

Министерство образования и науки Российской Федерации
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Курганский государственный университет»

В.К. Волк

БАЗЫ ДАННЫХ

Часть 2

АДМИНИСТРИРОВАНИЕ

Учебное пособие

Курган 2018

УДК 004.658.2(075.8)

ББК 32.973я73

В 67

Рецензенты

кафедра прикладной информатики и автоматизации бизнес-процессов Шадринского государственного педагогического университета (канд. физ.-мат. наук, профессор В.Ю. Пирогов);

кафедра информационных систем Тюменского государственного университета (канд. физ.-мат. наук, доцент П.К. Моор).

Печатается по решению методического совета Курганского государственного университета.

Научный редактор – канд. физ.-мат. наук, проф. В.А. Симахин

Волк В. К.

Базы данных. Часть 2. Администрирование : учебное пособие. – Курган : Изд-во Курганского гос. ун-та, 2018. – 128 с.

Учебное пособие содержит пять тематических разделов, в которых рассматриваются основные аспекты технологии администрирования баз данных: первый раздел – архитектура физической модели данных и типовые алгоритмы управления данными; второй и третий разделы – индексные структуры данных, алгоритмы поиска и процедурные планы выполнения SQL-запросов; четвертый раздел – концепции и методы защиты информации, реализуемые серверами баз данных; пятый раздел – архитектура системы информационной безопасности сервера MS SQL-Server и программные средства управления правами доступа.

Завершает учебное пособие лабораторный практикум, включающий семь лабораторных работ исследовательского характера, выполнение которых предполагает использование инструментальных средств администрирования, предоставляемых разработчиком сервера баз данных.

Изложение учебного материала иллюстрировано практическими примерами и сопровождается ссылками на официальные интернет-ресурсы разработчика.

Пособие предназначено для студентов IT-специальностей и может быть рекомендовано широкому кругу IT-специалистов для начального освоения технологии администрирования баз данных.

Рисунков – 19, таблиц – 11, листингов программного кода – 34.

ISBN 978-5-4217-0440-9

УДК 004.658.2(075.8)

ББК 32.973я73

© Курганский
государственный
университет, 2018
© Волк В.К., 2018

Содержание

ПРЕДИСЛОВИЕ	5
Глава 1. ФИЗИЧЕСКАЯ МОДЕЛЬ ДАННЫХ.....	7
1.1 Файловая структура базы данных	7
1.1.1 Файлы	7
1.1.2 Группы файлов	9
1.2 Структура файла типа DATA.....	12
1.2.1 Страницы.....	13
1.2.2 Экстенты.....	14
1.3 Типы файловых страниц.....	15
1.4 Служебные структуры данных	22
1.5 Программные средства анализа структуры файла данных.....	23
Глава 2. УПРАВЛЕНИЕ ИНДЕКСАМИ.....	25
2.1 Доступ к неупорядоченным данным	25
2.2 Линейный индекс.....	26
2.3 Многоуровневый иерархический индекс.....	29
2.4 Кластеризованный многоуровневый индекс	33
2.5 Фактор заполнения индексных страниц	36
2.6 Рекомендации по использованию индексов	37
Глава 3. ОПТИМИЗАЦИЯ ПРОЦЕДУРНЫХ ПЛАНОВ SQL-ЗАПРОСОВ	39
3.1 SQL – язык программирования декларативного типа.....	39
3.2 Типовая схема трансляции SQL-запроса	40
3.3 Исполнение процедурного плана выполнения запроса	47
3.4 Средства визуализации процедурных планов	48
3.4.1 Инструкции TransactSQL.....	48
3.4.2 Средства графического отображения процедурных планов	50
Глава 4. ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ БАЗ ДАННЫХ.....	53
4.1 Целостность информации.....	54
4.2 Доступность и конфиденциальность информации.....	60
4.3 Дискреционная защита информации.....	62
4.4 Мандатная защита информации	65
Глава 5. УПРАВЛЕНИЕ ДОСТУПОМ К ДАННЫМ MS SQL-SERVER.....	68
5.1 Двухуровневая архитектура управления доступом.....	68
5.2 Управление доступом на уровне сервера	70
5.2.1 Режимы аутентификации.....	70

5.2.2	Учетные записи и разрешения уровня сервера	71
5.2.3	Фиксированные роли сервера	73
5.2.4	Хранение информации об учетных записях	76
5.3	Управление доступом на уровне базы данных	77
5.3.1	Объекты доступа: таблицы, представления, команды и схемы	77
5.3.2	Субъекты доступа: пользователи и роли базы данных	78
5.3.3	Хранение информации о субъектах доступа	80
5.3.4	Программные средства управления пользователями и ролями	82
5.3.5	Программные средства управления правами доступа	83
Глава 6.	ЛАБОРАТОРНЫЙ ПРАКТИКУМ	90
	ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ.....	90
	Лабораторная работа №1	
	АНАЛИЗ ФАЙЛОВОЙ СТРУКТУРЫ БАЗ ДАННЫХ.....	91
	Лабораторная работа №2	
	АНАЛИЗ АЛГОРИТМОВ РЕЗЕРВИРОВАНИЯ ДИСКОВОЙ ПАМЯТИ	94
	Лабораторная работа №3	
	ИССЛЕДОВАНИЕ ИНДЕКСНЫХ СТРУКТУР ДАННЫХ.....	106
	Лабораторная работа №4	
	АНАЛИЗ ПРОЦЕДУРНЫХ ПЛАНОВ ВЫПОЛНЕНИЯ SQL-ЗАПРОСОВ	112
	Лабораторная работа №5	
	АНАЛИЗ АРХИТЕКТУРЫ СИСТЕМЫ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ	120
	Лабораторная работа №6	
	АНАЛИЗ СРЕДСТВ УПРАВЛЕНИЯ ДОСТУПОМ К ДАННЫМ	123
	Лабораторная работа №7	
	АНАЛИЗ ИЕРАРХИИ ПРАВ ДОСТУПА К ДАННЫМ	125
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	127

ПРЕДИСЛОВИЕ

Эксплуатационные характеристики хранилища данных, интегрированного в состав автоматизированной информационной системы, во многом определяют эффективность работы всего программного комплекса. Грамотное администрирование базы данных может существенно повысить производительность информационной системы, обеспечить высокую надежность хранения и требуемый уровень защиты информации.

Администрирование как вид профессиональной деятельности направлено на поддержание эффективной и бесперебойной работы баз данных, обеспечивающих функционирование информационных систем, и связано с выполнением следующих основных функций:

- эксплуатация серверов баз данных;
- поддержание баз данных в актуальном состоянии и обеспечение их эффективного функционирования;
- обеспечение доступности данных для легальных пользователей и защита от несанкционированного доступа к данным;
- мониторинг и идентификация потребностей пользователей.

Приведенный набор типовых функций администраторов баз данных закреплен как российскими, так и зарубежными профессиональными стандартами и классификационными системами, в которых явно разделяется деятельность по созданию баз данных (разработка концепций, проектирование и программирование) и деятельность по их сопровождению и поддержанию работоспособности (администрирование).

Так, в стандарте США SOC (Standard Occupational Classification) [1] работы по сопровождению баз данных включены в подкатегорию «Database Administrators» в составе категории «Database and Systems Administrators and Network Architects», а функции по созданию баз данных определены в подкатегории «Software Developers», входящей в состав категории «Software and Web Developers, Programmers and Testers».

Соответствующий европейский стандарт European ICT Professional Profiles [2] предусматривает профиль «Developer» для деятельности по проектированию баз данных и отдельный профиль «Database Administrator», в котором определены работы по обеспечению их функционирования в процессе эксплуатации.

Российские профессиональные стандарты «Архитектор программного обеспечения», «Руководитель разработки программного обеспечения» и

«Программист» определяют функции проектирования и программирования баз данных, а комплекс работ по их сопровождению регламентирован стандартом «Администратор баз данных» [3], в котором приведен детальный перечень трудовых функций с указанием для каждой из них квалификационного уровня, требований к профессиональной компетентности, базовому образованию и опыту работы. Стандарты «Специалист по информационным системам» [4], «Системный администратор информационно-коммуникационных систем» [5] и «Специалист по защите информации в автоматизированных системах» [6] также регламентируют отдельные аспекты администрирования баз данных.

Учебное пособие не является полным практическим руководством по администрированию баз данных и, разумеется, не охватывает весь объем трудовых функций, определенных соответствующими профессиональными стандартами. При подготовке учебного пособия основное внимание было уделено вопросам организации физической модели данных, оптимизации выполнения SQL-запросов и разграничения доступа к данным – то есть решению наименее рутинных задач администрирования баз данных, требующих профессиональной компетентности в вопросах функционирования серверов баз данных, организации хранения данных на файловом уровне и методах поиска информации, основанных на использовании индексных структур данных.

Использование пособия предполагает наличие у студентов базовой подготовки в области реляционных баз данных (теория и технология проектирования, методы управления, язык SQL), а также знание основных понятий и концепций информационной безопасности.

При подготовке учебного пособия учтен опыт преподавания технологий баз данных студентам IT-специальностей Курганского государственного университета. Автор выражает благодарность всем студентам, использовавшим предварительный вариант этого пособия и сделавшим ряд полезных замечаний по его содержанию. Особая благодарность студентам П. Казакову и Д. Стенникову, проделавшим большую работу по анализу структуры системных объектов баз данных MS SQL Server, и М.Останину, исследовавшему индексные структуры данных и процедурные планы выполнения SQL-запросов.

ФИЗИЧЕСКАЯ МОДЕЛЬ ДАННЫХ

Модель, или структура данных – одно из основополагающих понятий технологий хранения и обработки информации, хорошо знакомое каждому программисту. В теории и технологии разработки реляционных баз данных рассматриваются три уровня представления модели данных:

- концептуальный уровень – так называемая ER-модель, описывающая объектную структуру предметной области в понятиях «сущность-атрибут-связь»;
- логический уровень, представляющий реляционную схему базы данных в "табличной" терминологии и обеспечивающий SQL-взаимодействие программных приложений с сервером баз данных;
- физический уровень, на котором объекты логической модели данных отображаются в элементы файловой структуры вычислительной системы.

MS SQL-Server поддерживает двухуровневую структуру физической модели данных:

- верхний уровень физической модели обеспечивает взаимосвязь с файловой системой и представлен объектами «файл» и «группа файлов»;
- нижний уровень обеспечивает взаимосвязь с логической моделью данных и определяет внутреннюю структуру файла базы данных и представлен объектами «страница» и «экстент» (группа страниц).

1.1 Файловая структура базы данных

Файловая структура базы данных представляет верхний уровень физической модели данных, на котором сервер баз данных обеспечивает взаимодействие с файловой системой. Структура базы данных MS SQL-Server представлена объектами двух категорий: *файлы* и *группы файлов*.

1.1.1 Файлы

MS SQL-Server поддерживает файлы двух типов:

- Единственный (в базе данных) *файл журнала транзакций* (log file), в котором сервер сохраняет информацию обо всех активных операциях доступа к базе данных, то есть о тех операциях, которые изменили ее состояние (например, об операциях **Update**, **Delete** или **Insert**).

Имена файлов этого типа имеют стандартное расширение **.ldf** (**l**og **d**a-ta **f**ile).

- Множество *файлов данных* (data-files), в которых хранятся экземпляры всех логических объектов базы данных – пользовательских и системных таблиц, хранимых представлений, процедур, функций, ограничений целостности данных и т.д., а также служебных структур данных, например, индексов. Имена файлов этого типа могут иметь одно из двух стандартных расширений: **.mdf** (**m**aster **d**ata **f**ile) или **.ndf** (**s**econdary **d**ata **f**ile).

Если в базе данных всего один файл типа data, он имеет статус *первичного* (*primary file* - .mdf). Администратор может создавать и другие файлы этого типа – все они будут иметь статус вторичных файлов (*secondary file* - .ndf), при этом в базе данных может быть только один первичный файл.

Все данные системного каталога базы данных хранятся в первичном файле, а пользовательские данные могут храниться как в первичном, так и во вторичных файлах.

Файл типа data – это, по существу, сегмент дискового пространства определенного размера, зарезервированный сервером баз данных у файловой системы. Начальный размер файла либо явно указывается в момент создания базы данных SQL-инструкцией **CREATE DATABASE**, как это показано в примере (листинг 1.1), либо определяется по умолчанию в соответствии с параметрами системной базы данных **Model**.

При этом предполагается, что файл имеет объем свободного пространства, достаточный для функционирования базы данных в течение некоторого периода времени, а когда свободного места в файле не остается, сервер запрашивает у файловой системы дополнительный сегмент дискового пространства, и файл увеличивается в размере.

Каждый файл базы данных характеризуется рядом параметров, значения которых хранятся в системной таблице **SysFiles** этой базы данных:

- **File_ID** – внутренний идентификатор файла, уникальный в пределах базы данных; автоматически присваивается сервером файлу при его создании, используется как элемент адресной ссылки на страницу данных и/или на строку таблицы; выполняет роль первичного ключа таблицы **SysFiles**.

- **File_name** – имя файла в файловой системе, задается в стандартном формате (том:\путь\имя.расширение).
- **Name** – логическое имя файла (синоним `File_name`).
- **Size** – текущий размер файла (объем дисковой памяти, зарезервированной сервером баз данных у файловой системы); может автоматически изменяться в процессе эксплуатации базы данных.
- **MaxSize** – максимально-допустимый размер файла; может иметь значение `unlimited` – в этом случае размер файла ограничивается ресурсами файловой системы.
- **Growth increment** – шаг приращения размера файла – дополнительная "порция" дискового пространства, запрашиваемая сервером у файловой системы, когда текущий размер файла становится недостаточным для хранения данных; может задаваться в абсолютных размерных единицах или в процентах от текущего размера файла.
- **Group_ID** – внутренний идентификатор файловой группы, к которой принадлежит этот файл; выполняет роль внешнего ключа, обеспечивающего связь (M:1) между системными таблицами **SysFiles** и **SysFileGroups**.

1.1.2 Группы файлов

Для удобства администрирования и повышения эффективности хранения данных файлы типа **data** могут быть (логически !) распределены по *файловым группам*. В каждой базе данных по умолчанию создается единственная файловая группа, имеющая статус *первичной* (*primary file group*), при этом администратор может создавать дополнительные файловые группы, каждая из которых получает статус *вторичной* (*secondary file group*). Формируются вторичные файловые группы с помощью ключевого слова **FILEGROUP** в SQL-инструкциях **CREATE DATABASE** и **ALTER DATABASE**.

Первичный файл всегда принадлежит первичной файловой группе, вторичный файл может принадлежать как первичной, так любой из вторичных файловых групп. Файл не может одновременно входить в состав нескольких файловых групп.

Примечание 1: файловая группа является логической надстройкой над файловой структурой базы данных: принадлежность файла опреде-

ленной файловой группе не накладывает никаких ограничений на его физическое размещение в дисковом пространстве вычислительной системы.

Примечание 2: файл журнала транзакций (.ldf) не принадлежит ни одной из файловых групп базы данных.

Каждая файловая группа характеризуется рядом параметров, значения которых хранятся в системной таблице **SysFileGroups** базы данных:

- **Group_ID** – внутренний идентификатор файловой группы, уникальный в пределах базы данных; автоматически присваивается сервером при создании группы; выполняет роль первичного ключа системной таблицы **SysFileGroups**.

- **Group_name** – имя файловой группы; используется в операторах **Create Table** и **Create Index** для явного указания файловой группы, в файлах которой должны сохраняться данные таблицы или индекса. Первичная файловая группа всегда имеет имя PRIMARY.

- **Default** – присваивает группе статус «группа по умолчанию»:

- из множества групп только одна группа может иметь такой статус;
- если этот параметр явно не указан ни для одной из групп, группой по умолчанию будет назначена первичная группа;
- при создании нового файла в базе данных он будет ассоциирован с «группой по умолчанию», если иное явно не указано;
- при создании новой таблицы в базе данных она будет ассоциирована с «группой по умолчанию», если иное явно не указано, и все данные этой таблицы будут физически размещены в файлах этой группы.

- **Read Only** – присваивает группе статус «только для чтения»; если таблица базы данных ассоциирована с такой группой, для этой таблицы будет заблокирована возможность модификации данных.

Приведенный ниже пример (листинг 1.1) иллюстрирует языковые средства (TransactSQL) управления файловой структурой баз данных и содержит пакет из 4-х последовательных SQL-инструкций:

Инструкция **CREATE DATABASE** создает пользовательскую базу данных **MyDB**, содержащую первичный файл данных в первичной файловой группе, пользовательскую файловую группу **FG1**, содержащую два

вторичных файла, и файл журнала (вне файловых групп). Инструкция не устанавливает свойство `DEFAULT` ни одной из файловых групп, в результате чего статус «группы по умолчанию» получает первичная файловая группа.

Инструкция **CREATE TABLE** создает в базе данных таблицу **MyTable**, явно ассоциированную с пользовательской файловой группой **FG1** (а не с первичной группой, имеющей к этому моменту статус «группы по умолчанию»).

Инструкция **ALTER DATABASE** модифицирует структуру базы данных **MyDB** и придает файловой группе **FG1** статус группы по умолчанию.

Последняя инструкция **CREATE TABLE** создает в базе данных еще одну таблицу **MyTable1**, также (неявно) ассоциированную с пользовательской файловой группой **FG1**, имеющей к этому моменту статус «группы по умолчанию».

```
USE master;
CREATE DATABASE MyDB
ON PRIMARY
    (NAME='Primary_F',FILENAME='C:\data\Prm.mdf',
    SIZE=4MB, MAXSIZE=10MB, FILEGROWTH=1MB),
FILEGROUP FG1
    (NAME = 'Dat1',FILENAME = 'D:\data\FG1_1.ndf',
    SIZE = 1MB, MAXSIZE=5MB, FILEGROWTH=1MB),
    (NAME = 'Dat2',FILENAME = 'E:\data\FG1_2.ndf',
    SIZE = 15MB, MAXSIZE=50MB, FILEGROWTH=1MB)
LOG ON (NAME='MyDB_log',FILENAME = 'C:\data\MyDB.ldf',
    SIZE=1MB,MAXSIZE=10MB, FILEGROWTH=1MB);
USE MyDB;
CREATE TABLE MyTable(ID int PRIMARY KEY, Data char(8))
ON FG1;
ALTER DATABASE MyDB MODIFY FILEGROUP FG1 DEFAULT;
CREATE TABLE MyTable1 (ID int, Name char(16));
```

Листинг 1.1 – Создание и модификация файловой структуры базы данных

Лабораторная работа №1, представленная в шестом разделе учебного пособия, посвящена исследованию файловой структуры базы данных и практическому освоению соответствующих инструментальных программных средств.

1.2 Структура файла типа DATA

Основным объектом файловой модели базы данных является файл типа DATA, ассоциированный с некоторой файловой группой. База данных может содержать несколько таких файлов, главное назначение которых – хранение экземпляров объектов логической модели данных (например, строк таблиц) для их последующего извлечения в соответствии с поступившими серверу SQL-запросами.

При создании новой таблицы она будет связана с одной из файловых групп: или явной ссылкой на имя файловой группы в SQL-инструкции CREATE TABLE, или неявно – с группой, имеющей в этот момент статус «группы по умолчанию». В любом случае при вставке строк в таблицу серверу баз данных потребуется решать задачу поиска свободного пространства в файлах, включенных в соответствующую файловую группу, и задачу эффективного распределения строк таблицы между этими файлами с сохранением соответствующей адресной информации, необходимой для последующего поиска строк таблицы.

Приведенный ниже пример (листинг 1.2) иллюстрирует задачу поиска свободного пространства в файлах базы данных и задачу поиска строк заданной таблицы, удовлетворяющих определенному условию.

Пакет содержит четыре последовательных SQL-инструкций:

Инструкция **CREATE DATABASE** создает пользовательскую базу данных, содержащую три файла: один файл в первичной файловой группе и два файла – во вторичной группе FG1. Инструкция не содержит явного указания на группу со статусом DEFAULT, следовательно, «группой по умолчанию» будет считаться первичная файловая группа.

Инструкция **CREATE TABLE** создает таблицу MyTable, явно ассоциированную с вторичной группой FG1.

Следующая инструкция **INSERT INTO** вставляет в таблицу MyTable 10000 строк, заполняя их случайными данными.

Последняя инструкция **SELECT ... INTO** производит выборку из таблицы MyTable тех её строк, которые удовлетворяют заданному условию, и вставку выбранных строк в новую таблицу NewTable.

Вставка строк в таблицу MyTable потребует поиска свободного пространства в двух вторичных файлах группы FG1, а строки таблицы

NewTable будут записаны в единственный первичный файл первичной файловой группы, так как именно эта группа имеет статус DEFAULT.

Очевидно, что для реализации алгоритмов поиска свободного пространства в файле и алгоритмов поиска строк таблиц, сохраненных в файле, необходимо определить информационную структуру самого файла и предусмотреть наличие служебных структур данных, обеспечивающих хранение адресной информации.

```
USE Master
CREATE DATABASE MyDB
  ON PRIMARY (NAME='MyDB_Primary',FILENAME='C:\Prm.mdf'),
  FILEGROUP FG1
  (NAME = 'MyDB_Dat1',FILENAME='D:\F1.ndf',
   SIZE = 2MB, MAXSIZE=5MB, FILEGROWTH=1MB),
  (NAME = 'Dat2',FILENAME='E:\F2.ndf',
   SIZE = 15MB, MAXSIZE=50MB, FILEGROWTH=5MB);
USE MyDB
CREATE TABLE MyTable (Key_0 INT IDENTITY,Key_1 INT,Key_2
INT,
  FullName CHAR(60))
  ON FG1;
DECLARE @key0 INT, @key1 INT, @key2 INT, @name CHAR(60)
SET @key1=1000*RAND(),@key2=1000*RAND()
SET @name=STR(@key1)+STR(@key2)
INSERT INTO MyTable values (@key1,@key2,@name)
Go 10000
SELECT key_1, FullName INTO NewTable
FROM MyTable WHERE Key_2>500;
```

Листинг 1.2 – Пример выборки и вставки строк в таблицы

1.2.1 Страницы

Базовым элементом файла типа DATA является *файловая страница*. Сервер баз данных представляет файл типа DATA как линейный список из страниц фиксированного размера (по 8 Кб), которые последовательно (позиционно) пронумерованы, начиная с нулевой страницы*. Та-

* Физически, на уровне файловой системы, файл может быть фрагментирован, однако этот факт игнорируется, и с точки зрения сервера баз данных, файл – это неразрывная цепочка последовательно расположенных и позиционно пронумерованных страниц.

ким образом, номер страницы **PageNum** однозначно определяет ее позицию внутри файла: смещение (в килобайтах) от начала файла до начала страницы вычисляется, как $8 * \text{PageNum}$.

Учитывая тот факт, что файлы базы данных тоже "пронумерованы", и каждый из них при регистрации получил свой уникальный идентификатор `FileID`, сохраненный в системной таблице `SysFiles`, агрегат «`SysFiles.PageNum`» однозначно определяет адрес страницы в файловой структуре базы данных.

Страница является минимальным объектом физической модели данных, который может быть выделен сервером логическому объекту, при этом один логический объект (например, таблица) может быть владельцем одной или нескольких страниц одного или нескольких файлов, а одна страница не может иметь более, чем одного владельца.

Из сказанного, в частности, следует, что номера страниц, владельцем которых является таблица базы данных, могут использоваться в качестве адресных ссылок, используемых для поиска и выборки всех строк этой таблицы.

1.2.2 Экстенты

Если страница является «единицей занятости» файла данными логических объектов, то для хранения информации о свободном пространстве файлов используется другая, более крупная единица, называемая *экстентом*. Экстент имеет фиксированный размер 64 Кб и представляет собой блок из восьми соседних страниц одного файла.

Экстенты, так же, как и страницы, последовательно пронумерованы в пределах файла, начиная с нулевого экстента. При этом номер экстента **ExtNum** и номера входящих в него страниц **PageNum** связаны простой арифметической зависимостью. Например, страница 18 принадлежит экстен-ту №2, а экстент №5 включает страницы 40 – 47.

Сервер баз данных может присвоить экстен-ту статус *однородного* (uniform) или *смешанного* (mixed, shared): смешанный экстент может совместно использоваться разными владельцами страниц, а владельцем всех восьми страниц однородного экстента является какой-либо один логический объект базы данных (рисунок 1.1).

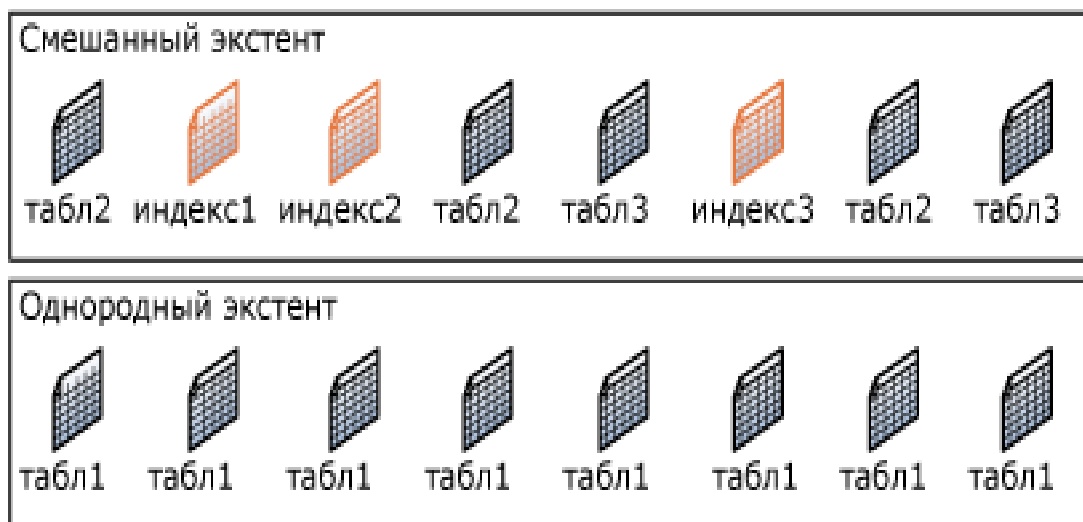


Рисунок 1.1 – Схема размещения страниц
в смешанных и однородных экстентах

Очевидно, что для небольших таблиц эффективнее оперировать смешанными экстентами, иначе некоторые страницы однородных экстентов всегда будут оставаться не занятыми. С другой стороны, что также очевидно, наличие однородных экстентов ускоряет работу алгоритмов поиска данных в таблицах большого объема (от 64 Кб), так как в этом случае адрес (порядковый номер) какой-либо одной страницы, принадлежащей логическому объекту, фактически определяет и адреса еще семи страниц этого же объекта.

Совсем не очевидна логика сервера баз данных, принимающего решение о выделении логическому объекту экстентов того или иного типа – исследованию этого вопроса посвящено одно из заданий лабораторной работы №2, представленной в 6-м разделе учебного пособия.

1.3 Типы файловых страниц

Перечень типов страниц, поддерживаемых MS SQL-Server'ом, приведен в таблице 1.1.

Страницы типа **Data/Index** – основной тип файловых страниц, предназначенных для хранения экземпляров логических объектов – строк таблиц базы данных, а также строк индексных таблиц, подобных по своей структуре классическим реляционным таблицам. Упрощенная схема хранения данных в странице типа DATA/INDEX иллюстрируется рисунком 1.2.

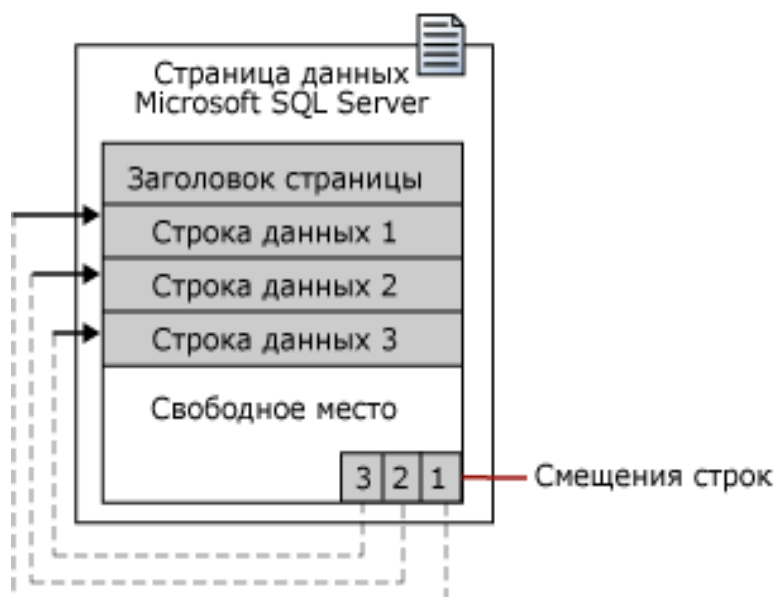


Рисунок 1.2 – Структура страницы типа Data/Index

Все пространство страницы (8 Кб) разделено на три области:

- *заголовок страницы* – начальная область страницы, имеет фиксированный размер 96 байтов, содержит служебную информацию, специфическую для страниц различных типов;
- *тело страницы* – основная область страницы (максимум 8060 байтов); содержит множество контейнеров («слотов»), каждый из которых предназначен для хранения одной строки таблицы;
- *хвостовик страницы* – конечная область страницы переменной длины (минимум 36 байтов); содержит массив указателей на слоты тела страницы (т.е., по существу, – на строки таблицы).

