Белорусский национальный технический университет
Международный институт дистанционного образования
Кафедра «Информационные системы и технологии»

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

Системы управления базами данных

для студентов специальностей:
I – 40 01 01 «Программное обеспечение информационных технологий» и
I – 40 05 01 «Информационные системы и технологии»

Составитель: Бухвалова И.А.

Минск, БНТУ 2020

Перечень материалов

Электронный учебно-методический комплекс включает:

- теоретический раздел, который содержит следующие подразделы:

система управления базами данных: структура, основные функции,
архитектура современной СУБД на примере MS SQL Server,
методы обеспечения целостности базы данных,
использование транзакций для обеспечения безопасности и параллелизма в работе с базой данных,
программирование в MS SQL SERVER,

- практический раздел, который состоит из восьми лабораторных работ,

- контроль знаний, который состоит их контрольных вопросов и тестовых заданий к восьми лабораторным работам,

- вспомогательный раздел, который содержит список рекомендуемой литературы.

Пояснительная записка

Электронный учебно-методический комплекс разработан для студентов специальности I – 40 01 01 «Программное обеспечение информационных технологий» и I-40 05 01 «Информационные системы и технологии» направление I – 40 05 01 – 04 «Информационные системы и технологии, обработка и представление информации». Информационное наполнение ЭУМК соответствует программе дисциплины «Системы управления базами данных». Комплекс предназначен для студентов дневного и заочного отделений.

Внедрение ЭУМК будет способствовать более эффективному овладению теоретическими и практическими основами систем управления базами данных.

ЭУМК разработан в виде pdf-файла, что делает его универсальным и позволяет применять как на локальном компьютере, так и в локальной или глобальной сети. ЭУМК не требует установки специального программного обеспечения. Для работы с ним достаточно иметь операционную систему семейства WINDOWS.

ЭУМК может использоваться для изучения теоретических основ дисциплины, при проведении лабораторных занятий, контрольных работ, тестирования, а также в ходе подготовки студентов к экзамену по дисциплине «Системы управления базами данных», «Базы данных».

Оглавление

ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

Система управления базами данных: структура, основные функции.

СУБД– программный комплекс поддержки интегрированной совокупности данных, предназначенный для создания, ведения и использования базы данных многими пользователями (прикладными программами).
Перечислим основные *функции системы управления базами данных*.

1. Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки.
Как правило, создание структуры **базы данных** происходит в режиме диалога. *СУБД* последовательно запрашивает у пользователя необходимые данные. В большинстве современных *СУБД* **база данных** представляется в виде совокупности таблиц. Рассматриваемая функция позволяет описать и создать в памяти структуру таблицы, провести начальную загрузку данных в таблицы.
2. Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация).
Такие возможности в *СУБД* представляются либо на основе использования специального языка программирования, входящего в состав *СУБД*, либо с помощью графического интерфейса. Для клиент-серверных *СУБД* существуют средства, позволяющие выполнять запросы, и программные средства, позволяющие создавать графический интерфейс пользователя.
3. Обеспечение независимости прикладных программ и данных (логической и физической независимости).
Важнейшим свойством *СУБД* является возможность поддерживать два независимых взгляда на базу данных – "взгляд пользователя", воплощаемый в логическом представлении данных, и его отражения в прикладных программах; и "взгляд системы" – физическое представление данных в памяти ЭВМ. Обеспечение логической независимости данных предоставляет возможность изменения (в определенных пределах) логического представления базы данных без необходимости изменения физических структур хранения данных. Таким образом, изменение логического представления данных в прикладных программах не приводит к изменению структур хранения данных. Обеспечение физической независимости данных предоставляет возможность изменять (в определенных пределах) способы организации базы данных в памяти ЭВМ не вызывая необходимости изменения "логического" представления данных. Таким образом, изменение способов организации базы данных не приводит к изменению прикладных программ.
4. Защита логической целостности базы данных.
Основной целью реализации этой функции является повышение достоверности данных в базе данных. Достоверность данных может быть нарушена при их вводе в БД или при неправомерных действиях процедур

обработки данных, получающих и заносящих в БД неправильные данные. Для повышения достоверности данных в системе объявляются так называемые ограничения целостности, которые в определенных случаях "отлавливают" неверные данные. Так, во всех современных СУБД проверяется соответствие вводимых данных их типу, описанному при создании структуры. Система не позволит ввести символ в поле числового типа, не позволит ввести недопустимую дату и т.п. В развитых системах ограничения целостности описывает программист, исходя из содержательного смысла задачи, и их проверка осуществляется при каждом обновлении данных. Более подробно разные аспекты логической целостности базы данных будут рассматриваться в последующих разделах.

5. Защита физической целостности.

При работе ЭВМ возможны сбои в работе (например, из-за отключения электропитания), повреждение машинных носителей данных. При этом могут быть нарушены связи между данными, что приводит к невозможности дальнейшей работы. Развитые СУБД имеют средства восстановления базы данных. Важнейшим используемым понятием является понятие " *транзакции* ". *Транзакция* – это единица действий, производимых с базой данных. В состав транзакции может входить несколько операторов изменения базы данных, но либо выполняются все эти операторы, либо не выполняется ни один. СУБД, кроме ведения собственно базы данных, ведет также журнал *транзакций*. Необходимость использования *транзакций* в базах данных проиллюстрируем на упрощенном примере. Предположим, что база данных используется в некотором банке и один из клиентов желает перевести деньги на счет другого клиента банка. В базе данных хранится информация о количестве денег у каждого из клиентов. Нам нужно сделать два изменения в базе данных – уменьшить сумму денег на счете одного из клиентов и, соответственно, увеличить сумму денег на другом счете. Конечно, реальный перевод денег в банке представляет собой гораздо более сложный процесс, затрагивающий много таблиц, а возможно, и много баз данных. Однако суть остается та же – нужно либо совершить все действия (увеличить счет одного клиента и уменьшить счет другого), либо не выполнить ни одно из этих действий. Нельзя уменьшить сумму денег на одном счете, но не увеличить сумму денег на другом. Предположим также, что после выполнения первого из действий (уменьшения суммы денег на счете первого клиента) произошел сбой. Например, могла прерваться связь клиентского компьютера с базой данных или на клиентском компьютере мог произойти системный сбой, что привело к перезагрузке операционной системы. Что в этом случае стало с базой данных? Команда на уменьшение денег на счете первого клиента была выполнена, а вторая команда – на увеличение денег на другом счете – нет, что привело бы к противоречивому, неактуальному состоянию базы данных. Использование механизма *транзакций* позволяет находить решение в этом и подобных случаях. Перед выполнением первого действия выдается команда начала транзакции. В транзакцию включается операция снятия денег на одном счете

и увеличения суммы на другом счете. Оператор завершения транзакций обычно называется COMMIT. Поскольку после выполнения первого действия транзакция не была завершена, изменения не будут внесены в базу данных. Изменения вносятся (фиксируются) только после завершения транзакции. До выдачи данного оператора сохранения данных в базе не произойдет.В нашем примере, поскольку оператор фиксации транзакции не был выдан, база данных "откатится" в первоначальное состояние – иными словами, суммы на счетах клиентов останутся те же, что и были до начала транзакции. Администратор базы данных может отслеживать состояние транзакций и в необходимых случаях вручную "откатывать" транзакции. Кроме того, в очевидных случаях СУБД самостоятельно принимает решение об "откате" транзакции.

