SQL SERVER BASICS

Basic concepts on Database objects, file systems, storage structure and query processing

Presenter : M.M.Al-Farooque (Shubho)

http://www.linkedin.com/in/thisisshubho

Why the basics are important to know?

Followings are the reasons

- Helps us to know the internal working procedure and tuning performance and resource usage
- Helps us to understand and troubleshoot problems
- Helps us to better design and develop systems

Topics discussed

1. SQL server Databases

3. SQL server storage structure and file systems

5. Data organization

7. Query processing

1.SQL server Databases

Two types of databases are there in SQL server

- <u>System database : </u>Some pre-built databases
- <u>User database : </u>User created databases

System Databases

- **Master :** The default selected database after logging in. This database keeps an eye on the system. It records information about other databases in the system, Disk space, Usage, DBMS configuration. It represents the system schema.

- **Model :** A database template. When a database is created in SQL server 2005, the model database is inherited. So, if anything is changed in the model database, the newly created databases will get those changes.

- **TempDb :** Stores all the temporary data from start to end of the SQL server instance that are used by the other databases. This database also inherits the model database.

- Msdb : Used by the SQL server notification and agent service.

SQL server storage structure

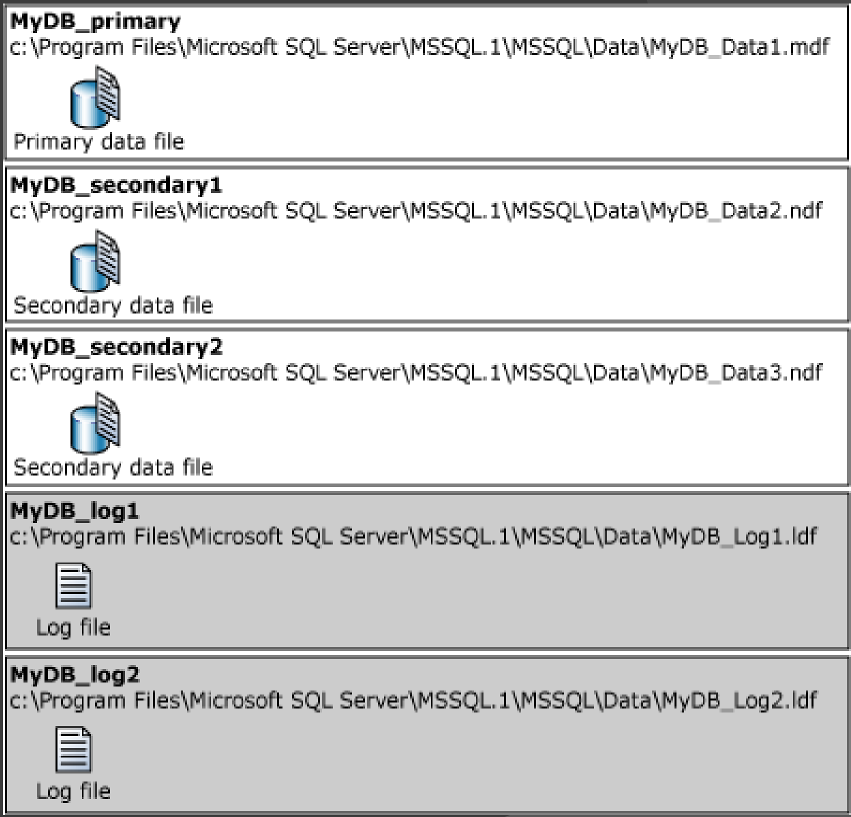
Followings are the units of storage Structure in SQL server

- Database Files
- Extents
- Pages

Database Files

A database is comprised of three kinds of files

- **.mdf file :** Primary data file. Contains data
- **.ndf file :** Secondary or other data files. Optional. Contains Data.
- **.ldf file :** Transaction log file, one or many. Contains transaction logs