Таблица 1.1 – Типы страниц, поддерживаемых MS SQL-Server'ом

Тип страницы	Тип хранимой информации
Data	Данные логических объектов, кроме данных типов LOB (LargeOBjects)
Index	Данные индексов
Text/Image	<ul style="list-style-type: none"> • Данные "длинных" типов, экземпляры которых превышают размер файловой страницы: text, ntext, image, xml, nvarchar(max), varchar(max), varbinary(max); • Данные типов переменной длины varchar(), nvarchar(), varbinary(), sql_variant в условиях, когда размер строки на основной странице типа Data превышает размер 8060 байтов
IAM	Index Allocation Map – битовая карта размещения страниц логического объекта по экстендам файла
GAM, SGAM	Global Allocation Map, Shared Global Allocation Map – глобальные битовые карты, содержащие информацию о свободных экстендах, их типах и степени заполнения
PFS	Page Free Space – информация о степени заполнения страниц
BCM	Bulk Changed Map (карта массовых изменений) – глобальная битовая карта, содержащая информацию об экстендах, измененных с момента последнего выполнения операции резервного копирования журнала транзакций
DCM	Differential Changed Map (карта разностных изменений) – глобальная битовая карта, содержащая информацию об экстендах, измененных с момента последнего выполнения операции резервного копирования базы данных

Согласно канонам реляционной модели данных все кортежи одного отношения (т.е. все строки одной таблицы) имеют одинаковую арность (количество атрибутов/столбцов), и при этом соответствующие атрибуты кортежей содержат данные одного типа. Именно в этом смысле верно утверждение о "прямоугольности" реляционных таблиц: действительно,

все строки таблицы имеют одинаковую длину, равную сумме длин типов данных всех ее столбцов.

Например, все строки таблицы `MyTable` (листинг 1.2) имеют одинаковую длину, равную 72 байта ($3*4 + 60$), и эта таблица действительно является "прямоугольной" как на логическом уровне, так и на уровне физической модели данных, так как для описания всех ее атрибутов были использованы типы данных постоянной длины, в том числе и строковый тип `CHAR(60)` для атрибута `FullName`.

Скорее всего, использование типа `CHAR(60)` в данном случае было ошибочным, так как независимо от реальной длины текста атрибут `FullName` будет занимать ровно 60 байтов в каждой строке таблицы. Было бы эффективнее (с точки зрения экономии памяти) использовать вместо `CHAR(60)` тип данных переменной длины `VARCHAR(60)`, и тогда затраты памяти на хранение экземпляров атрибута `FullName` будут минимальными, но таблица при этом потеряет свою "прямоугольность", что явно усложнит систему хранения адресов ее строк в файловых страницах типа `DATA`.

Приведенная на рисунке 1.2 структура страницы обеспечивает эффективную адресацию строк переменной длины.

Массив указателей на слоты, расположенный в хвостовике страницы, содержит целые числа, трактуемые как смещения от начала страницы до начала соответствующего слота. Первый элемент массива указателей всегда содержит число 96 – это указатель на первый слот, начинающийся по смещению 96 байтов от начала страницы (сразу после её заголовка). Когда страница пуста, все остальные указатели в этом массиве отсутствуют.

При записи на страницу первой строки (длина которой, естественно, известна) она помещается в первый слот, а в массив указателей заносится ссылка на начало второго слота (равное сумме длины записанной строки и числа 96).

При записи второй и всех последующих строк сканируется (справа налево) массив указателей, определяется номер очередного еще не заполненного слота, этот слот заполняется, а в массив указателей заносится ссылка на начало следующего слота. Процесс заполнения страницы может продолжаться до тех пор, пока обе эти структуры (массив слотов и массив указателей на них) не "встретятся" – в этот момент страница получит статус заполненной на 100%.

Если задан номер слота `SlotNum`, в котором размещена искомая строка таблицы, то адрес этого слота легко определяется прямым доступом к соответствующему элементу массива указателей.

Таким образом, адрес строки таблицы (`RID` – Row Identifier) – это агрегат «`SysFiles.PageNum`», используемый для адресации страницы в файловой структуре базы данных, дополненный номером соответствующего слота страницы: `RID = SysFiles.PageNum.SlotNum`. Именно такие `RID`-адреса используются в индексах для обеспечения прямого доступа к строкам таблиц, содержащих искомые значения ключевых полей таблицы.

Страницы типа **Text/Image** используются для хранения "больших объектов" (`LOB` – Large Object) – значений столбцов таблиц, превышающих объем файловой страницы (длинных текстов, графических объектов с хорошим разрешением и пр.). Если, например, столбец таблицы имеет один из таких типов данных, и при этом в какой-либо строке таблицы экземпляр данных этого типа имеет действительно большой объем, для хранения этого экземпляра выделяется отдельная страница типа **Text/Image**, а в соответствующем слоте основной страницы типа **Data** сохраняется указатель на эту страницу. Если для хранения экземпляра оказывается недостаточно одной страницы типа **Text/Image**, сервер дополнительно выделяет необходимое количество таких страниц, связывая их в линейный список специальными указателями (`NextPage` и `PrevPage` в заголовках страниц).

Страницы типа **Text/Image** используются также для хранения данных типов переменной длины в условиях, когда размер строки на основной странице типа **Data** превышает максимально-допустимый размер 8060 байтов. Если в результате вставки или обновления данных в таблице размер строки выходит за указанный предел, происходит перемещение данных столбца на страницу типа **Text/Image** с сохранением указателя на эту страницу в соответствующем слоте основной страницы типа **Data**. Если в дальнейшем размер строки уменьшается, данные перемещаются обратно на исходную страницу типа **Data**.

Страницы типа **IAM** (Index Allocation Map) формируются для каждого логического объекта базы данных (таблицы или индекса) в момент вставки в этот объект первого экземпляра данных. **IAM**-страница содер-

жит информацию о номерах всех экстентов, содержащих страницы логического объекта, и представляет собой битовую карту размером около 8 Кб, в которой номер позиции каждого бита ассоциируется с номером экстента: если i -й экстент файла содержит хотя бы одну страницу, владельцем которой является логический объект, то в **IAM**-странице этого объекта **IAM[i]=1**.

Нетрудно подсчитать, что одной **IAM**-страницы будет достаточно для представления таблицы объемом около 4 Гб. Если таблица имеет больший размер, она получит дополнительные **IAM**-страницы, связанные в линейный список указателями NextPage и PrevPage в заголовках этих страниц.

Если известен номер первой **IAM**-страницы, владельцем которой является таблица базы данных, легко определить номера всех используемых таблицей экстентов и получить доступ к страницам и строкам этой таблицы – именно такой подход и реализуется в низкоуровневом методе TableScan(), используемом для выборки строк из некластеризованных таблиц.

Страницы типа **GAM** (Global Allocation Map) и **SGAM** (Shared Global Allocation Map) содержат глобальные битовые карты (объемом около 8000 байтов), в которых каждый бит несет определенную информацию о соответствующем экстенте файла.

Битовая карта **GAM** содержит информацию о свободных или занятых экстентах: если **GAM[i]=1**, то i -й экстент свободен, в противном случае хотя бы одна страница этого экстента занята.

Битовая карта **SGAM** содержит информацию о типах экстентов и степени их заполнения: если **SGAM[i]=1**, то i -й экстент используется как смешанный и при этом имеет хотя бы одну свободную страницу; в противном случае этот экстент либо является однородным, либо смешанным, но полностью занятым.

Таким образом, двухбитовый код i -го экстента (**GAM[i],SGAM[i]**) несет информацию о его типе и степени заполнения (таблица 1.2), что позволяет серверу реализовывать несложные алгоритмы поиска свободного пространства при вставке строк в таблицы базы данных:

- если **GAM[i]=1**, то i -й экстент свободен, его тип еще не определен, и значение бита **SGAM[i]** может быть любым;

- если серверу требуется свободный однородный экстенст (например, для массового заполнения строк большой таблицы), производится поиск $GAM[i]=1$ и, после заполнения этого экстенста, для него устанавливается $GAM[i]=0$ и $SGAM[i]=0$;
- для поиска смешанного экстенста со свободными страницами сканируются обе битовые карты, и выбирается экстенст с кодом (0;1). После 100% -го заполнения выбранного экстенста для него устанавливается $SGAM[i]=0$;
- при отсутствии смешанного экстенста со свободными страницами выбирается свободный экстенст ($GAM[i]=1$), после его частичного заполнения устанавливаются $GAM[i]=0$ и $SGAM[i]=1$. В результате этот экстенст получит статус смешенного экстенста, имеющего свободные страницы;
- при освобождении i -го экстенста (например, в результате массового удаления строк соответствующих таблиц), для него устанавливается $GAM[i]=0$, и он получает статус свободного экстенста.

•

Таблица 1.2 – Кодирование состояний экстенстов

Состояние i -го экстенста	$GAM [i]$	$SGAM [i]$
Свободен, в текущий момент не используется	1	0
Однородный или заполненный смешанный	0	0
Смешанный со свободными страницами	0	1

Пара страниц $GAM/SGAM$ описывают файл размером около 4 Гб. Для файлов большего размера формируются дополнительные пары $GAM/SGAM$ - страниц, связанные в линейные списки указателями NextPage и PrevPage в заголовках соответствующих страниц.

Основное назначение служебных страниц типа PFS (Page Free Space) – хранение информации о степени заполнения страниц файла базы данных. Тело PFS -страницы состоит из единственного слота размером 8092 байта, содержимое которого трактуется как числовой массив байтового типа, каждый элемент которого $PFS[i]$ представляет 8-битовый код i -й страницы файла. Младшие 4 бита этого кода определяют степень заполнения соответствующей страницы (таблица 1.3), а старшие биты кода

несут дополнительную информацию о странице (например, о ее типе и принадлежности смешанному или однородному экстену).

Таблица 1.3 – Кодирование степени заполнения страниц

Состояние <i>i</i> -й страницы	PFS [i]
Страница не занята	0
Страница заполнена от 1 до 50%	1
Страница заполнена от 51 до 80%	2
Страница заполнена от 81 до 95%	3
Страница заполнена от 96 до 100%	4

Одна **PFS**-страница описывает файл размером чуть меньше 64 Мб. Для файлов большего размера выделяется необходимое количество дополнительных **PFS**-страниц, образующих линейный список с помощью указателей NextPage и PrevPage в заголовках этих страниц.

На рисунке 1.3 представлена типовая структура нулевого экстента файла: нулевая страница – это заголовок файла, первая из **PFS**-страниц файла всегда имеет порядковый номер «1», затем расположены другие служебные страницы, и далее – все остальные страницы файла.



Рисунок 1.3 – Стандартное расположение служебных страниц в начальной области файла

1.4 Служебные структуры данных

Информация о принадлежности файловых страниц логическим объектам базы данных и об адресах этих страниц содержится в системной таблице SysIndexes, входящей в состав системного каталога базы данных. Каждая строка этой таблицы представляет один логический объект (таблицу или индекс), поля **First**, **Root** и **FirstIAM** содержат идентификаторы («адреса») соответственно первой страницы типа **Data** (для объектов – таблиц), корневой индексной страницы (для объектов – индексов) и первой **IAM**-страницы (для любых логических объектов).

Идентификатор страницы **ID_Page** хранится в формате **BINARY (6)** и представляет собой агрегат из двух чисел, в котором младшие два байта представляют идентификатор файла (**ID_File**), а старшие четыре байта – порядковый номер страницы (**PageNum**) в этом файле: **ID_Page = ID_File.PageNum**.

1.5 Программные средства анализа структуры файла данных

Хранимая процедура **sp_spaceused** принимает имя таблицы и возвращает пять ее параметров: количество строк (**rows**), общий объем (в килобайтах) зарезервированного дискового пространства (**reserved**), в том числе – занятого страницами данных (**data**), индексными (включая IAM) страницами (**index_size**) и неиспользуемыми (**unused**) страницами, расположенными в однородных частично заполненных экстендах.

Команда **EXTENTINFO** системной утилиты DBCC позволяет получить более детальную информацию о страницах, занятых данными таблицы и всеми ее индексами: номера страниц, степень их заполнения и количество страниц в экстендах.

Команда **PAGE** позволяет получить информацию о содержимом страницы и отображает данные ее заголовка, содержимое всех слотов и массив указателей на строки таблицы (рисунок 1.2).

Приведенный ниже пример (листинг 1.3) содержит пакет SQL-инструкций, иллюстрирующих использование средств анализа структуры файла базы данных, а на рисунке 1.4 приведен результат выполнения первых трех инструкций этого пакета.

```

USE MyDB_2
SELECT ID,IndID,root,first,firstiam
FROM sysindexes WHERE ID=Object_ID('MyTable_6')
GO
EXEC sp_spaceused MyTable_6
GO
DBCC EXTENTINFO(MyDB_2,MyTable_6,-1)
GO
DBCC TRACEON (3604)
DBCC PAGE ('MyDB_2'1,1,2)
GO

```

Листинг 1.3 – Пример использования средств анализа структуры файла базы данных

ID	IndID	first	root	firstiam
1	2105058535	0	0x990000000100	0x000000000000
2	2105058535	2	0xAA0000000100	0xAD0000000100
3	2105058535	3	0xC32700000100	0xC62700000100
4	2105058535	4	0xDC2700000100	0xDF2700000100

name	rows	reserved	data	index_size	unused
MyTable_6	10000	80672 KB	80000 KB	512 KB	160 KB

file_id	page_id	pg_alloc	ext_size	object_id	index_id	partition_number	partition_id	iam_chain_type	pfs_bytes
1	1	170	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
2	1	172	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
3	1	173	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
4	1	174	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
5	1	175	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
6	1	10176	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
7	1	10177	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
8	1	10178	1	2105058535	2	1	72057594038845440	In-row data	0x6000000000000000
9	1	10184	8	2105058535	2	1	72057594038845440	In-row data	0x4040404040404040
10	1	10192	4	2105058535	2	1	72057594038845440	In-row data	0x4040404000000000

Рисунок 1.4 – Результат выполнения примера (листинг 1.3)

2.1 Доступ к неупорядоченным данным

Вспомним, что реляционная модель данных не гарантирует упорядоченности расположения строк в таблице, и, если пользователю необходимо представить выборку данных, отсортированную по значениям её столбцов, программист должен явно указать на это в разделе `Order By` соответствующего SQL-запроса.

На уровне физической модели данных строки реляционной таблицы хранятся в файловых страницах типа `DATA`, объединенных (по умолчанию) в структуры типа «куча» (`heap`), при этом файловые страницы, принадлежащие одной таблице, могут располагаться в произвольных местах файла (или нескольких файлов) базы данных.

Информация о принадлежности групп файловых страниц (экстентов) определенной таблице хранится в специальной IAM-странице, также принадлежащей таблице, а номер IAM-страницы хранится в системном каталоге базы данных (рисунок 2.1).

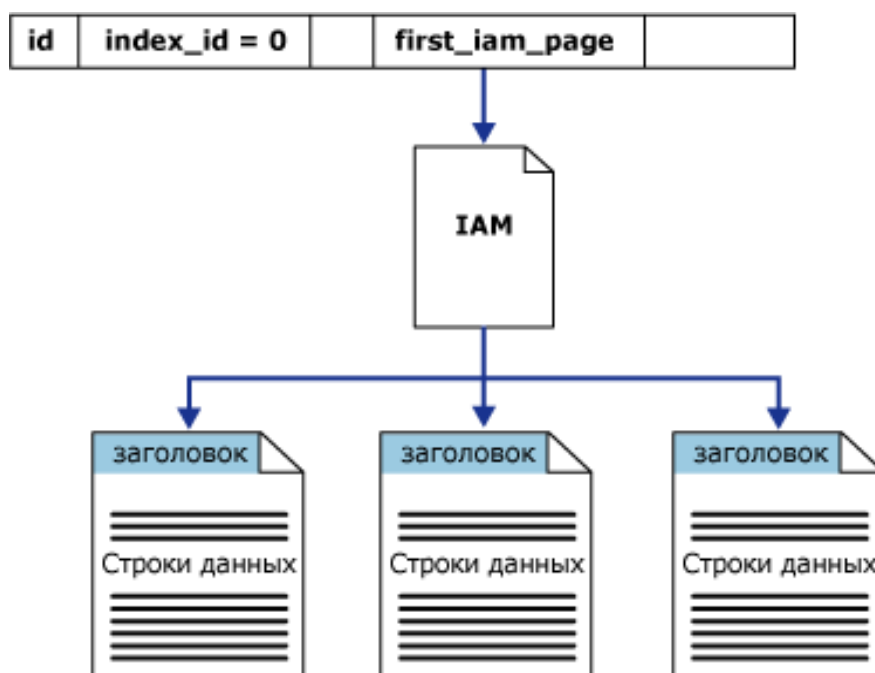


Рисунок 2.1 – Упрощенная схема доступа к данным типа «куча» методом последовательного сканирования

Таким образом, в условиях отсутствия дополнительной информации адресного характера, единственным методом поиска в «куче» строк таб-

лицы, удовлетворяющих заданному критерию отбора (диапазону значений указанных в запросе столбцов), является метод последовательного сканирования «кучи» (TableScan), работающий по следующему алгоритму:

- чтение системного каталога, определение номера IAM-страницы;
- загрузка IAM-страницы, определение номеров экстентов, включающих файловые страницы, принадлежащие таблице;
- последовательная загрузка каждой страницы выбранных экстентов в оперативную память;
- выполнение циклической процедуры сравнения значений столбцов с критерием отбора в каждой загруженной странице;
- формирование результирующей выборки.

Стоимость такого метода весьма высока, она пропорциональна количеству занятых таблицей страниц и не зависит от степени селективности предиката выборки. Например, единственная строка таблицы, удовлетворяющая критерию поиска, может оказаться как в первой, так и в последней странице, но в любом случае придется просканировать все страницы, принадлежащие таблице.

2.2 Линейный индекс

Естественным способом снижения стоимости выборки данных из «кучи» является отказ от технологии последовательного сканирования и применение методов прямого доступа, предусматривающих наличие специальных указателей (адресных ссылок) на строки таблиц, соответствующие критерию выборки. Применение таких методов требует наличия служебных структур данных (индексов), формируемых на базе основных таблиц базы данных.

Простейшей индексной структурой является линейный индекс, представляющий собой классический двунаправленный список, реализованный в виде бинарной таблицы, один из столбцов которой содержит все значения индексируемого поля (называемого «ключом индекса»), а второй – указатели на соответствующие строки индексируемой таблицы.

Отметим основные особенности устройства индексов:

- 1) если актуальна задача ускорения поиска по нескольким полям таблицы (ключам), то для одной таблицы может потребоваться несколько индексов – по одному для каждого поискового ключа;

- 2) мощность (количество строк) индексной таблицы соизмерима с мощностью основной (индексируемой) таблицы базы данных, а в предельном случае обе эти таблицы имеют одинаковую мощность;
- 3) физический размер индексной таблицы существенно (в сотни раз) меньше размера основной индексируемой таблицы и, соответственно, индекс занимает существенно меньше файловых страниц по сравнению с индексируемой таблицей;
- 4) строки индексной таблицы отсортированы по значению ключа индекса, что позволяет существенно ускорить поиск указателя по значению ключа непосредственно в самой индексной странице, применяя, например, метод дихотомии;
- 5) если индексируемое поле таблицы не является уникальным, одинаковые его значения могут многократно встречаться в различных строках таблицы и, как следствие, могут оказаться в нескольких файловых страницах. В этом случае поле ключа индекса в индексной таблице будет содержать множества значений-дубликатов, причем пара значений «ключ – ссылка» будет оставаться уникальной в пределах всей индексной таблицы;
- 6) индексные страницы не хранятся в структуре типа «кучи» – все страницы одного индекса организованы (логически) в линейный список (в порядке возрастания или убывания значения ключа индекса), реализуемый с помощью специальных указателей, присутствующих в заголовках всех индексных страниц;
- 7) операции вставки/удаления строк таблицы, а также операции модификации индексированных столбцов, требуют оперативной перестройки соответствующих индексов этой таблицы, что может негативно сказаться на показателях производительности.

Ниже приведено пошаговое описание алгоритма поиска строк таблицы по значению линейно-индексированного столбца в структуре данных типа «куча». Этот алгоритм актуален как для случая индексирования таблицы по уникальному ключу, так и для случая, когда значение индексируемого столбца многократно дублируется в разных строках таблицы.

- последовательная загрузка всех индексных страниц (начиная с первой и далее до последней по указателям на следующий элемент списка), в каждой из которых:

- выполняется циклическая процедура сравнения значений ключей индекса с критерием отбора и, при их совпадении, формируется массив указателей на страницы и строки таблицы.
- последовательная загрузка страниц «кучи», номера которых попали в массив указателей, в каждой из которых:
 - выбираются строки таблицы в соответствии с массивом указателей;
 - формируется результирующая выборка.

Пример

Размер файловой страницы - 8 kb (2^{13} b)

Индексируемая таблица базы данных:

мощность – около 16 млн ($\sim 2^{24}$) строк;

средняя длина одной строки – 2 kb,

длина индексируемого поля – 8 b;

всего строк таблицы в одной странице – 4,

всего страниц, занятых таблицей – около 4-х млн ($2^{24}/2^2$).

Индексная таблица:

мощность – около 16 млн ($\sim 2^{24}$) строк;

длина строки – 16 b (ключ индекса 8 b + указатель 8 b);

строк индекса в одной индексной странице – 512 ($2^{13}/2^4$);

всего индексных страниц – порядка 32 тысяч ($2^{24}/2^9$).

Сравним два рассмотренных выше метода поиска данных в «куче» по критерию стоимости реализации простого SQL-запроса:

`Select * From T1 Where T1.Col = const,`

где T1 – таблица из рассмотренного выше примера;

const – произвольное числовое значение.

В модели стоимости будем учитывать только операции доступа к внешней памяти для загрузки файловых страниц.

Метод полного прямого сканирования дает оценку стоимости в четыре миллиона единиц, независимо от степени селективности запроса Sel, равной количеству строк таблицы, в которых T1.Col=const.

Метод доступа с использованием линейного индекса потребует последовательной загрузки некоторого количества K_T индексных страниц, до

тех пор, пока значение ключа индекса не выйдет за пределы диапазона поиска, и дополнительно – загрузки тех K_T страниц индексированной таблицы $T1$, ссылки на которые были найдены ($Ind.Col=const$) при обработке индексных страниц. Количество $K_T(Se1)$ таких «дополнительных» страниц зависит от степени селективности запроса $Se1$, а общая оценка стоимости для этого метода составит $K_I + K_T(Se1)$.

Пессимистическая оценка количества загружаемых индексных страниц $K_I = 32768$ (когда совпадение будет найдено только в последней из них), оптимистическая оценка $K_I = 1$, а усредненная оценка $K_I \cong 16000$.

Пессимистическая оценка количества загружаемых страниц индексированной таблицы $K_T \cong 4000\ 000$, что соответствует ситуации, когда в таблице $T1$ достаточно много (больше, чем страниц в «куче») одинаковых и равных **const** значений столбца **Col**, и при этом они равномерно распределены по всем файловым страницам этой таблицы.

Ясно, что вероятность второго пессимистического случая крайне мала, а вероятность одновременного проявления двух рассмотренных выше пессимистических ситуаций равна нулю (что в целом должно нам немного добавить оптимизма в оценке эффективности линейных индексов).

Завершая краткий обзор линейных индексных структур, следует отметить, что линейные индексы не нашли практического применения в системах управления реляционными базами данных по причине своей низкой эффективности.

В рассмотренном примере усредненная оценка количества операций чтения индексных страниц, необходимого для выборки данных из таблицы мощностью в 16 миллионов строк, составила 16 тысяч единиц. Это, конечно, существенно лучше, чем оценка в 4 миллиона единиц для метода полного сканирования, но все-таки весьма посредственно по сравнению с многоуровневыми индексными структурами, построенными на базе сбалансированных деревьев.

2.3 Многоуровневый иерархический индекс

Основная причина низкой эффективности линейного индекса заключается в его «линейности». Обеспечив возможность прямого (адресного) доступа к файловым страницам «кучи», сам индекс остался классической структурой последовательного доступа к данным – линейным списком, на

котором, как известно, заданы два основных «поисковых» метода: переход к следующему и переход к предыдущему элементам списка.

В отличие от линейных индексов, иерархические («древовидные») структуры данных обеспечивают прямой доступ не только к страницам «кучи», но и к самим индексным страницам, организованным на каждом уровне индекса в последовательную списковую структуру.

Порядком индекса (P) будем называть количество строк индекса, помещающихся в одной индексной странице. Для нашего примера $P=512$ (2^9), и при этом весь линейный индекс занимает 32768 (2^{15}) файловых страниц.

Рассмотрим процесс построения многоуровневого иерархического индекса на базе существующего линейного индекса, который теперь будем считать конечным («листовым») уровнем многоуровневого индекса.

Все индексные страницы и все строки внутри индексных страниц упорядочены по значению ключа индекса, и каждая индексная строка листового уровня содержит указатель на соответствующую строку таблицы базы данных, включающий номер файловой страницы из «кучи» и номер строки этой страницы.

Построим еще один уровень иерархического индекса – линейный индекс, страницы которого будут содержать указатели на индексные страницы листового уровня. Для нашего примера потребуется 64 индексных страницы для этого «предлистового» уровня ($2^{15}/2^9$), так как каждая индексная страница может содержать $P=512$ (2^9) ссылок на страницы листового уровня, а всего на листовом уровне 32768 (2^{15}) страниц.

Продолжим построение многоуровневого индекса – создадим еще один уровень, страницы которого будут содержать указатели на индексные страницы предлистового уровня. Здесь нам будет достаточно одной индексной страницы (с восьмикратным запасом, так как указателей надо всего 64 , а порядок индекса $P=512$). Эта страница называется корневой страницей многоуровневого индекса или, более кратко, – «корнем дерева».

В результате построен многоуровневый индекс, страницы которого на каждом уровне связаны в двунаправленные списки, а переходы от страниц верхних (родительских) уровней на страницы нижних (дочерних) уровней выполняются по прямым ссылкам в соответствии со значениями ключей индекса. Построенный индекс содержит 32833 страницы ($1+64+32768$), распределенные по трем уровням индекса.

Количество уровней индекса (**H**) называют «глубиной индекса», которая зависит от количества страниц «кучи» **N** и порядка индекса **P**: $H = \text{Log}_P(N) + 1$. Для нашего примера $H = \text{Log}_{512}(32768) + 1 \cong 2,665 \rightarrow 3$.

Уровни индекса принято последовательно нумеровать в порядке возрастания, начиная от корневого (**Level=0**) и заканчивая листовым (**Level=H-1**) уровнем. На корневом (нулевом) уровне индекса всегда находится единственная индексная страница: $N[0]=1$, а количество страниц $N[i]$ на каждом дочернем уровне зависит от количества значений ключа индекса на всех страницах родительского уровня и, в частности, от степени заполнения индексных страниц. Максимально возможное количество страниц на i -м* уровне индекса (при 100%-м заполнении всех индексных страниц) $N[i]=P^i$.

Ниже приведено описание алгоритма реализации метода поиска строки в «куче» по значению индексированного столбца, включающего «спуск» от корня индекса до листового уровня с последующей загрузкой страниц «кучи», указатели на которые были получены на листовом уровне индекса (рисунок 2.2):

- получение адреса корневой страницы индекса запросом в системном каталоге базы данных (*адрес страницы = file_ID.page_Num*);
- загрузка корневой индексной страницы, поиск индексной строки, удовлетворяющей критерию отбора по ключу индекса, определение адреса индексной страницы нижележащего уровня индекса;
- загрузка индексной страницы промежуточного уровня индекса;
- поиск индексной строки, удовлетворяющей критерию отбора по ключу индекса, определение адреса индексной страницы нижележащего уровня;
- и т.д. по всем промежуточным уровням индекса;
- загрузка индексной строки листового уровня:
 - выполнение циклической процедуры сравнения значений ключей индекса с критерием отбора и формирование массива указателей на строки таблицы в «куче» (*указатель на строку = file_ID.page_Num.slot_Num*);

* В системе MS SQL-Server принята иная нумерация уровней индекса: нулевым считается листовый уровень, а корневой уровень индекса имеет наибольший номер.

- последовательная загрузка в оперативную память страниц «кучи», номера которых попали в массив указателей;
 - в каждой из загруженных страниц: выборка строк таблицы в соответствии с массивом указателей;
- формирование результирующей выборки.

Как видно из рисунка 2.2 и приведенного выше описания алгоритма «спуска по дереву», для получения ссылки на страницу «кучи», содержащую уникальное значение искомого ключа, потребуется загрузка одной индексной страницы на каждом уровне индекса (в нашем примере $H=3$) и дополнительно – загрузка одной страницы «кучи». Общая оценка стоимости запроса в этом случае составит 4 единицы (сравните с оценкой 16 000 единиц для линейного индекса).

Если многоуровневый индекс построен по неуникальному столбцу таблицы, его преимущество по сравнению с линейным индексом будет не настолько радикальным: потребуется загрузка одной индексной страницы на каждом нелистовом уровне индекса (в нашем примере – 2 страницы) и дополнительно – загрузка K_I страниц листового уровня индекса, количество которых зависит от степени селективности запроса Sel и порядка индекса P : $K_I = Sel/P$.

Если, например, $P=512$ (как в нашем примере), а степень селективности предиката выборки по индексируемому ключу составляет 2%, то при мощности таблицы в 16 миллионов строк получим значение оценки стоимости $K_I = 320000/512 \cong 640$, что существенно меньше 16 000 единиц для линейного индекса.