Транзакции не обязательно могут быть короткими. Бывают транзакции, которые длятся несколько часов или даже несколько дней. Увеличение количества действий в рамках одной транзакции требует увеличения занимаемых системных ресурсов. Поэтому желательно делать транзакции по возможности короткими. В журнал транзакций заносятся все транзакции – и зафиксированные, и завершившиеся "откатом". Ведение журнала транзакций совместно с созданием резервных копий базы данных позволяет достичь высокой надежности базы данных.

Предположим, что база данных была испорчена в результате аппаратного сбоя компьютера, на котором был установлен сервер СУБД. В этом случае нужно использовать последнюю сделанную резервную копию базы данных и журнал транзакций. Причем применить к базе данных нужно только те транзакции, которые были зафиксированы после создания резервной копии. Большинство современных СУБД позволяют администратору воссоздать базу данных исходя из резервной копии и журнала транзакций. В таких системах в определенный момент БД копируется на резервные носители. Все обращения к БД записываются программно в журнал изменений. Если база данных разрушена, запускается процедура восстановления, в процессе которой в резервную копию из журнала изменений вносятся все произведенные изменения.

6. Управление полномочиями пользователей на доступ к базе данных.

Разные пользователи могут иметь разные полномочия по работе с данными (некоторые данные должны быть недоступны; определенным пользователям не разрешается обновлять данные и т.п.). В СУБД предусматриваются механизмы разграничения полномочий доступа, основанные либо на принципах паролей, либо на описании полномочий.

7. Синхронизация работы нескольких пользователей.

Достаточно часто может иметь место ситуация, когда несколько пользователей одновременно выполняют операцию обновления одних и тех же данных. Такие коллизии могут привести к нарушению логической целостности данных, поэтому система должна предусматривать меры, не допускающие обновление данных другим пользователям, пока работающий с этими данными пользователь полностью не закончит с ними работать.

Основным используемым здесь понятием является понятие " *блокировка* ". *Блокировки* необходимы для того, чтобы запретить различным пользователям возможность одновременно работать с базой данных, поскольку это может привести к ошибкам.
Для реализации этого запрета СУБД устанавливает блокировку на объекты, которые использует транзакция. Существуют разные типы блокировок – табличные, страничные, строчные и другие, которые отличаются друг от друга количеством заблокированных записей. Чаще других используется строчная блокировка – при обращении транзакции к одной строке блокируется только эта строка, остальные строки остаются доступными для изменения.
Таким образом, процесс внесения изменений в базу данных состоит из следующей последовательности действий: выдается оператор начала транзакции, выдается оператор изменения данных, СУБД анализирует оператор и пытается установить блокировки, необходимые для его выполнения, в случае успешной блокировки оператор выполняется, затем процесс повторяется для следующего оператора транзакции. После успешного выполнения всех операторов внутри транзакции выполняется оператор фиксации транзакции. СУБД фиксирует изменения, сделанные транзакцией, и снимает блокировки. В случае неуспеха выполнения какого-либо из операторов транзакция "откатывается", данные получают прежние значения, блокировки снимаются.
8. Управление ресурсами среды хранения.
БД располагается во внешней памяти ЭВМ. При работе в БД заносятся новые данные (занимается память) и удаляются данные (освобождается память). СУБД выделяет ресурсы памяти для новых данных, перераспределяет освободившуюся память, организует ведение очереди запросов к внешней памяти и т.п.
9. Поддержка деятельности системного персонала.
При эксплуатации базы данных может возникать необходимость изменения параметров СУБД, выбора новых методов доступа, изменения (в определенных пределах) структуры хранимых данных, а также выполнения ряда других общесистемных действий. СУБД предоставляет возможность выполнения этих и других действий для поддержки деятельности БД обслуживающему БД системному персоналу, называемому администратором БД.

Архитектура современной СУБД на примере MS SQL Server.

Для лучшего понимания принципов работы современных СУБД рассмотрим структуру одной из наиболее распространенных клиент-серверных СУБД - *Microsoft SQL Server. Под архитектурой (структурой) базы данных конкретной СУБД будем понимать основные модели представления данных, используемые в соответствующей СУБД а также взаимосвязи между этими моделями*. В соответствии с рассмотренными ранее различными

уровнями описания данных различают разные уровни абстракции архитектуры базы данных.

Логический уровень (уровень модели данных СУБД) - средство представления концептуальной модели. Здесь каждая СУБД имеет некоторые отличия, но они являются не очень значительными. Отметим, что у разных СУБД существенно отличаются механизмы перехода от логического к физическому уровню представления.

Физический уровень (внутреннее представление данных в памяти ЭВМ - физическая структура базы данных). Данный уровень рассмотрения подразумевает изучение базы данных на уровне файлов, хранящихся на жестком диске. Структура этих файлов – особенность каждой конкретной СУБД, в т.ч. и Microsoft SQL Server.

,*Microsoft SQL Server* представляет собой реляционную СУБД . Таким образом, основной структурой модели данных этой СУБД являются таблицы. Таблицы содержат данные о всех сущностях концептуальной модели базы данных. При описании каждого столбца (поля) пользователь должен определить тип соответствующих данных.*Microsoft SQL Server 2012* поддерживает как уже ставшие традиционными типы данных (символьная строка с разным представлением, число с плавающей точкой длиной 8 или 4 байта, целое число длины 2 или 4 байта, дата и время, поле примечаний и т. д.), так и новые типы данных. Кроме этого *Microsoft SQL Server 2012* предоставляет специальный аппарат для создания *пользовательских типов данных*.

Для каждой таблицы должен быть определен *первичный ключ – минимальный набор атрибутов, уникально идентифицирующий каждую запись в таблице*. Для реализации связи между таблицами в одну из связанных таблиц включается дополнительное поле (несколько полей) – первичный ключ другой таблицы. Дополнительно включенные поле или поля в этом случае называются внешним ключом соответствующей таблицы.

Кроме таблиц, в модель данных *Microsoft SQL Server 2012* входит еще целый ряд компонентов. Дадим краткую характеристику основным из них.

Индексы создаются для ускорения поиска нужной информации и содержат информацию об упорядоченности данных по различным критериям. Индексирование может быть выполнено по одному или нескольким столбцам. Индексирование может быть произведено в любой момент. Индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении. Эти ключи хранятся в виде структуры *сбалансированного дерева*, которая поддерживает быстрый поиск строк по их ключевым значениям в SQL Server.

Представление — это виртуальная таблица, содержимое которой определяется запросом. Представление формируется на основе SQL-запроса SELECT, формируемого по обычным правилам. Таким образом, представление есть поименованный запрос SELECT.

Ограничения позволяют задать метод, с помощью которого компонент СУБД Database Engine автоматически обеспечивает целостность базы

данных. Ограничения задают правила допустимости определенных значений в столбцах и представляют собой стандартный механизм обеспечения целостности. Рекомендуется использовать ограничения, а не триггеры , правила и значения по умолчанию. *Оптимизатор запросов* также использует определения ограничений для построения высокопроизводительных *планов выполнения* запросов.

