principles of writing DTD definitions:
7. Disadvantages and features of DTD definitions.
8. Data modeling using XML. Information model. Mapping to XML. Schemas.
9. Document Object Model (DOM).
10. Namespace and schemas.
11. Connections and requests.
12. Transformation of an XML document.
13. XML Document Design.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

LABORATORY WORK #13.Methods for creating web services. Subject - CMethods for creating web services.

The purpose of the work ismaster the methods of creationWeb services. The competencies being formed or their parts -OPK-1 Theoretical part.

In order to create the simplest Web service that will return the current date or a combination of date and time upon user request in the Visual Studio .NET development environment, you should execute the menu commandFile | New | Projectand in in the dialog box that appearsNew Projectin the setTemplatesselect icon ASP.NET Web Service. In the fieldNameyou should specify the name of the object being created project. After all the necessary preparations have been made by the development environment, two new pages will appear on the main workspace. One will be intended for developing the visual design, and the second will contain the code of our service. Naturally, the service being created cannot have an appearance as such, it has nothing to display, so the page for design can be safely closed.

A single project may include several separate services. For our example, we will need only one Web service, a template for which is created by the Visual Studio development environment. NET automatically, so we will not need to change anything, and we can immediately go to the page named Service1.asmx.vb. On this page, we need to place the code for our Web service:

```
Imports System.Web.Services
<WebService(Namespace := "http://tempuri.org/")>
_ Public Class Service1
Inherits System.Web.Services.WebService
# Region " Web Services Designer Generated
Code " Public Sub New()
MyBase.New()
'This call is required by the Web Services
Designer. InitializeComponent()
'Add your own initialization code after the InitializeComponent()
call End Sub
'Required by the Web Services Designer
Private components As System.ComponentModel.IContainer
'NOTE: The following procedure is required by the Web Services
Designer 'It can be modified using the Web Services Designer.
```

```
'Do not modify it using the code editor. <System.Diagnostics.DebuggerStepThrough()>
Private Sub InitializeComponent() components = New
System.ComponentModel.Container()

End Sub
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
'CODEGEN: This procedure is required by the Web Services Designer
'Do not modify it using the code editor.
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End
If
End
If
MyBase.Dispose(disposing)
End Sub
# End Region
<WebMethod()> Public Function MyDate(ByVal ShowTime As Boolean
) As String
Dim MD As
DateTime If
ShowTime Then
MyDate = MD.Now
Else
MyDate = MD.Today
End If
End Function
End of class
```

Of all this code, only a small part was written by the developer. This is the MyDate function, located at the end of the listing.

After compilation, the created Web service can be accessed even from a browser. The figure shows the appearance of the Web page that is generated by the Web server when attempting to access*To*Web service.

This web page contains a link to a formal description of the web service structure, and also lists all the functions supported by the web service. In our case, naturally, only one function, MyDate, is listed. The name of the function is also a hyperlink, clicking on which you can go to the web page that allows you to use this function. After clicking on the link in the browser

A web page of the following type will be displayed. This page contains an input field in which the user can specify the value of the parameter passed to the function.

As a result of the service function we get: ***<?xml***

***version="1.0" encoding="utf-8" ?>***

***<string xmlns="***http://tempuri.org/***">****11.08.2006 23:03:54****</* ***string***>

Client side

To create a client application, you must first create a new project. To do this, execute the menu commandFile | New | Project. In the dialog box that appearsNew Projectin the setTemplatesselect templateWindows Applicationand in the fieldNameSpecify the name of the application being

created. After this, Visual Studio .NET will create a form with the default name Form1. In this form window, we will need to place one Checkbox element, one Label text field, and one Button button. Using the independent switch CheckBox1, we will set the parameter passed to the MyDate function, which is part of the service. The text field will contain the result returned by this function, and the button will start the process of establishing a connection with the Web service and receiving the required data from it.

After this, you should establish a connection with the desired Web service. To do this, in the windowSolution Explorerselect the name of the application you are creating and right-click by clicking the mouse you can call up a context menu for it. In this context menu you need to execute the commandAdd Web Reference, after which the dialog box of the same name will be activated, allowing you to set links to the services used.

To get links to Web service functions and transfer them to the project, you need to enter the following in the text field:AddressspecifyURL of the required service and download the resource located at this address. After that, in the left part of the desired windowAdd Web Referencethe contents of the start page will be displayedWeb pages.

Once the required links to the Web service functions have been found, use the buttonAdd Referencethey need to be added to the project under development.

After which you can access the Web service functions in the application (Button_Click function).