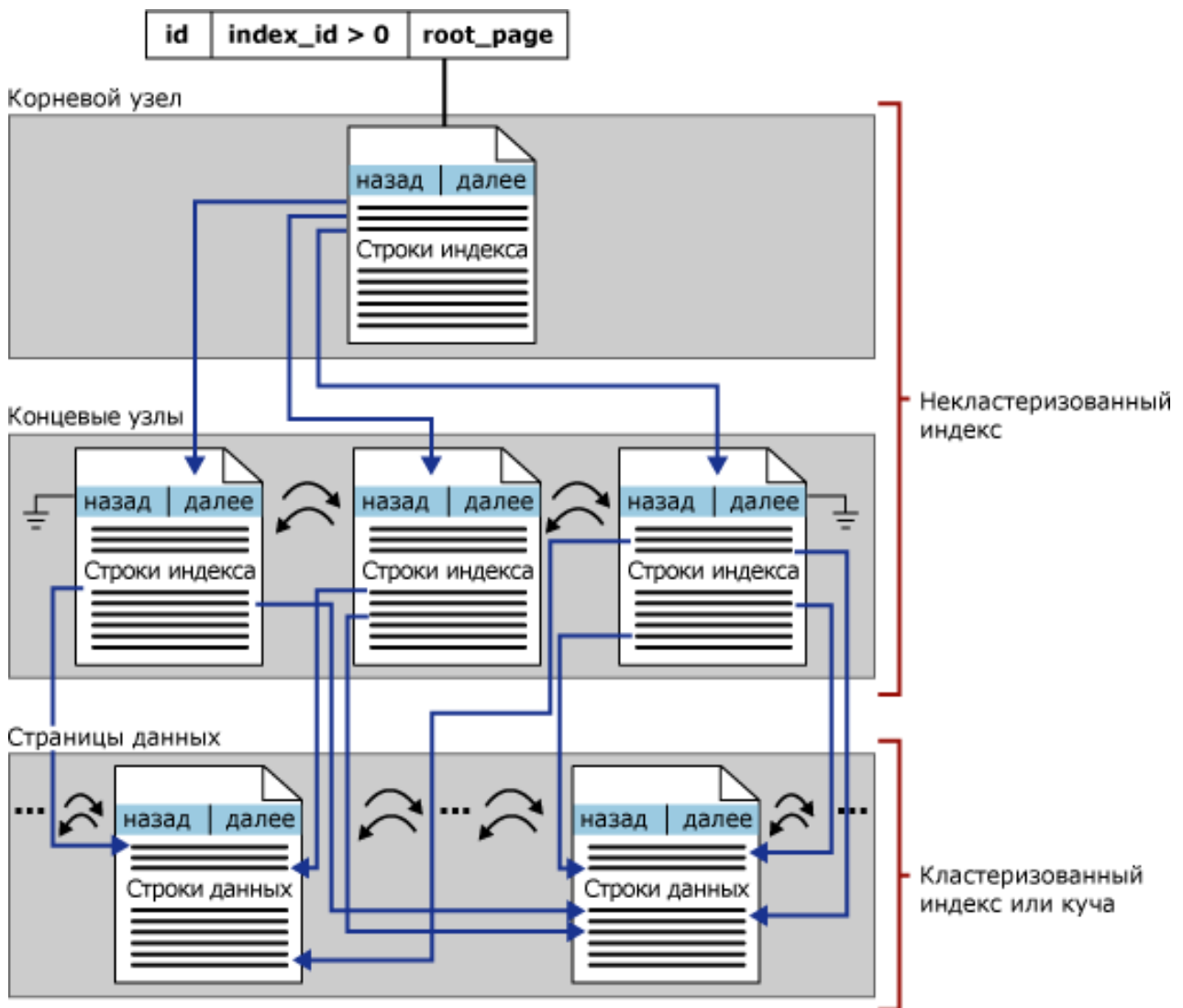


Рисунок 2.2 – Схема доступа к данным типа «куча» методом спуска по многоуровневому индексу

2.4 Кластеризованный многоуровневый индекс

Рассмотренный выше многоуровневый индекс – автономный объект базы данных, существующий отдельно от структуры «кучи», на базе которой он был создан. Такие индексы (в терминологии MS SQL-Server) принято называть «некластеризованными» (nonclustered), подчеркивая тот факт, что данные индексированной таблицы и индексы, связанные с этой таблицей, хранятся отдельно друг от друга, т.е. в разных «кластерах».

Некластеризованные индексы существенно ускоряют поиск данных по уникальным ключам, но при высокой степени селективности предиката выборки их эффективность резко падает.

Некластеризованные индексы хорошо работают в запросах с точечными условиями выборки (`Select ... Where Col=Const`) и гораздо

хуже – если условие выборки содержит диапазон значений ключевого поля (`Select...Where Col Between Const1 And Const2`).

Некластеризованные индексы недостаточно эффективны при выполнении запросов, требующих соединения таблиц или группировки их строк, реализация которых основана на алгоритмах сортировки и слияния данных.

Кластеризованный (`clustered`) индекс – это структура данных, объединяющая в единый «кластер» и «кучу», и собственно индекс. В кластеризованном индексе файловые страницы, содержащие строки индексируемой таблицы, перестают быть «кучей» - они упорядочиваются по значению ключа (как внутри страниц, так и между страницами) и в таком виде занимают место листового уровня многоуровневого индекса, страницы которого образуют двунаправленный список, обеспечивающий возможность быстрого перехода на последующую или предыдущую страницу. Остальные (верхние) уровни кластеризованного индекса устроены так же, как и в некластеризованных индексах (рисунок 2.3).

Так как данные таблиц в кластеризованном индексе хранятся в уже упорядоченном виде, такие индексы оказываются эффективными в запросах с диапазонными условиями выборки, а также при реализации методов, требующих сортировки данных.

Физическая упорядоченность строк таблицы в кластеризованном индексе накладывает существенное ограничение на его использование: в одной таблице может быть создано не более одного такого индекса. Как правило, кластеризованный индекс создается по одному из уникальных столбцов таблицы, а при наличии в таблице первичного ключа сервер автоматически (по умолчанию) создает по нему кластеризованный индекс.

Если кластеризованный индекс создается до заполнения таблицы, то при последующей вставке строк они сразу упорядочиваются по значению ключа индекса, и формируется листовый уровень индекса как двусвязный список страниц.

Если кластеризованный индекс создается на базе уже заполненной таблицы, сервер автоматически перестраивает «кучу» в «список» листового уровня индекса и затем достраивает все остальные его уровни, вплоть до корневого.

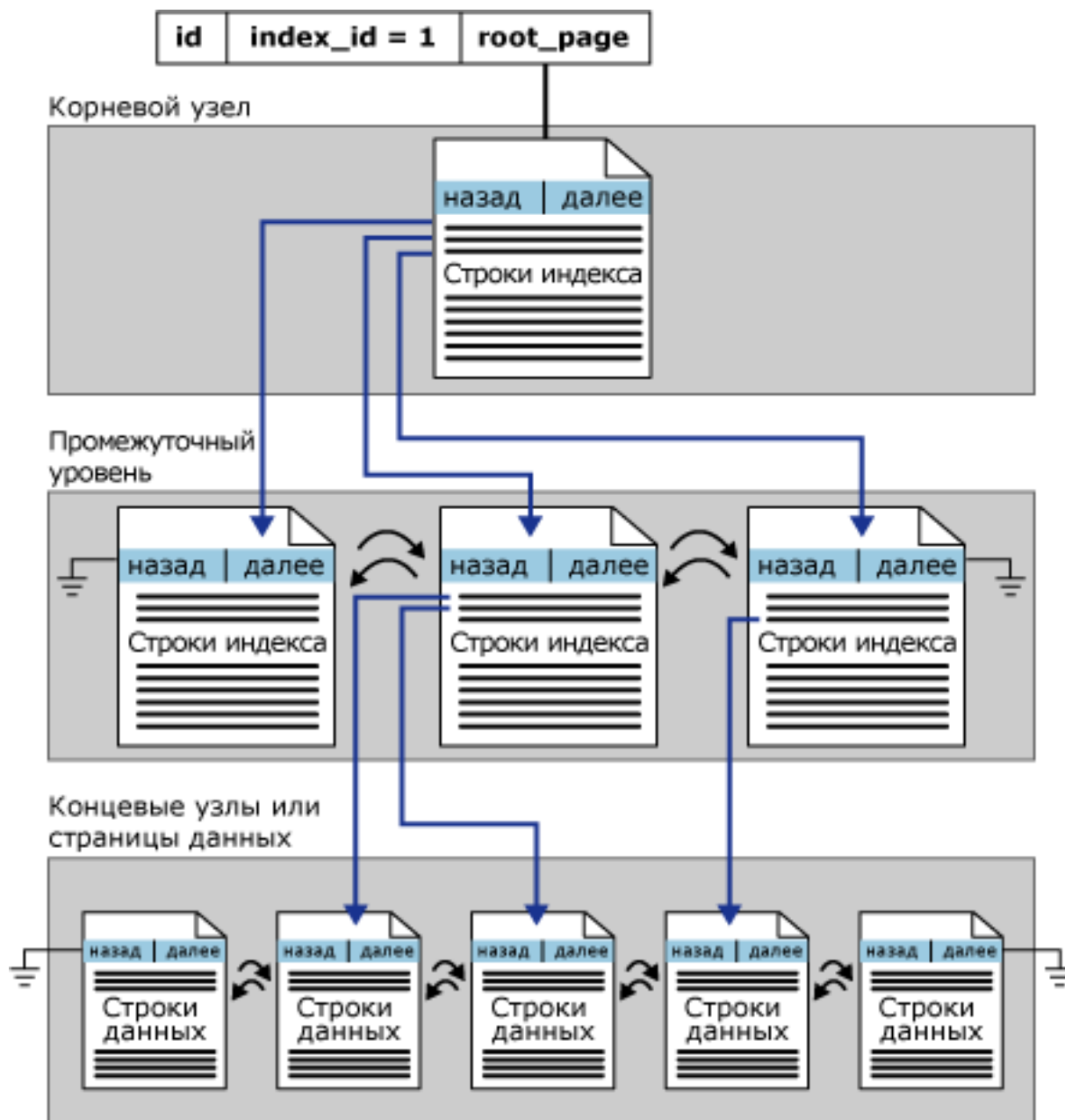


Рисунок 2.3 – Схема доступа к данным типа «кластеризованный многоуровневый индекс»

Если к этому моменту в таблице уже существовали некластеризованные индексы, то все они будут автоматически перестроены, так как в этом случае изменяется формат указателя листового уровня некластеризованного индекса: вместо ссылки на строку таблицы (номер слота файловой страницы) этот указатель теперь будет содержать значение ключа кластеризованного индекса, соответствующего этой строке (которое однозначно определяет номер слота).

Такой подход несущественно увеличивает стоимость поиска по значению ключа индекса, так как требует дополнительного «спуска» по кла-

стеризованному индексу, однако он дает существенную экономию при реализации процедур реиндексации – перестройки индексов таблицы после модификации ее данных в условиях, когда в таблице, кроме кластеризованного индекса, создано большое количество некластеризованных индексов.

Если в таблице модифицируется значение неиндексированного столбца, и это приводит к перераспределению данных между страницами кластеризованного индекса, это никак не скажется на некластеризованных индексах, так как в них отсутствуют указатели на физическое размещение страниц. Если же в аналогичной ситуации изменение коснулось одного из индексированных столбцов, то это потребует перестройки только этого индекса.

2.5 Фактор заполнения индексных страниц

Как было показано выше, порядок индекса P , определяющий количество индексных строк на одной индексной странице, влияет на глубину индекса $H = \text{Log}_P(N) + 1$, и, следовательно, на «стоимость» алгоритма спуска от корневого до листового уровня индекса.

Если оптимизировать индексные структуры по критерию производительности выполнения операций поиска данных, следует стремиться к увеличению порядка индекса P – как за счет минимизации длины индексных записей (ключ+ссылка), так и за счет максимального использования пространства индексных страниц, то есть 100%-го заполнения этих страниц индексными записями.

Однако, 100%-е заполнение индексных страниц приведет к резкому снижению производительности выполнения операций модификации данных, так как вставка и удаление строк таблицы потребует оперативной перестройки всех её индексов, связанной с массовым проведением весьма дорогостоящих операций расщепления и слияния индексных страниц на всех уровнях индексов.

В этих условиях администратор базы данных, учитывая специфику структуры базы данных, характер типовых запросов и результаты мониторинга работы пользователей, может принять компромиссное решение и установить требуемое значение *фактора заполнения* индексных страниц в процентах от значения порядка индекса P (например, 20% – при интен-

сивном выполнении операций модификации данных и 80% – в противном случае).

Сервер баз данных будет учитывать установленное значение фактора заполнения при начальном формировании индексов, то есть, фактически, будет занижать допустимый порядок индекса, что приведет к увеличению глубины индекса и, как следствие – к снижению производительности выполнения операций выборки данных.

В процессе эксплуатации базы данных реальное значение фактора заполнения будет изменяться (вспомним про расщепление и слияние страниц при вставке и удалении строк таблицы), однако, при каждом выполнении *операции реиндексации* сервер автоматически перестроит индексы в соответствии с установленным начальным значением фактора заполнения.

По умолчанию, сервер устанавливает фактор заполнения индексных страниц на уровне 50%.

2.6 Рекомендации по использованию индексов

Прежде, чем принимать решение об индексировании таблиц, следует провести детальный анализ и попытаться понять, индексирование каких столбцов позволит повысить производительность работы базы данных. Для этого рекомендуется проанализировать активность пользователей, собрать и обработать статистику выполняемых запросов к базе данных, обращая внимание на интенсивность выполнения поисковых и модифицирующих запросов, наборы соединяемых в запросах таблиц, критерии поиска данных в таблицах, условия группировки и сортировки данных и т.д.

Кластеризованные индексы

- по умолчанию, сервер автоматически создаст кластеризованный индекс для первичного ключа таблицы, и с таким его решением следует согласиться в подавляющем большинстве случаев;
- если создается подчиненная таблица, будет полезно создать кластеризованный индекс для составного ключа, включающего пару «первичный ключ – внешний ключ», так как в процессе эксплуатации базы данных, как правило, будут часто выполняться запросы с эквисоединением таблиц, требующие сортировки данных;
- если планируется создать единственный индекс в таблице, пусть он будет кластеризованным;

- если не планируется создавать в таблице первичный ключ (и такое тоже случается), следует создать кластеризованный индекс по столбцу, который часто используется в условиях группировки и операторах сравнения **between**, **>**, **>=**, **<=** или **<**;

- не рекомендуется создавать кластеризованные индексы для столбцов с «длинными» типами данных: во-первых, это приведет к уменьшению порядка самого этого индекса и, как следствие, к увеличению его глубины, а, во-вторых, что гораздо важнее, этот «длинный тип» будет многократно дублироваться в качестве конечной ссылки на страницах листовых уровней всех некластеризованных индексов этой таблицы, что также негативно скажется на их структуре.

Некластеризованные индексы

- некластеризованные индексы целесообразно создавать для тех столбцов таблицы, которые участвуют в SQL-запросах в разделах **Where**, **Having**, **Group BY**, а также в условиях соединения таблиц **Join**;

- не следует применять такие индексы для часто изменяемых столбцов таблиц: ускорение поиска данных станет незаметным на фоне существенных временных потерь на реиндексацию после каждого такого изменения;

- не следует также применять некластеризованные индексы для таблиц малой мощности: при небольшом количестве файловых страниц метод полного сканирования «кучи» может оказаться более эффективным по сравнению с методами поиска по ключу индекса;

- наибольший эффект от применения некластеризованного индекса достигается, когда индексируемый столбец содержит относительно много различных (уникальных) значений, а также при небольшой степени селективности предиката выборки (т.е. когда количество строк, соответствующих критерию отбора по ключу индекса, значительно меньше общего количества строк в таблице).

Глава 3

ОПТИМИЗАЦИЯ ПРОЦЕДУРНЫХ ПЛАНОВ ВЫПОЛНЕНИЯ SQL-ЗАПРОСОВ

Повышение производительности информационных систем (ИС), активно использующих операции доступа к данным, – комплексная задача, решение которой не ограничивается оптимизацией исключительно базы данных и может затрагивать различные аспекты функционирования ИС – организационные, технические, архитектурные, программные, эксплуатационные.

Важнейшей эксплуатационной характеристикой ИС, влияющей на её производительность, является *время отклика на запрос к базе данных*, которое во многом зависит от реализации физической модели данных и, в частности, от правильности построения индексных структур данных.

- "Почему запрос выполняется так долго" ?
- "Почему длительности выполнения двух практически одинаковых запросов так сильно отличаются" ?
- "Почему сегодня этот запрос выполняется гораздо дольше, чем он выполнялся вчера" ?
- "Почему все так плохо даже при наличии индексов в таблице" ?

Для получения ответов на такие вопросы, часто задаваемые администратору пользователями и программистами баз данных, необходимо рассмотреть процесс формирования *процедурного плана выполнения SQL-запроса*, генерируемого сервером баз данных в результате трансляции соответствующего SQL-кода.

3.1 SQL – язык программирования декларативного типа

Программируя на классическом языке высокого уровня, таком, например, как Basic, Pascal или C#, мы, по существу, описываем на этом языке алгоритм решения некоторой задачи в виде последовательности операторов, то есть явно определяем процедуру обработки данных, реализация которой должна привести к получению желаемого результата. Такие высокоуровневые языки относятся к категории *процедурных* языков программирования.

Процедурными являются и низкоуровневые (машинные) языки, так как машинная программа – это детальное описание алгоритма обработки данных в виде последовательности команд процессора. Из этого, в част-

ности, следует, что компиляция исходного кода программы, написанной на процедурном языке высокого уровня, в её машинный код – это (всего лишь!) преобразование одного описания процедуры обработки данных в другое описание этой же процедуры.

В отличие от процедурных языков программирования, язык SQL является языком *декларативного* типа, и текст SQL-запроса к базе данных не содержит описания алгоритма получения результата, а только *декларирует требования* к этому результату.

Например, SQL-запрос вида `Select * From T Where T.x>T.y` требует выборки из таблицы **T** только тех её строк, в которых значение поля **x** больше значения поля **y**, не описывая при этом алгоритма выполнения такой операции.

3.2 Типовая схема трансляции SQL-запроса

Если SQL-запрос не содержит описания процедуры поиска данных, а в результате компиляции SQL-кода все же получается процедурный машинный код, то, очевидно, задачу выбора необходимой процедуры транслятор решает самостоятельно, без прямого участия программиста.

В состав транслятора с языка SQL входит специализированный компонент – *оптимизатор запросов*, задача которого сгенерировать *оптимальный процедурный план выполнения запроса*, то есть, фактически, "переписать" исходный непроцедурный SQL-код в эквивалентный ему код на некотором промежуточном процедурном языке. На завершающем этапе трансляции этот процедурный план будет скомпилирован в исполнимый машинный код.

Процесс трансляции SQL-запроса (рисунок 3.1) включает несколько последовательных фаз его обработки. Перед тем, как попасть на вход оптимизатора, исходный SQL-код подвергается предварительной обработке (на рисунке не показано), включающей его синтаксический анализ, лексическое и логическое преобразования.

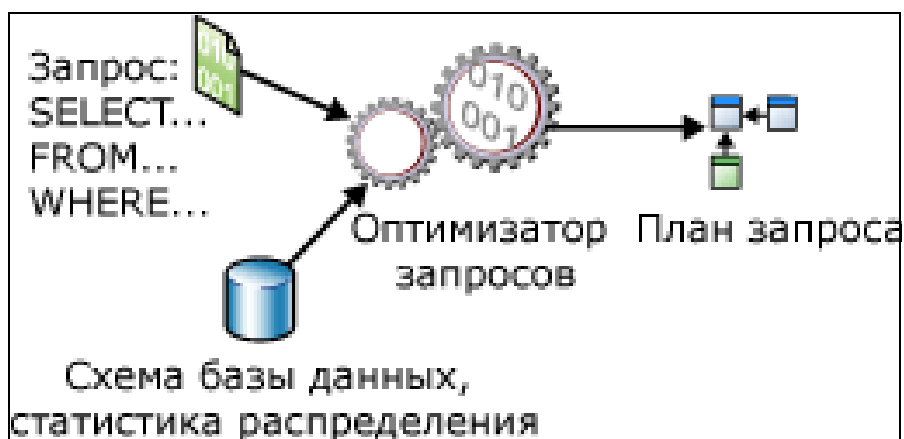


Рисунок 3.1 – Упрощенная схема трансляции SQL-запроса

Фаза 1 – Синтаксический анализ

Стандартная для любых трансляторов фаза – синтаксический разбор (parsing) исходного кода. Синтаксический анализатор просматривает инструкцию SELECT, разбивает ее на логические единицы (ключевые слова, выражения, операторы и идентификаторы), контролирует правильность написания языковых конструкций.

Фаза 2 – Лексические преобразования

На этой фазе проверяется корректность использования имен логических объектов (таблиц, индексов, полей таблиц), а сами эти имена заменяются на соответствующие им внутренние идентификаторы (вспомним системный каталог базы данных и, в частности, таблицы SysObjects, SysColumns, SysIndexes).

В результате формируется новое (все еще непроцедурное) представление запроса, синтаксически эквивалентное исходному SQL-коду.

Фаза 3 – Логические преобразования

Дальнейшая обработка запроса связана с его логическими преобразованиями с целью упрощения процесса дальнейшего анализа и принятия решений, связанных с генерацией процедурных планов. При этом применяются как *синтаксические*, так и *семантические* преобразования.

Синтаксические преобразования запроса выполняет специализированный компонент транслятора – *algebrizer* («алгебраизатор»), в котором входное представление запроса приводится к виду, удобному для его последующей трансформации в последовательность операций реляционной алгебры.

В результате синтаксических преобразований:

- производится упрощение предикатов ограничения выборки и предикаты соединений (логические выражения разделов Where и Join) путем приведения их к некоторой канонической форме;

- исключается вложенность запросов, например, запрос вида:

```
Select R1.a From R1 Where R1.b IN
  (Select R2.d From R2 Where R2.e = R1.c)
```

приводится к виду:

```
Select R1.a
  From R1 Join R2 ON (R1.b=R2.d AND R1.c=R2.e);
```

- запросы, заданные на представлениях, преобразуются путем объединения кода запроса с кодом представления, например:

пусть создано представление **V(C)** :

```
Create View V(C) AS
  Select T.C1 From T Where T.C1>6
```

и на базе этого представления написан следующий запрос:

```
Select * From V Where V.C<6
```

План выполнения такого запроса (без предварительного логического преобразования) состоял бы из двух фаз:

- 1) "материализация" представления **V** путем выборки из таблицы **T** строк, соответствующих ограничению **T.C1>6**, с сохранением результатов во временной таблице, например, в таблице **Tmp.V**;
- 2) выборка из временной таблицы **Tmp.V** тех строк, которые удовлетворяют ограничению **Tmp.V.C<6**.

После объединения кодов запроса и представления получится запрос вида:

```
Select T.C1 From T Where T.C1>6 AND T.C1<6,
```

в котором раздел **Where** содержит тождественно ложное логическое выражение, из чего явно следует, что строить и, тем более, оптимизировать процедурный план вообще не потребуется – достаточно вернуть в результат запроса пустое множество строк.

В процессе *семантических преобразований* формируется новый запрос, синтаксически не эквивалентный исходному запросу, но дающий точно такой же результат.

Пусть, например, в базе данных, обслуживающей систему кадрового учета компании, имеются две связанные таблицы: таблица **Employees**, содержащая данные обо всех сотрудниках компании, в том числе размер

должностного оклада сотрудника (поле **Salary**), и таблица **Posts** – справочник наименований должностей компании (поле **Title**).

Следующий исходный SQL-запрос производит выборку всех сотрудников компании, занимающих должности начальников («**Head**»):

```
Select Employees.Name
From Employees Inner Join Posts
ON Employees.Post_ID=Posts.Post_ID
Where Posts.Title Like "Head";
```

План реализации такого запроса будет включать процедуру внутреннего соединения двух таблиц, например, методом вложенных циклов, и процедуру фильтрации строк временной таблицы, сформированной первой процедурой. Заметим, что процедура соединения таблиц – одна из наиболее "дорогостоящих" процедур, даже при наличии соответствующих индексов.

Пусть для поля **Salary** таблицы **Employees** задано следующее ограничение целостности CONSTRAINT, соответствующее утверждению, что зарплата начальника не может быть меньше \$1000:

```
ALTER TABLE Employees
ADD CONSTRAINT MaxHeadSalary
CHECK (
  If (Select Posts.Title
      WHERE Posts.Post_ID=Employees.Post_ID) Like
  "Head"
  Then Employees.Salary>=$1000);
```

С помощью такого ограничения можно, например, контролировать правильность заполнения данных о заработной плате сотрудников.

В этих условиях исходный запрос может быть семантически преобразован в другой запрос, синтаксически отличающийся от исходного, но эквивалентный ему по результату выполнения, и при этом гораздо более простой и "удобный" для оптимизатора, генерирующего процедурный план:

```
Select Employees.Name From Employees
Where Employees.Salary>=$1000;
```

К тому же, план исполнения такого запроса не содержит процедуры соединения таблиц, следовательно, он будет существенно менее дорогостоящим по сравнению с процедурным планом выполнения исходного запроса.

Четвертая и пятая фазы трансляции SQL-запроса реализуется непосредственно *оптимизатором запросов*, который получает непроцедурное представление запроса, прошедшее предварительную обработку на предшествующих фазах трансляции, и генерирует процедурный план выполнения запроса. В своей работе оптимизатор использует дополнительную статистическую информацию о текущем состоянии базы данных.

Фаза 4 – Генерация альтернативных планов выполнения запроса

Генератор процедурных планов получает внутреннее непроцедурное представление запроса (результат его предшествующих преобразований) и запрашивает дополнительную информацию о текущем состоянии объектов базы данных, указанных в запросе: например, данные о мощности таблиц, о наличии и типах индексов, созданных в этой таблице по столбцам, затрагиваемым запросом.

В распоряжении генератора имеется набор типовых стратегий реализации запросов – по существу, набор правил, применяемых в определенных условиях. Генератор анализирует информацию о текущем состоянии объектов базы данных и принимает решение о возможности использования определенных стратегий при формировании планов.

Результатом данной фазы трансляции запроса является множество альтернативных планов, каждый из которых представлен в виде соответствующего *дерева логических операторов*, описывающего на концептуальном уровне последовательность выполнения операций реляционной алгебры (вот где оказывается полезной проведенная ранее "алгебраизация" непроцедурного представления запроса).

Примеры логических операторов:

- **Cross Join** – соединяет каждую строку из первого (верхнего) входного параметра с каждой строкой второго (нижнего) входного параметра;
- **Distinct** – удаляет дубликаты из набора строк;
- **Lazy Spool** – сохраняет все строки входных данных в скрытом временном объекте, который хранится в системной базе данных **TempDB**.

Фаза 5 – Оценка стоимости и выбор оптимального плана

На этой фазе оптимизатор решает задачу эффективной реализации логических операторов, описывающих альтернативные планы выполнения запроса. Для каждого логического оператора выбирается подходящий физический оператор (или несколько физических операторов) и определяется "стоимость" их выполнения. В результате альтернативный план представляется *деревом физических операторов* и вычисляется его суммарная стоимость.

Каждый физический оператор является объектом или процедурой, выполняющей соответствующую логическую операцию, например, сканирование таблицы или индекса, соединение таблиц, вычисление, статистическую обработку, проверку целостности данных и др.

Примеры физических операторов:

- **Filter** – просматривает входные данные и возвращает только строки, удовлетворяющие критерию фильтрации.
- **Nested Loops** – выполняет логические операции внутреннего соединения методом вложенных циклов.
- **Clustered Index Delete** – удаляет строки из кластеризованного индекса.
- **Index Scan** – получает все записи некластеризованного индекса, которые удовлетворяют условию, указанному в предикате.

Некоторые низкоуровневые операторы (например, **Index Scan**) являются и логическими, и физическими. С полным перечнем операторов, используемых для представления процедурных планов, можно ознакомиться на официальном ресурсе корпорации Microsoft [8].

На рисунке 3.2 приведен пример графического представления процедурного плана выполнения следующего SQL-запроса:

```
Select * From MyTable_4 Where Key0=4 AND Key1>50
```

Таблица MyTable_4 содержит около 10000 строк, не имеет кластеризованного индекса, а по полям Key0 и Key1 этой таблицы созданы некластеризованные неуникальные индексы. Результирующая выборка составила 11 строк.