Правила – еще один специальный механизм, предназначенный для обеспечения целостности базы данных, по функциональности напоминающие некоторые типы ограничений. Microsoft отмечает, что при соответствующей возможности использование ограничений по ряду причин предпочтительнее и, возможно, в будущей версии эта возможность будет удалена.

Значения по умолчанию определяют, какими значениями заполнять столбец, если при вставке строки для этого столбца значение не указано. Значение по умолчанию могут быть любым выражением, результат которого — константа, например, собственно константой, встроенной функцией или выражением.

Физический уровень - это представление данных в памяти ЭВМ. Основными понятиями, используемыми для представления структуры хранения (физического уровня) являются понятия файла (физического) и единицы обмена между внешней и оперативной памятью (физической записи или страницы).

На физическом уровне база данных в *Microsoft SQL Server 2012* представляется набором файлов операционной системы. Данные и сведения журналов транзакций всегда размещаются в разных файлах. Отдельные файлы используются только одной базой данных. Файловые группы представляют собой именованные коллекции файлов и используются для упрощения размещения данных и выполнения задач администрирования, например, резервного копирования и восстановления.

Базы данных SQL Server содержат файлы трех типов:

- **Первичные файлы данных.**
  Первичный файл данных является отправной точкой базы данных. Он указывает на остальные файлы базы данных. В каждой базе данных имеется один *первичный файл* данных. Для имени первичного файла данных рекомендуется использовать расширение MDF.
- **Вторичные файлы данных.**
  Ко вторичным файлам данных относятся все файлы данных, за исключением первичного файла данных. Базы данных могут вообще не содержать *вторичных файлов* данных, или содержать один или несколько *вторичных файлов* данных. Для имени *вторичного файла* данных рекомендуется использовать расширение NDF.
- **Файлы журналов.**
  Файлы журналов содержат все сведения журналов, используемые для восстановления базы данных. В каждой базе данных должен быть по меньшей мере один файл журнала, но их может быть и больше. Для

имен файлов журналов рекомендуется использовать расширение MDF, NDF и LDF. Однако эти расширения помогают пользователю идентифицировать различные виды файлов и правильно их использовать.

В SQL Server расположение всех файлов базы данных записывается в *первичный файл* базы данных и в специальную служебную структуру СУБД SQL Server, называемою базой данных master. В большинстве случаев при работе с базой данных компонент СУБД (SQL Server *Database Engine*) использует сведения о размещении файлов, хранимые в базе данных master. Однако в некоторых случаях (например, при восстановлении базы данных master из копии, при определенным образом проводимым присоединении базы данных) компонент *Database Engine* использует сведения о расположении файлов из первичного файла, чтобы инициализировать записи о расположении файлов в базе данных master.

Файлы SQL Server имеют два имени:

- logical_file_name — имя, используемое для ссылки на физический файл во всех инструкциях Transact-SQL. *Логическое имя файла* должно соответствовать правилам для идентификаторов SQL Server и быть уникальным среди логических имен файлов в соответствующей базе данных.
- os_file_name — это имя физического файла, включая путь к каталогу. Оно должно соответствовать правилам для имен файлов операционной системы.

Изначально можно указать максимальный размер каждого файла. Если максимальный размер файла не указан, файлы SQL Server могут автоматически увеличиваться в размерах, превосходя первоначально заданные показатели, пока не займут все доступное место на диске. При определении файла пользователь может указывать требуемый шаг роста. Каждый раз при заполнении файла его размер увеличивается на указанный шаг роста. Если в файловой группе имеется несколько файлов, их автоматический рост начинается лишь по заполнении всех файлов. Затем файлы увеличиваются в размерах по кольцевому списку. Эта функция особенно полезна в случаях, когда SQL Server используется в качестве базы данных, внедренной в приложение, где пользователь не имеет удобного доступа к системному администратору. По мере необходимости пользователь может предоставить файлам возможность увеличиваться в размерах автоматически, тем самым снимая с администратора часть забот по наблюдению за свободным пространством базы данных и по распределению дополнительного пространства вручную.

Методы обеспечения целостности базы данных.

Одним из основополагающих понятий в технологии баз данных является понятие целостности. В общем случае, это понятие, прежде всего, связано с тем, что база данных отражает в информационном виде некоторый

объект реального мира или совокупность взаимосвязанных объектов реального мира. В реляционной модели объекты реального мира представлены в виде совокупности взаимосвязанных отношений. Целостность базы данных – это правила и средства, обеспечивающие надежную реализацию установленных межтабличных связей между всеми данными, содержащимися в базе. Поддержание целостности данных является достаточно серьезным и сложным вопросом. При эксплуатации базы данных ее повреждение может возникнуть по нескольким причинам: при сбое компьютера, вследствие ошибок в программном обеспечении, из-за некорректных действий пользователя.

Любое изменение в предметной области, значимое для построенной модели, должно отражаться в базе данных, и при этом должна сохраняться однозначная интерпретация информационной модели в терминах предметной области.

Поддержка целостности в реляционной модели данных в ее классическом понимании, включает в себя 3 аспекта.

Во-первых, это поддержка структурной целостности, которая трактуется как то, что реляционная СУБД должна допускать работу только с однородными структурами данных типа «реляционное отношение». При этом понятие «реляционного отношения» должно удовлетворять всем ограничениям, накладываемым на него в классической теории реляционной БД. Отсутствие дубликатов кортежей (строки отношений), соответственно, обязательное наличие первичного ключа (один или несколько столбцов (атрибутов), которые однозначно идентифицируют каждую запись в таблице, т.е. позволяют четко отличить одну запись от другой), отсутствие понятия упорядоченности кортежей.

В дополнение к структурной целостности необходимо рассмотреть проблему неопределенных Null значений. Неопределенное значение в реляционной базе данных интерпретируется как значение, неизвестное на данный момент времени. Это значение при появлении дополнительной информации в любой момент времени может быть заменено на некоторое конкретное значение.

Во-вторых, это поддержка языковой целостности, которая состоит в том, что реляционная СУБД должна обеспечивать языки описания и манипулирования данными не ниже стандарта SQL. Не должны быть доступны иные низкоуровневые средства манипулирования данными, не соответствующие стандарту.

Именно поэтому доступ к информации, хранимой в базе данных, и любые изменения этой информации могут быть выполнены только с использованием операторов языка SQL.

В-третьих, это поддержка ссылочной целостности (Declarative Referential Integrity, DRI), означает обеспечение одного из заданных принципов взаимосвязи между экземплярами кортежей взаимосвязанных отношений:

· кортежи подчиненного отношения уничтожаются при удалении кортежа основного отношения, связанного с ним;

· кортежи основного отношения модифицируются при удалении кортежа основного отношения, связанного с ним, при этом на месте ключа родительского отношения ставится неопределенное Null значение.
Ссылочная целостность обеспечивает поддержку непротиворечивого состояния БД в процессе модификации данных при выполнении операций добавления или удаления.
Кроме указанных ограничений целостности, которые в общем виде не определяют семантику БД, вводится понятие семантической поддержки целостности.
Структурная, языковая и ссылочная целостности определяют правила работы СУБД с реляционными структурами данных. Требования поддержки этих трех видов целостности говорят о том, что каждая СУБД должна уметь это делать, а разработчики должны это учитывать при построении БД с использованием реляционной модели. Эти три аспекта никак не касаются содержания БД. Для определения некоторых ограничений, которые связаны с содержанием БД, требуются другие методы. Именно эти методы и сведены в поддержку семантической целостности.
Семантическая поддержка может быть обеспечена двумя путями:
· декларативный, выполняемый средствами языка SQL;
· процедурный, выполняемый посредством триггеров и хранимых процедур.
Декларативный путь связан с наличием механизмов в рамках СУБД, обеспечивающих проверку и выполнение ряда декларативно заданных правил-ограничений, называемых чаще всего декларативными ограничениями целостности.