Public Class Form1

Inherits System.Windows.Forms.Form

# Region "Windows Form Designer generated code"

```
Public Sub New()
MyBase.New()
'This call is required by the Windows Form
Designer. InitializeComponent()
'Add any initialization after the InitializeComponent()
call End Sub
'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean) If
disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End
If
End
If
MyBase.Dispose(disposing)
End Sub
'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer
'NOTE: The following procedure is required by the Windows Form
Designer 'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents CheckBox1 As
System.Windows.Forms.CheckBox Friend WithEvents Label1 As
System.Windows.Forms.Label
Friend WithEvents Button1 As System.Windows.Forms.Button
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
Me.CheckBox1 = New System.Windows.Forms.CheckBox()
Me.Label1 = New System.Windows.Forms.Label()
Me.Button1 = New System.Windows.Forms.Button()
Me.SuspendLayout()
'CheckBox1
'
Me.CheckBox1.Location = New System.Drawing.Point(16, 16)
Me.CheckBox1.Name = "CheckBox1"
Me.CheckBox1.Size = New System.Drawing.Size(168, 24)
Me.CheckBox1.TabIndex = 0
Me.CheckBox1.Text = "Show time"
```

```
        '
        'Label1
        '
        Me.Label1.Location = New
        System.Drawing.Point(16, 64) Me.Label1.Name =
        "Label1"
        Me.Label1.Size = New System.Drawing.Size(264, 23)
        Me.Label1.TabIndex = 1
        '
        'Button1
        '
        Me.Button1.Location = New System.Drawing.Point(16, 128)
        Me.Button1.Name = "Button1"
        Me.Button1.Size = New System.Drawing.Size(256, 32)
        Me.Button1.TabIndex = 2
        Me.Button1.Text = "Launch Web Service" '

        40
        'Form1
        '
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(292, 273)
        Me.Controls.AddRange(New                                    {Me.Button1,
        System.Windows.Forms.Control()
Me.Label1, Me.CheckBox1})
        Me.Name = "Form1"
        Me.Text = "Form1"
        Me.ResumeLayout(Fals
        e)
        End Sub
        # End Region
        Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
        Button1.Click
        Dim t1 = New WindowsApplication1.localhost.Service1()
        Label1.Text = t1.MyDate(CheckBox1.Checked)
        End Sub
        End of class
```

Once launched, the application will look like this: XML document as a data source

Let's create a Web service that will include a single function that returns a regular DataSet containing information taken from an XML file.

To do this, create a new Web service project. Drag one DataSet component from the Data tab to its page. This will activate the dialog box AddDataset,in which the developer is asked to specify the type of set being created data. Since in our case the XML file can contain data of any structure, the structure cannot be specified in advance for the data set. Therefore, in this dialog box the developer must select the radio buttonUntyped dataset.

The web service code is shown below.

```
ImportsSystem.Web.Services
<WebService(Namespace:=
"http://tempuri.org/")> _ Public ClassService1
InheritsSystem.Web.Services.WebService
#Region"Web Services Designer Generated Code"
Public Sub New() MyBase.New()

'This call is required by the Web Services
Designer. InitializeComponent()
'Add your own initialization code after the InitializeComponent()
call End Sub
'Required by the Web Services Designer
PrivatecomponentsAsSystem.ComponentModel.IContain
er
'NOTE: The following procedure is required by the Web Services
Designer 'It can be modified using the Web Services Designer.
'Do not modify it using the code editor.
Friend With EventsDataSet1AsSystem.Data.DataSet
<System.Diagnostics.DebuggerStepThrough()>Private SubInitializeComponent() Me
.DataSet1 =NewSystem.Data.DataSet()
CType(Me.DataSet1, System.ComponentModel.ISupportInitialize).BeginInit() '

'DataSet1
'
```

```
Me.DataSet1.DataSetName = "NewDataSet"
Me.DataSet1.Locale =NewSystem.Globalization.CultureInfo("ru-RU") CType(Me
.DataSet1, System.ComponentModel.ISupportInitialize).EndInit() End Sub

Protected Overloads Overrides SubDispose(ByValdisposingAs Boolean)
'CODEGEN: This procedure is required by the Web Services Designer
'Do not modify it using the code editor.
IfdisposingThen
If Not(componentsIs Nothing)Then
components.Dispose()
End
If
End
If
MyBase.Dispose(disposing)
End Sub
#End Region
<WebMethod()>Public FunctionXMLData(ByValfileAs String)AsDataSet
DimFSAsIO.FileStream DimReaderAsIO.StreamReader

FS =NewIO.FileStream(Server.MapPath(file),                IO.FileMode.Open,
IO.FileAccess.Read)
Reader =NewIO.StreamReader(FS)
DataSet1.ReadXml(Reader)
FS.Close()
XMLData = DataSet1
End Function
End of class
```

If the content of the XML file used in our example looks like this:

```
<Document>
<Row>
<Columnl>l</Columnl>
<Column2>2</Column2>
<Column3>Textl</Column3>
<Column4>Text2</Column4> </
Row>
```

<Row>
<Columnl>3</Columnl>
<Column2>4</Column2>
<Column3>Text3</Column3>
<Column4>Text4</Column4> </
Row>
</Document>,
then as a result of the work of this Web service we will
receive: <? xml version="1.0" encoding="utf-8" ?>
- <DataSetxmlns="http://tempuri.org/">
- <xs:schema id="Document" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema
"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
- <xs:element name="Document"msdata:IsDataSet="true"msdata:Locale="ru-RU">
- <xs:complexType>
- <xs:choice maxOccurs="unbounded">
- <xs:element name="Row">
- <xs:complexType>
- <xs:sequence>
<xs:element name="Column"type="xs:string"
minOccurs="0" />
<xs:element name="Column2"type="xs:string"
minOccurs="0" />
<xs:element name="Column3"type="xs:string"
minOccurs="0" />
<xs:element name="Column4"type="xs:string"
minOccurs="0" /> </xs:sequence>

</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
- <diffgr:diffgramxmlns:msdata="urn:schemas-microsoft-com:xml-msdata"

```
xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
- <Documentxmlns="">
- <Row
diffgr:id="Row1"msdata:rowOrder="0"diffgr:hasChanges="inserted"> <
Column>l</Column> <Column2>2</Column2>
<Column3>Textl</Column3> < Column4>Text2</Column4> </Row>

- <Row
diffgr:id="Row2"msdata:rowOrder="1"diffgr:hasChanges="inserted">      <
Column>3</Column>                                    <Column2>4</Column2>
<Column3>Text3</Column3> < Column4>Text4</Column4> </Row>

</Document>
</diffgr:diffgram>
</DataSet>
```

To create a client application, you first need to create a new project. To do this, as in the previous example, you should execute the menu commandFile | New | Project . In the dialog box that appearsNew Projectin the setTemplateschoose templateWindows Applicationand in the fieldNamespecify the name of the created applications.