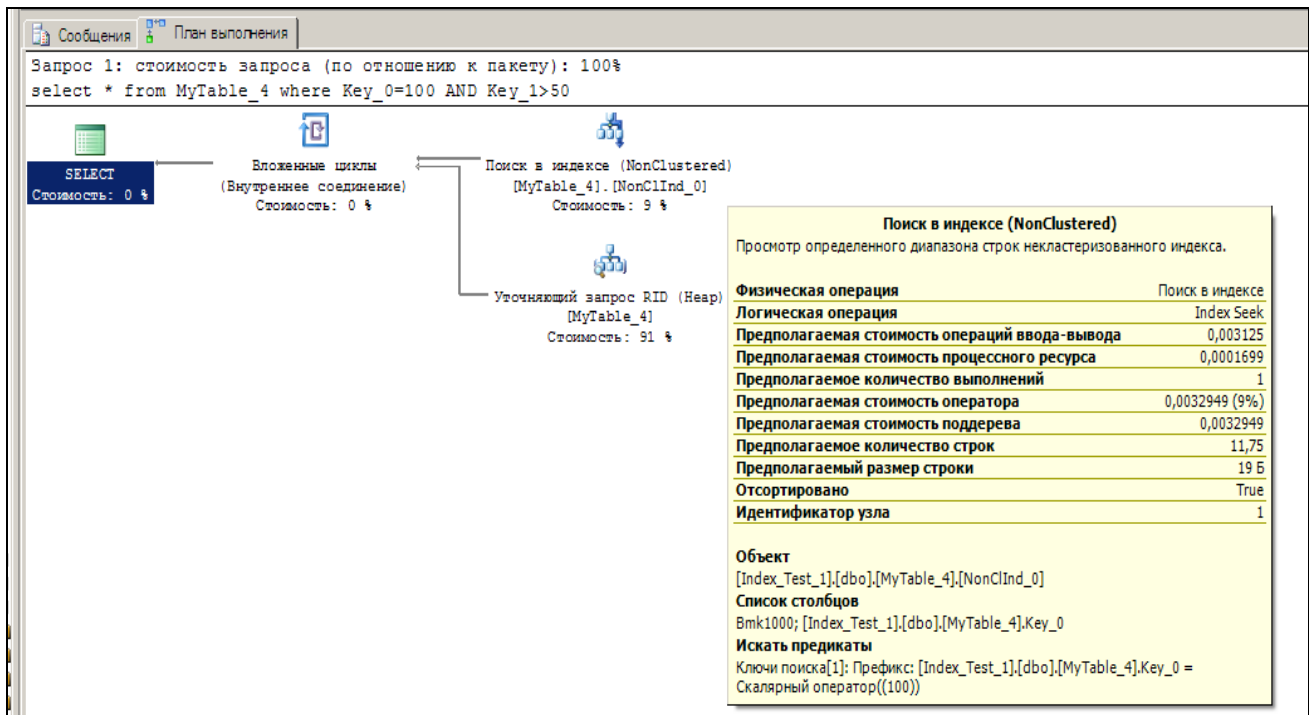


Рисунок 3.2 – Пример графического представления процедурного плана

Для оценки стоимости выполнения каждого физического оператора оптимизатор запроса использует соответствующую этому оператору модель стоимости.

Например, для оператора **Index Seek** эта модель определена, как сумма стоимостей операций ввода-вывода (**IO_Cost**=0,003125) и обработки данных (**ProcessingCost**=0,0001699), умноженная на количество операций (=1).

Оценка стоимости операторов процедурного плана выполняется на основании статистических данных, характеризующих состояние объектов, затрагиваемых SQL-запросом. Статистические данные содержат усредненную информацию о таблицах и индексах (например, количество занимаемых страниц, степень селективности предикатов выборки данных, гистограммы распределения значений полей таблиц и т.д.).

MS SQL-сервер собирает статистику в фоновом режиме в соответствии со сценарием, заданным администратором базы данных. Например, процедура сбора статистики может запускаться в фиксированное время суток или после каждой модификации данных, при этом данные таблиц могут собираться как полностью (для первых 200 строк), так и случайной выборкой части строк больших таблиц.

Соответствующий инструмент среды SQL Server Management Studio (вкладка «Статистика») позволяет в любой момент времени обновить статистику или просмотреть время её последнего обновления.

Просмотреть текущее состояние статистики индекса **Ind** таблицы **T1** можно командой **DBCC show_statistics (T1,Ind)**.

Тот из альтернативных планов, стоимость которого оказалась минимальной, получает статус *предполагаемого плана (estimated execution plan)* и записывается в хранилище процедурных планов (*PlanCache*), где временно хранится вместе с планами других запросов в прекомпилированном виде (в виде дерева физических операторов) для последующего извлечения и многократного использования.

На этом работа оптимизатора запроса завершается, и управление передается подсистеме выполнения запросов (Storage Engine) сервера баз данных.

3.3 Исполнение процедурного плана выполнения запроса

Предполагаемые планы выполнения запросов, записанные в хранилище, не хранятся там "вечно" – сервер ведет статистику использования сохраненных планов и регулярно удаляет из хранилища редко используемые планы.

При поступлении на обработку очередного SQL-запроса производится проверка наличия в хранилище предполагаемого плана его выполнения.

Если нужного плана в хранилище нет, запускается описанный выше полный процесс трансляции исходного SQL-кода (фазы с 1-й по 5-ю), по завершении которого сформированный оптимизатором запросов предварительный план записывается в хранилище.

При наличии соответствующего предварительного плана он извлекается из хранилища, проверяется возможность его выполнения в текущем состоянии базы данных и, если план осуществим, он получает статус действительного (*actual*) плана, который затем компилируется в машинный код и исполняется.

Если принимается решение о невозможности реализации ранее сохраненного предполагаемого плана, повторно запускается процесс оптимизации (4-я и 5-я фазы трансляции запроса – генерация альтернативных процедурных планов, их оценивание и выбор оптимального предвари-

тельного плана). По завершению процесса оптимизации старый план заменяется в хранилище новым планом.

Во многих случаях предполагаемый и действительный планы будут совпадать. Типичные причины нереализуемости сохраненного ранее предварительного плана:

- статистика, на основе которой был сформирован предполагаемый план выполнения запроса, к текущему моменту либо устарела («*out of date*»), либо была обновлена;
- логические объекты базы данных, затрагиваемые запросом, были модифицированы после создания процедурного плана (например, были созданы новые индексы, изменены или удалены старые, удалены временные таблицы, на которые ссылается запрос, и т.д.).

3.4 Средства визуализации процедурных планов

На завершающем этапе процесса отладки SQL-запроса (храняемого представления, процедуры или пакета, состоящего из множества таких объектов в любых их комбинациях), можно просмотреть не только результат его выполнения, но также и соответствующий процедурный план, сгенерированный оптимизатором запросов. При этом предусмотрена возможность визуализации как предполагаемого плана, извлекаемого из хранилища, так и действительного плана, актуального в текущих условиях реализации запроса.

3.4.1 Инструкции TransactSQL

Язык TransactSQL содержит инструкции группы SET, позволяющие сохранить в текстовом формате или в формате XML-документа как предполагаемый, так и действительный план выполнения запроса (таблица 3.1).

Инструкция **SET SHOWPLAN_... ON/OFF** включает/выключает соответствующий режим сохранения предварительного плана, извлекаемого из хранилища, блокируя при этом исполнение запроса.

Инструкция **SET STATISTICS_... ON/OFF** включает/выключает режим отображения действительного плана, не препятствуя исполнению запроса.

Таблица 3.1 – Операторы управления отображением процедурных планов

SET SHOWPLAN_XML	Возвращает сведения о приблизительном плане выполнения в виде XML-документа
SET SHOWPLAN_TEXT	Возвращает сведения о приблизительном плане выполнения
SET SHOWPLAN_ALL	Возвращает полную информацию о приблизительном плане выполнения запроса
SET STATISTICS XML	Возвращает сведения о действительном плане выполнения в виде XML-документа
SET STATISTICS PROFILE	Возвращает полную информацию о действительном плане выполнения запроса
SET STATISTICS IO	Отображает сведения о дисковой активности во время выполнения запроса
SET STATISTICS TIME	Отображает время (в миллисекундах), которое потребовалось для синтаксического анализа, компиляции и выполнения запроса

На рисунке 3.3 приведен пример отображения действительного плана выполнения следующего SQL-запроса:

```

SET STATISTICS XML ON
SET STATISTICS PROFILE ON
SET STATISTICS IO ON
SET STATISTICS TIME ON
Select * From T1 Inner Join T2 ON T1.c2 > T2.c2
Where T2.c3<500 OR T1.c3<500;

```

Такой формат табличного отображения плана запроса достаточно информативен: каждая строка представляет один физический оператор и содержит код этого оператора, информацию о времени его выполнения и количестве обработанных строк.

Примечание. Приведенный пример – результат реализации запроса в MS SQL Server 2005. В более поздних версиях возможность отображения планов в текстовом формате не предусмотрена, хотя соответствующие команды оставлены для совместимости со старшими версиями. В более поздних версиях в результате применения любой из команд, приведенных в

таблице 3.1, план запроса будет сохранен в XML-формате, однако утилита просмотра такого плана отобразит его на экране в графическом формате, как показано на рисунках 3.2 и 3.5.

RowID	Rows	Executes	StmtText	EstimateCPU	TotalSubtreeCost	EstimateRows	EstimateIO	EstimateExecutions	AvgR
1	363040	1	select * from T1 inner join T2 on T1.c2>T2.c2 where T2.c3<500 or T1.c3>500 order by 2	NULL	1.877784	358642.6	NULL	NULL	NULL
2	363040	1	↳Nested Loops(Inner Join, OUTER REFERENCES: ((db1].[dbo].[T1].[c3]))	0.941416	1.877784	358642.6	0	1	31
3	1000	1	↳Sort(ORDER BY: ((db1].[dbo].[T1].[c2] ASC))	0.01564664	0.03277139	1000	0.01126...	1	19
4	1000	1	↳Clustered Index Scan(OBJECT: ((db1].[dbo].[T1].[PK__T1__3213663B21B6055D]))	0.001257	0.005863482	1000	0.00460...	1	19
5	363040	1000	↳Filter(WHERE: ((db1].[dbo].[T2].[c3]<(500) OR (db1].[dbo].[T1].[c3]>(500)))	0.000264	0.9035968	225.2191	0	1000	19
6	481771	1000	↳Index Spool(SEEK: ((db1].[dbo].[T2].[c2] < (db1].[dbo].[T1].[c2]))	0.001587	0.6395967	300	0.03413...	1000	19
7	1000	1	↳Clustered Index Scan(OBJECT: ((db1].[dbo].[T2].[PK__T2__3213663B29572725]))	0.001257	0.005863482	1000	0.00460...	1	19

3040 row(s) affected

le 'Worktable'. Scan count 1000, logical reads 5408, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0

le 'T2'. Scan count 1, logical reads 5, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

le 'T1'. Scan count 1, logical reads 5, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.

row(s) affected)

row(s) affected)

L Server Execution Times:
 CPU time = 484 ms, elapsed time = 6021 ms.
 Server parse and compile time: |
 CPU time = 0 ms, elapsed time = 0 ms.

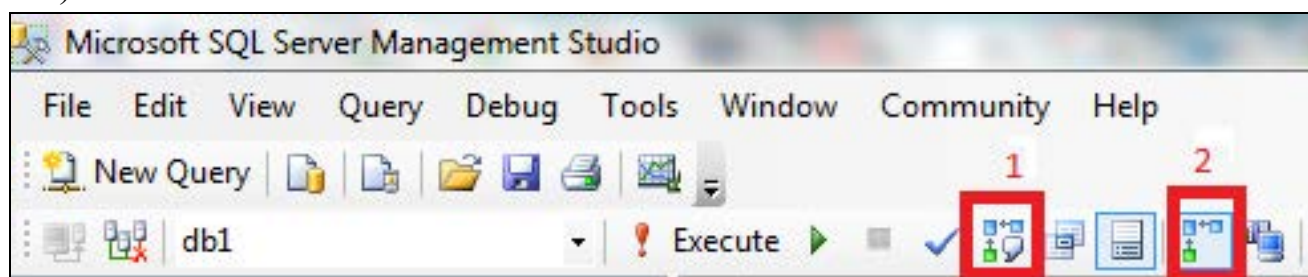
L Server Execution Times:
 CPU time = 0 ms, elapsed time = 0 ms.

Рисунок 3.3 – Пример табличного представления процедурного плана

3.4.2 Средства графического отображения процедурных планов

Среда SQL Server Management Studio предоставляет возможность графического представления предварительного и действительного плана выполнения запроса в виде дерева физических процедурных операторов.

Для включения/отключения режимов графического отображения планов на экранной панели имеются соответствующие "кнопки" (рисунок 3.4).



1) предполагаемый план; 2) действительный план

Рисунок 3.4 – Режимы просмотра графического плана

Следует помнить, что при включении режима отображения предварительного плана (так же, как и в случае с применением команды **SET SHOWPLAN ...ON**) выполнение запроса блокируется.

На рисунке 3.5 приведен пример графического отображения действительного плана выполнения запроса из приведенного выше примера.

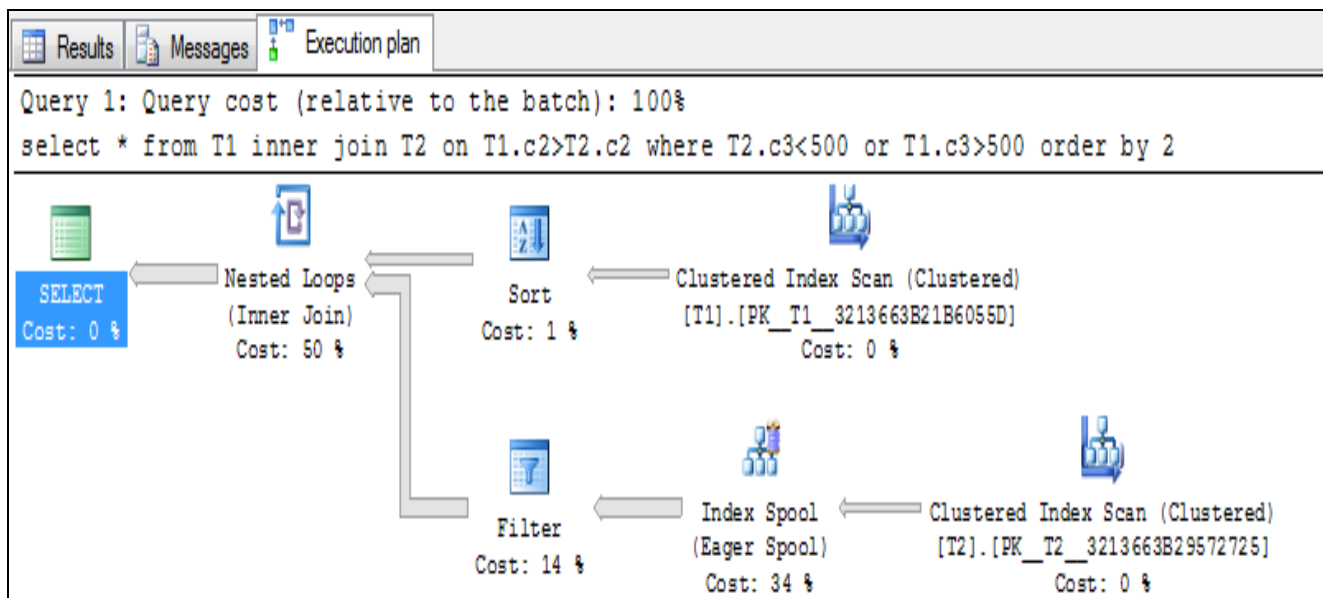


Рисунок 3.5 – Пример графического представления плана запроса

Каждый оператор на схеме плана представляется соответствующим графическим символом (таблица 3.2), под которым отображается код этого оператора. При наведении курсора на оператор на экране появляется всплывающая подсказка (ToolTips), как это показано на рисунке 3.6. Подсказка содержит имена логического и физического операторов, количество строк, возвращаемых оператором (предполагаемое и фактическое), стоимость операции, количество исполнений и другую информацию, полезную для анализа процедурного плана.

Читать графические планы следует справа налево, в соответствии с направлением стрелок. Толщина стрелки также информативна – она пропорциональна количеству передаваемых строк. При наведении курсора на стрелку на экране также появляется соответствующая подсказка.

С полным перечнем низкоуровневых операторов процедурных планов можно ознакомиться на официальном ресурсе разработчика [7-9].

Index Spool	
Reformats the data from the input into a temporary index, which is then used for seeking with the supplied seek predicate.	
Physical Operation	Index Spool
Logical Operation	Eager Spool
Actual Number of Rows	481771
Estimated I/O Cost	0.034136
Estimated CPU Cost	0.001587
Number of Executions	1000
Estimated Number of Executions	1000
Estimated Operator Cost	0.6337335 (34%)
Estimated Subtree Cost	0.639597
Estimated Number of Rows	300
Estimated Row Size	19 B
Actual Rebinds	634
Actual Rewinds	366
Node ID	4
Output List	
[db1].[dbo].[T2].c1, [db1].[dbo].[T2].c2, [db1].[dbo].[T2].c3	
Seek Predicate	
Seek Keys[1]: End: [db1].[dbo].[T2].c2 < Scalar Operator ((db1).[dbo].[T1].[c2])	

Рисунок 3.6 – Всплывающая подсказка

Таблица 3.2 – Графическое представление операторов процедурных планов

	Table Scan	Сканирует таблицу («кучу»)
 	Clustered Index Scan Index Scan	Сканирует кластеризованный индекс / некластеризованный индекс
 	Clustered Index Seek Index Seek	Производит поиск по кластеризованному индексу / некластеризованному индексу
	Key Lookup	Производит поиск закладок в таблице с кластеризованным индексом
	RID Lookup	Производит поиск закладки в «куче» по заданному идентификатору строки
  	Table Insert Clustered Index Insert Index Insert	Вставляет строки в таблицу / кластеризованный индекс / некластеризованный индекс
	Filter	Просматривает входные данные и возвращает строки, удовлетворяющие критерию фильтрации
	Nested Loops	Соединяет таблицы по методу вложенных циклов

Глава 4

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ БАЗ ДАННЫХ

Защита информации не ограничивается только рамками базы данных и даже рамками информационной системы в целом. Решение этой задачи требует рассмотрения различных аспектов информационной безопасности всего программно-аппаратного комплекса, включая использование операционных систем, сетевого оборудования, средств защиты сетевого трафика и, разумеется, средств разграничения доступа пользователей к объектам баз данных [10; 11].

Требования к уровню защищенности информационной системы обеспечиваются как соответствующими проектными решениями, так и действиями персонала сопровождения в процессе ее эксплуатации, в том числе и действиями администраторов баз данных, реализующих (а в ряде случаев и определяющих) политику информационной безопасности предприятия.

Обеспечение информационной безопасности баз данных не является чисто технологической задачей, решаемой администраторами на стадии эксплуатации информационной системы. Многие проблемы с безопасностью "закладываются" в систему на стадии ее проектирования и вызваны недостаточной компетентностью разработчиков, другая категория проблем связана с действиями пользователей, непреднамеренно или сознательно нарушающих установленные правила.

При проектировании базы данных должны быть решены специфические задачи обеспечения защиты информации: классификация, группировка и определение ролевых функций субъектов доступа, разграничение прав доступа субъектов к объектам, детальное описание способов доступа к информации и определение используемых для этой цели серверных и клиентских программных приложений.

При этом следует понимать, что обеспечение жесткого режима безопасности неизбежно снижает производительность информационной системы, увеличивает трудоемкость ее администрирования, а также усложняет работу пользователей, "подталкивая" их к нарушениям установленных правил доступа к информации. Поэтому разработчики и администраторы баз данных сталкиваются с проблемой, существенно более сложной, чем техническая реализация методов защиты данных – это проблема вы-

бора разумного компромисса между безопасностью и доступностью информации.

Информационная безопасность базируется на трех основных концепциях защиты информации: *целостность, доступность и конфиденциальность*.

4.1 Целостность информации

Целостность (integrity) определяется, как *способность сохранения неизменности информации в условиях ее случайного и/или преднамеренного искажения*. В теории и технологии реляционных баз данных рассматриваются, как минимум, четыре аспекта целостности информации, каждый из которых связан с определенной проблемой, методами и инструментальными средствами, применяемыми для ее решения.

1 Нормализация базы данных

В слабо нормализованной базе данных, обслуживающей OLTP-систему, ориентированную на интенсивное выполнение модифицирующих транзакций, могут проявляться аномалии включения, удаления и изменения данных. Решение этой проблемы (или сведение к минимуму ее негативных проявлений) достигается при проектировании базы данных путем ее нормализации: в хорошо нормализованной базе данных каждая таблица приведена к одной из сильных нормальных форм путем ее декомпозиции на несколько взаимосвязанных таблиц.

Заметим, что нормализация базы данных, решая задачу обеспечения целостности информации, может приводить и к негативным последствиям, снижая производительность выполнения запросов, связанных с поиском и чтением данных (вспомним процедурные планы выполнения запросов с соединением таблиц, рассмотренные в предыдущем разделе учебного пособия).

2 Ссылочная целостность

Обеспечение ссылочной целостности берет на себя сервер баз данных, контролируя соответствие типов данных и значений первичных и внешних ключей в строках связанных таблиц в процессе модификации соответствующих данных.

Единственное, что должен сделать разработчик базы данных для обеспечения ссылочной целостности – это явно определить ключевые поля таблиц, с помощью которых будут реализованы связи между ними, ис-

пользуя соответствующие конструкции SQL-операторов Create/Alter Table:

```
FOREIGN KEY  
REFERENCES <имя главной таблицы> (имя первичного  
ключа)
```

3 Явные ограничения целостности

Явные (*checked* – проверяемые) ограничения целостности могут быть определены для полей таблиц и отражают, по существу, семантические ограничения, накладываемые на значения атрибутов соответствующих сущностей предметной области. Простейшие ограничения этого типа – это ограничения типов данных и диапазонов допустимых значений полей, в более сложных случаях такие ограничения могут содержать логические выражения и вложенные SQL-запросы.

Серверы баз данных предоставляют несколько инструментов для работы с явными ограничениями целостности. Один из таких инструментов позволяет явно задать ограничение в процессе описания схемы таблицы SQL-операторами Create/Alter Table, используя конструкцию Constraint. Сервер будет автоматически проверять выполнение ограничения при модификации данных таблицы и блокировать ввод некорректных значений. Пример использования конструкции Constraint приведен в разделе 3.2 учебного пособия.

Другая ситуация, требующая контроля выполнения явного ограничения целостности информации, возникает в случаях, когда существует зависимость между значениями полей нескольких таблиц базы данных. Классический пример: таблица «Студенты» содержит список всех студентов университета, а связанная с ней таблица «Группы» содержит список всех студенческих групп, причем одно из полей этой таблицы – количество студентов в группе. Очевидно, что любая из типовых операций с контингентом студентов (зачисление, отчисление или перевод) потребует внесения изменений в таблицу «Студенты» и, как следствие – перерасчета и изменения значения зависимого поля таблицы «Группы».

Одно из возможных решений проблемы – включение этих двух операций в одну транзакцию, что исключит возможность "одионого" выполнения любой из них и, как следствие, сохранит непротиворечивость данных.

Другое решение связано с использованием *триггеров* – специальных хранимых процедур, выполнение которых обусловлено наступлением определенных событий. В нашем примере триггер, обрабатывая таблицу «*Студенты*», должен рассчитывать количество студентов в каждой группе и записывать полученные значения в соответствующие строки таблицы «*Группы*». Если связать такой триггер с событиями модификации данных в таблице «*Студенты*», то логическая целостность базы данных будет надежно обеспечена.

4 Физическая согласованность и надежность хранения данных

Завершая обсуждение концепции целостности информации, рассмотрим задачу сохранения физической целостности базы данных в условиях, когда в процессе эксплуатации информационной системы происходят технические сбои в работе оборудования. Обычно рассматривают две категории таких сбоев: "мягкий сбой", связанный с потерей данных, накопленных в оперативной памяти и еще не сохраненных в файлах базы данных, и "жесткий сбой", связанный с потерей работоспособности внешних запоминающих устройств.

Физическая целостность информации обеспечивается определенными действиями администраторов баз данных с использованием стандартных средств поддержки надежности, предоставляемых серверами баз данных.

Жесткий сбой. Практически единственным способом борьбы с жесткими сбоями является регулярное резервное копирование баз данных с последующим их восстановлением из резервных копий. Администратор базы данных может создать определенный сценарий, задающий периодичность создания резервных копий и сохраняющий их на хорошо защищенных и надежных устройствах внешней памяти. Такой сценарий будет выполняться автоматически и не потребует оперативного вмешательства администратора.

Важнейшей проблемой резервного копирования баз данных является существенная длительность этой операции, причем на время ее выполнения сервер накладывает монопольную блокировку доступа клиентских приложений ко всем физическим объектам базы данных (экстентам файлов типа Data), затрагиваемым процедурой резервного копирования.

Острота этой проблемы частично снимается применением технологии создания *разностных копий* базы данных, согласно которой копиру-

ются не все объекты базы данных, а только те из них, в которых произошли изменения с момента последнего резервного копирования базы данных. MS SQL-Server оперативно сохраняет информацию о номерах измененных экстенгов в специальной файловой DCM-странице (Differential Changed Map, таблица 1.1).

Мягкий сбой. Проблема потери данных в оперативной памяти решается путем сохранения в специальном журнале (Log-файле) информации обо всех транзакциях, изменивших состояние базы данных, в том числе и о транзакциях, содержащих операции резервного копирования базы данных. Журнал транзакций – это LOG-файл специального формата, который обычно размещается на хорошо защищенном и надежном устройстве внешней памяти, и содержит хронологически упорядоченную последовательность записей, каждая из которых описывает активную операцию доступа к данным с указанием идентификатора транзакции, в составе которой бала выполнена эта операция.

Сервер баз данных формирует журнал транзакций в соответствии с протоколом WAL, гарантирующим актуальность состояния журнала транзакций к моменту наступления мягкого сбоя. Ниже приведено несколько упрощенное описание процесса формирования журнала транзакций и алгоритма его использования в процессе восстановления базы данных после мягкого сбоя.

Если транзакция пытается изменить содержимое строк таблицы, удалить или вставить дополнительные строки в таблицу, соответствующие файловые страницы считываются из файла в оперативную память, и все необходимые изменения производятся в виртуальных копиях этих страниц. Параллельно информация обо всех таких изменениях оперативно регистрируется в сегменте журнала транзакций, также размещенном в буфере оперативной памяти.

В соответствии с ACID-требованиями к реализации механизма транзакций, любая транзакция должна "получить" базу данных в согласованном состоянии и сохранить ее в согласованном состоянии после своего завершения (как успешного, так и прерванного), и при этом все изменения, произведенные в оперативной памяти операциями успешно завершённой транзакции, должны быть гарантированно зафиксированы (Commit) в базе данных, а операции прерванной транзакции, в том числе и уже зафик-

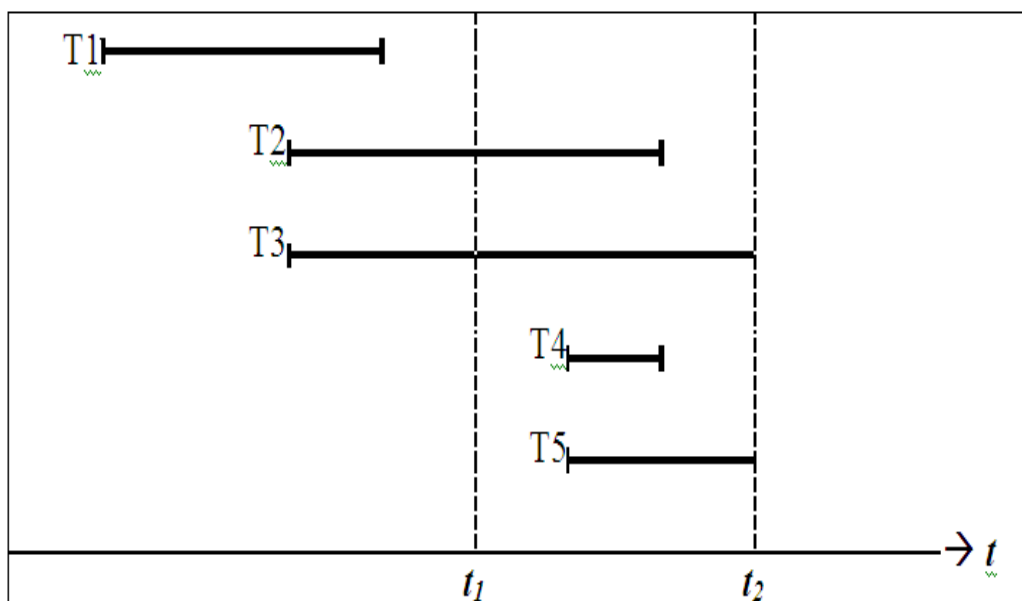
сированные в базе данных, должны быть отменены путем отката (Roll-Back) транзакции.

Из этого, в частности, следует, что если транзакция успешно завершилась оператором Commit, то все виртуальные страницы, измененные операциями этой транзакции, должны быть сохранены в файле типа DATA.

Фактически все немного сложнее, так как SQL-код транзакции может содержать специальные метки – промежуточные "точки сохранения" (save points), в которых гарантируется физическая согласованность данных и при достижении которых все промежуточные результаты еще не завершённой транзакции должны быть сохранены в файле точно так же, как и при её успешном завершении оператором Commit. При этом в журнале транзакций сохраняются и временные метки физической согласованности данных (так называемые TPC – Time of Physical Consistency), соответствующие промежуточным точкам сохранения транзакций.

В соответствии с протоколом WAL (Write Ahead Log – «прежде запиши в журнал»), любой операции записи виртуальных страниц в файл типа DATA должна предшествовать операция записи в LOG-файл соответствующего виртуального сегмента журнала транзакций. При соблюдении этого условия LOG-файл будет содержать записи обо всех транзакциях, как завершённых, так и прерванных в момент наступления мягкого сбоя и "оставивших" базу данных в рассогласованном состоянии, что позволяет реализовать процедуру восстановления согласованного состояния базы данных.