Использование транзакций для обеспечения безопасности и параллелизма в работе с базой данных.

Если с базой данных работает несколько человек, то неизменно возникает проблема совместного использования данных.
Основы транзакций

Транзакция (transaction) представляет собой набор из одной или более команд, обрабатываемых как единое целое. То есть будет выполнен либо весь набор команд, либо не выполнена ни одна из них. Кроме того, с помощью распределенных транзакций можно изменять строки в разных базах данных.

По умолчанию каждая команда Transact-SQL рассматривается как отдельная транзакция. Имеются определенные требования к выполнению транзакций. Эти требования, известные как требования ACID (Atomicity, Consistency, Isolation и Durability), описывают то, как должны обрабатываться данные и в каком состоянии они должны находиться после завершения транзакции:

- Атомарность (Atomicity). Все изменения данных, выполненные в транзакции, рассматриваются как единый минимальный блок. Не может быть такого, что изменения, внесенные одной командой, будут зафиксированы, а изменения, выполненные остальными командами,

будут отменены. Зафиксированными могут оказаться либо все изменения, выполненные в транзакции, либо данные будут восстановлены в состоянии, в котором они были до начала транзакции.

- Согласованность (Consistency). После того, как транзакция будет успешно завершена, данные должны удовлетворять всем ограничениям целостности, определенным в базе данных. Кроме того, все связанные с измененными данными индексы должны находиться в корректном состоянии. Каждая команда производит изменение данных в таблице. В ходе этих изменений могут быть нарушены правила и ограничения целостности, наложенные на данные. SQL Server 2000 позволяет контролировать целостность данных двумя способами: целостность данных может проверяться после выполнения каждой команды или только при её фиксировании.
- Изолированность (Isolation). Изменения данных, выполняемые различными транзакциями, должны быть независимыми друг от друга, т. е. быть изолированными. Если транзакция выбирает строки по определенному логическому условию, то никакая другая транзакция не должна изменять, добавлять или удалять строки, которые соответствуют указанному логическому условию. Такое поведение известно как "сериализуемость".
- Устойчивость или долговечность (Durability). После того, как транзакция выполнит все необходимые изменения и ее работа будет завершена, система выполняет фиксирование транзакции (commit transaction). После этого система не может быть возвращена в состояние, в котором она была до начала транзакции.

Выполнение требования ACID по обработке транзакций распространяется на систему управления базами данных. То есть пользователь не должен предпринимать никаких дополнительных усилий, чтобы соблюсти перечисленные требования.

Чтобы обеспечить требования изолированности транзакций используются блокировки (locks). По возможности следует включать в транзакцию как можно меньше команд, чтобы блокировать ресурсы минимальное количество времени.

Основы блокировок

Блокировкой (locks) называется временное ограничение, накладываемое системой на использование тех или иных ресурсов. Блокировки используются для обеспечения изолированности транзакций друг от друга. Простейшим способом обеспечения изолированности транзакций является запрещение любого обращения к данным, уже используемым в одной из транзакций. Однако часто достаточно просто запретить изменение данных, оставив возможность чтения. Тем не менее, в случае, когда транзакция изменяет данные, необходимо полностью блокировать их. Это гарантирует, что другие транзакции не будут использовать промежуточные данные.

В SQL Server имеется несколько различных типов блокировок. По возможности система старается применить как можно менее жесткий режим блокирования. Управлением блокировок, а также разрешением конфликтов занимается менеджер блокировок. Блокировки могут налагаться как на отдельную строку таблицы, так и на всю таблицу. Кроме того, блокировки могут возникать на уровне страницы или группы страниц.

Проблемы, возникающие в системах управления базами данных, не имеющих механизмов изоляции транзакций:

- Проблема последнего изменения (The lost update problem). При одновременной попытке нескольких транзакций изменить одни и те же данные часть их будет утеряна. Основываясь на первоначальном состоянии данных, несколько транзакций могут начать изменение данных. Однако, т. к. транзакции выполняются независимо друг от друга, то каждая из них не знает об изменениях, которые делают другие транзакции. В процессе сохранения изменений транзакции последовательно сохраняют новые данные. При этом изменения, сделанные ранее закончившимися транзакциями, будут утеряны. В итоге останутся изменения, выполненные последними транзакциями. При этом транзакции не будут знать о том, что их изменения были потеряны.
- Проблема "грязного" чтения (The uncommitted dependency problem). Эта проблема возникает, когда транзакция пытается считать данные, обрабатываемые другой транзакцией, и находящиеся на стадии обработки. При этом данные могут нарушать ограничения целостности и правила, тем самым нарушая общую целостность данных.
- Проблема неповторяемого чтения (The inconsistent analysis problem). Эта проблема связана с многократным чтением транзакцией одних и тех же данных. Между операциями чтения некоторая транзакция может изменить данные, так что при следующем сканировании первая транзакция будет оперировать уже другими данными.
- Проблема чтения фантомов (The phantom read problem). Эта проблема возникает, когда во время выполнения транзакции в таблицу вставляются новые строки, которые могут быть обработаны транзакцией. Предположим, что транзакция осуществляет несколько раз выборку данных из таблицы на основе одного и того же логического условия. Перед началом очередной выборки в таблицу добавляются (или удаляются) строки, удовлетворяющие логическому условию. В результате при сканировании будет обработан иной набор данных, чем при предыдущих сканированиях.

Стандарт, определяющий уровни блокировки:

- ✓ Level 0 — No trashing of data (запрещение "загрязнения" данных). На этом уровне решается проблема последнего изменения, т. е. обеспечивается изолированность изменений данных транзакциями. Одни и те же данные в каждый момент времени может изменять

только одна транзакция. Если какая-то другая транзакция пытается изменить эти же данные, то она должна ожидать завершения работы первой транзакции.

- ✓ Level 1 — No dirty read (запрещение "грязного" чтения). Когда транзакция начинает изменение данных, СУБД должна блокировать ресурсы, чтобы ни одна другая транзакция не смогла прочитать изменяемые данные до тех пор, пока транзакция не будет зафиксирована или отменена. Транзакции, подавшие запрос на чтение данных, должны будут ожидать разблокирования ресурсов.
- ✓ Level 2 — No nonrepeatable read (запрещение неповторяемого чтения). Когда транзакция обращается к каким-то данным, СУБД организовывает блокировку таким образом, чтобы ни одна другая транзакция не могла изменить эти данные. Если транзакция только читает данные, то достаточно запретить только изменение данных. Если же транзакция изменяет данные, то необходимо полностью блокировать ресурсы, запретив также чтение данных.
- ✓ Level 3 — No phantom (запрещение фантомов). Если транзакция производит выборку данных по логическому условию, то никакая другая транзакция не должна вставлять в таблицу или удалять из нее строки, удовлетворяющие этому логическому условию.