After this, Visual Studio .NET will create a form with the default name Form1. In this form window, we will need to place one TextBox element, one Label text field, one Button, one DataGrid component, and one DataSet component from the Data tab. An example of the form is shown below.

After this, you should establish a connection with the desired Web service. To do this, in the windowSolution Explorerselect the name of the application you are creating and right-click by clicking the mouse you can call up a context menu for it. In this context menu you need to execute the commandAdd Web Reference, after which the dialog box of the same name will be activated, allowing you to set links to the services used.

To get links to Web service functions and transfer them to the project, you need to enter the following in the text field:AddressspecifyURL of the required service and download the resource,

located at this address. After that, in the left part of the search windowAdd Web Referencethe contents of the start page will be displayedWeb pages.

Once the required links to the Web service functions have been found, use the buttonAdd Referencethey need to be added to the project under development.

After which you can access the Web service functions in the application (Button_Click function).

```
Public Class Form1
Inherits System.Windows.Forms.Form
# Region " Windows Form Designer generated
code " Public Sub New()
MyBase.New()
'This call is required by the Windows Form
Designer. InitializeComponent()
'Add any initialization after the InitializeComponent()
call End Sub
'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean) If
disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End
If
End
If
MyBase.Dispose(disposing)
End Sub
'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer
'NOTE: The following procedure is required by the Windows Form
Designer 'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
Friend WithEvents Label1 As System.Windows.Forms.Label Friend
WithEvents Button1 As System.Windows.Forms.Button Friend
WithEvents DataGrid1 As System.Windows.Forms.DataGrid Friend
WithEvents DataSet1 As System.Data.DataSet
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
```

```
Me.TextBox1 = New
System.Windows.Forms.TextBox() Me.Label1 = New
System.Windows.Forms.Label() Me.Button1 = New
System.Windows.Forms.Button() Me.DataGrid1 =
New System.Windows.Forms.DataGrid()
Me.DataSet1 = New System.Data.DataSet()
CType(Me.DataGrid1, System.ComponentModel.ISupportInitialize).BeginInit()
CType(Me.DataSet1, System.ComponentModel.ISupportInitialize).BeginInit()
Me.SuspendLayout()
'
'TextBox1
'
Me.TextBox1.Location = New System.Drawing.Point(56, 24)
Me.TextBox1.Name = "TextBox1"
Me.TextBox1.Size = New System.Drawing.Size(224, 20)
Me.TextBox1.TabIndex = 0
Me.TextBox1.Text = "TextBox1" '

'Label1
'
Me.Label1.Location = New System.Drawing.Point(8, 24)
Me.Label1.Name = "Label1"
Me.Label1.Size = New System.Drawing.Size(48, 23)
Me.Label1.TabIndex = 1
Me.Label1.Text = "File" '

'Button1
'
Me.Button1.Location = New System.Drawing.Point(16, 56)
Me.Button1.Name = "Button1"
Me.Button1.Size = New System.Drawing.Size(264, 24)
Me.Button1.TabIndex = 2
Me.Button1.Text = "Get data" '

'DataGrid1
'
```

```
Me.DataGrid1.DataMember = "" Me.DataGrid1.DataSource = Me.DataSet1
Me.DataGrid1.HeaderForeColor = System.Drawing.SystemColors.ControlText
Me.DataGrid1.Location = New System.Drawing.Point(16, 88)

Me.DataGrid1.Name = "DataGrid1" Me.DataGrid1.Size =
New System.Drawing.Size(264, 80)
Me.DataGrid1.TabIndex
= 3
'
'DataSet1
'
Me.DataSet1.DataSetName = "NewDataSet"
Me.DataSet1.Locale = New System.Globalization.CultureInfo("ru-RU") '

'Form1
'
Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(292, 273)
Me.Controls.AddRange(New System.Windows.Forms.Control()          {Me.DataGrid1,
Me.Button1,
Me.Label1, Me.TextBox1})
Me.Name = "Form1"
Me.Text = "Form1"
CType(Me.DataGrid1,
System.ComponentModel.ISupportInitialize).EndInit()
CType(Me.DataSet1,
System.ComponentModel.ISupportInitialize).EndInit()
Me.ResumeLayout(False)
End Sub
# End Region
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
Button1.Click
Dim xml = New WindowsApplication2.localhost.Service1()
DataSet1.Merge(xml.XMLData(TextBox1.Text))
End Sub
End of class
```

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Learn the basic principles of creationWeb services.
2. Createweb service that will use the XML document as

data source. That is, to create a Web service that returns a regular data set containing information taken from an XML file.