MyDB_primary
c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\MyDB_Data1.mdf
Primary data file
MyDB_secondary1
c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\MyDB_Data2.ndf
Secondary data file
MyDB_secondary2
c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\MyDB_Data3.ndf
Secondary data file
MyDB_log1
c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\MyDB_Log1.ldf
Log file
MyDB_log2
c:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Data\MyDB_Log2.ldf
Log file

File Groups

- **Logical** grouping of database files used for performance and to improve administration on large databases
- Particular database objects could be put into particular file groups. For example, tables in a group, and, Nonclastered indexes in another group
- Files within different file groups are written/updated concurrently, so this makes the database server efficient.
- In a newly created database, the primary file group is created by default. The default mdf file(Primary data file) is included in the primary file group.
- Any other file group created is user defined file group
- When creating a database object, the database file or file group could be specified

Multiple data files

Advantages

- Data files could be placed to separate physical disks to improve performance
- Data files could be backed up one by one
- Taking advantage of file groups, performance could be improved by explicitly placing different files in different file groups

Pages and Extents

SQL Server 2005 has two main types of storage structures:

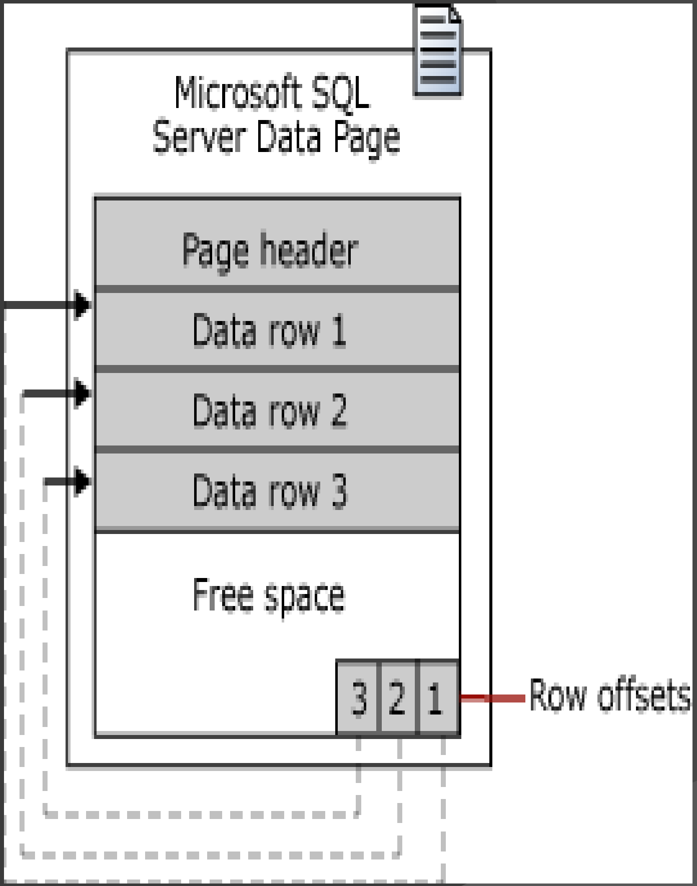
- Extents
- Page

All type of database objects are stored in the data files (mdf and ndf files), which spans across the Pages and Extents in the data files

Pages

At the most basic level, every data is stored in 8KB pages. Each page has a header portion and data is stored in the body portion

- The page is the smallest unit of input/output (I/O) in SQL Server.
- Every time data is either read from or written to a database, this occurs in page units.
- Pages has a fill factor. It determines how tightly a page is going to be packed. For example, if fill factor for a data pages is 80%, then, at most 80% of the data pages can be filled with data. Defalut fill factor for index pages is 0% (Equivalent to 100%)



Extents

- Collection of eight physical contiguous pages that are used to efficiently manage the pages. All pages are stored in extents.