Иллюстрация этой процедуры приведена на рисунке 4.1.



t – время фиксации записей в журнале транзакций;
 t_1 – последняя точка сохранения (tpc);
 t_2 – момент наступления мягкого сбоя

Рисунок 4.1 – Иллюстрация алгоритма восстановления базы данных после мягкого сбоя

На рисунке 4.1 показаны 5 типовых ситуаций:

- транзакция T1 началась, успешно завершилась и была зафиксирована в журнале транзакций и в базе данных до наступления момента мягкого сбоя – никаких действий по ее восстановлению не потребуется;
- транзакция T2 успешно завершилась до момента наступления мягкого сбоя, все ее операции были зафиксированы в журнале транзакций, но часть результатов их выполнения (на рисунке – правее последней точки сохранения t_1) по каким-то причинам не были зафиксированы в базе данных до момента наступления сбоя. Для восстановления согласованности база данных в этой ситуации потребуется повторно выполнить все операции (ReDo) этой транзакции, зарегистрированные в журнале транзакций после последней точки сохранения t_1 ;
- транзакция T3 не успела завершиться до момента наступления сбоя, однако в момент t_1 (точка физической согласованности) часть результатов ее выполнения были сохранены в базе данных. Несмотря на то, что транзакция не нарушила согласованного состояния ба-

зы данных, потребуется выполнить откат транзакции (Undo) путем последовательного выполнения операций, "противоположных" операциям, зафиксированным в журнале транзакций раньше последней точки сохранения t_1 ;

- транзакция T4 началась позднее последней точки сохранения t_1 и успешно завершилась. Все операции этой транзакции были записаны в журнал транзакций, однако результаты их выполнения по каким-то причинам не были зафиксированы в базе данных до наступления сбоя. Данная транзакция не нарушила целостность базы данных, однако для восстановления ее результатов потребуется повторно выполнить все операции, зарегистрированные в журнале от имени этой транзакции;
- транзакция T5 началась позднее последней точки сохранения t_1 и не успела завершиться до момента сбоя t_2 . Часть операций транзакции были зарегистрированы в журнале транзакций, но результаты их выполнения не были зафиксированы в базе данных. Эта транзакция не нарушила согласованного состояния базы данных, и никаких действий по ее восстановлению не потребуется.

4.2 Доступность и конфиденциальность информации

Обеспечение доступа к информации легальным пользователям и защита конфиденциальной информации от несанкционированного доступа – две взаимосвязанные задачи, для решения которых могут использоваться различные методы и средства, определяемые требованиями к уровню защищенности системы.

Таблица 4.1 – Терминология системы разграничения доступа

Термины		Определения
Доступ к информации	Access to information	Ознакомление с информацией и/или ее обработка (копирование, модификация, уничтожение)
Объекты доступа	Access objects, Securables	Единицы информации, доступ к которым регламентируется правилами разграничения доступа.
Субъекты доступа	Access subjects, Principals	Пользователи, группы пользователей или процессы, действия которых регламентируются правилами разграничения доступа
Права доступа (разрешения)	Permissions	Набор операций над объектом доступа, выполнение которых разрешено субъекту доступа
Правила разграничения доступа	Security policy	Совокупность правил, регламентирующих права субъектов по отношению к объектам
Идентификация	Identification	Присвоение объектам и субъектам доступа идентификатора и/или сравнение предъявляемого идентификатора с перечнем присвоенных идентификаторов
Аутентификация	Authentication	Проверка принадлежности субъекту предъявленного им идентификатора, подтверждение подлинности
Уровень полномочий субъекта	Subject privilege	Совокупность прав доступа (привилегий) субъекта доступа
Метка конфиденциальности	Sensitivity label	Информация, характеризующая уровень конфиденциальности объекта доступа
Дискреционная защита (логическая)	Discretionary secure	Разграничение доступа между поименованными субъектами и поименованными объектами
Мандатная защита (физическая)	Mandatory secure	Защита, обеспечивающая разграничение доступа субъектов с различными правами доступа к объектам различных уровней конфиденциальности

Пользователь информационной системы является важнейшим элементом системы разграничения доступа. По отношению к серверу баз

данных пользователей можно разделить на три категории: конечные пользователи, прикладные программисты и администраторы.

Конечные пользователи, как правило, не имеют непосредственного доступа к серверу баз данных и получают доступ к базам данных через клиентские приложения, которые в этом случае получают статус "конечных пользователей". Права доступа конечных пользователей определяются их должностными обязанностями в информационной системе.

Прикладные программисты разрабатывают программы (клиентские и серверные приложения), использующие базы данных в интересах конечных пользователей. Прикладные программисты могут иметь права создания, модификации и выполнения программных компонентов баз данных, однако они лишены прав модификации структуры компонентов данных и схем баз данных.

Администраторы образуют особую категорию пользователей. Они наделяются правами создания баз данных и модификации их структуры, осуществляют контроль функционирования серверов баз данных и мониторинг работы конечных пользователей, в необходимых случаях оказывают консультативную помощь прикладным программистам. В обязанности администратора входит обеспечение высокой производительности работы системы, резервное копирование и восстановление работоспособности баз данных после сбоев оборудования, определение и контроль за соблюдением политики безопасности, а также регистрация пользователей и управление правами их доступа к данным.

Система управления доступом пользователей к объектам баз данных, обеспечивающая разграничение доступа к конфиденциальной информации, имеет многоуровневую архитектуру: на уровне сервера баз данных производится идентификация и аутентификация пользователей, контроль их принадлежности определенным категориям, наделение их глобальными правами выполнения определенных операций с базами данных, управляемыми этим сервером; на уровне базы данных решаются локальные задачи управления доступом со стороны индивидуальных пользователей и/или их групп.

4.3 Дискреционная защита информации

Технология дискреционного управления доступом (discretionary access control, от *discretion* – разделение, разграничение) обеспечивает так

называемую *логическую* защиту данных путем *разграничения прав доступа* субъектов базы данных к *поименованным объектам* логического уровня.

Информация о зарегистрированных субъектах доступа – *пользователях* или *группах пользователей* (иначе называемых *ролями*), как и информация о логических объектах (таблицах, представлениях, процедурах и пр.), хранится в системном каталоге базы данных (например, MS SQL Server хранит такую информацию в системных таблицах SysUsers и SysObjects). Наделение субъекта правом доступа к логическому объекту – это, по существу, установление связи (порядка M:N) между экземплярами (строками) соответствующих системных таблиц, а определение типа (наименования) права доступа – это присвоение соответствующего значения атрибуту такой связи. Таким образом, право доступа также является именованным логическим объектом базы данных.

Язык SQL предоставляет функционально полный набор средств дискреционного управления доступом, состав и синтаксис которых могут несущественно отличаться в различных реализациях. Ниже приведен краткий обзор таких языковых средств.

Категории прав доступа:

- **CREATE / ALTER** – право создания / модификации объектов доступа: баз данных, таблиц, представлений, процедур, функций и др.;
- **SELECT** – право выборки (чтения) строк или отдельных столбцов таблицы или представления;
- **INSERT** – право вставлять в таблицу или представление новые строки;
- **UPDATE** – право изменять данные в таблице или ее отдельных столбцах;
- **DELETE** – право удалять строки из таблицы или представления;
- **REFERENCES** – право ссылаться на указанный объект – разрешает пользователю создавать внешние ключи в подчиненных таблицах без права доступа к главным таблицам;
- **EXECUTE** – право выполнения хранимых процедур и функций.

SQL-операторы управления правами доступа:

- **CREATE USER ... , CREATE ROLE ...** – создание субъектов доступа соответствующих типов;

- **GRANT** <привилегия> **ON** <объект> **TO** <субъект> [**WITH GRANT OPTION**] – разрешение доступа: оператор предоставляет субъекту указанные права доступа ("привилегии") к объекту (опционально – с правом передачи полученных прав другим субъектам);
- **DENY** <привилегия> **ON** <объект> **TO** <субъект> – запрет доступа: оператор запрещает субъекту доступ к объекту с указанными правами ("привилегиями");
- **REVOKE** <привилегия> **ON** <объект> **TO** <субъект> – оператор отменяет действие выполненных ранее операторов **GRANT** или **DENY**.

Дополнительно к перечисленным выше SQL-средствам прямого управления доступом, серверы баз данных предоставляют администраторам программные компоненты, а также неязыковые средства экранного интерфейса.

Дискреционная защита информации удовлетворяет потребностям большинства коммерческих приложений, однако для систем повышенного уровня защищенности ее возможностей оказывается явно недостаточно.

Завершая обзор методов дискреционного управления доступом, перечислим основные проблемы защиты конфиденциальной информации, которые либо не решаются, либо решаются лишь частично в рамках этой технологии.

Во-первых, дискреционная защита не позволяет надежно разграничить доступ к *различным строкам* одной таблицы. Частичное решение проблемы – запрет доступа к таблице и разрешение доступа к отдельным представлениям, созданным на базе этой таблицы и содержащим различные условия выборки строк. Слабость такого решения очевидна: невозможно разграничить доступ к нескольким строкам таблицы, удовлетворяющим одному условию выборки.

Во-вторых, разграничение доступа к *различным столбцам* одной таблицы также создает определенные проблемы. Защита, основанная на создании представления, не содержащего информации "секретных" столбцов таблицы, легко обходится средствами SQL. Более радикальное решение предполагает модификацию схемы базы данных: защищаемые столбцы таблицы выделяются в отдельную таблицу, связанную с исходной таблицей, в которой остается только общедоступная информация. Такое решение может оказаться достаточно надежным, однако оно приведет

к снижению производительности системы (еще раз вспомним процедурные планы выполнения SQL-запросов с соединением таблиц).

Третий, возможно, главный недостаток дискреционного управления доступом – это отсутствие средств защиты от несанкционированного *распространения* конфиденциальной информации: ничто не препятствует пользователю, легально получившему доступ (с правом чтения *Select*) к таблице с конфиденциальной информацией, сделать эту информацию доступной другим пользователям путем вставки (*Insert*) этой информации в другую (например, временную) общедоступную таблицу.

И последнее, дискреционная защита не позволяет физически изолировать технического специалиста, выполняющего функции администратора базы данных, от управления конфиденциальными данными (в том числе и от ознакомления с ними), что требует повышения меры ответственности администратора и вряд ли соответствует политике информационной безопасности, реализуемой в хорошо защищенной системе.

Как видим, дискреционная логическая защита является довольно слабой, и основная причина такой "слабости" заключается в том, что информация о защите данных хранится отдельно от самих защищаемых данных. Существенно более высокий уровень защищенности информации обеспечивает так называемая «мандатная», или физическая защита.

4.4 Мандатная защита информации

Мандатное управление доступом (*mandatory access control*) основано на использовании «меток безопасности», присваиваемых экземплярам защищаемых объектов базы данных, и официальном «мандате», выдаваемом субъекту и разрешающем ему доступ к информации с определенными «метками безопасности».

Классическая модель системы мандатного управления доступом впервые была описана Дэвидом Беллом и Леонардом Лападулой в 1975 году. Согласно этой модели каждому субъекту и каждому объекту доступа присваивается метка конфиденциальности: от самой высокой («особой важности») до самой низкой («несекретный» или «общедоступный»).

При этом субъект не может читать информацию объекта, метка конфиденциальности которого выше его собственной, но также ему запрещена запись информации в объекты с более низким уровнем безопасности.

На рисунке 4.2* показана диаграмма информационных потоков модели Белла – Лападулы.

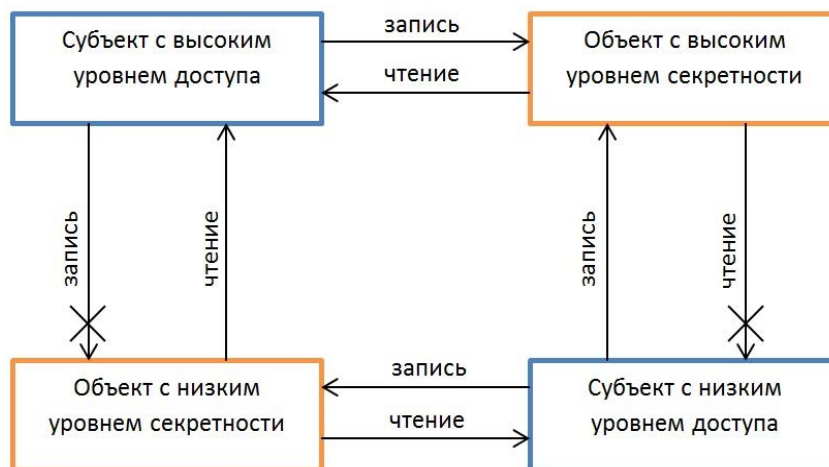


Рисунок 4.2 – Диаграмма информационных потоков модели мандатной защиты информации

Метка безопасности объекта (всей таблицы, отдельного ее столбца, строки или поля) может иметь сложную структуру, позволяющую определить группу принадлежности объекта, уровень его ценности и конфиденциальности, а также ряд других параметров, например, уровень защищенности устройства, на котором хранится объект. Метка безопасности объекта неотделима от самого объекта, физически хранится вместе с ним (а не в системном каталоге, как это происходит при логической защите) и уничтожается только в момент уничтожения объекта.

Мандатная защита обеспечивается специализированными серверами баз данных, которые блокируют «обход» меток безопасности при получении доступа к информации и, кроме того, предоставляют средства аудита доступа субъектов к защищаемым объектам.

В качестве примера рассмотрим систему мандатного управления доступом, реализованную сервером баз данных «Линтер», имеющим сертификат по второму классу защиты от несанкционированного доступа, что соответствует классу В3 американского национального стандарта.

Группы принадлежности

Субъекты доступа группируются (например, по отделам организации), при этом могут устанавливаться «доверительные отношения» между различными группами принадлежности субъектов. Объекты доступа,

* Автор рисунка – М. Савельев. URL: <https://ru.wikipedia.org/w/index.php?curid=2894287>

независимо от иерархии в базе данных, также распределяются по группам принадлежности, причем объект может принадлежать только одной такой группе. Группы принадлежности субъектов связываются с группами принадлежности объектов, при этом субъект получает права доступа только к объектам групп, связанных с группой этого субъекта, а также с группами, находящимися в доверительных отношениях с его группой.

Уровни доступа

Каждому **объекту** присваивается определенный *уровень ценности* (*WAL – Write Access Level*), ограничивающий возможность его модификации, и *уровень конфиденциальности* (*RAL – Read Access Level*), ограничивающий возможность просмотра (чтения) этого объекта. *WAL*- и *RAL*-уровни присваиваются также и **субъектам** доступа. По умолчанию объект наследует уровни того субъекта, который создал или изменил этот объект.

Метки безопасности

Метка безопасности *объекта доступа* включает следующие компоненты: группа принадлежности субъекта, создавшего объект; *RAL*-уровень объекта; *WAL*-уровень объекта.

Метка безопасности *субъекта доступа*: группа принадлежности субъекта; *RAL*-уровень субъекта, равный максимальному *RAL*-уровню объекта, доступному для прочтения этим субъектом; *WAL*-уровень субъекта, равный минимальному *RAL*-уровню объекта, доступному для модификации этим субъектом.

Субъект не получит права чтения объекта доступа, *RAL*-уровень которого выше его собственного *RAL*-уровня, тем самым конфиденциальная информация будет защищена от несанкционированного прочтения. *WAL*-уровень субъекта доступа ограничивает его возможности по понижению уровня конфиденциальности объекта: субъект не получит права модификации (изменения, удаления или вставки) объекта доступа, *RAL*-уровень которого выше *WAL*-уровня самого этого субъекта. Иными словами, пользователь не сможет сделать доступную ему информацию менее конфиденциальной, чем задано ее *RAL*-уровнем.

Глава 5

УПРАВЛЕНИЕ ДОСТУПОМ К ДАННЫМ MS SQL-SERVER

5.1 Двухуровневая архитектура управления доступом

MS SQL-Server поддерживает двухуровневую архитектуру системы управления доступом к данным (рисунок 5.1).

На верхнем уровне (уровень сервера) решаются задачи аутентификации пользователей и выдача им глобальных прав выполнения операций с базами данных, управляемыми этим сервером.

Объектами доступа на этом уровне являются пользовательские базы данных и типовые операции над ними.

В качестве субъектов доступа используются *учетные записи пользователей* (имена входа, login), и *фиксированные роли сервера*, за каждой из которых закреплено *имя роли* и определенный набор *разрешений* на выполнение операций с базами данных. Учетная запись – это средство идентификации и аутентификации пользователя, а роль сервера – средство группировки учетных записей для более удобного администрирования. Имеется два способа предоставления пользователю разрешений на выполнение глобальных операций с базами данных: индивидуальное назначение требуемых разрешений учетной записи пользователя или включение учетной записи в качестве члена в соответствующую серверную роль.

На нижнем уровне (уровень базы данных, доступ к которой получили пользователи через соответствующие учетные записи), решаются задачи разграничения прав их доступа к логическим объектам этой базы данных, а также задача шифрования данных объектов доступа.

Основные *субъекты доступа* нижнего уровня – это *пользователи базы данных* и *роли базы данных*, используемые для группового управления правами доступа пользователей к объектам базы данных. При этом *пользователь базы данных* всегда связан с определенной учетной записью, "предъявляемой" им на этапе аутентификации.

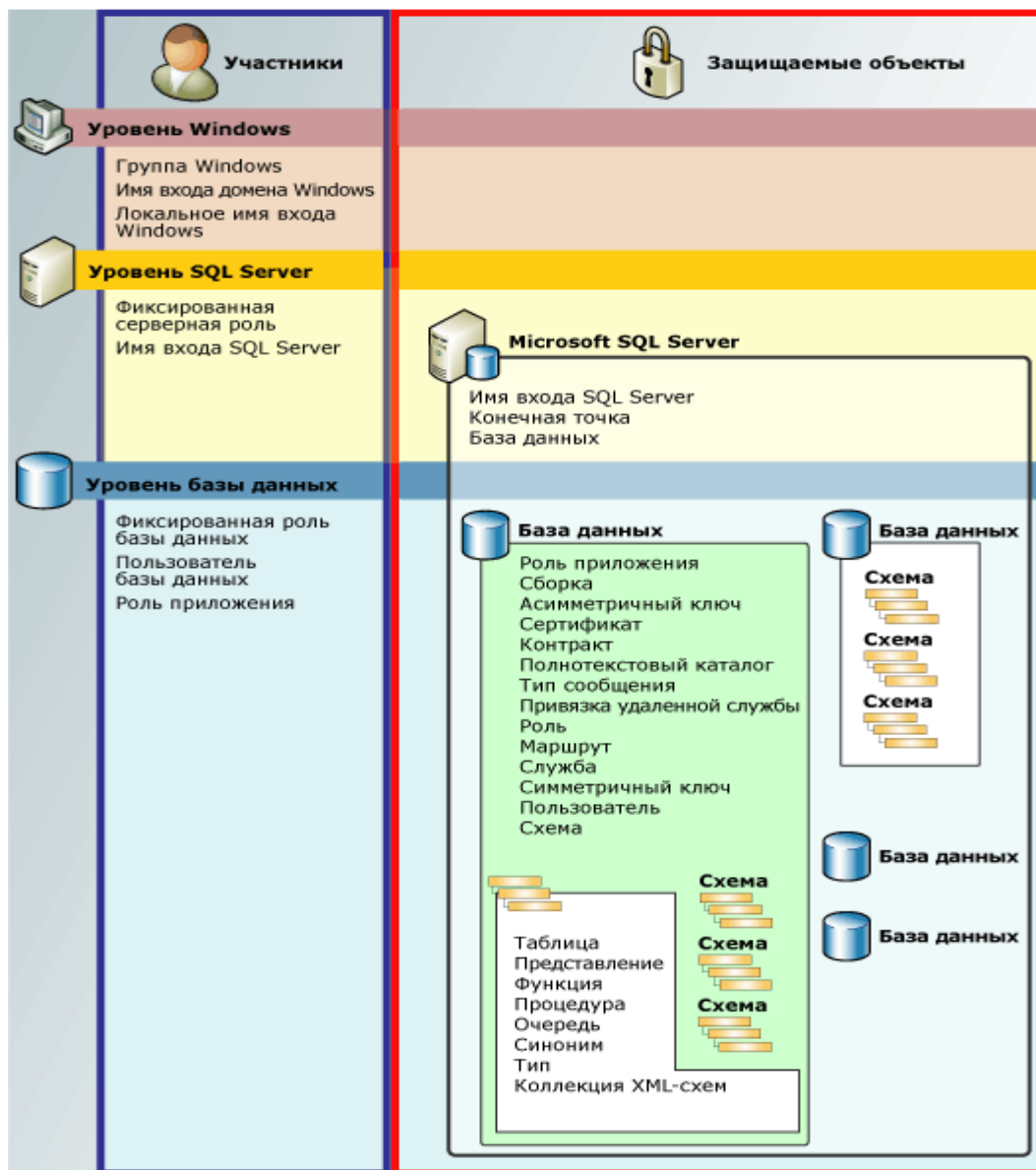


Рисунок 5.1 – Иерархия субъектов и объектов доступа MS SQL_Server

На уровне базы данных поддерживается три категории ролей:

- *предопределенный набор фиксированных ролей базы данных*, через членство в которых пользователи получают соответствующие права доступа ко всем объектам базы;
- *формируемые администратором пользовательские роли*, члены которых могут получать необходимые права доступа к любым логическим объектам базы данных;
- *роли приложений*, членство пользователей в которых не предусмотрено, а права доступа, назначенные этой роли, предоставляются клиентскому приложению, через которое пользователь подключается к базе данных.

Состав, имена и права доступа фиксированных ролей базы данных ко всем ее объектам также фиксированы, у администратора остается право управления членством пользователей в ролях этого типа. Пользовательские роли базы данных администратор вправе создавать, именовать и назначать им права доступа к логическим объектам базы по своему усмотрению (например, в соответствии с организационной структурой предприятия и должностными функциями сотрудников).

Основные объекты доступа на уровне базы данных – это схемы, таблицы и представления, отдельные столбцы таблиц и представлений, а также программные компоненты базы данных – хранимые процедуры и функции, которые, заметим, одновременно являются и субъектами доступа и могут наделяться соответствующими правами.

5.2 Управление доступом на уровне сервера

5.2.1 Режимы аутентификации

Аутентификация (проверка подлинности пользователя) – это первый рубеж, успешное преодоление которого дает право пользователю на соединение с сервером баз данных. MS SQL-Server реализует традиционный способ аутентификации пользователей – путем сравнения введенного пользователем пароля с "правильным" паролем, соответствующим имени учетной записи этого пользователя.

MS SQL-Server поддерживает два режима аутентификации – «Аутентификация Windows» и «Аутентификация SQL Server».

В процессе установки сервера баз данных по умолчанию назначается первый режим аутентификации, при котором пользователи, успешно прошедшие проверку подлинности при входе в операционную систему, автоматически получают право соединения с сервером. Такое решение позволяет использовать средства безопасности, предоставляемые операционной системой (как правило, более надежные по сравнению с соответствующими средствами, реализуемыми на уровне приложений), и, что также немаловажно, упрощает работу пользователей и снижает риск небезопасного хранения паролей, так пользователю требуется помнить единый пароль доступа к различным приложениям Windows.

Второй режим аутентификации (называемый также «смешанным») предполагает дополнительную регистрацию пользователей средствами сервера баз данных, выдачу им имен учетных записей и паролей с после-

дующей проверкой подлинности пользователей при их соединении с сервером.

Такой режим аутентификации предназначен, в основном, для клиентских приложений, функционирующих на платформах, отличных от Windows. Аутентификация средствами программного приложения считается менее безопасной, однако MS SQL Server поддерживает шифрование трафика между клиентом и сервером, в том числе с помощью сертификатов, сгенерированных самим сервером.

5.2.2 Учетные записи и разрешения уровня сервера

MS SQL Server использует два типа учетных записей: учетные записи Windows (*Windows Domain login* и *Windows Local login*) и учетные записи сервера (*SQL Server login*), используемые в случае выбора смешанного режима аутентификации.

При подключении к серверу от имени учетной записи Windows и при создании учетной записи сервера имя и идентификатор учетной записи сохраняются в таблице SysLogins базы данных Master (таблица 5.3). Для учетных записей сервера в этой таблице дополнительно сохраняется пароль.

Для создания учетной записи сервера можно воспользоваться SQL-командой CREATE LOGIN, например:

```
CREATE LOGIN NewUser WITH PASSWORD = 'NewP@sssW000rd'
```

Графический интерфейс SQL Server Management Studio содержит средства создания и удаления учетных записей сервера, а также средства просмотра и редактирования выданных им разрешений, доступные через вкладку «Свойства» учетной записи и вкладку «Разрешения» свойств сервера.

На рисунках 5.2 и 5.3 приведены примеры соответствующих экранных форм.

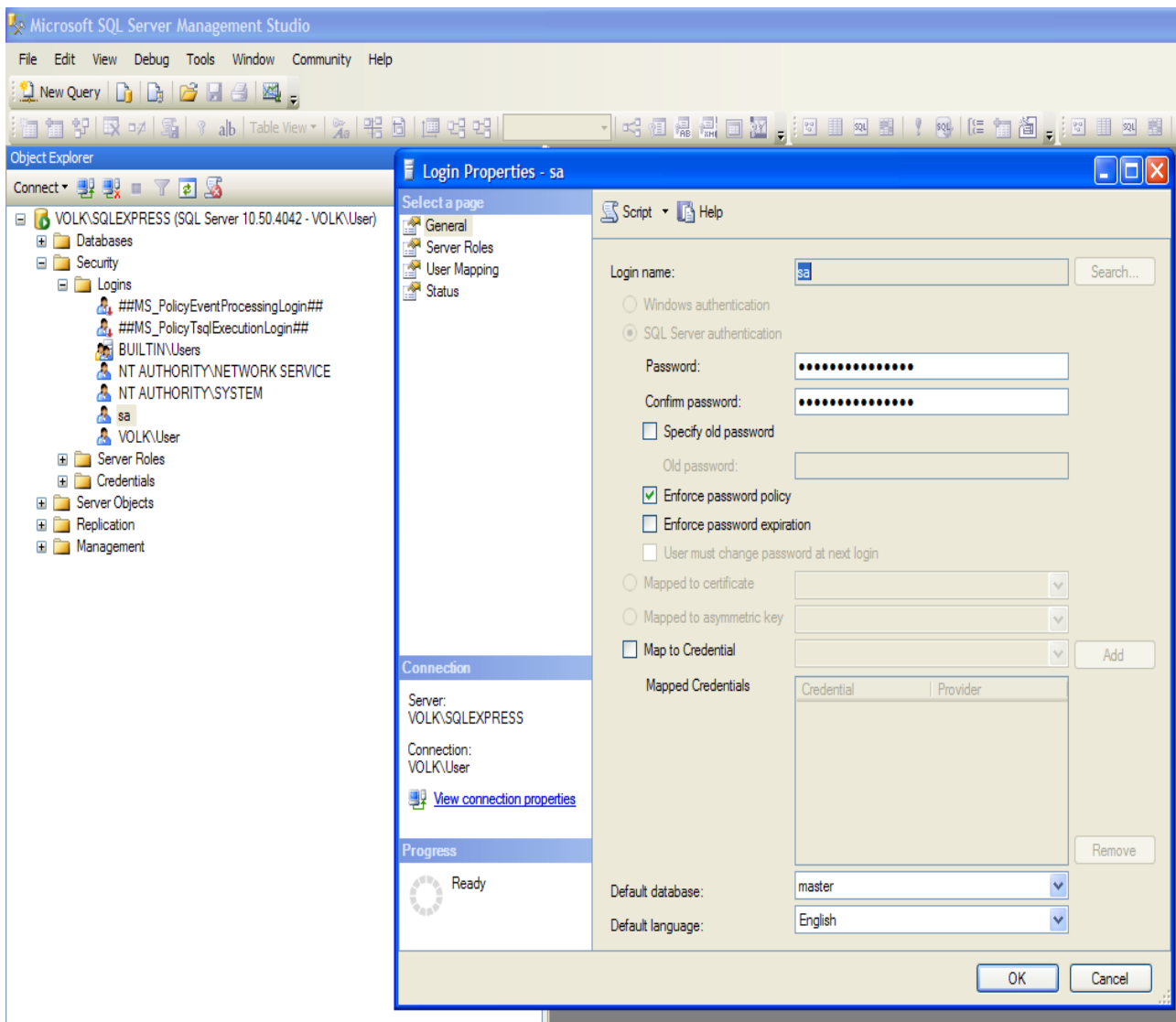


Рисунок 5.2 – Средства просмотра свойств учетных записей

На вкладке **Logins** отображается набор создаваемых автоматически (так называемых "встроенных") учетных записей Windows и учетная запись **sa** (*System Administrator*) – единственная учетная запись сервера, создаваемая по умолчанию. Эта учетная запись обладает правами системного администратора сервера, но использовать ее можно только в случае, если выбран режим аутентификации средствами SQL Server.