3. Supplement the created oneweb service procedure that allows you to save data in XML file.
4. Create a client application.
5. Prepare a report including the statement of the problem, the text of the programs, demonstration of the programs' operation.
6. Defend laboratory work Test

questions:

1. Web services. Basic concepts. Purpose.
2. History of Web services development. COM/DCOM model. CORBA/IIOP standard. Disadvantages of the listed technologies.
3. .Net Web Services. Advantages. Architecture.
4. Basic web services technologies: SOAP, UDDI, WSDL.

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and Computer Science. Bachelor's degree) - Neck: Add. UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: With. http://biblioclub.ru/index.php?page=book&id=231519

# LABORATORY WORK #14.Creation and validation technologiesXML-documents

Subject -Creation and validation technologiesXML documents

The purpose of the work ismaster the technologies of creation and validationXML documents. The competencies being formed or their parts -OPK-5 Theoretical part.

1. Creating XML documents using Microsoft Visual Web Developer *Methodology of implementation*

Create a new XML document

Start | Programs | Microsoft Visual Web Developer 2008 Express Edition File → New File… → New File f. | select Xml File → Open tab

Define the processing instruction (if the processing instruction already exists) is set, then skip this item):

f. New File.xml | Enter the line of code: <?xml version="1.0" encoding="utf-8"?> go to new line

Set the root element root

f. New File.xml | Enter an opening angle bracket (<); Specify the element name (root); Enter a closing angle bracket (>) → the closing tag of the root element will be automatically generated

1.4 Create a first level nested element test1

f. New File.xml | between the opening and closing tags of the root element, specify the corresponding tags of the test1 element (the document will be formatted automatically)
Set attribute attr1 with value value1

f. New File.xml | place the cursor in the opening tag of the test1 element after the element name before the closing angle bracket; enter a space; enter the attribute name attr1; enter the symbol "=" → double quotes will be automatically generated to represent the attribute value; at the current cursor position, enter the text "value1".

Similarly, introduce a nested element of the second level test1-1 with attribute attr1-1 with value value1-1 (test1-1 is a child element of the test1 element)
Save the document.

*Task to complete*

According to the given scheme (Fig. 1) and variant, construct an XML document. For each element of the first level, provide at least 3 copies, for each element of subsequent levels – at least two copies.

Option A. Student performance

Option B. Bookstore Option C.

Cinema

Option D. Airport

Option D. Human Resources Department

2. Checking the correctness of XML documents using a browser

*Methodology of implementation*

Open the XML document in the browser. If the document is correct, the window will show The browser will display a hierarchical XML structure.

Make the document incorrect according to the basic requirements of correctness.

Copy the root element root along with its contents to the end document (as a result, there should be two consecutive root elements). After opening the document in the browser, you will receive an error message: "Only one top-level element is allowed in XML documents.»

Replace the name of the root element only in the opening tag with Root. After opening the document in the browser, you will receive an error message: "The end tag "root" does not match the start tag "Root".»

Move the closing tag of the test1-1 element after the closing tag element test1. After opening the document in the browser, you will receive an error message: "End tag "test1" does not match start tag "test1- 1".»

Undo all changes made so that the document becomes correct again. *Task to complete*

Include in the report messages about document inaccuracies in a manner similar to the implementation methodology.

3. Creating an XSD schema of an XML document using the Microsoft Visual environment Web Developer

*Methodology of implementation*

Create a new XSD schema

Start | Programs | Microsoft Visual Web Developer 2008 Express Edition File → New File… → New File f. | select Xml Schema → Open tab

Change prefix from xs to xsd:

f. XMLSchema1.xsd | in the opening and closing tags of the schema element, replace xs:schema with xsd:schema; in the line

xmlns:xs="http://www.w3.org/2001/XMLSchema" replace xs with xsd

Remove the targetNamespace, elementFormDefault, xmlns, xmlns:mstns attributes along with their values.

Define the description of the root element root

f. XMLSchema1.xsd | after the closing angle bracket of the opening xsd:schema tag, go to a new line and enter the opening angle bracket ("<") → from the list that opens, select the value "xsd:element" and press [Enter] → enter a space → from the list that opens, select the value "name" and press [Enter] → the name attribute with double quotes will be automatically created for the value
→ at the current cursor position (in quotation marks) enter the value "root" → Enter a closing angle bracket (>) → the closing tag of the xsd:element element will be automatically generated

Define the structure of the root element (complexType with elements / sequence)

f. XMLSchema1.xsd | between the opening and closing tags of the xsd:element element, enter an opening angle bracket ("<") → from the list that opens, select the value "xsd:complexType" and press [Enter] → Enter a closing angle bracket (>) → the closing tag of the xsd:complexType element will be automatically generated. Similarly, within the xsd:complexType element, create an xsd:sequence element.

3.4. Create a description of the nested element of the first level test1 (complex / complexType with elements / sequence)

f. New File.xml | between the opening and closing tags of the xsd:sequence element, create an xsd:element element with the name property value equal to "test1". Within the new xsd:element element, define the xsd:complexType and xsd:sequence elements.

By analogy with paragraph 3.4, define an element (xsd:element) with the name test1-1 between
opening and closing tags of the xsd:sequence element.