- All database objects are stored in Extent level (Not at page level)

- Two types of extents

  Mixed : In one extent, more than one object is stored (That object is too small to occupy the whole extent)

  Uniform : The whole extent is allocated to a single database object. When an extent is not enough to contain a table or index, another extent is allocated for that table/index.

Extents : Continued

Allocating extent instead of pages to objects serves some important purpose

- Allocation is time consuming and allocating an extent is actually a batch allocation process
- This process allows a contiguous allocation of pages within an extent

Types of Pages

8 types of pages are there. Most important type is the **Data page.**

- A Data page is owned by a particular database object (Say, table)
- Data pages Hold the actual data records. A Page may contain one or more data rows
- Data rows are put on the page serially, starting immediately after the header. A row offset table starts at the end of the page, and each row offset table contains one entry for each row on the page.
- Rows are not allowed to span more than one page. When a page is not sufficient to contain a variable length column, the rest data is moved to a page in the ROW_OVERFLOW_DATA allocation unit, and, a 24 bit pointer is left behind
- The Page header contains info about the page number, page type free space, owner object info etc

Other types of pages

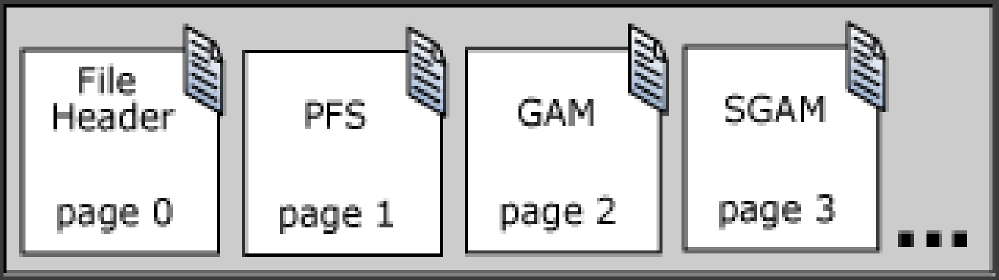
- Index pages : Stores the index keys and levels
- Text/Image page : Contains the actual Text/Image data, and the 16 byte pointers to the Text/Image data is stored in the Data page
- Global Allocation Map pages (GAM) : Keeps tracks of which extents in a data file are allocated and which extents in a data file are free

  GAM : Contains bitmaps that keeps track of uniform extents within the data file

  SGAM : Contains bitmaps that keeps track of shared extents within the data file

Other types of pages : Continued

- Index Allocation Map pages (IAM) :These pages keep track of the extents are allocated to database objects
- Page free space Pages (PFS) : Pages that keep track of page allocation data for all pages within the data file.
- Bulk changed map (BCM) pages : Contains information about other pages that have been modified by bulk operation since the last BACKUP LOG statement
- Differential Changed Map (DCM) pages : Contains information about other pages that have changed since last BACKUP DATABASE statement



Data organization

Data in the tables in SQL server are organized in two ways:

Heap:

- When a table is created without any index, and, data is stored in the table, the table is said to be in a Heap.
- The data rows are not stored in any particular order, and there is no particular order to the sequence of the data pages. Any data retrieval from the table in heaps results in a full scan of the table.

Data organization : Continued

Indexed table:

If a table as a clustered index (Primary key), then, the table is said to be an indexed table.

- The data rows and data pages are physically stored in the order based on the clustered Index key
- Indexes are represented in the system using B+ Tree (Balanced tree) . Indexes are preserved in Memory for faster access. Indexes are represented in tree structure (Using doubly linked list) for faster accesses of the index keys.

Types of Indexes

- Two types of indexes are there

  <u>Clustered :</u>

  - Creation of a primary key on a table results in creating a clustered index on the table.
  - Each tree node in the balanced tree of a Clustered index contains the actual row (Page).
  - Rows and (Pages) are physically sorted according to the clustered key.
  - There could be only one clustered index per table.