Microsoft SQL Server поддерживает все уровни блокирования.

Использование транзакций

Начало транзакции:

BEGIN TRAN[SACTION] [tran_name | @tran_var

[WITH MARK ['description']]]

- tran_name – имя транзакции;
- @tran_ variable – переменная типа char, varchar, nchar, nvarchar;
- WITH MARK ['description'] – метка, используемая при описаниях в transact log.

При создании транзакций изменяется значение системной переменной @@TRANCOUNT, которая содержит номер активной транзакции для текущего соединения.

Для распределенных транзакций:

BEGIN DISTIBUTED TRAN[SACTION] [tran_name | @tran_var]

Создание точек сохранения:

SAVE TRAN[SACTION] { save_point | @save_var}

ROLLBACK TRAN. С помощью данной команды выполняется откат транзакции. Данные, которые изменялись в ходе выполнения транзакции, восстанавливаются в состояние, в котором они были до начала транзакции. Кроме того, с помощью этой команды можно выполнить восстановление точки сохранения.

ROLLBACK [TRAN [SACTION]

[tran_name | @tran_var| save_point | @savet_var]]

или

ROLLBACK [WORK]

Команда ROLLBACK WORK – отменяет последнюю начатую транзакцию.

SQL Server 2000 контролирует работу транзакций на низком уровне, проверяя лишь соответствие данных наложенным ограничениям целостности и правилам. При нарушении целостности система автоматически выполняет откат транзакции.

COMMIT TRAN. Эта команда выполняет фиксирование транзакции.

COMMIT [TRAN[SACTION] [tran_name | @tran_var]]

Если команда commit выполняется без указания имени транзакции, то происходит фиксирование последней транзакции. Если транзакции создаются друг из друга, то происходит образование вложенных транзакций. Пользователь может производить поочередно фиксирование каждой транзакции либо выполнить фиксирование транзакции высокого уровня с указанием имени транзакции.

Неявное определение транзакции

При работе SQL Server в этом режиме система автоматически начинает новую транзакцию после того, как будет завершена предыдущая транзакция. Пользователь не должен явно указывать начало транзакции.

При открытии нового соединения также создается новая транзакция. В отличие от режима автоматического определения транзакций, после выполнения очередной команды не происходит автоматического фиксирования транзакции. Пользователь должен явно выполнить фиксирование или откат транзакции с помощью команд commit tran или rollback tran.

Кроме того, система автоматически выполняет фиксирование транзакции, если пользователь пытается выполнить одно из следующих действий:

- изменение таблицы (alter table);
- создание в базе данных нового объекта;
- удаление данных из таблицы;
- удаление объекта базы данных (drop);
- выборка данных из курсора и т. д.

Для переключения SQL Server в режим неявного начала транзакции существует команда:

SET IMPLICIT TRANSACTION ON

Распределенные транзакции

Распределенные транзакции (distributed transaction) представляют собой совокупность двух или более локальных транзакций, выполняемых

одновременно в различных базах данных. Каждая из локальных транзакций выполняется самостоятельно и не подозревает о существовании других транзакций и о том, что она является частью распределенной транзакции. Необходимо каким-то образом синхронизировать все действия, выполняемые в каждой из локальных транзакций. В качестве такого центрального менеджера в SQL Server 2000 используется компонент MSDTC (Microsoft Distributed Transaction Coordinator). Координатор распределенных транзакций контролирует всю работу по инициализации, откату и фиксирование локальных транзакций. Сам координатор распределенных транзакций реализован в виде службы операционной системы и может запускаться отдельно от SQL Server. В терминологии распределенных транзакций источник данных, к которому происходит обращение, называется менеджером ресурсов. То есть распределенная транзакция лишь передает запрос на выборку этих данных менеджеру ресурсов. Последний, в свою очередь, должен обеспечивать возможность создания, фиксирования и отката транзакций, а также позволять управлять ходом работы полученного запроса с целью синхронизации действий множества менеджеров ресурсов, задейтвованных в распределенной транзакции. Эти требования выполняются с помощью специального модуля менеджера ресурсов, называемого обработчиком распределенных запросов.

Обработчик распределенных запросов принимает от координатора запросы на запуск локальных транзакций, которые являются частью распределенный транзакции. Координатор принимает от обработчиков распределенных транзакций результат выполнения локальных транзакций. На основе полученной информации он принимает решение о том, какие команды должны быть выполнены следующими в каждой локальной транзакции.

Завершение распределенных транзакций

Как и при работе с обычными транзакциями, работа с распределенными транзакциями состоит из двух фаз: начало и завершение. Начало распределенной транзакции не вызывает определенных трудностей — как только будут инициированы все локальные транзакции, координатор может начать выполнение распределенной транзакции.

Завершение транзакции возможно двумя способами — откат и фиксирование. Сложности возникают при фиксации распределенной транзакции.

При использовании стандартных механизмов фиксирования транзакций возможна ситуация, когда часть локальных транзакций окажется принятой, а фиксирование другой части будет невозможно. В этом случае необходимо будет выполнить откат всех локальных транзакций, использованных для выполнения распределенной транзакции. Однако требование устойчивости ACID говорит о том, что система не может вернуться состояние, в котором она была до начала транзакции, после того, как последняя была зафиксирована. То есть требование атомарности ACID будет нарушено.

Поэтому необходим какой-то механизм, позволяющий синхронизировать фиксирование всех локальных транзакций. В качестве такого механизма выступает двухфазный протокол фиксирования транзакции 2PC (Two Phase Commit protocol). Протокол состоит из следующих фаз:

- Фаза подготовки (Prepare Phase). На этом этапе каждому из локальных менеджеров ресурсов посылается сообщение о подготовке к фиксированию транзакции. Менеджеры ресурсов должны выполнить проверку ограничений целостности и соответствие данных наложенным правилам, т. е. гарантировать целостность данных. Кроме того, на этом этапе выполняется физическая запись данных из буферов на диск. Однако в журнале транзакций не отмечается, что транзакция завершена, и блокировки не снимаются.
- Фаза фиксирования (Commit Phase). Информация об успешном (или неуспешном) выполнении фазы подготовки от локальных менеджеров ресурсов поступает координатору распределенных транзакций, который переходит к фазе фиксирования после прохождения фазы подготовки. Если все локальные менеджеры ресурсов успешно выполнили фазу подготовки, то координатор посылает им команду на окончательное фиксирование транзакции. После это считается, что распределенная транзакция успешно зафиксирована.

В случае, когда один или более локальных менеджеров ресурсов по каким-то причинам не смогут зафиксировать локальную транзакцию, координатор посылает всем менеджерам ресурсов, участвующим в распределенной транзакции, команду с требованием отката локальных транзакций. Таким образом, выполняется откат распределенной транзакции.

Программирование в MS SQL SERVER

Компоненты языка Transact-SQL

Переменные

В MS SQL Server , как и во многих других СУБД, разрешается создание временных объектов. Эти объекты предназначены для хранения промежуточных значений приложениями, транзакциями, хранимыми процедурами и т. д. Использование временных объектов удобно тем, что пользователь может не заботиться об удалении ненужных объектов, т. к. SQL Server сделает это автоматически.