Create a description of the attr1 attribute

f. New File.xml | place cursor after closing tag

xsd:sequence before the closing xsd:complexType tag from section 3.4; enter an opening angle bracket (“<”) → from the list that opens, select the value “xsd:attribute” and press [Enter] → put a space → from the list that opens

select the value "name" and press [Enter] → the name attribute with double quotes will be automatically created for the value → at the current cursor position (in quotes) enter the value "attr1" → Similarly, specify the type parameter with the value xsd:string
→ Enter a closing angle bracket (>) → the closing tag of the xsd:attribute element will be automatically generated

Similarly, create a description of the attr1-1 attribute.

Save XSD schema.

*Task to complete*

Create an XSD schema according to your version of the XML document.

4. Checking the validity of an XML document using JavaScipt

Open the XML document created in step 1.

Enter a reference to the schema in the root element:

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="*Scheme name*.xsd" 4.3. Create a text document.

Enter the following code into the document:

```
var sOutput = validateFile("XML file name.xml");
WScript.Echo(sOutput);
function validateFile(strFile) {

var x = new ActiveXObject("MSXML2.DOMDocument.4.0"); x.async = false;
x.validateOnParse = true; x.resolveExternals = true; x.load(strFile);
if (x.parseError.errorCode != 0) {

return("NOT VALID document " + strFile + "\n======================" +
"\nReason: " + x.parseError.reason +
"\nSource: " + x.parseError.srcText +
"\nLine: " + x.parseError.line + "\n");
}

else
return("Valid document " + strFile +
```

"\n====================\n" +

x.xml + "\n");

}

Save the document as validate.js.

Run the validate.js script.

Equipment and materials

To perform the laboratory work, it is recommended to use a personal computer with the following characteristics: 32-bit (x86) or 64-bit (x64) processor with a clock frequency of 1 GHz or higher, RAM - 1 GB or higher, free disk space - at least 1 GB, graphics device DirectX 9. Software: operating system WINDOWS 7 and higher, MongoDB 2.4.3.

Safety instructions

Safety precautions when performing laboratory work coincide with those generally accepted for users of personal computers. Do not repair the personal computer, install or remove software on your own; in case of a personal computer malfunction, inform the laboratory service personnel (operator, administrator); observe safety precautions when working with electrical equipment; do not touch electrical outlets with metal objects; the user's workplace of the personal computer must be kept clean; it is not allowed to eat or drink near the personal computer.

Contents of the report

1. Learn the basic principles of creationWeb services.

2. Createweb service that will use the XML document as data source. That is, to create a Web service that returns a regular data set containing information taken from an XML file.

3. Supplement the created oneweb service procedure that allows you to save data in XML file.

4. Create a client application.

5. Prepare a report including the statement of the problem, the text of the programs, demonstration of the programs' operation.

6. Defend laboratory work

A list of literature recommended for use on this topic

1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., rev. - Moscow: Academy, 2012. - 314, [1] p. : ill. ; 22. - (Higher

professional education. Computer science and Grif: Computer Science. Bachelor's degree) - Additional UMO. - 1000 copies. - ISBN 978-5-7695-9308-6.

2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL:

With.

http://biblioclub.ru/index.php?page=book&id=231519

RECOMMENDED REFERENCES


1. Kuzin, Alexander Vladimirovich. Databases: a textbook for universities / A.V. Kuzin, S.V. Levonisova. - 5th ed., corrected. - Moscow: Academy, 2012. - 314, [1] p.: ill.; 22. - (Higher professional education. Computer science and computing technology. Bachelor's degree) - Stamp: Additional UMO. - 1000 copies - ISBN 978-5-7695-9308-6.
2. Shnyrev, S.L. Databases: a tutorial / S.L. Shnyrev. - M.: MEPhI, 2011. - 224 p. - ISBN 978-5-7262-1483-2 ; Ditto [Electronic resource]. - URL: http://biblioclub.ru/index.php?page=book&id=231519
3. Partitioned tables and indexes of SQL Server 2005, Based on the article Kimberly L. Tripp: SQL Server 2005. Partitioned Tables and Indexes, Translation by Alexey Safonov, http://www.sql.ru/articles/mssql/2005/073102PartitionedTablesAndIndexes.shtml
4. Microsoft MSDN,http://msdn.microsoft.com/ru-ru/
5. Bratchenko, N. Yu. (NorthCaucasian State Technical University). Databases: a tutorial / N. Yu. Bratchenko ; North Caucasus state tech. univ. - Stavropol: SevKavGTU, 2011. - 195 p. - Bibliography: p. 194
6. Bratchenko, N. Yu. (North Caucasus State Technical University). Databases. Basics of Working with DBMS ORACLE: study guide (laboratory practical training) / N. Yu. Bratchenko; FGBOU North-Caucasian State Technical University. - Stavropol: Publishing House of North-Caucasian State Technical University, 2011. - 129 p.: ill. - Bibliography: pp. 128-129
7. Characteristics of database administration tools and their prospects development [Electronic resource] / N. I. Kusyutin. - M.: Laboratory of the book, 2011. - 99 p. - Access mode:
8. Ilyushechkin V. M. Fundamentals of using and designing databases. Textbook [Electronic resource] / V. M. Ilyushechkin. - M.: YURAYT, 2011. - 214 p.
9. Sovetov, B. Ya. Databases: theory and practice: textbook / B. Ya. Sovetov, V.V. Tsekhanovsky, V.D. Chertovskoy. - 2nd ed. - M.: Yurait, 2012. - 464 p. - (Bachelor). - Stamp: Rec. UMO. - Bibliography: pp. 459-460. - ISBN 978-5-9916-1479-
10. ShashaD., BonneF., Database Optimization. Principles, Practice, Solution problems, M.:KUDITS-Image, 2004.