Types of indexes : Continued

<u>Non Clustered :</u>

- Typically, creation of a Unique constraint results in creating a non-clustered index on the table.
- Each tree node in the B+ tree of a non-clustered index contains the pointer to the actual row (Page).
- The pointer could be a pointer to the clustered index key, or, could be a pointer to the actual row (Page).
- There could be at most 255 Non clustered indexes per table.

Allocation Unit

An allocation unit is a collection of pages to manage data based on their page type. types of allocation units are:

IN_ROW_DATA : Used to manage Data or index rows that contain all
data, except large object (LOB) data

LOB_DATA: Used to manage Large object data stored in one or more of these data types: text,ntext, image, xml, varchar(max), nvarchar(max), varbinary(max)

ROW_OVERFLOW_DATA: Used to manage Variable length data stored in **varchar**, **nvarchar**, **varbinary**, or **sql_variant** columns that exceed the 8,060 byte row size limit

Index Allocation Map (IAM)

- Pages Used to map the extents within a particular type of allocation unit . Used to find the allocated extents for a particular database object (Say, table).
- IAM pages are allocated as required for each allocation unit and are located randomly in the file.
- Each allocation unit has at least one IAM page for each file. If allocation unit size is more than 4 GB, more than one IAM page is required and, the IAM pages are linked

Data Insertion

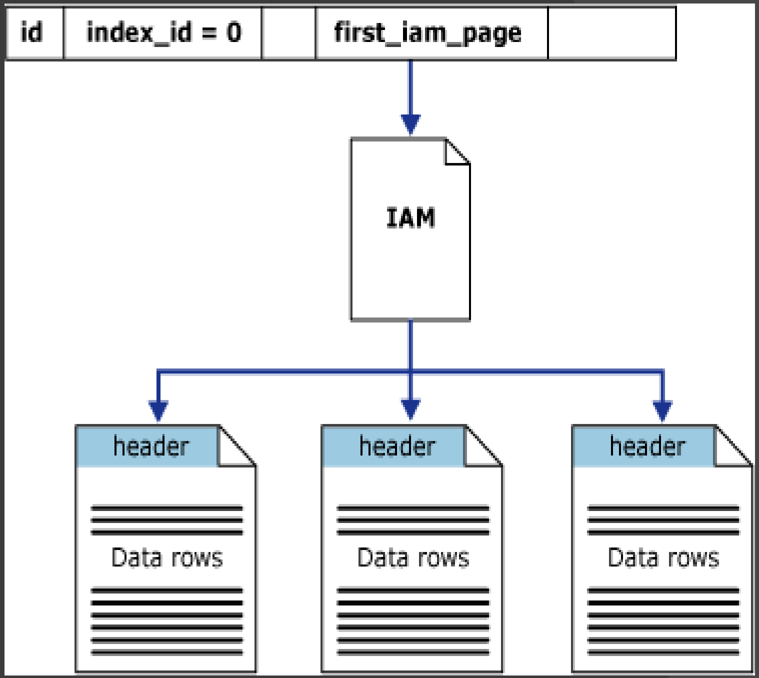
When the SQL Server Database Engine has to insert a new row :

- If space is available in the current page, the row is inserted there
- If not, The Database Engine uses the IAM pages to find the extents allocated to the appropriate allocation unit
- For each extent, the Database Engine searches the PFS pages to see if there is a page that can be used
- IAM and PFS pages covers lots of extents and pages, so, they are few in numbers. So, these are recides in memory in the SQL Server buffer pool, so they can be searched quickly

Data selection through full table scan

Issuing a select query that has no filter condition on any indexed column results in a full table scan. In this case, the SQL server must go through all corresponding pages of the table

- SQL server goes to the corresponding table's row in the sysindexes table, and, finds the "indid" column value is 0. In this case, the SQL server reads the FirstIAM column's value that points to the First Index Allocation Map
- The SQL server scans each bit in the IAM page, reads the contiguous pages in the extents one by one to find the pages belonging to the table. Then, the SQL server reads the rows in the matching pages to return result.