В SQL Server имеются два типа временных объектов – переменные и таблицы. Каждая переменная имеет определенный тип данных и используется для хранения одной величины.

Прежде чем начать использовать переменную, ее необходимо объявить, при этом указывается ее имя и тип данных. Для объявления переменной предназначена команда DECLARE:

DECLARE @var type

Типы text, ntext или image не используются для переменных. С помощью одного оператора DECLARE может быть создано несколько локальных переменных. Для этого переменные нужно указывать через запятую:

DECLARE @var1 type, @var2 type

В SQL Server 2000 появилась новая возможность создавать перемененные табличного типа:

DECLARE @var Table (определение столбцов)

Определение таблицы идентично обычному определению CREATE TABLE, за исключением того, что разрешается использование лишь следующих ограничений: PRIMARY KEY, UNIQUE KEY и NULL.

Другим полезным типом переменных является sql_variant. Он может содержать любой из тип данных.

После создания локальная переменная первоначально имеет значение NULL. Для присвоения значения переменной можно использовать различные способы:

- ✓ использование команды SET

```
DECLARE @var1 int, @var2 varchar(5)
SET @var1=10
SET @var2='Str'
```

- ✓ с помощью команды SELECT

```
DECLARE @var1 int, @var2 varchar(5)
SELECT @var1=10, @var2='Str'
```

Можно использовать оператор SELECT для присвоения значений переменным на основании значений полей таблиц, а также с использованием подзапросов:

```
select @var = (select count(job_id) from jobs)
```

или

```
select @var = count(job_id) from jobs
```

Пример табличной переменной:

```
declare @tbl table(job_id int primary key, job_desc varchar(50))
insert into @tbl
select job_id, job_desc from jobs
```

Глобальные переменные

Глобальные переменные обозначаются двойным символом @. Они не могут создаваться пользователем. Предоставляют информацию о текущем статусе SQL Server. Разделяются на 3 группы:

- конфигурированные переменные;
- статические переменные;
- системные переменные.

Временные таблицы

Временные таблицы бывают двух типов: глобальные и локальные. Все временные таблицы всегда создаются в системной базе данных tempdb и автоматически уничтожаются при завершении работы.

Доступ к глобальной временной таблице может быть получен из любого соединения. Глобальные временные таблицы могут быть использованы для обмена данными между различными приложениями.

Для создания глобальной временной таблицы нужно в начале ее имени указать символы ##. Глобальная временная таблица существует до тех пор, пока пользователь явно не уничтожит ее с помощью команды DROP TABLE или пока не будет закрыто соединение, в котором она создавалась. При приостановке сервера все временные таблицы также уничтожаются.
Локальные временные таблицы видны только из того соединения, в котором они были созданы. При закрытии соединения такая таблица автоматически уничтожается. Если локальная временная таблица была образована в хранимой процедуре, то после выхода из этой процедуры такая таблица также уничтожается. В разных соединениях могут создаваться временные локальные таблицы с одинаковыми именами. Локальные временные таблицы часто используются в работе хранимых процедур. Для создания локальной временной таблицы необходимо перед ее именем указать один символ #.

Временные таблицы могут создаваться при помощи команды CREATE TABLE либо с помощью конструкции:

```
SELECT au_id, au_lname, au_fname
INTO #tempTable
FROM authors
```

Выражения

Выражения состоят из комбинаций констант, идентификаторов объектов, операций, функции, подзапросов и т.д. Выражения автоматически вычисляются при выполнении команд Transact-SQL, и вместо них подставляется конкретное значение.

Операнды

- константы и NULL;
- переменные;
- функции: встроенные и пользовательские;
- имена столбцов;
- подзапросы. Обычно используются подзапросы, возвращающие набор строк с одним единственным столбцом;
- функции CASE, ISNULL.

Функция CASE – возвращает значение из списка возможных в зависимости от входного выражения.

```
CASE input_expr
        WHEN when_expr THEN result_expr
                [...]
        [ELSE result_expr]
END

SELECT GETDATE(), CASE DATEPART (dw, GETDATE ( ))
        WHEN 1 THEN 'Понедельник'
        WHEN 2 THEN 'Вторник'
        ELSE 'Другие дни'
END
```

Функция ISNULL – предназначена для замены выражения, равного NULL, другим значением.

```
ISNULL(check_expr, replace_value)
```

Операторы

простейшие;

присваивания;

арифметические;

конкатенации строк;

сравнения;

битовые;

логические:

NOT

AND

OR

BETWEEN

ALL:

```
scalar_expression { = | <> | != | > | >= | !> | < | <= | !< } ALL (subquery)
```

Сравнивает скалярное значение со всеми значениями в наборе subquery. Если условие выполняется для всех значений набора, то возвращается TRUE, иначе – FALSE.

```
IF ( 'Moscow' != ALL(SELECT city FROM authors) )
        PRINT 'Нет автора из Москвы'
ELSE
        PRINT 'Как минимум один автор из Москвы'
```

ANY и SOME:

```
scalar_expression { =|<>|!=|>|>=|!>|<|<=|!< } {SOME|ANY} (subquery)
```

Оператор сравнивает указанное скалярное выражение со всеми значениями набора и возвращает TRUE, если логическое условие выполняется хотя бы для одного значения набора.

```
IF ( 'Moscow' = ANY (SELECT city FROM authors) )
        PRINT 'Как минимум один автор из Москвы'
ELSE
```

```
        PRINT 'Нет автора из Москвы'
```
IN:
test_expr [NOT] IN (subquery | expr [,...n])
С помощью этого оператора можно определить, совпадает ли указанное (или не совпадает, если NOT) выражение хотя бы с одним значением набора.
EXISTS:
EXISTS subquery
Этот оператор возвращает TRUE, если подзапрос возвращает хоть одну строку, и FALSE в противном случае.
```
IF EXISTS (SELECT * FROM authors WHERE city='Moscow')
        PRINT 'Как минимум один автор из Москвы'
ELSE
        PRINT 'Нет автора из Москвы'
```
Оператор EXISTS возвратит TRUE, даже если подзапрос вернет единственную строку, во всех столбцах которой будет NULL.
LIKE:
match_expr [NOT] LIKE pattern
Проверяется соответствие шаблону pattern. Сам шаблон выглядит как строковая константа, в которой могут использоваться специальные символы:
символ % – любое количество любых символов (в том числе и ни одного);
символ _ – один произвольный символ ;
в [ ] указывается набор символов.
```
SELECT city FROM authors WHERE city LIKE '_[^erta]%[^a-ghj]
```

Управление ходом выполнения

BEGIN…END предназначена для объединения двух и более команд в единый блок, воспринимаемый сервером как одно целое.
IF…ELSE – конструкция, реализующая ветвление алгоритмов.
WHILE…CONTINUE
В SQL Server 2000 имеется всего один тип цикла — с предусловием, который и реализуется с помощью команды WHILE.
```
WHILE Boolean_expr
        { sql | statement }
[ BREAK ]
        { sql | statement }
[ CONTINUE ]
```
CONTINUE позволяет начать новый виток цикла, не дожидаясь выполнения всех команд цикла.
```
DECLARE @aa int, @fact bigint
SELECT @aa=1, @fact=l
WHILE @aa>0
        SELECT @fact=@fact*@aa, @aa=@aa-l
SELECT @fact
```