MINISTRY OF SCIENCE AND HIGHER EDUCATION
OF THE RUSSIAN FEDERATION
FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION OF HIGHER EDUCATION
"NORTH CAUCASUS FEDERAL UNIVERSITY"

# METHODOLOGICAL INSTRUCTIONS

on organizing independent work on the subject "Database tuning" for master's students in the field 09.04.03 "Applied informatics" focus (profile) "Management knowledge"

Stavropol
2022

# Introduction

The discipline "Database Tuning" develops the professional skills of a master in the field of training 09.04.03 "Applied Informatics".

The study of the discipline "Tuning Databases" is based on laboratory work and individual lessons under the guidance of a teacher, independent work of students

The objectives of the course are:

- development of logical and algorithmic thinking; study of the
- principles of operation of software and hardware tools and

organization of data in information systems using databases;

- mastering work with modern DBMS;
- development of the ability to independently solve processing problems

text and non-text information in the database;

- gaining skills in problem algorithmization, programming

in algorithmic language, debugging and executing tasks on a personal computer;

- study of the prospects for the development of information technologies in

information systems in the subject area;

- study of information resource markets and their features

use.

1. General characteristics of independent work of a student studying the disciplineDatabase tuning – formation of a set professional competencies of the future master's degree student in the field of study

09.04.03 "Applied Informatics".

2. Objectives of mastering the discipline:

- development of logical and algorithmic thinking;
- study of the principles of operation of software and hardware tools and organization of data in information systems using databases;
- mastering work with modern DBMS;
- development of the ability to independently solve text processing problems and non-textual information in the database;
- acquiring skills in problem algorithmization, programming algorithmic language, debugging and executing tasks on a personal computer;

- studyingprospects for the development of information technology in information systems in the subject area;
- study of information resource markets and their features use.

3. Formed competencies- OPK-5

4. Schedule for completing independent work

| No. p/p | Chapter disciplines | Week semesters | SRS in hours | Forms of the current control academic performance |
|---|---|---|---|---|
| 1 | Lab 1. Index structures | 1. | 3 | Interviews on laboratory work |
| 2 | Lab 2. Connecting data tables | 2. | 3 | Interviews on laboratory work |
| 3 | Lab 3. Table Join Algorithms | 3. | 3 | Interviews on laboratory work |
| 4 | Lab 4. Queries with subqueries | 4. | 3 | Interviews on laboratory work |
| 5 | Lab 5. Query Execution Plans | 5. | 3 | Interviews on laboratory work |
| 6 | Lab 6. Transactions | 6. | 3 | Interviews on laboratory work |
| 7 | Lab 7. Working with XML. Part 1 | 7. | 3 | Interviews on laboratory work |
| 8 | Lab 8. Working with XML. Part 2 | 8. | 3 | Interviews on laboratory work |
| 9 | Lab 9. Hierarchically organized data | 9. | 3 | Interviews on laboratory |

| | | | | |
|---|---|---|---|---|
| | | | | work |
| 10 | Lab #10. Table Partitioning | 10. | 3 | Interviews on laboratory work |
| 11 | Laboratory work No. 11. Principles of constructing statistical and dynamic model | 11. | 3 | Interviews on laboratory work |
| 12 | Lab #12. XML language constructs | 12. | 3 | Interviews on laboratory work |
| 13 | Laboratory work No. 13.WITHWays to create web services | 13. | 2.5 | Interviews on laboratory work |
| 14 | Lab #14. Technologies for creating and validating XML documents | 14. | 2 | Interviews on laboratory work |
| | | TOTAL | 40.5 | |

5. Methodological recommendations for studying theoretical material

At the first stage, it is necessary to familiarize yourself with the working program of the discipline, which examines the content of the topics of the lecture course, the relationship of lecture topics with practical classes, topics and types of independent work. For each type of independent work, certain reporting forms are provided.

Lab #1 Index structures

7. Check for indexes on key fields of the table.

Create clustered indexes if necessary. To create a new index, use the CREATE INDEX command, or in Microsoft SQL Management Studio, under Tables/table_name/Indexes, use the New Index… command.

8. Create nonclustered indexes on foreign key fields database tables. Explain why such indexes are needed?

9. Create non-clustered indexes on the information fields:

Name and Date in all tables of the database. Explain why such indexes are needed?

10. For the clustered index and the index on the Date field of the records table in check, get information about the extended properties of indexes. Explain the meaning of the information provided in the "Fragmentation" section of the "Properties" page. Explain how the index tree depth, number of leaves, and fragmentation factor are calculated.

11. Rebuild the clustered index on the BillItem table using the ALTER INDEX command or by using the Rebuild command in the index context menu.

12. Prepare material for inclusion in the final presentation on course "Database Tuning".

Lab #2 Connecting Data Tables

1. Filter by a single product name. Use a single filter condition "=". Select the product name yourself.
2. Filter by multiple product names. Use the condition filtering in. Select the list of products yourself.
3. Filter by a single date. Use a single condition filtering "=". Select the check date yourself.
4. Filter with range limitation by receipt dates. Use filter by range between or together "<=", "=>". Choose the date range yourself.
5. Filter by half-open check date range. Use filtering by the condition "<=" or "=>". Select the date range yourself.