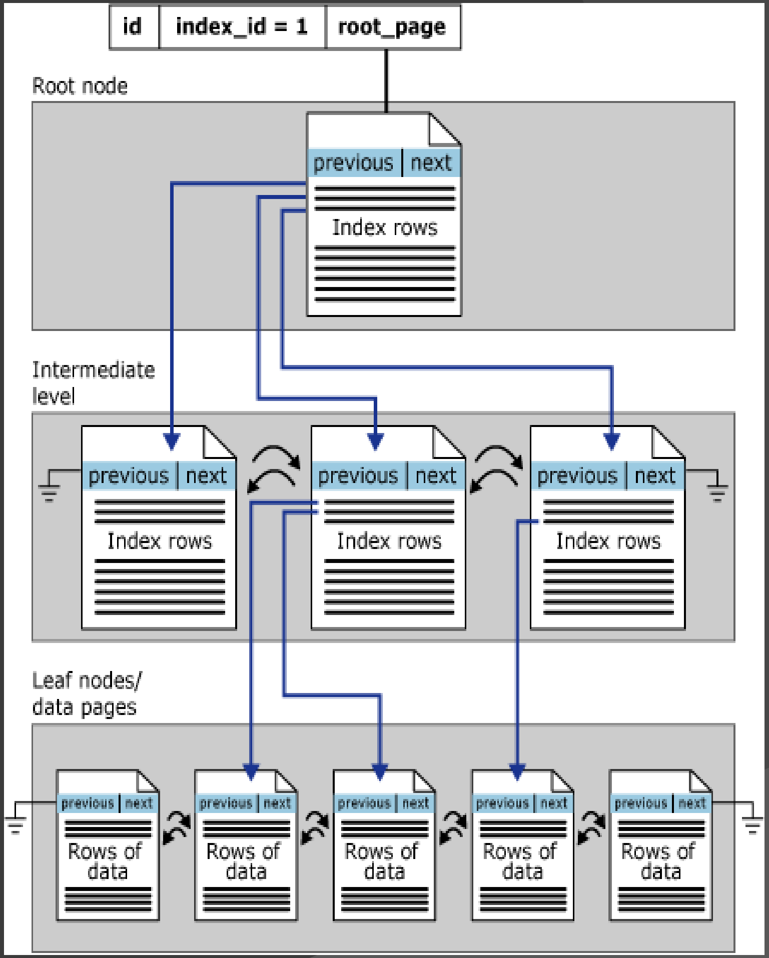
Clustered index tree

- In a clustered index, the leaf nodes contain the data pages of the underlying table.
- The root and intermediate level nodes contain index pages holding index rows

Data selection through clustered index

Issuing a select query that has filter condition on any primary key (Clustered index) column results in a clustered Index tree search.

- SQL server goes to the corresponding table's row in the sysindexes table, and, finds the "indid" column value is 1. In this case, the SQL server reads the "Root"column's value that points to the root node of the clustered index tree.
- The SQL server scans the index rows at each level to match the searched data, and, goes to the next deeper level, until the matched pages are found
- The leaf nodes in the clustered index tree are the physically sorted actual pages containing the data. The data is/are returned from these pages