USE – все команды выполняются в контексте какой-то базы данных. Такая база данных называется текущей. При обращении к объектам текущей базы данных не требуется указание ее имени. Команда USE служит для переключения текущей базы данных.
WAITFOR – предназначена для организации паузы.
WAITFOR { DELAY 'time' | TIME 'time' }
При указании ключевого слова TIME выполнение команд будет остановлено до указанного времени (абсолютная точка). Если же требуется приостановить работу лишь на определенный отрезок времени (относительная точка), то необходимо использовать ключевое слово DELAY.
WAITFOR DELAY '23:17'
WAITFOR TIME '18:00'
GO – клиент отправляет серверу команды на выполнение так называемыми пакетами (batch). После работы всех команд пакета сервер возвращает клиенту результат. После этого клиент может отправить следующий пакет. При работе с Query Analyzer в качестве пакета рассматривается набор команд, которые пользователь выделил и запускает на выполнение. Некоторые команды должны запускаться отдельно от остальных. К таким командам можно отнести: CREATE, BACKUP и RESTORE, предназначенные, соответственно, для выполнения создания и восстановления резервных копий баз данных. Локальные объекты (переменные и курсоры) существуют в пределах пакета.

Курсоры в Transact SQL.

В ответ на запросы пользователей SQL Server может возвращать сотни тысяч строк, общим объемом в десятки мегабайт. Передача такого объема данных по сети одновременно многими пользователями может вызвать значительную загрузку, что отрицательно скажется на работе всех пользователей сети. Кроме того, не каждый клиент имеет достаточный объем памяти, чтобы сохранить все полученные данные. Обычно клиент работает лишь с небольшой частью данных, например отдельной строкой, а не со всем набором строк. Часто бывает просто необходимо иметь возможность обратиться к конкретной строке выборки по ее номеру, однако с помощью команды SELECT сделать это довольно трудно.

Механизмом, обеспечивающим хранение результата выборки на сервере и предоставляющим пользователю возможность доступа к любой строке выборки по ее номеру, являются курсоры (cursors). Пользователь может работать в каждый момент времени только с одной строкой, но, перемещая окно, он способен получить доступ к любой строке выборки.

При создании курсора пользователь указывает запрос SELECT, на основе которого создается результирующий набор данных.

Использование курсоров SQL Server дает преимущества, когда обработка данных большей частью происходит на сервере, а не на клиенте. То есть приложение формирует на сервере набор данных, после чего

отправляет запросы, обращающиеся к этому набору данных. В ответ может передаваться только результат обработки данных, размер которого значительно меньше объема всего результирующего набора.

Примечание:

Операции обработки данных с использованием курсоров выполняются заметно медленнее обычных команд обработки данных и не позволяют работать со всеми строками одновременно. Поэтому всегда, когда это возможно, следует избегать применения курсоров и пользоваться командами SELECT, UPDATE, INSERT и DELETE.

Виды курсоров:

По месту хранения и принципам работы курсоры классифицируются следующим образом:

Курсоры Transact-SQL (Transact-SQL Cursors). Создание курсоров этого типа и работа с ними ведется средствами команд Transact-SQL. Эти курсоры создаются на сервере. Курсоры Transact-SQL могут создаваться и работать в транзакциях, хранимых процедурах и триггерах.

Курсоры API сервера (API Server Cursors). Этот тип курсоров используется приложениями при работе с различными механизмами доступа к данным (ODBC, OLE DB, DB Library и т. д.). Работа с этим курсором выполняется средствами API, реализующего все базовые операции с курсорами и, возможно, некоторые дополнительные операции.

Курсоры клиента (Client Cursors). Этот тип курсоров создается непосредственно на клиенте. Сервер обрабатывает отправленный клиентом запрос и возвращает ему результирующий набор. Клиент получает весь результирующий набор и уже сам организует нужные механизмы доступа к данным. Такой подход весьма удобен при работе с небольшим набором данных.

По способу обращения к данным курсоры можно разделить на две следующие категории:

Последовательные (Forward-only). Этот тип курсоров разрешает только последовательное считывание строк, начиная с первой строки и заканчивая последней. После выполнения команды выборки сервер автоматически перемещает указатель на следующую строку. Сам пользователь не может управлять ходом выборки строк.

Прокручиваемые (Scrollable). Курсоры этого типа позволяют обращаться к произвольной строке результирующего набора. В распоряжении пользователей имеются средства как последовательного обращения к строкам курсора, так и средства работы со строками по их порядковому номеру в результирующем наборе. Направление перебора строк при последовательном обращении может быть не только прямым (от первой строки к последней), но и обратным (от последней к первой). Кроме того, можно в произвольном порядке комбинировать команды последовательного и произвольного обращения к строкам курсора.

В SQL Server реализовано четыре типа курсоров, обладающих разными возможностями. Тип курсора указывается в момент его создания и после этого не может быть изменен.

Статические курсоры

При создании статического курсора (static cursor) сервер сохраняет полученные данные в системной базе данных Tempdb, используемой для временного хранения данных. При работе с курсором сервер обращается не к данным исходных таблиц, а к данным, сохраненным в базе данных Tempdb.

Во время создания статического курсора сервер блокирует все данные, удовлетворяющие критериям выборки. На момент формирования курсора ни одна транзакция не может изменить даже одну строку, входящую в курсор. Таким образом и получается моментальный снимок.

В статических курсорах не поддерживается отображение изменений, сделанных в исходных данных. При использовании статических курсоров пользователь может работать с данными, которые уже не существуют в исходных таблицах или были изменены таким образом, что при повторном создании статического курсора они бы уже не вошли в результирующий набор, т. к. не удовлетворяли бы условию горизонтальной фильтрации. Кроме того, в исходные таблицы могут быть добавлены новые строки, которые удовлетворяют условию горизонтальной фильтрации курсора. Помимо того, что изменения в исходных таблицах никак не отображаются в статическом курсоре, пользователь даже не получает никаких сообщений о выполненных изменениях.

Статические курсоры всегда открываются в режиме "только для чтения"(read only.)

Ключевые курсоры

Курсоры, базирующиеся на наборе ключей (keyset driven cursors) или ключевые курсоры, представляют собой набор ссылок на строки, которые удовлетворяют условиям горизонтальной фильтрации, указанным при создании курсора в разделе WHERE запроса SELECT. То есть, в отличие от статического курсора, ключевые курсоры выбирают не всю строку, а лишь ключевые поля. Полученный набор уникальных ссылок называется набором ключей (keyset).

При обращении к строке курсора сервер находит в наборе ключей нужный идентификатор и выбирает данные непосредственно из полного набора данных. Набор ключей формируется только в момент открытия курсора и впоследствии не изменяется. Если в исходные таблицы были добавлены новые строки, удовлетворяющие условиям горизонтальной фильтрации курсора, то такие строки не будут отображены в курсоре. Кроме того, пользователи могут удалить строки, входящие в результирующий набор курсора. Такие строки показываются в курсоре как поврежденные (row missing).