Lab #3 Table Join Algorithms

1. Queries created in points 1,2 of the previous laboratory Rework the work so that the nested loop join, hash join, and merge join algorithms are used to join tables.
2. We estimate the execution time of each of the received requests.
3. Save the received queries in views, adding to name of the original representation name of the algorithm used.
4. Prepare material for inclusion in the final presentation on course "Database Tuning".

Lab #4 Queries with Subqueries

1. Select all checks for a specific date that are not unique and there are other checks for this date. To form a query, use the EXISTS operator.
2. Select all customers who did not make a purchase on a specific date. Use NOT EXISTS operators to form a query.
3. Create a temporary table (CREATE TABLE command) or a table type variable (DECLARE TABLE command) with the same structure as the customer table, to which you add information about five random customers (use the INSERT operator). Select information about the 5 added customers and their purchase dates. Use a connection of a temporary table (a table type variable) with the Bill receipts table. Use the IN operator to form a query.
4. Save the received requests.
5. Prepare material for inclusion in the final presentation on course "Database Tuning".

Lab #5 Query Execution Plans

1. For the queries created in points 1 and 2 of Lab 2 and
queries created in point 1, lab 3, generate a graphical and textual plan for executing queries. Study the execution plans and explain the order of joining tables.

2. Estimate the complexity of individual nodes in the query plan tree. Select the most labor-intensive operations, study them and explain why these operations are the most labor-intensive compared to the others.

3. Estimate the volume of data transferred between plan operators
execution of the request.

4. Prepare material for inclusion in the final presentation on
course "Database Tuning".

Lab #6 Transactions

1. Add a Quantity field to the Product table. Fill the
field with arbitrary values from 0 to 100.

2. Create a stored procedure AddBill that will generate
check, and then return its identifier. The input parameters of the procedure are BuyerID and EmployeeID. All statements of this stored procedure must be executed in a transaction.

3. Create a stored procedure for adding a product to a receipt. Name it its AddBillItem. Input parameters of the procedure: BillID, ProductID, Quantity. Set the transaction isolation level to SERIALIZABLE mode. After opening the transaction (BEGIN TRANSACTION), it is necessary to check the quantity of goods in the store. If there is not enough goods in the store, the procedure must complete the transaction (ROLLBACK TRANSACTION). Otherwise, the stored procedure reduces the available quantity of goods in the store and adds a purchase record to the receipt (records are added to the BillItem table). Close the transaction (COMMIT TRANSACTION). Explain the choice of transaction isolation level.

4. Make a selection of the check contents using query #1 from laboratory
work #2.

5. Check the functionality of the written stored procedures.
Try creating a receipt and adding 2 items with sufficient quantity in the store and 1 item with insufficient quantity. Make sure that the receipt only shows 2 sold items.

6. Prepare material for inclusion in the final presentation on
course "Database Tuning".

Lab #7 Working with XML. Part 1

7. Create a new stored procedure AddBillXML. Input
The parameter of this stored procedure will be an XML document, an example of which is shown below.

```
<root>
<bill billId="1473">
<product productId="1" quantity="5"/>
```

<product productId="23" quantity="1"/>
<product productId="7" quantity="34"/> </ bill>
</root>

8. Using sp_xml_preparedocument, sp_xml_removedocument and OPENXML parse the XML document, selecting the fields productId and quantity for the receipt record. Create a flat temporary table with the contents of the XML document.

9. After parsing the XML document and forming a flat table open a transaction to generate a new receipt. Create a new receipt (use the AddBill stored procedure from worksheet #6). Add to the created receipt using the INSERT…SELECT statement only those products that are in sufficient quantity. Using the UPDATE statement, update the Quantity field of the Product table, subtracting the products that were purchased. Choose the transaction isolation level yourself and comment on your choice.

10. Prepare material for inclusion in the final presentation on course "Database Tuning".

Lab #8 Working with XML. Part 2

1. Write a query that returns the check number, the total amount check, employee's full name, check date; information about each product, name, price, quantity, cost. Return the result as XML, presented in the previous lab work.

2. Prepare material for inclusion in the final presentation on course "Database Tuning".

Lab #9 Hierarchically Organized Data

1. Add a manager column to the employees table (foreign key to the employee code). Add data to the table so that all existing employees are subordinate to 2 new managers (each seller to only one of his own managers), and the 2 new managers in turn are subordinate to the store director.

2. Add 2 new columns to the table (right key, left key). Write a query to calculate keys and enter keys using Joe Selko's method.

3. Add a new column (materialized path) to table 1. Write a query to calculate and update the materialized path.

4. Write a query to select all subordinates (at all levels) some employee for each of the three ways of representing the hierarchy.

5. Prepare material for inclusion in the final presentation on course "Database Tuning".

Lab #10 Table Partitioning

1. Write 2 queries to join the Bill and BillItem tables in the first case with filtering Bill.date between '02/05/2011' and '02/20/2011', in the second case with filtering Bill.date between '02/05/2011' and '03/20/2011'. Study the query execution plans, explain why they are different. Save the query execution plans.

2. Partition the BillItem table according to the scenario, described in this lab. Use BillItem.Date as the partition key. Explain whether the partitioned tables Bill and BillItem are aligned?