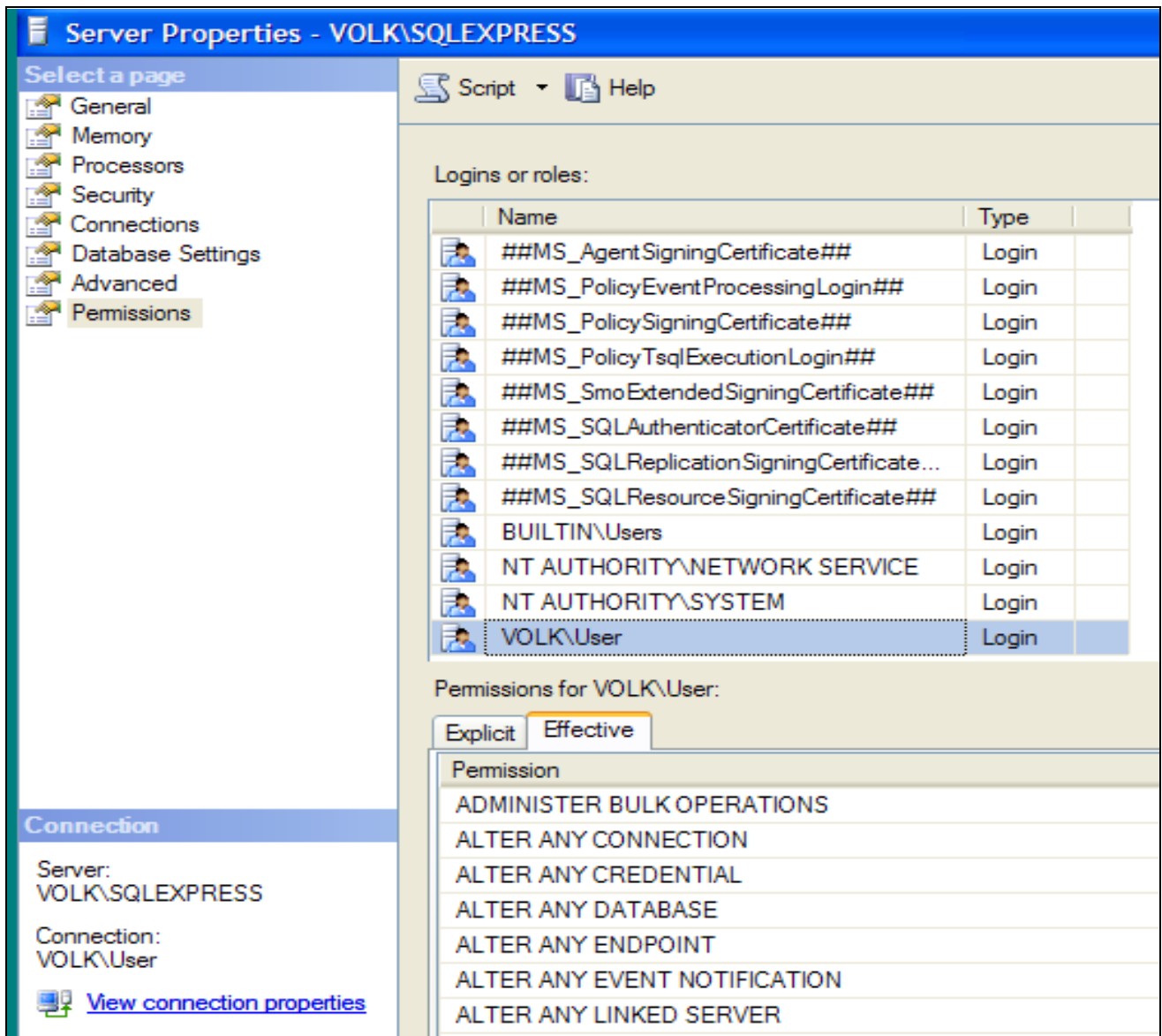


Рисунок 5.3 – Средства просмотра и редактирования разрешений

5.2.3 Фиксированные роли сервера

Для предоставления пользователю фиксированного набора разрешений на выполнение работ по администрированию сервера учетную запись этого пользователя следует включить в соответствующую серверную роль. Состав таких ролей, их наименования и связанный с каждой ролью набор административных задач строго фиксированы (таблица 5.1), при этом ни один из администраторов сервера не может ни создавать новые серверные роли, ни удалять существующие роли, ни изменять их свойства.

Таблица 5.1 – Фиксированные роли сервера

Имя роли	Назначение роли и права членов роли
sysadmin	Роль для системного администратора сервера баз данных. Полные права на SQL Server, в том числе и право передачи этих прав другим пользователям
serveradmin	Роль для оператора, обслуживающего сервер с правами отключения сервера и изменения любых его настроек. Запрещен доступ к данным, отсутствует право изменения разрешений на доступ к объектам
dbcreator	Члены роли получают право создания баз данных (и автоматически становится ее владельцем с полными правами в созданной базе данных)
securityadmin	Роль для администратора безопасности с правами управления "обычными" пользователями: создание учетных записей, изменение паролей, предоставление прав доступа к данным. Запрещено любое вмешательство в работу сервера, изменение административных прав и учетных записей администраторов
setupadmin	Члены роли получают право подключения внешних (так называемых "связанных") серверов баз данных
processadmin	Члены роли получают единственное право: право закрытия пользовательских подключений к серверу (например, зависших)
diskadmin	Члены роли получают право выполнять любые операции с файлами баз данных
bulkadmin	Роль для сотрудников, которые выполняют специфическую операцию – массовую загрузку данных в таблицы

Члены любой фиксированной роли сервера могут добавлять *в эту роль* любую учетную запись. Члены роли sysadmin могут добавлять учетные записи *в любую фиксированную роль* сервера.

Есть еще одна серверная роль – это роль PUBLIC, членами которой автоматически становятся все пользователи, подключившиеся к серверу, причем лишить пользователя членства в этой роли нельзя. Роль PUBLIC используется для предоставления разрешений всем пользователям сервера.

ра. По умолчанию у этой роли есть только два разрешения: разрешение «CONNECT» на соединение с сервером и разрешение «VIEW ANY DATABASE» на просмотр списка баз данных, управляемых сервером.

Роль PUBLIC занимает особое место в составе серверных ролей:

- во-первых, отсутствуют средства управления членством учетных записей в этой роли (и в этом нет необходимости, так как членами роли автоматически становятся все учетные записи, получившие доступ к серверу);

- во-вторых, эта роль не отображается в списке серверных ролей, возвращаемых хранимой процедурой `sp_helpsrvrole` (таблица 5.2), однако она присутствует на вкладке \Безопасность\Роли_сервера обозревателя объектов SQL Server Management Studio;

- в-третьих, роль PUBLIC, строго говоря, не является "фиксированной" – в отличие от других ролей сервера в окне свойств этой роли можно изменить её текущие разрешения.

Управление фиксированными серверными ролями реализуется соответствующими системными хранимыми процедурами, состав и форматы обращения к которым приведены в таблице 5.2.

Таблица 5.2 – Средства управления фиксированными серверными ролями

Хранимая процедура	Входные параметры	Результат
<code>sp_helpsrvrole</code>	–	Список серверных ролей
<code>sp_helpsrvrolemember</code>	<code>['role']</code>	Список членов роли 'role'. Если входной параметр опущен, процедура возвращает список членов всех серверных ролей
<code>sp_addsrvrolemember</code>	<code>'login', 'role'</code>	Добавление учетной записи 'login' в роль 'role'
<code>sp_dropsrvrolemember</code>	<code>'login', 'role'</code>	Удаление учетной записи 'login' из роли 'role'
<code>sp_srvrolepermission</code>	<code>['role']</code>	Список разрешений указанной роли. Если входной параметр опущен, процедура возвращает список разрешений для всех серверных ролей

5.2.4 Хранение информации об учетных записях

Для хранения информации об учетных записях пользователей и их членстве в фиксированных серверных ролях используется системная таблица SysLogins системной базы данных Master. Каждая строка этой таблицы содержит информацию об одной учетной записи (таблица 5.3).

Таблица 5.3 – Структура системной таблицы SysLogins

Имя поля	Тип данных	Описание
sid	varbinary(85)	(Security Identifier) – уникальный идентификатор безопасности учетной записи
createdate	datetime	Дата добавления учетной записи
updatedate	datetime	Дата обновления учетной записи
name	sysname	Имя учетной записи
dbname	sysname	Имя базы данных по умолчанию для пользователя при установлении соединения
password	nvarchar(128)	Хешированное значение пароля (= NULL для учетных записей Windows)
language	sysname	Язык по умолчанию для пользователя
denylogin	int	= 1 – учетной записи Windows отказано в доступе к серверу
hasaccess	int	= 1 – учетной записи Windows разрешен доступ к серверу
isntname	int	= 1 – для учетных записей Windows; = 0 – для учетных записей SQL Server
isntgroup	int	= 1 – для учетных записей групп Windows
isntuser	int	= 1 – для учетных записей пользователей Windows
sysadmin	int	= 1, если учетная запись является членом одноименной серверной роли
securityadmin	int	
serveradmin	int	
setupadmin	int	
processadmin	int	
diskadmin	int	
dbcreator	int	
bulkadmin	int	

5.3 Управление доступом на уровне базы данных

5.3.1 Объекты доступа: таблицы, представления, команды и схемы

Пользователь, прошедший процедуру аутентификации и получивший через свою учетную запись доступ к серверу с соответствующими глобальными разрешениями на выполнение операций с базами данных, должен получить у администратора требуемый ему набор прав доступа к логическим объектам базы данных.

Объекты доступа базы данных – это *таблицы, представления, хранимые процедуры и функции*, а также *SQL-команды*, предназначенные для создания логических объектов (CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, CREATE FUNCTION, CREATE RULE, CREATE DEFAULT) и резервных копий (BACKUP DATABASE, BACKUP LOG) базы данных, и *схемы*, используемые для группировки объектов.

Схемы

Схема – это именованный объект-контейнер, предназначенный для группировки логических объектов базы данных с целью упрощения процесса управления правами доступа к ним со стороны пользователей базы данных.

При создании базы данных в ней автоматически создаются схемы, владельцами которых являются фиксированные роли и специальные (встроенные) пользователи базы данных. Этим схемам присваиваются имена их владельцев.

Объект базы данных (таблица или представление) не может одновременно входить в состав нескольких схем, при этом допускается перемещение объектов между схемами одной базы данных (ALTER SCHEMA).

Схема может быть удалена (DROP SCHEMA) при условии, что она пуста. Перед удалением схемы все ее объекты должны быть либо удалены, либо перемещены в другие схемы базы данных.

При создании схемы (CREATE SCHEMA) должен быть задан ее владелец – субъект доступа любого типа, получающий соответствующие права доступа ко всем объектам, включенным в схему.

5.3.2 Субъекты доступа: пользователи и роли базы данных

Права на выполнение операций с логическими объектами базы данных предоставляются именованным *субъектам доступа* – *пользователям* и *ролям* базы данных, а также *ролям приложений*.

Пользователи базы данных

Основным субъектом доступа на уровне базы данных является пользователь, которому могут быть индивидуально выданы права доступа к объектам базы данных. При создании пользователя базы данных сервер связывает его с определенной учетной записью (субъектом доступа уровня сервера), и, таким образом, владелец учетной записи получает соответствующие права доступа к объектам базы данных. При этом учетная запись не может быть связана более, чем с одним пользователем одной базы данных.

При создании базы данных в ней автоматически создаются два специальных пользователя: **dbo** и **guest**. Пользователь **dbo** (database owner) является *владельцем* базы данных и имеет в этой базе абсолютные права, в том числе и право наделения других пользователей правами владельца базы данных. С пользователем **dbo** связана та учетная запись, которой на уровне сервера было выдано разрешение CREATE DATABASE и владелец которой воспользовался этим разрешением и создал базу данных. Пользователь **dbo** автоматически становится членом фиксированной роли базы данных **db_owner**.

Если учетной записи явно не предоставлен доступ к базе данных, то она автоматически отображается сервером в пользователя **guest**, следовательно, права доступа к объектам базы данных, назначенные пользователю guest, автоматически получают все учетные записи сервера. Для снижения рисков нарушения безопасности рекомендуется удалять пользователя guest из базы данных.

Роли базы данных

Для группировки пользователей базы данных используются *роли*, которые сами являются именованными субъектами доступа, то есть для ролей, как и для пользователей, могут быть определены права доступа к логическим объектам базы данных. При этом права роли автоматически получают все ее члены. Роль всегда имеет *владельца*, в качестве которого может выступать пользователь или другая роль базы данных, и *набор схем*, принадлежащих этой роли.

Фиксированные роли базы данных

В каждой базе данных всегда присутствует предопределенный набор фиксированных ролей (таблица 5.4), с каждой из которых связано определенное множество глобальных разрешений – прав доступа ко всем объектам базы данных.

Роли этого типа предназначены, в основном, для пользователей, выполняющих функции администрирования базы данных.

Таблица 5.4 – Фиксированные роли базы данных

Роль	Права членов роли
db_owner	Права владельца, выполнение любых действий с базой данных
db_securityadmin	Управление правами доступа к объектам базы данных других пользователей и членством их в ролях
db_accessadmin	Управление пользователями базы данных: создание, удаление и изменение
db_ddladmin	Создание, изменение и удаление объектов базы данных
db_datawriter	Разрешено изменение / просмотр данных любых таблиц или представлений базы данных
db_datareader	
db_denydatawriter	Запрещено изменение / просмотр данных любых таблиц или представлений базы данных независимо от выданных разрешений
db_denydatareader	
db_backupoperator	Резервное копирование базы данных
public	Любой пользователь, созданный в базе данных, автоматически включается в роль public и не может быть удален из нее. Роль предназначена для предоставления <i>прав доступа по умолчанию</i> (default right) всем пользователям базы данных

Владельцем всех фиксированных ролей базы данных является пользователь **dbo**, а членами фиксированной роли могут быть только пользователи базы данных.

Пользовательские роли базы данных

В отличие от фиксированных ролей *пользовательские роли* формируются администратором и предназначены для группировки пользователей, которым следует назначить одинаковые права доступа к логическим объектам базы данных. Набор объектов, связанных с пользовательской ролью, и права доступа к каждому из них не являются предопределенными и могут быть сформированы индивидуально для каждой такой роли.

Другим существенным отличием пользовательских ролей от фиксированных является возможность членства пользовательской роли в другой пользовательской роли с соответствующим наследованием дочерними ролями прав доступа к объектам, установленным для родительских ролей.

Роли приложений

Имеется еще одна категория ролей уровня базы данных – это *роли приложений*. Членство пользователей в ролях приложений не предусмотрено – роли этого типа предназначены для предоставления прав доступа клиентским приложениям, через которые пользователи подключаются к базе данных. Отличительной особенностью ролей приложений является наличие у каждой из них специального пароля, который должен быть отправлен приложением серверу для получения прав доступа, назначенных соответствующей роли.

5.3.3 Хранение информации о субъектах доступа

Для хранения информации о пользователях и ролях базы данных всех трех перечисленных выше типов используется системная таблица `SysUsers` пользовательской базы данных (таблица 5.5).

Каждая строка этой таблицы представляет один субъект доступа – пользователя, фиксированную или пользовательскую роль базы данных или роль приложения.

Таблица 5.5 – Структура системной таблицы SysUsers

Имя столбца	Тип данных	Описание
uid	smallint	Идентификатор субъекта доступа (пользователя или роли), уникальный в пределах базы данных. uid = 1 – для пользователя dbo uid = 2 – для пользователя guest $3 \leq \text{uid} < 16384$ – для прочих субъектов доступа uid ≥ 16384 – для фиксированных ролей
name	sysname	Имя субъекта доступа
sid	varbinary(85)	Идентификатор безопасности учетной записи пользователя или владельца роли
roles	varbinary(2048)	Битовая строка, определяющая членство субъекта доступа в ролях сервера. Если roles[uid] = 1, то субъект доступа является членом роли, идентификатор которой равен uid (в новых версиях SQL Server не используется)
createdate	datetime	Дата создания субъекта доступа
updatedate	datetime	Дата последнего изменения субъекта доступа
password	varbinary(256)	Пароль для ролей приложения. = null для других типов субъектов доступа
gid	smallint	Идентификатор роли (в старых версиях сервера – группы, откуда и название поля): если gid = uid, субъект доступа – фиксированная или пользовательская роль базы данных; для других типов субъектов доступа gid = 0
hasdbaccess	int	Если = 1, то пользователь имеет доступ к базе данных
islogin	int	Если = 1, то субъект доступа соответствует учетной записи любого типа
isntname	int	Если = 1, то пользователь является отображением учетной записи Windows
isntgroup	int	Если = 1, то пользователь является отображением учетной записи группы Windows
isntuser	int	Если = 1, то пользователь является отображением учетной записи пользователя Windows
isqluser	int	Если содержит 1, то пользователь является отображением учетной записи пользователя SQL Server
isaliased	int	Если = 1, то пользователь является псевдонимом другого пользователя. (Актуально для учетных записей SQL Server 6.x)
isqlrole	int	Если = 1, то субъект доступа – пользовательская или фиксированная роль базы данных
isapprole	int	Если = 1, то субъект доступа – роль приложения

5.3.4 Программные средства управления пользователями и ролями

Команда `CREATE USER` создает пользователя базы данных, сопоставляя его с учетной записью (сертификатом, асимметричным ключом) или схемой по умолчанию:

```
CREATE USER user_name
  [ FOR
    {
      LOGIN login_name
      | CERTIFICATE cert_name
      | ASYMMETRIC KEY asym_key_name
    }
  | WITHOUT LOGIN
]
[ WITH DEFAULT_SCHEMA = schema_name ]
```

Листинг 5.1 – Формат SQL-команды `CREATE USER`

Параметр `WITHOUT LOGIN` создает пользователя, который не сопоставляется с учетной записью, такой пользователь может подключиться к базе данных как `guest`. Если параметр `DEFAULT_SCHEMA` не задан, пользователю в качестве схемы по умолчанию будет назначена схема `dbo`.

Команда `ALTER USER` позволяет переименовать пользователя базы данных, сопоставить его с новой учетной записью или определить для него новую схему по умолчанию:

```
ALTER USER userName
WITH
{
  NAME = newUser_name
| DEFAULT_SCHEMA = schemaName
| LOGIN = loginName
}
[ , ...n ]
```

Листинг 5.2 – Формат SQL-команды `ALTER USER`

Команда `DROP USER userName` удаляет пользователя базы данных.

Команда `CREATE ROLE role_name [AUTHORIZATION owner_name]` создает роль базы данных `role_name`, владельцем которой

назначается *owner_name*. Если параметр `AUTHORIZATION` не задан, владельцем созданной роли назначается субъект, выполнивший команду `CREATE ROLE`.

Команда `ALTER ROLE` позволяет переименовать роль, включить в нее новых членов или удалить членов из роли:

```
ALTER ROLE role_name
{
    [ADD MEMBER database_principal]
  | [DROP MEMBER database_principal]
  | WITH NAME = new_role_name
}
```

Листинг 5.3 – Формат SQL-команды `ALTER ROLE`

5.3.5 Программные средства управления правами доступа

В разделе 4.3 учебного пособия приведен краткий обзор типов прав доступа к объектам базы данных, которые поддерживает MS SQL Server. Каждый из этих типов может быть отнесен к одной из следующих категорий:

- права доступа к данным таблиц и представлений базы данных;
- права на выполнение хранимых процедур и функций;
- права на выполнение команд Transact-SQL.

При создании нового пользователя базы данных он автоматически становится членом роли `PUBLIC` и наделяется правами доступа, установленными для этой роли.

Субъекту доступа (пользователю или пользовательской роли базы данных) можно предоставить (или отнять) права *неявно* – через членство в фиксированных или пользовательских ролях базы данных, или явно, например, SQL-операторами `Grant`, `Deny` или `Revoke`.

Grant – предоставление доступа

Для предоставления субъектам прав доступа к таблицам базы данных, а также к представлениям, процедурам и функциям, используется SQL-оператор `GRANT`, имеющий следующий синтаксис (листинг 5.4 а):

```
GRANT
{ ALL | permission [,...n] }
ON table | view [( column [,...n] )]
| ON procedure | ON function
TO security_account [,...n]
[WITH GRANT OPTION]
[AS (role)]
```

а)

```
GRANT
{ALL | statement[,...n]}
TO security_account [, .. .n]
```

б)

а) представление прав доступа к объектам;

б) предоставление прав выполнения SQL-команд

Листинг 5.4 – Формат SQL-команды GRANT

Параметры команды GRANT:

- ALL – позволяет предоставить все права доступа к объекту (актуальные для типа объекта, указанного в разделе ON);
- permission [,...n] – вид предоставляемого права доступа (SELECT, INSERT, DELETE, UPDATE, REFERENCES, или EXECUTE), при этом одной командой может быть предоставлено несколько видов прав;
- ON table | view – имя объекта доступа (таблицы или представления);
- [(column [,...n])] – имена столбцов таблицы или представления: если задан этот необязательный параметр, то права (в этом случае только SELECT или UPDATE) будут предоставлены только указанным столбцам; предоставление прав на уровне таблицы или представления отменяет все права, ранее предоставленные на уровне столбцов;
- ON procedure – имя стандартной или расширенной хранимой процедуры (разрешается выдавать только право EXECUTE);
- ON function – имя пользовательской функции (для скалярных функций разрешается выдавать права EXECUTE или REFERENCES, для TVF-функций, возвращающих табличные значения, дополнительно разрешается выдавать права SELECT, INSERT и DELETE);

- TO security_account [, ...n] – список субъектов доступа (пользователей и/или ролей базы данных), которым предоставляются права;

- [WITH GRANT OPTION] – при наличии этого параметра субъекты доступа, получившие указанные в операторе права, получают право предоставления этих прав другим субъектам доступа (за исключением прав доступа к отдельным столбцам таблиц и представлений);

- [AS (role)] – использование этого необязательного параметра дает возможность пользователю, не имеющему явно выданных ему прав на предоставление прав другим субъектам доступа, выдавать права доступа от имени своей "родительской" роли, имеющей такие права.

Право выполнения SQL-операторов CREATE TABLE, CREATE VIEW, CREATE PROCEDURE, CREATE FUNCTION, CREATE RULE, CREATE DEFAULT, BACKUP DATABASE, BACKUP LOG также предоставляется командой GRANT (листинг 5.4 б), в которой параметр statement [, ...n] задает список SQL-операторов из приведенного выше перечня.

Deny - отклонение (запрет) доступа

Получив от администратора набор прав доступа, пользователь сможет выполнять разрешенные ему действия. Но в некоторых случаях бывает необходимо наложить запрет на выполнение им определенных операций с определенными объектами базы данных.

Глобальный запрет доступа субъекта ко всем объектам базы данных может быть реализован путем включения этого субъекта в фиксированные роли базы данных db_denydatareader (глобальный запрет чтения данных) или db_denydatawriter (глобальный запрет модификации данных).

Если же требуется запретить субъекту доступ к отдельным объектам базы данных, сохранив при этом предоставленные ему ранее права доступа к другим объектам, администратор может воспользоваться SQL-оператором DENY, синтаксис которого приведен ниже (листинг 5.5 а).

```
DENY
{ ALL | permission [,...n] }
ON table | view [(column
[,...n])]
| ON procedure | ON function
TO security_account [,...n]
[CASCADE]
```

а)

а) запрет доступа к объектам базы данных;

б) запрет выполнения SQL-команд

```
DENY
{ALL | statement[,...n]}
TO security_account [,.. .n]
```

б)

Листинг 5.5 – Формат SQL-команды DENY

Эта же команда позволяет запретить субъекту право выполнения отдельных SQL-операторов, сохранив предоставленные ему ранее права (листинг 5.5 б).

Параметры команды DENY аналогичны соответствующим параметрам команды GRANT. Необязательный параметр [CASCADE] предписывает *каскадирование* запрета доступа, указанного в команде DENY: если пользователь, которому эта команда запрещает доступ к объекту, ранее предоставлял к нему доступ другим пользователям, то команда запретит доступ к этому объекту и этим пользователям, а также всем их "потомкам" всех уровней.

Если запрет доступа требуется распространить на группу из нескольких пользователей, будет целесообразно создать специальную пользовательскую роль, SQL-оператором DENY запретить ей доступ к соответствующим объектам и затем включить в эту роль всю группу пользователей.

Согласно базовому правилу функционирования системы ограничения доступа, *запрет доступа* (Deny) имеет более высокий приоритет, чем его предоставление (Grant).

Если администратор запрещает пользователю (или пользовательской роли) доступ к объекту (явно или через членство в соответствующей роли), а ранее такой доступ был ему предоставлен, то система безопасности гарантирует отсутствие прав доступа до момента отмены (Revoke) этого запрета или исключения пользователя из числа членов "запрещающей" роли.

REVOKE – отмена явно предоставленных прав и запретов доступа

Если пользователю был предоставлен доступ к объекту через членство в роли базы данных, но при этом ему был явно запрещен (DENY) доступ к этому же объекту, то запрет, как более приоритетный по сравнению с разрешением, будет действовать, и пользователь будет лишен соответствующего доступа. В такой ситуации явный запрет может быть снят оператором REVOKE, после чего пользователь вновь получит доступ к объекту, как член роли.

Оператор REVOKE может также отменить явно выданное субъекту право (GRANT) доступа к объекту (листинг 5.6 а) или отменить явное предоставление или запрет права выполнения SQL-операторов (листинг 5.6 б).

```
REVOKE [GRANT OPTION FOR]
{ ALL | permission [,...n] }
ON table | view [(column
[,...n])]
| ON procedure | ON function
TO security_account [,...n]
[CASCADE]
[AS (role)]
```

а)

а) отмена явно выданных прав доступа к объекту;

б) отмена прав выполнения SQL-команд

```
REVOKE
{ALL | statement[,...n]}
TO security_account [,... .n]
```

б)

Листинг 5.6 – Формат SQL-команды REVOKE

Параметры оператора REVOKE аналогичны соответствующим параметрам команд GRANT и DENY, за исключением необязательного параметра [GRANT OPTION FOR]. Если в операторе REVOKE присутствует параметр [GRANT OPTION FOR], и при этом отменяемое право доступа было предоставлено субъекту оператором GRANT без использования параметра [WITH GRANT OPTION], то само право доступа субъекта отменено не будет, а будет отменено только его право на предоставление указанного права другим субъектам; в противном случае будет отменено само право доступа.

Если в операторе REVOKE присутствует параметр [CASCADE], то каскадная отмена прав доступа, предоставленных субъекту с помощью параметра [WITH GRANT OPTION], приведет к отмене этих прав, независимо от наличия параметра [GRANT OPTION FOR] в операторе REVOKE.

Просмотр прав доступа

Среда SQL Management Studio содержит средства просмотра прав доступа через вкладку «свойства» соответствующего объекта, пользователя или роли базы данных. Эту же информацию можно получить средствами Transact-SQL, используя системную хранимую процедуру sp_helpprotect:

```
sp_helpprotect ['object' | 'statement']
               [, 'security_account']
               [, 'grantor'] [, 'type']
```

Листинг 5.7 – Формат вызова системной процедуры sp_helpprotect

Параметры процедуры sp_helpprotect:

- ['object' | 'statement'] – имя объекта, категории объектов базы данных, или команды Transact-SQL, о правах доступа к которым предполагается получить информацию, например: 'sysusers', 'tables'; 'views', 'procedures', 'Create Table', 'Backup Database';
- ['security_account'] – имя субъекта доступа;
- ['grantor'] – имя субъекта доступа, который предоставил право;
- ['type'] – тип прав доступа: 'o' – только права доступа к объектам; 's' – только права выполнения команд; 'os' – оба типа прав (значение по умолчанию).

Ни один из параметров не является обязательным: при выполнении процедуры без параметров она возвратит список всех прав, выданных всем субъектам на все объекты базы данных. Указание (позиционно!) любого из параметров накладывает соответствующий фильтр на результирующий список прав доступа например: sp_helpprotect

'Create Role', sp_helprotect Null,'User1' или
sp_helprotect Null,Null,'User2'.

Правом вызова хранимой процедуры sp_helprotect по умолчанию владеет фиксированная роль базы данных Public, следовательно, каждый пользователь базы данных может получить информацию о правах доступа любого из её пользователей к любому её объекту.

Системное представление sys.database_permissions [12] и TVF-функция fn_builtin_permissions() [13] позволяют получить более детальную информацию о правах доступа.

Общие методические указания

Структура и содержание. Лабораторный практикум содержит семь лабораторных работ, отражающих следующие аспекты администрирования:

- управление физической моделью базы данных (работы №1 и №2);
- управление индексными структурами данных (работа №3);
- анализ процедурных планов выполнения SQL-запросов (работа №4);
- анализ архитектуры системы информационной безопасности (работа №5);
- управление правами доступа пользователей (работы №6 и №7).

Каждая лабораторная работа содержит несколько взаимосвязанных практических заданий, выполнение которых направлено на решение поставленных в работе задач и требует освоения и применения соответствующих инструментальных средств.

Программное обеспечение. Все лабораторные работы выполняются в системе SQL-Server Management Studio, рекомендуемая версия сервера баз данных – 2008R2 *Express*. Дополнительная информация об использовании программных средств размещена на официальном ресурсе корпорации Microsoft*.

Отчет по лабораторной работе должен содержать:

- титульный лист (кафедра, дисциплина, номер и наименование работы, исполнитель, дата представления отчета);
- цели и задачи, описание методики проведения работы, используемых структур данных и инструментальных программных средств;
- иллюстративный материал (листинги программных компонентов, выводимые на экран результаты их работы, графический материал и пр.);
- анализ полученных результатов с собственными выводами;
- ответы на контрольные вопросы (при их наличии).

Защита. Лабораторная работа выполняется индивидуально, защита работы проводится в форме собеседования по материалу представленного отчета. В процессе защиты оценивается полнота и качество выполнения практических заданий, грамотность использования инструментальных средств, правильность и обоснованность выводов по результатам работы, качество оформления отчета.

* URL: [https:// technet. microsoft.com/ru-ru/library/](https://technet.microsoft.com/ru-ru/library/)

ЛАБОРАТОРНАЯ РАБОТА №1

Анализ файловой структуры баз данных

Цель работы: ознакомление с программной архитектурой сервера баз данных и приобретение практических навыков применения инструментальных программных средств, используемых разработчиками и администраторами для управления файловой структурой баз данных.

Задачи:

- изучить пользовательский интерфейс программной среды *SQL-Server Management Studio*;
- исследовать файловую структуру системной базы данных «*Model*», используемую в качестве шаблона для создания пользовательских баз данных;
- освоить технику создания пользовательских баз данных средствами *MS SQL-Server Management Studio*;
- освоить технику создания пользовательских баз данных соответствующими средствами языка *Transact SQL* (операторы `Create Database`, `Create Table`, `Alter Table`);
- освоить технику модификации параметров файловой структуры пользовательских баз данных средствами *MS SQL-Server Management Studio*;
- освоить технику модификации параметров файловой структуры баз данных средствами языка *Transact SQL* (оператор `Alter Database`);
- исследовать объекты системного каталога базы данных, ответственные за хранение параметров её файловой структуры.