Non-clustered index tree

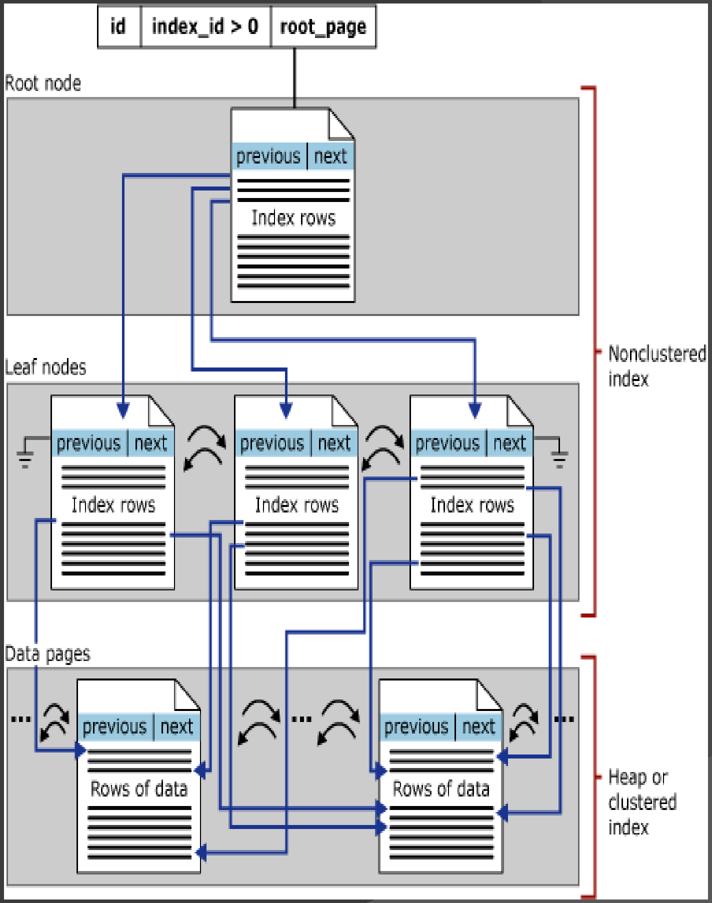
Nonclustered indexes have the same B-tree structure as Clustered indexes, except for the following significant differences:

- The data rows of the underlying table are not sorted and stored in order based on their nonclustered keys.
- The leaf layer of a nonclustered index is made up of index pages instead of data pages. That is, the leaf nodes points to the clustered index key

Data selection through non-clustered index

Issuing a select query that has filter condition on any field that is an indexed column (Non PK column), results in a non-clustered Indexed tree search.

- SQL server goes to the corresponding table's row in the sysindexes table, and, finds the "indid" column value is >0. In this case, the SQL server reads the "Root"column's value that points to the root node of the non-clustered index tree.
- The SQL server scans the index rows at each level to match the searched data, and, goes to the next deeper level, until the matched leaf nodes are found
- The leaf nodes in the non-clustered index tree contains the clustered index key. So, the SQL server again searches the clustered index tree to find the corresponding result.

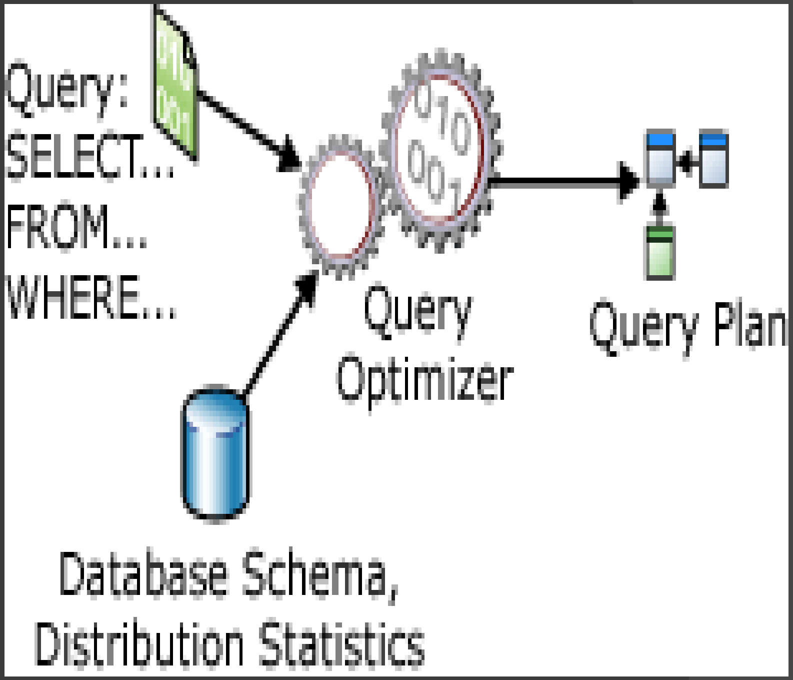


Query processing

The SQL Server query processor consists of two components:

<u>The query optimizer :</u>
Responsible for generating good query plans, using the input SQL and database statistics

<u>The query execution engine:</u>
Takes the query plans generated By the query optimizer and, as Its name suggests, runs them



Iterators/Operators

SQL Server breaks queries down into a set of fundamental building blocks that we call operators or iterators.

- Each iterator implements a single basic operation such as scanning data from a table, updating data in a table, filtering or aggregating , joining etc
- An iterator/operator may have 0 or more children, and, the iterators/operators form a query execution tree
- When SQL Server executes a query plan, control flows down this execution tree.

Execution plan optimization

The query optimizer analyzes several possible execution plans.

Each possible execution plan has an associated cost in terms of the amount of computing resources used.

The query optimizer analyzes the possible plans and choose the one with the lowest estimated cost.

Execution plan structure

SQL Server 2005 execution plans have the following main components:

- Query Plan

  The bulk of the execution plan is a, read-only data structure used by any number of users. This is referred to as the query plan. No user context is stored in the query plan.

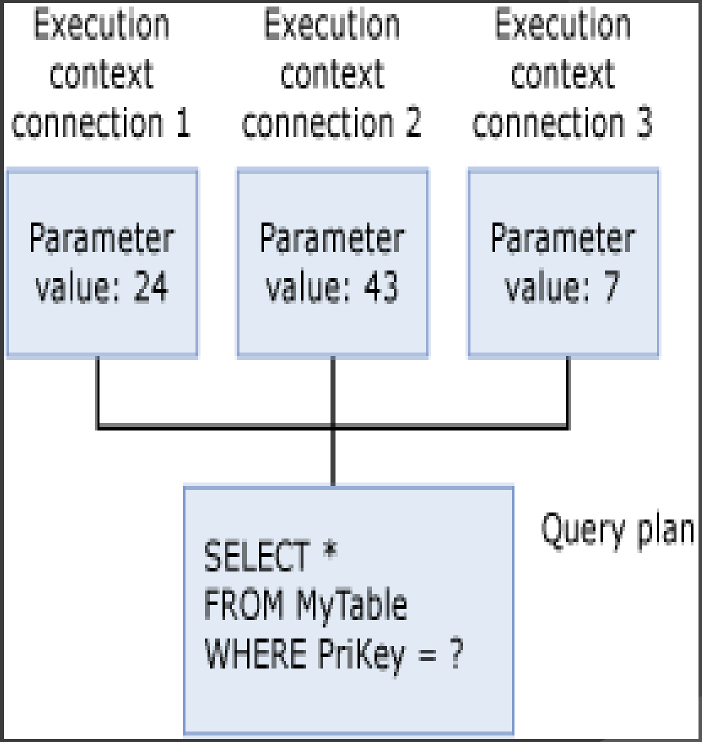
- Execution Context

  Each user that is currently executing the query has a data structure that holds the data specific to their execution, such as parameter values. This data structure is referred to as the execution context.

Execution plan structure : Continued

- When any SQL statement is executed, the relational engine first looks through the procedure cache to use an existing execution plan for the same SQL statement .

- If no cached execution plan is found for the SQL , the SQL Server generates a new execution plan for the query and put in cache


Execution context connection 1
Execution context connection 2
Execution context connection 3
Parameter value: 24
Parameter value: 43
Parameter value: 7
SELECT * FROM MyTable WHERE PriKey = ?
Query plan