3. Execute the queries from point 1 and study their execution plans. Explain what has changed in the query execution plans.

Laboratory work No. 11 Principles of constructing a statistical and dynamic model

1. Provide a definition of a static information model.

2. What stages must be completed when constructing a static information model.

3. What is the “concept identification” stage? construction of a static information model.

4. What is the stage of “organizing concepts into a hierarchy” classes" when constructing a static information model.

5. What is the “definition of connections” stage in construction? static information model.

6. What is the “property description” stage when constructing static information model.

7. Provide a definition of a dynamic information model.

8. What are workflow models?

9. What are data flow models?

10. What are object models?

11. What do life cycles of objects describe?

12. What use cases describe.

13. What do object interaction diagrams describe?

Lab #12 XML language constructs

1. What are markup languages. Stylistic, structural and semantic markup.

2. History of the development of markup languages: SGML, HTML, XML.

3. What is XML. How XML came into being. XML as data

4. XML document structure. Correctly formatted and valid documents.

5. XML parser. Event-driven parsers. Tree analyzers.

6. Why do we need a Document Type Definition (DTD). General principles for writing DTD definitions:

7. Disadvantages and features of DTD definitions.

8. Modeling data With using XML.

Information model. Mapping to XML. Schemas.

9. Document Object Model (DOM).
10. Namespace and schemas.
11. Connections and requests.
12. Transformation of an XML document.
13. XML Document Design.

Lab #13 Methods for creating Web services

1. Web services. Basic concepts. Purpose.
2. History of Web Services Development. COM/DCOM Model. .Standard CORBA/IIOP. Disadvantages of the listed technologies.
3. Web services .Net. Advantages. Architecture. Basic technologies of web services: SOAP, UDDI, WSDL.

6. Guidelines for preparing for the exam

Before the exam, the student must fully complete all assignments for laboratory work. If there are outstanding debts for the current certification in this discipline, the student is not allowed to take the exam. The exam in the discipline is provided in oral form using tickets.

Basic level.
Questions to check your level of knowledge Know:

1. Caching in databases.
2. Physical organization of tables. Growth and aging of physical tables.
3. Indexes in B-trees. B-trees and B+-trees.
4. Indexes with and without clustering. Hash tables.
5. Calculating selectivity. Filter selectivity and selectivity on the range.
6. Types of connections. Internal and external connections.
7. Methods for handling joins: nested loops, hash join, merge sort join.
8. Locks. Parallel execution of transactions.
9. Setting up blocking.
10. Transaction isolation levels.
11. Transaction logs and data recovery.
12. RAID levels.
13. Optimization of database file placement.
14. Partitioning data tables.
15. Organization of hierarchical data in relational databases.
16. Managing XML documents in relational databases.

17. Methods of internal data sorting.
18. Methods of external data sorting.
19. Methods of searching for data in main memory.
20. Methods of searching for data in external memory. ***Advanced level:***

Know:

1. *Transformation of entities and attributes*
2. *Transformation of binary relations*
3. *Testing the DB model using sequential normalization concepts*
4. *Checking data integrity support*
5. *Physical organization of data*
6. *Page organization of data in DBMS*
7. *Indexed-direct files. Indexed-sequential files*
8. *Organizing indexes as B-trees*
9. *Inverted lists*
10. *Relational calculus*
11. *Target list and defining expression*
12. *Existential quantifier*
13. *Universal quantifier*
14. *The select operator. Forming queries to the database*
15. *Aggregate functions of the language*
16. *Nested queries*
17. *Multi-table queries*
18. *Data entry operator INSERT*
19. *Data deletion operator DELETE*
20. *UPDATE data update operation*
21. *Operators for creating and deleting indexes*
22. *Transaction recovery*
23. *DB parallelism*
24. *DB locking*

7. Methodological materials defining procedures assessment of knowledge, skills, abilities and (or) experience of activities characterizing the stages of formation of competencies

Procedure of implementationexamis carried out in accordance with Regulations on the current monitoring of academic performance and midterm assessment of students in higher education programs at SKFU.

The examination ticket includes 3 questions: 2 theoretical questions (basic level) and 1 theoretical question (advanced level).

You have 40 minutes to prepare for your ticket.

Current certification of masters is carried out by teachers, conducting lectures and laboratory work on the discipline in the form of an interview, a written report.

Admission to laboratory work occurs if the master's student has a written report in the notebook for laboratory work. By the lesson, the master's student must prepare answers to questions for the laboratory work. During the laboratory lesson, the master's student must complete the laboratory work in its entirety, including the individual assignment. The report is defended in the form of a master's report on the work performed and answers to the teacher's questions within the framework of this laboratory work.

A master's student receives the maximum number of points if he actively participates in the work, has a command of the material, can express his thoughts logically and clearly, creatively approaches the solution of the main issues of the topic, and demonstrates independent thinking.

The grounds for reducing the rating are:

- poor quality work on a personal computer
- the written report is done in a careless manner and contains errors
- the graduate student does not give a comprehensive answer to the teacher's question
- is confused or has difficulty demonstrating work on a computer The report may be sent back for revision in the following cases:
- executed on a separate sheet
- careless execution and with errors
- abridged report content

The criteria for assessing interviews and defending laboratory work, studying the topics of the discipline are given in the Fund of assessment tools for the discipline "Tuning Databases".