Практические задания:

Задание 1. Анализ файловой структуры базы данных «*Model*».

- 1.1 Активизируйте системную базу данных «*Model*». Определите свойства файловой структуры (состав и имена файлов и файловых групп, размеры и прочие параметры файлов) этой базы данных.
- 1.2 Просмотрите и проанализируйте схемы и содержимое системных таблиц `sysFileGroups` и `sysFiles` этой базы данных (через соответствующие им «одноименные» системные пред-

ставления (`sys.sysfilegroups` и `sys.sysfiles`) и прямым доступом к этим таблицам SQL-оператором `Select`).

- 1.3 На базе таблиц `sysfilegroups` и `sysfiles` напишите запрос (представление – *view*) для визуализации информации о технических параметрах файлов базы данных и их распределении по группам.
- 1.4 Сохраните результаты выполнения задания в отчете по лабораторной работе.

Задание 2. Создание пользовательских баз данных

- 2.1 Создайте пользовательскую базу данных и сформируйте её схему (2-3 связанных таблицы) средствами *SQL-Server Management Studio*.
- 2.2 Активизируйте созданную базу данных и, не заполняя таблиц данными, выполните в контексте этой базы данных задание 1.2.
- 2.3 В контексте этой базы данных выполните запрос, созданный при выполнении задания 1.3. Проанализируйте результат выполнения запроса.
- 2.4 Создайте еще одну пользовательскую базу данных средствами *Transact SQL* (оператор `Create Database`). Создайте в этой базе данных две вторичные файловые группы, одной из которых установите свойство «по умолчанию». Создайте по два вторичных файла в каждой из вторичных файловых групп. Создайте в этой базе данных 4-5 простых таблиц, определите для этих таблиц файловые группы.
- 2.5 Повторите задания 2.2 и 2.3.
- 2.6 Сохраните результаты выполнения задания в отчете.

Задание 3. Модификация файловой структуры баз данных

- 3.1 Используя средства *SQL-Server Management Studio*, измените параметры файловой структуры одной из пользовательских баз данных, созданных при выполнении предыдущего задания:
 - увеличите в 2 раза начальный размер первичного файла базы данных;
 - уменьшите в 2 раза шаг приращения размера этого файла;
 - создайте две дополнительные (вторичные) файловые группы;
 - создайте во вторичных файловых группах по два (вторичных) файла базы данных.

3. 2 В контексте этой (модифицированной) базы данных выполните запрос, созданный при выполнении задания 1.3. Проанализируйте результат.
3. 3 Измените по своему усмотрению параметры файловой структуры системной базы данных «*Model*» (размер первичного файла базы данных, количество вторичных файловых групп и вторичных файлов).
3. 4 Создайте новую пользовательскую базу данных.
3. 5 В контексте этой базы данных выполните задания 2.2 и 2.3.
3. 6 Сохраните результаты выполнения задания в отчете.

Задание 4. Анализ файловой структуры внешней базы данных

4. 1 Подключите к MS SQL-Server пользовательскую базу данных, реализованную в среде MS Access (используйте результаты своей контрольной работы, выполненной ранее).
4. 2 Проконтролируйте результаты преобразования в формат *MS SQL-Server* схемы этой базы данных, содержимого таблиц и SQL-запросов.
4. 3 Определите свойства файловой структуры этой базы данных (состав и имена файлов и файловых групп, размеры и прочие параметры файлов) и сравните их с соответствующими параметрами системной базы данных «*Model*».
4. 4 Сохраните результаты выполнения задания в отчете.

ЛАБОРАТОРНАЯ РАБОТА №2

Анализ алгоритмов резервирования дисковой памяти

Цель работы: исследование типовых алгоритмов управления процессами распределения файловых страниц между логическими объектами базы данных, и приобретение практических навыков использования инструментальных средства администратора для анализа и управления физической моделью данных.

Задачи:

- изучить внутреннюю организацию файлов базы данных в системе MS SQL-Server и внутреннюю организацию файловых страниц;
- изучить структуру объектов системного каталога базы данных, ответственных за хранение параметров её физической модели;
- освоить языковые (TransactSQL) средства создания и модификации логических объектов базы данных;
- освоить технику анализа физической модели базы данных с использованием команд системной утилиты DBCC и системных хранимых процедур;
- исследовать алгоритмы выделения дисковой памяти для хранения логических объектов базы данных (таблиц и индексов), реализованные в MS SQL-Server.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Лабораторная работа содержит четыре задания, каждое из которых предполагает постановку несложного эксперимента с привлечением программных компонентов базы данных и последующий анализ полученных результатов.

В процессе выполнения заданий продолжается знакомство с компонентами системного каталога базы данных (таблицами SysObjects, SysIndexes и соответствующими им системные представлениями), а также приобретается опыт использования встроенных программных средств, предназначенных для создания и анализа компонентов файловой структуры базы данных:

- SQL-операторы подмножества DDL языка TransactSQL: CreateDatabase, CreateTable, Create Proc;
- команды системной утилиты DBCC (*DataBase Console Command*): DBCC TraceON, DBCC Page, DBCC ExtentINFO, DBCC ShowContig;

– системные хранимые процедуры и функции: `sp_helptext`, `sp_helpfile`, `sp_spaceused`, `Object_ID()`.

Задание 1. Анализ системного каталога пользовательской базы данных

1.1 Используя SQL-команду `CREATE DATABASE`, создайте пользовательскую базу данных с простейшей файловой структурой: один файл данных и одна файловая группа (листинг 6.1).

```
Use master;
CREATE DATABASE MyDB-1
ON
  (NAME = MyDB-1_Dat,
   FILENAME = 'C:\...\...\MyDB-1.mdf',
   SIZE = 4MB,
   MAXSIZE = UNLIMITED,
   FILEGROWTH = 1MB)
LOG ON
  (NAME = MyDB-1_log,
   FILENAME = 'D:\...\...\MyDB-1.ldf',
   SIZE = 2MB,
   MAXSIZE = 10MB,
   FILEGROWTH = 15%);
```

Листинг 6.1 – Пример создания базы данных

1.2 Проконтролируйте (и сохраните в отчете) параметры базы данных, созданной в результате выполнения приведенного выше SQL-кода:

- a) прямым доступом оператором `Select` к таблицам `SysFiles` и `SysFileGroups` системного каталога базы данных `MyDB-1`;
- b) с использованием хранимой процедуры `sp_helpfile` (листинг 6.2).

```
Use MyDB-1;
EXEC sp_helpfile;
```

Листинг 6.2 – Пример выполнения хранимой процедуры

1.3 Используя SQL-команду `CREATE TABLE`, создайте в этой базе данных простую унарную таблицу `MyTable_1`, каждая строка которой будет занимать ровно одну дисковую страницу (листинг 6.3):

```
Use MyDB-1;  
CREATE TABLE MyTable_1  
(column1 char(8000) NOT NULL  
default 'One row in one page');
```

Листинг 6.3 – Пример создания унарной таблицы

1.4 Прямым доступом оператором `Select` к таблице `SysObjects` системного каталога базы данных `MyDB-1` определите (и сохраните в отчете):

- a) параметры `name`, `Id`, `xtype`, и `crdate` для вновь созданного объекта (таблицы `MyTable_1`);
- b) общее количество объектов такого же типа (`xtype`) в этой базе данных;
- c) общее количество объектов с такой же датой создания (`crdate`) в этой базе данных;
- d) общее количество пользовательских таблиц (`xtype='U'`), системных таблиц (`xtype='S'`), хранимых представлений (`xtype='V'`) и хранимых процедур (`xtype='P'`) в этой базе данных.

1.5 Используя встроенную функцию `Object_ID()`, определите идентификатор объекта (таблицы `MyTable_1`) по его имени:

```
Use MyDB-1;  
Select Ob-  
ject_ID('MyTable_1');
```

Листинг 6.4 – Пример использования функции `Object_ID()`

1.6 Прямым доступом оператором `Select` к таблице `SysIndexes` системного каталога базы данных `MyDB-1` определите (и сохраните в отчете):

- a) количество строк в этой таблице, соответствующих объекту `MyTable1` (для ограничения выборки используйте конструкцию: `where ID=Object_ID('MyTable_1')`);
- b) значения полей `Id`, `IndId`, `First`, `Root`, `FirstIAM` для таблицы `MyTable_1`;
- c) количество и номера (адреса `PageNum`) дисковых страниц, владельцем которых является таблица `MyTable_1` (помним, что в текущей ситуации эта таблица еще не содержит ни одной строки данных).

1.7 Сформулируйте (и сохраните в отчете) ответы на следующие вопросы:

- a) каковы результаты трансляции SQL-запроса `CREATE TABLE ...` и в каких таблицах системного каталога MS SQL-Server сохраняет эти результаты ?
- b) каково назначение системной таблицы `SysObjects` ?
- c) какую информацию несут значения полей `name`, `Id`, `xtype` и `crdate` в таблице `SysObjects` ?
- d) каково назначение системной таблицы `SysIndexes` ?
- e) какую информацию несут значения полей `Id` и `IndId` в таблице `SysIndexes` ?
- f) какую информацию несут значения полей `First`, `Root` и `FirstIAM` и в каком формате представлена эта информация в таблице `SysIndexes` ?
- g) сколько страниц занимает в файле данных "пустая" таблица ?
- h) для чего можно использовать встроенные хранимые процедуры `sp_helpfile` и `sp_helptext` и встроенную функцию `Object_ID()` ? Приведите примеры.

Задание 2. Исследование алгоритма резервирования памяти в базах данных с простой файловой структурой

Продолжим эксплуатировать базу данных MyDB-1, созданную при выполнении 1-го задания. Эта база содержит единственный (primary) файл данных, ассоциированный с единственной файловой группой.

В соответствии с результатами выполнения задания 1.4, часть страниц типа Data этого файла заняты объектами системного каталога, а единственная пользовательская таблица MyTable_1 пока пуста и не владеет ни одной из файловых страниц этого типа.

Проведя несложный эксперимент, исследуйте процесс резервирования и заполнения файловых страниц и распределения их по экстенстам соответствующих типов (смешанных – mixed или однородных – uniform) при последовательной вставке строк в таблицы.

2.1 Создайте в базе данных MyDB-1 еще одну таблицу, например MyTable-2, с такой же схемой, как у таблицы MyTable-1 (используйте SQL-код, подобный приведенному в листинге 6.3).

2.2 Вставьте по одной строке в обе эти таблицы (листинг 6.5) и затем повторно выполните задание 1.6 для двух таблиц (теперь обе таблицы не пусты, и каждая из них является владельцем, как минимум, двух страниц: одной страницы типа Data и одной IAM-страницы).

```
Use MyDB-1;  
INSERT INTO MyTable_1 DEFAULT VALUES;  
INSERT INTO MyTable_2 DEFAULT VALUES;
```

Листинг 6.5 – Вставка одной строки

Определите номера экстенстов, содержащих страницы, выделенные двум таблицам при вставке в них строк данных. К какому типу относятся эти экстенсты – uniform или mixed? Ответ обоснуйте и сохраните в отчете.

2.3 С помощью команды PAGE системной утилиты DBCC просмотрите заголовки и основное содержимое этих страниц (для каждой из двух таблиц):

```
DBCC TRACEON (3604)  
DBCC PAGE ('MyDB-1', <Id файла>, <Id страницы>, 0)
```

Листинг 6.6 – Пример выполнения команды PAGE

С четвертым параметром команды PAGE придется поэкспериментировать: значение этого параметра (0, 1, 2 или 3) влияет (по-разному для различных типов страниц) на объем и формат выводимой на экран информации.

2.4 Используя хранимую процедуру `sp_spaceused`, определите количество страниц, занятых каждой из этих таблиц:

```
Use MyDB-1;  
EXEC sp_spaceused MyTable_1;  
EXEC sp_spaceused MyTable_2;
```

Листинг 6.7 – Пример выполнения процедуры `sp_spaceused`

Прокомментируйте результаты работы этой процедуры, представленные на экране в табличном виде, как это показано ниже на рисунке 6.1.

NAME	ROWS	RESERVED	DATA	INDEX_SIZE	UNUSED
MyTable_1	1

NAME	ROWS	RESERVED	DATA	INDEX_SIZE	UNUSED
MyTable_2	1

Рисунок 6.1 – Форма представления результатов выполнения процедуры `sp_spaceused`

Подтвердите (или скорректируйте) свой ответ на вопрос, сформулированный в задании 2.2.

2.5 Последовательно добавляя в обе таблицы еще по 4 строки (листинг 6.8), определите после каждой вставки общее количество зарезервированных страниц (поле `RESERVED`), количество страниц, занятых строками таблиц (поле `DATA`), и количество зарезервированных, но еще не использованных страниц (поле `UNUSED`).

```
Use MyDB-1;
INSERT INTO MyTable_1 DEFAULT VALUES;
EXEC sp_spaceused MyTable_1;
Go 4;
INSERT INTO MyTable_1 DEFAULT VALUES;
EXEC sp_spaceused MyTable_2;
Go 4;
```

Листинг 6.8 – Вставка 4 строк с контролем занятого пространства

Обратите внимание на динамику изменения трех указанных выше параметров при изменении количества вставленных в таблицы строк. Теперь в каждой таблице по 5 строк, и, соответственно, каждая таблица является владельцем пяти файловых страниц типа DATA и одной IAM-страницы.

В экстендах какого типа (*uniform* или *mixed*) размещены страницы, выделенные двум этим таблицам ?

2.6 Повторите предыдущий опыт – вставьте еще по 5 строк в обе эти таблицы (*Go 5*) и проанализируйте полученный результат.

2.7 Вставьте в эти таблицы еще по 50 строк, проанализируйте полученный результат с помощью процедуры *sp_spaceused* и дополнительно – с помощью команды *DBCC EXTENTINFO* (листинг 6.9), которая отобразит на экране информацию об идентификаторах занятых страниц (поля *file_id* и *page_id*), количестве выделенных (*ext_size*) и фактически заполненных (*pg_alloc*) страниц.

2.8 Если SQL Server все еще выделяет таблицам страницы в смешанных экстендах, продолжайте вставлять строки в таблицы до тех пор, пока сервер начнет резервировать однородные экстенды для страниц каждой из таблиц.

```
DBCC EXTENTINFO (MyDB-1, MyTable_1, -1*)
```

Листинг 6.9 – Пример выполнения команды *EXTENTINFO*

* Последним параметром команды *EXTENTINFO* можно указывать либо имя индекса таблицы (для вывода информации о страницах этого индекса), либо число «0» (для вывода информации о страницах, занятых строками таблицы), либо число «-1» (для вывода информации о страницах, занятых строками таблицы и всеми ее индексами).

2.9 С помощью команды DBCC PAGE просмотрите содержимое страниц типа GAM и SGAM (позиции этих страниц в файле фиксированы: №2 – для GAM и №3 – для SGAM). Какие свойства (двухбитовые коды) получили экстенды, зарезервированные для хранения строк таблиц MyTable_1 и MyTable_2 ?

2.10 С помощью команды DBCC PAGE просмотрите содержимое страницы типа PFS (фиксированная позиция №1 в файле данных). Определите степень заполнения нескольких страниц, выделенных таблицам MyTable_1 и MyTable_2.

2.11 Сформулируйте (и сохраните в отчете) ответы на следующие вопросы:

1 В каком формате хранятся номера страниц в таблице SysIndexes ?

2 Может ли логический объект базы данных (например, таблица) быть владельцем единственной файловой страницы ?

3 Может ли логический объект базы данных быть владельцем нескольких файловых страниц типа DATA ?

4 Может ли одна файловая страница типа DATA иметь более, чем одного владельца ?

5 Может ли одна файловая страница типа DATA входить в состав более, чем одного экстенда ?

6 В каких случаях MS SQL Server резервирует смешанные экстенды ?

7 В каких случаях MS SQL Server резервирует однородные экстенды ?

8 В каких структурах данных и в каком формате SQL Server хранит информацию о свободных экстендах, о типах зарезервированных экстендов и о свободном пространстве внутри файловых страниц ?

9 Какие эксплуатационные показатели использовались в качестве критериев при реализации стратегии резервирования экстендов ?

Задание 3. Исследование алгоритма распределения памяти в базах данных со сложной файловой структурой

Если в предыдущем задании анализировался процесс резервирования экстендов и страниц единственного файла данных, то теперь ставится задача экспериментального исследования алгоритма распределения страниц одного логического объекта (таблицы) между несколькими файлами данных.

Рабочая гипотеза, которую следует подтвердить, опровергнуть или уточнить в результате выполнения этого задания, может быть сформулирована следующим образом: «Если файловая группа содержит более одного файла типа DATA, то при вставке строк в таблицу, ассоциированную с этой файловой группой, количество файловых страниц, выделяемых сервером в каждом из файлов, будет пропорциональным их размерам».

Для выполнения задания потребуется создать несколько баз данных, имеющих более сложную (по сравнению с MyDB-1) файловую структуру: несколько файловых групп и несколько файлов разных размеров в каждой группе.

3.1 Создайте новую пользовательскую базу данных (например, MyDB-2) со следующей файловой структурой:

- две файловых группы (группа Primary со свойством «по умолчанию» и дополнительная группа Group2);
- четыре файла типа DATA (первичный файл и три вторичных файла File1, File2 и File3, принадлежащих группе Group2);
- установите начальные размеры вторичных файлов: Size=5, 10 и 15 Mb соответственно;
- установите остальные размерные параметры, одинаковые для всех этих трех файлов: MaxSize=Unlimited; Grows=1 Mb.

3.2 Проконтролируйте (и сохраните в отчете) полученный результат с использованием прямого доступа к системной таблице sysfiles SQL-инструкцией SELECT.

3.3 Создайте в базе данных MyDB-2 новую таблицу MyTable_3, аналогичную таблице MyTable_1. Каждая строка этой таблицы будет занимать ровно одну файловую страницу, и все эти страницы (в случае заполнения строк таблицы) будут размещены во вторичных файлах, включенных в группу Group2:

```

Use MyDB-2;
CREATE TABLE MyTable_3
  (column1 char(8000) NOT NULL
   default 'One row in one page')
ON Group2;

```

Листинг 6.10 – Пример создания таблицы, связанной с файловой группой

- 3.4 Учитывая тот факт, что все системные объекты базы данных MyDB-2 будут размещены в её первичном файле, а вторичные файлы будут заняты исключительно пользовательскими данными таблицы MyTable_3, рассчитайте (приблизительно) максимальное количество страниц этой таблицы, соответствующее начальному размеру каждого из трех файлов группы Group2.
- 3.5 Используя пример (листинг 6.11), напишите пользовательскую хранимую процедуру, при выполнении которой в таблицу будет вставлено заданное количество строк со значениями полей «по умолчанию», указанными при создании таблицы.

```

CREATE PROC AddRows @Tablename char(12), @maxrows int
AS
  SET nocount off
  DECLARE @count INT
  SET @count = 0
  WHILE @count < @maxrows
  BEGIN
    INSERT INTO @Tablename DEFAULT VALUES
    SET @count = @count + 1
  END

```

Листинг 6.11 – Пример SQL-кода для создания хранимой процедуры

- 3.6 Используя процедуру AddRows, вставьте в таблицу MyTable_3 расчетное количество строк так, чтобы меньший из трех вторичных файлов оказался заполненным примерно наполовину. Проконтролируйте (и сохраните в отчете) полученный результат.

- 3.7 Многократно повторяя п. 3.6, добейтесь ситуации, когда наибольший (по начальному размеру) из вторичных файлов увеличится по размеру примерно вдвое.
- 3.8 По результатам проведенного эксперимента опишите (и сохраните в отчете) алгоритм распределения страниц типа DATA между файлами одной файловой группы в случае, когда начальные размеры файлов различны, но их предельные размеры не ограничены.
- 3.9 Создайте новую пользовательскую базу данных (например, MyDB-3) с файловой структурой, аналогичной MyDB-2, но с различными параметрами MaxSize для вторичных файлов.
- 3.10 Повторите п. 3.2 – 3.7 этого задания.
- 3.11 По результатам проведенного эксперимента постройте графики зависимостей размеров файлов от количества строк таблицы, опишите (и сохраните в отчете) алгоритм распределения страниц типа DATA между файлами одной файловой группы в случае, когда различны и начальные, и предельные размеры файлов.

Задание 4. Исследование структуры файловой страницы типа DATA

При выполнении предыдущих заданий исследовался процесс резервирования файловых страниц при вставке строк в таблицы, и для этого было удобно использовать унарные таблицы с полями типа char(8000), чтобы каждая строка таблицы занимала целую страницу. В реальной ситуации страница может содержать несколько строк таблицы и при этом иметь свободное пространство для вставки в таблицу последующих строк.

Объектом исследования в этом задании является внутренняя структура файловой страницы и процесс её заполнения строками таблицы.

Для выполнения задания рекомендуется создать новую базу данных (например, MyDB-4) с простой файловой структурой.

- 4.1 Создайте бинарную таблицу MyTable_4(Key1 INT, Data CHAR(10)) с длиной строки 14 байтов.
- 4.2 Вставьте в таблицу 10 строк, заполнив поля случайными данными:

```
USE MyDB-4
DECLARE @key1 INT, @data CHAR(10)
SET @key1=1000*RAND(), @data=STR(@key1)
INSERT into MyTable_4 values(@Key1,@data)
Go 10
```

Листинг 6.12 – Вставка 10 строк в таблицу

Примечание. Функция `RAND ()` возвращает псевдослучайное число в диапазоне (0 ... 1), а функция `STR ()` преобразует число в цифровую строку.

- 4.3 Определите номер файловой страницы (`sysindexes.First`), выделенной этой таблице, и просмотрите страницу командой `DBCC PAGE ()`. Определите и сохраните в отчете:
 - количество слотов страницы, занятых строками таблицы;
 - смещения (в байтах) каждого слота;
 - длину (в байтах) каждого слота.
- 4.4 Вставьте еще 100 строк в таблицу и повторно выполните п.4.3.
- 4.5 Просмотрите PFS-страницу командой `DBCC PAGE`, определите процент заполнения первой страницы, выделенной таблице.
- 4.6 Выполняйте п. 4.4 и п. 4.5 до 100%-го заполнения первой страницы, выделенной таблице. Определите и сохраните в отчете:
 - количество слотов первой страницы, занятых строками таблицы;
 - суммарный объем страницы, занятый заполненными слотами;
 - суммарный объем страницы (в байтах), занятый областью обратных ссылок (`offset table`).
- 4.7 Просматривая первую страницу таблицы, выберите по своему усмотрению один из заполненных слотов, запомните его номер и значения полей соответствующей строки таблицы. Затем удалите из таблицы эту строку (`Delete MyTable_4 Where Key1=...`).
- 4.8 Повторно просмотрите страницу командой `DBCC PAGE`, обращая внимание на слот с удаленной строкой. Прокомментируйте и попытайтесь объяснить полученный результат.

ЛАБОРАТОРНАЯ РАБОТА №3

Исследование индексных структур данных

Цель работы: изучение индексных структур и приобретение навыков использования инструментальных средств управления индексами.

Задачи:

- освоить программные средства создания, модификации и анализа индексных структур данных;
- изучить структуру объектов системного каталога, ответственных за хранение параметров индексов;
- изучить формат индексных страниц для различных типов индексов.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Лабораторная работа содержит четыре взаимосвязанных задания, каждое из которых направлено на изучение многоуровневых индексных структур данных для индексов четырех различных типов:

- некластеризованного индекса по неуникальным столбцам таблицы при условии отсутствия кластеризованного индекса;
- кластеризованного уникального индекса;
- некластеризованного индекса при условии наличия в таблице кластеризованного индекса;
- некластеризованного индекса с включенными столбцами.

Каждое задание предполагает постановку несложного эксперимента с привлечением программных компонентов базы данных и последующее проведение анализа полученных результатов.

В процессе выполнения заданий продолжится знакомство с компонентами системного каталога базы данных (таблица `SysIndexes`) и программными средствами, используемыми для управления индексами:

- SQL-операторы `Create/Alter/Drop Index`;
- команды `Page` и `ShowContig` системной утилиты `DBCC`;
- системная хранимая процедура `sp_spaceused`;
- TVF-функция `sys.dm_db_index_physical_stats()`.

Задание 1. Анализ структуры индексных страниц
для неуникального некластеризованного индекса

1.1 Используя SQL-команду `Create Database`, создайте пользовательскую базу данных (например, `Index_Test_1`) с простейшей файловой структурой: один файл данных в одной группе.

1.2 Используя SQL-команду `Create Table`, создайте в этой базе данных таблицу `MyTable_4`, схема которой включает три целочисленных столбца и четвертый столбец строкового типа:

```
Use Index_Test_1
CREATE TABLE MyTable_4
(Key_0 INT NOT NULL,
 Key_1 INT NOT NULL,
 Key_2 INT NOT NULL,
 Data CHAR(61) NOT NULL)
```

Листинг 6.13 – Создание таблицы `MyTable_4`

Примечание. Длина поля `Data` (61 байт) выбрана для удобства просмотра файловых страниц командой `DBCC PAGE`: так как каждый слот страницы содержит два служебных поля суммарной длиной в 7 байтов, то общая длина каждого слота составит ровно 80 байтов ($7+3*4+61$).

1.3 Определите идентификатор этой таблицы и убедитесь в том, что системная таблица `sysindexes` содержит ровно одну запись, соответствующую таблице `MyTable_4`, и при этом таблица `MyTable_4` не является владельцем ни одной файловой страницы. Объясните этот факт.

1.4 Вставьте в таблицу `MyTable_4` одну строку данных:

```
USE Index_Test_1
DECLARE @key0 INT, @key1 INT, @key2 INT, @dat CHAR(30)
SET @key0=1000*RAND(),@key1=1000*RAND(),
@key2=1000*RAND()
SET @dat=STR(@key0)+STR(@key1)+STR(@key2)
INSERT into MyTable_4 values(@key0,@key1,@key2,@dat)
```

Листинг 6.14 – Вставка строки в таблицу `MyTable_4`

1.5 Используя хранимую процедуру `sp_spaceused`, определите количество страниц, занятых данными этой таблицы и всеми её индексами, включая IAM-страницу.

1.6 Повторным запросом к таблице `SysIndexes` определите:

- количество записей в системной таблице `SysIndexes`, соответствующих таблице `MyTable_4`.
- значения полей `Root`, `First`, `FirstIAM`, и `IndID` для каждой из этих записей;
- категории объектов (`Heap`, `ClusteredIndex`, или `NonClusteredIndex`), соответствующих таблице `MyTable_4`.

1.7 С помощью команды `DBCC PAGE` просмотрите содержимое всех страниц, владельцем которых является таблица `MyTable_4`.

1.8 Создайте *НЕ*кластеризованные *НЕ*уникальные индексы по полям `Key_0`, `Key_1` и `Key_2` таблицы `MyTable_4` (листинг 6.15), затем повторно выполните п. 1.5 и п. 1.6.

```
Use Index_Test_1
Create NONCLUSTERED Index NonClInd_0
ON MyTable_4 (Key_0)
```

Листинг 6.15 – Создание некластеризованного индекса `NonClInd_0` по полю `Key_0`

1.9 Командой `DBCC PAGE` просмотрите заголовки и основное содержимое корневых и листовых страниц всех трех индексов (используйте значения 1, 2 и 3 для четвертого параметра этой команды). Определите:

- глубину индексов;
- номера страниц корневого и листового уровней;
- формат ссылки с корневого уровня индекса на промежуточный (нелистовой) уровень;
- формат ссылки с листовой страницы на страницу данных в «куче» (`heap`).

1.10 Вставьте в таблицу еще 1000 строк.

- 1.11 Проанализируйте полученный результат с помощью хранимой процедуры `sp_spaceused`, команды `DBCC ShowContig` и TVF-функции `sys.dm_db_index_physical_stats()`.
- 1.12 Вставьте в таблицу еще 100000 строк (придется немного подождать) и выполните повторный анализ полученного результата.
- 1.13 Сформулируйте (и сохраните в отчете) ответы на следующие вопросы:
- 1 Для чего и в каких случаях рекомендуется использовать индексы следующих типов: «кластеризованный индекс», «некластеризованный индекс», «уникальный индекс», «индекс с включенными столбцами»?
 - 2 На какие эксплуатационные показатели работы базы данных оказывает влияние индексирование данных в таблицах ?
 - 3 В каких ситуациях наличие индексированных столбцов таблиц может привести к снижению производительности работы базы данных?
 - 4 Какие ограничения накладывает SQL-Server на использование индексов ?
 - 5 Каков формат ссылки с корневого уровня индекса на промежуточный (нелистовой) уровень ?
 - 6 Каков формат ссылки с листовой страницы некластеризованного индекса при условии, что в таблице отсутствует кластерный индекс ?
 - 7 Как связаны между собой значения параметров «порядок индекса», «глубина индекса» индексной структуры данных ?
 - 8 Какова глубина индексов, построенных при выполнении заданий 1.8, 1.10 и 1.12?

Задание 2. Анализ структуры индексных страниц для кластеризованного индекса

Для выполнения задания рекомендуется создать новую базу данных (например, `Index_Test_2`) с простейшей файловой структурой.

- 2.1 Создайте в базе данных `Index_Test_2` таблицу `MyTable_5` со схемой, аналогичной схеме таблицы `MyTable_4` (листинг 6.16).

```

use Index_Test_2
CREATE TABLE MyTable_5
  (Key_0 IDENTITY CONSTRAINT Key0_PK PRIMARY
  KEY,
  Key_1 INT, Key_2 INT, Data CHAR(68))

```

Листинг 6.16 – Пример создания таблицы с первичным ключом

Примечание 1. Параметр IDENTITY, установленный для целочисленного поля Key_0, присваивает полю статус идентификатора, значения которого при вставке строк в таблицу сервер будет присваивать автоматически (по умолчанию – автоинкрементно с шагом 1, начиная с 1).

Примечание 2. Ограничение первичного ключа (PRIMARY KEY) для поля Key_0 гарантирует уникальность значений этого поля в таблице, даже при отсутствии свойства IDENTITY. По умолчанию сервер автоматически создает кластеризованный индекс по первичному ключу таблицы.

- 2.2 Определите адреса файловых страниц Root, First и FirstIAM для кластеризованного индекса полю Key_0 таблицы MyTable_5.
- 2.3 Вставьте в таблицу 10 строк (листинг 6.17) и повторите п.2.2.

```

USE Index_Test_2
DECLARE @key0 INT, @key1 INT, @key2 INT, @dat CHAR(30)
SET @key1=1000*RAND(), @key2=1000*RAND()
SET @dat=STR(@key1)+STR(@key2)
INSERT into MyTable_5 values (@key1, @key2, @dat)
Go 10

```

Листинг 6.17 – Вставка 10 строк в таблицу MyTable_5

- 2.4 Выполните задания, аналогичные п.1.8 ... п.1.12, в контексте базы данных Index_Test_2 применительно к таблице MyTable_5.

**Задание 3. Анализ структуры индексных страниц
некластеризованного индекса при условии наличия
кластеризованного индекса**

3.1 Для выполнения задания будет использоваться ранее созданная база данных Index_Test_2 и уже существующая и заполненная таблица MyTable_5.

3.2 Создайте в таблице MyTable_5 некластеризованный индекс по полю Key_1:

```
use Index_Test_2
Create NONCLUSTERED Index NonClInd_1
ON MyTable_5(Key_1)
```

Листинг 6.18 – Создание индекса в ранее заполненной таблице

3.3 Дождитесь завершения процесса построения индекса и затем повторно выполните задания 2.2 ... 2.4.

**Задание 4. Анализ структуры индексных страниц
некластеризованного индекса с включенным столбцом**

4.1. Для выполнения задания будет использоваться ранее созданная база данных Index_Test_2 и заполненная таблица MyTable_5, в которой уже созданы индексы по двум полям: уникальный кластеризованный индекс по ключевому полю Key_0 и неуникальный некластеризованный индекс по полю Key_1. Остальные два поля таблицы неиндексированы.

4.2. Создайте в таблице MyTable_5 некластеризованный индекс по полю Key_2 с включенным полем Data:

```
use Index_Test_2
Create NONCLUSTERED Index incl_Ind_2
ON MyTable_5(Key_1) INCLUDE(Data)
```

Листинг 6.19 – Создание индекса с включенным столбцом

4.3. Повторно выполните задание 3.3.

ЛАБОРАТОРНАЯ РАБОТА №4

Анализ процедурных планов выполнения SQL-запросов

Лабораторная работа завершает цикл из четырех работ, направленных на изучение физической модели данных, поддерживаемой MS SQL-Server'ом.

Цель работы: изучение стратегий построения процедурных планов исполнения SQL-запросов, реализуемых оптимизатором запросов, и приобретение практических навыков анализа и управления производительностью работы сервера.

Задачи:

- ознакомиться с основными низкоуровневые операторами, используемыми для построения и описания процедурных планов;
- освоить технику анализа процедурных планов соответствующими языковыми средствами (SET SHOWPLAN, SET STATISTICS), а также средствами их графической визуализации, предоставляемыми средой SQL Server Management Studio;
- провести экспериментальное исследование влияния индексирования таблиц базы данных на производительность выполнения типовых SQL-запросов;
- по результатам проведенного анализа сделать выводы о стратегии работы генератора процедурных планов и эффективности применения различных индексных структур.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

Лабораторная работа содержит два задания, каждое из которых направлено на изучение стратегий построения процедурных планов исполнения следующих типовых SQL-запросов при использовании различных типов индексов по столбцам базовых таблиц:

- процедурные планы реализации простейших однотабличных SQL-запросов вида `Select ... From <Table> Where <condition>;`
- процедурные планы реализации SQL-запросов с соединением таблиц;
- процедурные планы реализации SQL-запросов с группировкой данных и использованием агрегатных функций;
- процедурные планы реализации модифицирующих SQL-запросов (Insert, Delete, Update).

Для выполнения лабораторной работы потребуется создать базу данных, состоящую из нескольких взаимосвязанных таблиц достаточно большой мощности, по столбцам которых сформированы индексы различных типов.

Каждое задание предполагает постановку несложных экспериментов с выполнением типовых SQL-запросов, визуализацией процедурных планов их исполнения и сравнительной оценкой планов по критерию производительности.

В процессе выполнения заданий потребуется использование следующих программных средств:

- SQL-операторы `Create/Alter DataBase/Table/Index` – для создания / модификации логических объектов базы данных;
- TVF-функция `sys.dm_db_index_physical_stats()` – для определения параметров индексов;
- команда `DBCC show_statistics(t1,ind)` – для отображения статистики индекса `Ind` таблицы `T1`;
- команды группы `SET` – для визуализации процедурных планов:
 - `SHOWPLAN_XML`, `SHOWPLAN_TEXT`, `SHOWPLAN_ALL` – отображают информацию о предполагаемом (estimated) процедурном плане выполнения запроса, блокируя при этом его выполнение;
 - `STATISTICS XML`, `STATISTICS PROFILE` – отображают информацию о фактическом (real) процедурном плане исполнения запроса;
 - `STATISTICS IO`, `STATISTICS TIME` – отображают информацию о дисковой активности и времени выполнения запроса;
- рассмотренные в разделе 3 учебного пособия логические и физические процедурные операторы (таблицы 3.1 и 3.2), используемые генератором для формирования планов исполнения SQL-запросов.

Дополнительная информация и методические рекомендации по использованию программных средств анализа процедурных планов выполнения SQL-запросов приведена в [8; 9].

Задание 1. Анализ процедурных планов **реализации однотабличных SQL-запросов**

1.1 Запросы выборки из структуры данных типа «куча»

Для проведения эксперимента создайте базу данных (например, Plan_Test1) и в этой базе данных создайте таблицу MyTable_6:

```
Use Plan_Test1;
CREATE TABLE MyTable_6
(Key_0 INT NOT NULL, Key_1 INT NOT NULL,
Key_2 INT NOT NULL, Data CHAR(8000) NOT NULL);
```

Листинг 6.20 – Создание таблицы MyTable_6

Заполните 10 000 строк этой таблицы случайными значениями:

```
USE Plan_Test1;
DECLARE @key0 INT,@key1 INT,@key2 INT,@dat CHAR(30)
SET
@key0=1000*RAND(),@key1=1000*RAND(),@key2=1000*RAND()
SET @dat=STR(@key1)+STR(@key2)
INSERT into MyTable_6 values(@key0,@key1,@key2,@dat);
Go 10000
```

Листинг 6.21 – Вставка 10 000 строк в таблицу MyTable_6

1.1.1 Используя запрос к таблице SysIndexes, убедитесь в том, что для таблицы MyTable_6 сформирована структура данных типа «куча», и определите количество страниц, занятых строками этой таблицы с помощью хранимой процедуры sp_spaceused.

1.1.2 Подготовьте и выполните SQL-запрос выборки всех данных таблицы (Select * From MyTable_6), оцените её мощность, сравните с результатами работы процедуры sp_spaceused.

1.1.3 Включите (рисунок 3.4) режим графического отображения предполагаемого плана выполнения запроса, определите и сохраните в отчете:

- графическую схему предполагаемого плана;
- используемые процедурные операторы;
- стоимость выполнения операторов;
- стоимость операций ввода-вывода;
- стоимость операций обработки данных.

1.1.4 Дополните пакет выполнения предыдущего запроса (п. 1.1.2) командами управления отображением процедурных планов (таблица 3.1):

- SET STATISTICS XML ON;
- SET STATISTICS PROFILE ON;
- SET STATISTICS IO ON;
- SET STATISTICS TIME ON.

1.1.5 Включите режим графического отображения фактического плана и повторно выполните запрос.

1.1.6 Просмотрите результаты выполнения запроса (вкладки «результаты», «сообщения» и «план выполнения»). Определите и сохраните в отчете параметры фактического процедурного плана.

1.1.7 Проведите анализ фактических планов выполнения следующих запросов, содержащих операторы ограничения и группировки строк таблицы:

- `Select * From MyTable_6 Where Key_1=555;`
- `Select * From MyTable_6 Where Key_2>666;`
- `Select Data From MyTable_6
Where Key_1>20 And Key_2<100;`
- `Select * From MyTable_6 Order By Data;`
- `Select Key_1, Count(*)
From MyTable_6
Group By Key_1;`

1.1.8 Сформулируйте (и сохраните в отчете) выводы о стратегии работы генератора процедурных планов выполнения SQL-запросов выборки данных из «кучи».

1.2 Запросы выборки данных из индексированных таблиц

1.2.1 Создайте таблицу MyTable_7 (аналогичную MyTable_6).

1.2.2 Создайте НЕкластеризованные НЕуникальные индексы по полям Key_0, Key_1 и Key_2 таблицы MyTable_7.

1.2.3 Вставьте **10 строк** в таблицу MyTable_7.

1.2.4 Используя запрос к таблице SysIndexes, убедитесь в том, что для таблицы MyTable_7 сформирована структура данных типа «куча» и дополнительно – три некластеризованных индекса.

- 1.2.5 Определите для каждого из индексов глубину индекса и количество индексных страниц на каждом уровне.
- 1.2.6 Выполните запросы (п. 1.1.2 и 1.1.7) на базе индексированной таблицы MyTable_7, сравните процедурные планы их выполнения с планами запросов выборки из «кучи».
- 1.2.7 Командой Insert вставьте в таблицу MyTable_7 еще **9990 строк** (теперь её мощность равна мощности таблицы MyTable_6).
- 1.2.8 Просмотрите процедурный план выполнения операции Insert, определите стоимость её выполнения, сохраните в отчете.
- 1.2.9 Командой DBCC show_statistics('MyTable_7',<ind>) отобразите статистику всех трех индексов таблицы. Прокомментируйте результаты.
- 1.2.10 Повторно выполните п. 1.2.6. Оцените повышение производительности запросов на индексированной таблице по сравнению с запросами выборки данных из «кучи».
- 1.2.11 Создайте (листинг 6.22) таблицу MyTable_8 с первичным ключом Key_0 автоинкрементного типа IDENTITY (по этому полю таблицы будет автоматически создан кластеризованный индекс):

```
USE Plan_Test1
CREATE TABLE MyTable_8
(Key_0 INT NOT NULL IDENTITY
 CONSTRAINT Key0_PK PRIMARY KEY,
 Key_1 INT NOT NULL,
 Key_2 INT NOT NULL,
 Data CHAR(8000) NOT NULL)
```

Листинг 6.22 – Создание таблицы с первичным ключом

- 1.2.12 Создайте *НЕ*кластеризованные *НЕ*уникальные индексы по полям Key_1 и Key_2 таблицы MyTable_8.
- 1.2.13 Вставьте в таблицу MyTable_8 10000 строк:

```

USE Plan_Test1
DECLARE @key1 INT
DECLARE @key2 INT
DECLARE @dat CHAR(30)
SET @key1=1000*RAND()
SET @key2=1000*RAND()
SET @dat=STR(@key1)+STR(@key2)
INSERT into MyTable_8
values (@key1,@key2,@dat)
Go 10000

```

Листинг 6.23 – Вставка 10 000 строк

- 1.2.14 Командой DBCC show_statistics() отобразите статистику всех трех индексов таблицы MyTable_8. Прокомментируйте результаты.
- 1.2.15 Просмотрите процедурный план выполнения операции Insert, определите стоимость её выполнения, сохраните в отчете.
- 1.2.16 Повторно выполните запросы, аналогичные запросам п. 1.1.2 и 1.1.7, сравните процедурные планы их выполнения с планами запросов выборки данных из «кучи» и с планами запросов выборки данных из таблицы с некластеризованными индексами.

Задание 2. Анализ процедурных планов выполнения SQL-запросов с соединениями таблиц

2.1 Запросы с соединением неиндексированных таблиц

В этом эксперименте будет использоваться имеющаяся таблица MyTable_6 (индексов нет, мощность – 10000 строк) и аналогичная ей по структуре новая таблица MyTable_9 (существенно меньшей мощности – 100 строк).

Создайте 4 запроса на базе таблиц MyTable_6 и MyTable_9 и проведите анализ фактических планов их выполнения:

- ```

1) Select MyTable_6.Key_0, MyTable_9.Key_1, MyTable_6.Data
 From MyTable_6 Inner Join MyTable_9
 ON MyTable_6.Key_0=MyTable_9.Key_1;
2) Select MyTable_9.Key_0, MyTable_6.Key_1, MyTable_6.Data
 From MyTable_9 Inner Join MyTable_6
 ON MyTable_9.Key_0 = MyTable_6.Key_1;
3) Select MyTable_9.Key_0, MyTable_6.Key_1, MyTable_6.Data
 From MyTable_9 Left Join MyTable_6
 ON MyTable_9.Key_0 = MyTable_6.Key_1;
4) Select MyTable_9.Key_0, MyTable_6.Key_1, MyTable_6.Data
 From MyTable_9 Right Join MyTable_6
 ON MyTable_9.Key_0 = MyTable_6.Key_1;

```

Листинг 6.24 – Запросы с соединением неиндексированных таблиц

## 2.2 Запросы с соединением индексированных таблиц

В этом эксперименте будет использоваться имеющаяся таблица MyTable\_8 (мощность – 10000 строк, кластеризованный уникальный индекс по полю Key\_0 и некластеризованные неуникальные индексы по полям Key\_1 и Key\_2) и аналогичная ей по структуре новая таблица MyTable\_10 мощностью в 100 строк.

Создайте 4 запроса на базе таблиц MyTable\_8 и MyTable\_10 (аналогичных запросам, приведенным в п. 2.1) и проведите анализ фактических планов их выполнения.

Сформулируйте (и сохраните в отчете) ответы на следующие вопросы:

- 1 В каких ситуациях наличие индексированных столбцов таблиц может привести к снижению производительности работы базы данных?
- 2 Какую информацию о состоянии базы данных использует генератор процедурных планов?
- 3 В каких типовых ситуациях «estimated plan» и «real plan» одного и того же SQL-запроса не совпадают?

### **Задание 3. Анализ процедурных планов выполнения SQL-запросов с группировкой строк**

Выполнение задания потребует проведения эксперимента по анализу стратегии работы генератора процедурных планов выполнения SQL-запросов, содержащих операторы группировки строк Group By, опе-

раторы фильтрации групп Having и агрегатные функции (такие, например, как COUNT(), SUM()или MAX()).

Методика проведения экспериментов и средства анализа процедурных планов – те же, что и при выполнении предыдущих заданий лабораторной работы.

Предлагается самостоятельно спланировать, подготовить и провести две серии экспериментов:

- 1) группировка строк таблицы с применением агрегатной функции;
- 2) группировка строк таблицы с фильтрацией групп.

Серия содержит несколько опытов, каждый из которых выполняется для определенных условий:

- таблица не имеет индексов, группировка – по одному из столбцов;
- таблица не имеет кластеризованного индекса, группировка – по одному из индексированных столбцов;
- таблица имеет первичный ключ, группировка – по одному из неиндексированных столбцов;
- таблица имеет первичный ключ, группировка – по одному из индексированных столбцов.

## ЛАБОРАТОРНАЯ РАБОТА № 5

### Анализ архитектуры системы информационной безопасности MS SQL-Server

**Цель работы:** ознакомление со средствами управления информационной безопасностью на уровне сервера и пользовательских баз данных.

#### **Задачи:**

- освоить компоненты пользовательского интерфейса *SQL-Server Management Studio*, обеспечивающие возможность просмотра и настройки параметров информационной безопасности;
- исследовать свойства учетных записей, пользователей и ролей сервера и базы данных;
- освоить технику создания и модификации учетных записей сервера, ролей и пользователей баз данных соответствующими средствами языка *Transact SQL*;
- освоить программные средства управления членством в ролях;
- исследовать объекты системного каталога, ответственные за хранение параметров информационной безопасности.

#### **Практические задания**

##### **Задание 1. Анализ серверных компонентов системы информационной безопасности**

- 1.1 Выполните вход в систему, используя режим аутентификации Windows. Просмотрите свойства учетных записей (имен входа, logins) сервера. Переименуйте учетные записи. Какие из учетных записей относятся к категории «Учетные записи SQL Server» ?
- 1.2 Создайте новую учетную запись: выберите для нее режим аутентификации SQL Server, задайте (и запомните !) пароль и установите свойства учетной записи по своему усмотрению.
- 1.3 Создайте учетную запись SQL-оператором CREATE LOGIN.
- 1.4 Просмотрите свойства созданных учетных записей, определите их членство в фиксированных ролях сервера.
- 1.5 Включите одну из созданных учетных записей в состав членов серверной роли Securityadmin, а другую – в состав членов серверной роли Sysadmin.
- 1.6 Прямым доступом к системной таблице SysLogins базы данных MASTER определите состав и основные параметры:
  - учетных записей Windows;



- учетных записей пользователей Windows;
  - учетных записей групп Windows;
  - учетных записей SQL Server;
- 1.7 Выполните вход в систему, используя режим аутентификации SQL Server и новую учетную запись, просмотрите свойства доступных учетных записей. Создайте новую базу данных. Прокомментируйте полученные результаты.
- 1.8 Просмотрите и проанализируйте свойства серверных ролей Sysadmin, Securityadmin и Public.
- 1.9 Используя системные хранимые процедуры (таблица 5.2):
- определите состав членов всех серверных ролей;
  - добавьте учетную запись в состав членов роли Securityadmin;
  - определите перечень разрешений, установленных для серверной роли Securityadmin;
- 1.10 Ознакомьтесь с перечнем и назначением хранимых процедур и функций информационной безопасности, права на выполнение которых получают члены фиксированной роли сервера Securityadmin (используйте документацию разработчика technet).
- 1.11 Сделайте выводы по результатам выполнения задания и сохраните их в отчете по лабораторной работе.

## **Задание 2. Анализ компонентов информационной безопасности уровня базы данных**

- 2.1 Выполните вход в систему, используя режим аутентификации Windows. Активизируйте одну из пользовательских баз данных, созданных при выполнении предыдущих лабораторных работ. Просмотрите свойства пользователей базы данных, их членство в фиксированных ролях базы данных.
- 2.2 Создайте в контексте этой базы данных нового пользователя, сопоставьте его с одной из учетных записей, созданных при выполнении предыдущего задания, включите его в фиксированную роль db\_owner.
- 2.3 Просмотрите и проанализируйте свойства ролей базы данных db\_owner, db\_securityadmin и public.

2.4 Используя SQL-команды CREATE USER, CREATE ROLE и ALTER ROLE, создайте нового пользователя базы данных, пользовательскую роль, включите пользователя в состав членов этой роли.

2.5 Прямым доступом к системной таблице SysUsers базы данных определите состав и основные параметры пользователей и ролей.

### ***Контрольные вопросы***

- 1 Дайте сравнительные характеристики, назовите преимущества и недостатки двух режимов проверки подлинности пользователей: аутентификация Windows и аутентификация SQL Server.
- 2 Членство в каких серверных ролях позволяет пользователю создавать, удалять и модифицировать учетные записи пользователей?
- 3 Выполнение каких операций разрешено членам серверной роли public?
- 4 Перечислите основные объекты и субъекты доступа уровня базы данных.
- 5 Какие пользователи автоматически создаются при создании базы данных?  
Какими правами обладают эти пользователи ?
- 6 Может ли пользователь базы данных одновременно быть членом нескольких ролей базы данных?
- 7 Может ли пользовательская роль базы данных быть членом фиксированной роли или другой пользовательской роли?

## ЛАБОРАТОРНАЯ РАБОТА №6

### Анализ средств управления доступом к данным

**Цель работы:** изучение средств управления правами доступа субъектов к логическим объектам базы данных MS SQL-Server.

#### **Задачи:**

- освоить компоненты пользовательского интерфейса *SQL-Server Management Studio* и программные средства, обеспечивающие возможность просмотра и настройки разрешений доступа;
- исследовать объекты системного каталога, ответственные за хранение информации о правах доступа.

#### **Практические задания**

##### **Задание 1. Управление пользователями и ролями базы данных средствами SQL-Server Management Studio**

- 1.12 Выполните вход в систему, используя режим аутентификации Windows. Активизируйте одну из пользовательских баз данных (или создайте новую). Просмотрите свойства пользователей и фиксированных ролей базы данных. Какие пользователи были созданы сервером "самостоятельно", а какие – в результате действий администратора ? Определите членство пользователей в ролях базы данных.
- 1.13 Создайте три новых учетных записи SQL-Server, установите для каждой из них текущую базу данных в качестве базы данных по умолчанию. Создайте в этой базе данных трех новых пользователей, сопоставив их с новыми учетными записями. Включите этих пользователей в состав нескольких фиксированных ролей базы данных (по своему усмотрению).
- 1.14 Создайте пользовательскую роль базы данных, включите в состав ее членов всех трех новых пользователей, а также пользователя *guest*. Затем удалите пользователя *guest*.
- 1.15 Прямым доступом к системной таблице SysUsers определите состав пользователей, фиксированных и пользовательских ролей базы данных.
- 1.16 Сделайте выводы по результатам выполнения задания и сохраните их в отчете по лабораторной работе.

## **Задание 2. Программные средства управления разрешениями уровня базы данных**

- 2.1 Используя системную хранимую процедуру `sp_helpprotect` (описание и примеры ее использования – в разделе 5.3.4 учебного пособия), определите права доступа к данным и права на выполнение операций для пользователя *dbo*, а также для созданных при выполнении предыдущего задания пользователя и пользовательской роли базы.
- 2.2 Используя SQL-команды `GRANT` и `DENY` предоставьте и запретите права доступа пользователям (по своему усмотрению), а затем SQL-командой `REVOKE` отмените некоторые из предоставленных разрешений и запретов.
- 2.3 Просмотрите информацию о правах доступа к объектам базы данных, используя хранимую процедуру `sp_helpprotect`, системное представление `sys.database_permissions` и системную функцию `fn_built_in_permissions()`.
- 2.4 Выполните вход в систему, используя режим аутентификации SQL Server и одну из учетных записей пользователей, созданных при выполнении задания 1.2. Активизируйте базу данных, в которой ранее был создан пользователь, выполнивший вход в систему, и экспериментально проверьте действие выданных ему разрешений и запретов.
- 2.5 Повторно выполните задание 2.4, используя учетную запись другого пользователя этой базы данных.
- 2.6 Сделайте выводы по результатам выполнения задания и сохраните их в отчете по лабораторной работе.

## ЛАБОРАТОРНАЯ РАБОТА №7

### Анализ иерархии прав доступа к данным

MS SQL Server поддерживает сложную иерархическую систему разрешений на выполнение операций с объектами баз данных и множество методов и инструментальных средств управления этими разрешениями. При этом параллельно существуют несколько (не всегда строгих) иерархий, например: «сервер – база данных», «роль – пользователь», «база данных – схема – таблица – столбец», «процедура – представление – таблица».

Все это позволяет администраторам реализовывать надежные, гибкие и технологичные системы защиты информации, но, с другой стороны, может создавать проблемы, связанные с необходимостью понимания и учета приоритетов разрешений, предоставленных субъектам доступа на разных уровнях и / или разными методами.

**Цель работы** – изучение системы приоритетов разрешений доступа к данным, реализованной в MS SQL Server.

**Задачи.** При выполнении лабораторной работы ставится комплексная задача экспериментального подтверждения, уточнения или опровержения следующих рабочих гипотез, касающихся приоритетов прав доступа к объектам баз данных:

**Гипотеза № 1.** Запрет доступа (Deny) к объекту всегда имеет более высокий приоритет по сравнению с предоставлением (Grant) доступа.

**Гипотеза № 2.** Глобальные права доступа к данным, предоставленные пользователю через членство в фиксированных ролях базы данных, имеют более высокий приоритет по сравнению с локальными правами, предоставленными ему явно или через членство в пользовательских ролях базы данных.

**Гипотеза № 3.** Права доступа, предоставленные пользователю неявно через членство в пользовательской роли базы данных, имеют более высокий приоритет по сравнению с правами, предоставленными ему явно (персонально).

**Гипотеза № 4.** Права доступа к отдельным столбцам таблицы имеют более низкий приоритет по сравнению с правами, предоставленными на таблицу в целом.

**Гипотеза № 5.** Право на выполнение (Execute) процедуры или хранимого представления, содержащих SQL-оператор доступа к таблице,

позволяет пользователю получить несанкционированный доступ к этой таблице, несмотря на:

- явный запрет (Deny) доступа к этой таблице;
- явный запрет на выполнение операций (например, Select);
- членство пользователя в фиксированных ролях базы данных DB\_denydatareader и DB\_denydatawriter.

**Гипотеза № 6.** Право на создание процедуры (Create Proc) позволяет пользователю включить в нее SQL-оператор чтения данных таблицы, доступ к которой ему явно запрещен, и получить таким образом несанкционированный доступ к этой таблице.

### Задание

1 Спланируйте эксперименты по проверке всех шести рабочих гипотез:

- определите состав субъектов и объектов доступа и соответствующих разрешений;
- подготовьте текстовый файл для формирования отчета по лабораторной работе с описанием каждого из запланированных экспериментов;
- включите в отчет бланки протоколов для регистрации результатов экспериментов.

2 Подготовьте базу данных для проведения экспериментов (можно использовать выполненные ранее собственные разработки или учебную базу данных).

3 Подготовьте SQL-запросы, хранимые представления и процедуры, необходимые для проверки рабочих гипотез.

4 Создайте в этой базе данных необходимое количество пользователей (сопоставленных с соответствующими учетными записями SQL Server) и пользовательских ролей.

5 Определите для пользователей и пользовательских ролей соответствующие права доступа к объектам базы данных.

6 Распределите пользователей по фиксированным и пользовательским ролям базы данных.

7 Проведите эксперименты, сохраните в отчете их результаты и собственные выводы.

## Список использованных источников

- 1 Standard Occupational Classification [Electronic resource] / U.S. Bureau of Labor Statistics. – Electronic data (1 file : 974848 bytes). URL: [https://www.bls.gov/soc/2018/soc\\_structure\\_2018.pdf](https://www.bls.gov/soc/2018/soc_structure_2018.pdf), free. Title from screen.
- 2 ГОСТ Р 56413-2015 Информационные технологии. Европейские профили профессий ИКТ-сектора /CWA 16458:2012 Information technologies. European ICT professional profiles. – Введ. 01.06.2016 приказом Федерального агентства по техническому регулированию и метрологии от 29 мая 2015 г. № 465-ст.
- 3 Профессиональный стандарт 06.011 «Администратор баз данных». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 12 декабря 2016 г. №727н.
- 4 Профессиональный стандарт 06.015 «Специалист по информационным системам». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 12 декабря 2016 г. №727н.
- 5 Профессиональный стандарт 06.026 «Системный администратор информационно-коммуникационных систем». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 5 октября 2015 г. №684н.
- 6 Профессиональный стандарт 06.033 «Специалист по защите информации в автоматизированных системах». Утвержден приказом Министерства труда и социальной защиты Российской Федерации от 15 сентября 2016 г. №522н.
- 7 Справочник по логическим и физическим операторам Showplan. URL: <https://msdn.microsoft.com/ru-ru/library/ms191158.aspx>.
- 8 Отображение планов выполнения с помощью параметров Showplan инструкции SET (Transact-SQL). URL: [http://msdn.microsoft.com/ru-ru/library/ms180765\(v=SQL.100\).aspx](http://msdn.microsoft.com/ru-ru/library/ms180765(v=SQL.100).aspx).
- 9 Графическое отображение планов выполнения (SQL Server Management Studio). URL: [http://msdn.microsoft.com/ru-ru/library/ms178071\(v=SQL.100\).aspx](http://msdn.microsoft.com/ru-ru/library/ms178071(v=SQL.100).aspx).
- 10 ISO 10181:1-7.ВOC.1996-1998. Структура работ по безопасности в открытых системах. Ч. 1. Обзор. Ч. 2. Структура работ по аутентификации. Ч. 3. Структура работ по управлению доступом. Ч. 4. Структура работ по безотказности. Ч. 5. Структура работ по конфиденциальности. Ч. 6. Структура работ по обеспечению целостности. Ч. 7. Структура работ по проведению аудита на безопасность.
- 11 Козленко Л. Информационная безопасность в современных системах управления базами данных. URL: <http://compress.ru/article.aspx?id=10099>.
- 12 Sys.database\_permissions (Transact-SQL). URL: <https://docs.microsoft.com/en-us/sql/relational-databases/system-catalog-views/sys-database-permissions-transact-sql>.
- 13 Sys.fn\_builtin\_permissions (Transact-SQL). URL: <https://docs.microsoft.com/en-us/sql/relational-databases/system-functions/sys-fn-builtin-permissions-transact-sql>.

Учебное издание

Владимир Константинович Волк

# **БАЗЫ ДАННЫХ**

## **Часть 2**

### **АДМИНИСТРИРОВАНИЕ**

Учебное пособие

Редактор Л.П. Чукомина

---

|                               |                   |                            |
|-------------------------------|-------------------|----------------------------|
| Подписано в печать 12.04.2018 | Формат 60×84 1/16 | Бумага 80 г/м <sup>2</sup> |
| Печать цифровая               | Усл. печ. л. 8,00 | Уч.-изд. л. 8,00           |
| Заказ № 83                    | Тираж 100         |                            |

---

Библиотечно-издательский центр КГУ.  
640020, г. Курган, ул. Советская, 63/4.  
Курганский государственный университет.