Замечание:

Набор ключей, как и данные статического курсора, хранится в системной базе данных Tempdb.

Последовательные курсоры

Если предполагается произвести лишь одиночное последовательное сканирование всех строк курсора, то рекомендуется использовать специально для этого предназначенные последовательные курсоры (forward-only cursors). Этот тип курсоров поддерживает только обращение к следующей строке курсора и не позволяет вернуться к любой из ранее выбранных строк. При создании последовательного курсора сервер не сохраняет значений ключей или копий строк. Сервер оперирует лишь запросом SELECT и номером текущей строки. При считывании следующей строки сервер выполняет поиск в исходных таблицах следующей после текущей строки, удовлетворяющей условиям горизонтальной фильтрации. Такой подход позволяет отображать в курсоре все изменения, выполненные в результирующем наборе другими пользователями. Кроме того, при работе с последовательными курсорами минимальны требования к объему оперативной памяти.

Динамические курсоры

Динамические курсоры (dynamic cursors) используют тот же принцип обращения к данным, что и последовательные, однако позволяют обращаться к любой произвольной строке результирующего набора. По способу доступа к данным и отображению в результирующем наборе изменений, сделанных другими пользователями, динамические курсоры напоминают ключевые курсоры. Однако между этими типами имеется одно существенное различие.

Природа же динамических курсоров такова, что при каждом обращении к строкам сервер заново обрабатывает ассоциированный с курсором запрос SELECT, обновляя тем самым результирующий набор. Если в исходный набор данных пользователями были внесены изменения, влияющие на список строк результирующего набора, то такие изменения будут автоматически отражены в курсоре.

При использовании динамических курсоров пользователь гарантированно работает со строками, удовлетворяющими условиям горизонтальной фильтрации курсора.

При работе с динамическими курсорами пользователи могут выполнять с помощью них изменения данных в исходных таблицах. Для этого предназначены команды UPDATE, INSERT и DELETE. Во время выполнения выборки или изменений данных с помощью курсора SQL Server блокирует соответствующим образом нужную строку. При чтении данных с помощью курсора другие пользователи не смогут изменить данные, читаемые в курсоре.

Работа с курсорами

После того, как был выбран нужный тип курсора и разработан запрос SELECT, на основе которого будет формироваться результирующий набор данных, можно приступать к реализации и использованию этого курсора.

Весь процесс использования курсора можно разбить на пять этапов:

- **Объявление курсора (declare cursor).** Прежде чем выполнить любую операцию с курсором, его необходимо объявить. Объявление курсора можно сравнить с объявлением переменной или созданием таблицы. Объявление курсора подразумевает указание его имени и запроса SELECT, который будет использоваться для формирования результирующего набора.
- **Открытие курсора (open cursor).** Сразу же после создания курсор не содержит никаких данных. В процессе открытия курсора выполняется ассоциированный с курсором запрос SELECT.
- **Выборка данных из курсора и использование курсора для изменения исходных данных.**
- **Закрытие курсора (close cursor).** После того, как из курсора были получены все интересующие данные и выполнены все необходимые изменения, его можно закрыть. Закрытие курсора подразумевает освобождение выделенного для него пространства в системной базе данных Tempdb.
- **Освобождение курсора (release cursor).** Закрытый курсор может быть удален. Удаление курсора подразумевает удаление из оперативной памяти описания курсора как объекта. Если курсор закрыт, но не удален, он может быть повторно открыт для использования. При этом в него будет помещен новый набор строк.

Объявление курсора

Объявление курсора выполняется с помощью команды DECLARE CURSOR. SQL

```
DECLARE <cursor_name> CURSOR
[LOCAL | GLOBAL]
[ FORWARD_ONLY | SCROLL] [READ_ONLY ][STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
...
FOR <select_statement>
```

Открытие курсора

Открытие курсора включает выполнение ассоциированного с курсором запроса SELECT. Дальнейшие действия с полученным результатом выборки зависят от типа курсора.

Открытие курсора производится с помощью команды OPEN, имеющей следующий синтаксис:

```
OPEN [GLOBAL] <cursor_name> | <cursor_var_name>
```

Имя курсора также можно указать с помощью переменной (параметр cursor_variable_name).

Выборка данных

После открытия курсора пользователь может приступать к выборке или изменению данных.

Для считывания строки данных из курсора используется команда FETCH, имеющая синтаксис:

```
FETCH
[ [NEXT | PRIOR | FIRST | LAST| ABSOLUTE {n | @nvar}
| RELATIVE {n | @nvar} ]
FROM ]
[GLOBAL] <cursor_name>
[INTO @var1 [,...n] ]
```

Непосредственно после слова FETCH указывается строка, которую нужно выбрать. Выбираемую строку можно определить, указав либо ее абсолютную, либо относительную позицию. Затем следует ключевое слово FROM, а далее приводится имя курсора, из которого будут выбираться данные. После ключевого слова INTO указываются имена переменных, в которые будут помещены значения столбцов выбираемой строки курсора.

Замечание:

Прежде чем выбрать данные, сервер выполняет переход на указанную строку. То есть сначала указанная строка становится текущей, и только после этого производится выборка данных. При этом возможен выход за пределы диапазона строк.

Изменение данных

Для выполнения изменений данных с помощью курсора предназначена команда UPDATE. Команда UPDATE производит изменения в одной из исходных таблиц в строке, соответствующей текущей строке курсора. В команде UPDATE нельзя указать, какой строке курсора должно соответствовать выполняемое изменение. Предварительно с помощью команды FETCH нужно сделать текущей ту строку, для которой предлагается выполнить изменения.

За одну операцию изменения данных, как и при работе с представлениями, допускается изменение значений столбцов, расположенных в одной таблице. Если строка курсора строится на основе данных трех таблиц и необходимо изменить все значения строки курсора, то для этого придется выполнить три команды UPDATE.

При работе с курсорами применяется следующий синтаксис команды UPDATE:

```
UPDATE table_name SET {column_name={DEFAULT | NULL | expression}}
[,...n]
WHERE CURRENT OF cursor_name
```

С помощью параметра table_name указывается имя таблицы, в которой необходимо выполнить изменения. Список столбцов, в которых необходимо осуществить изменения, а также значения, которые будут им присвоены, указываются после ключевого слова SET.

Раздел WHERE, в котором обычно определяется логическое условие, ограничивающее диапазон обрабатываемых строк, содержит конструкцию CURRENT OF. Эта конструкция позволяет однозначно идентифицировать

строку исходной таблицы, в которой необходимо выполнить изменения. Допускается изменение любых столбцов таблицы, в том числе и не входящих в результирующий набор курсора.

```
BEGIN TRAN
SELECT au_id, au_lname, au_fname, state FROM authors WHERE state!='CA'
DECLARE curs2 CURSOR LOCAL SCROLL FOR
SELECT au__id, au__lname, au_fname FROM authors WHERE state ! = 'CA'
OPEN curs2
FETCH FROM curs2
UPDATE authors SET state='XY', au_lname=au_lname+' '+LEFT(au_fname,1)+'.'
WHERE CURRENT OF curs2
SELECT * FROM authors WHERE state!='CA'
```

Замечание:

Как видно, пример выполняется в виде транзакции. В конце программы выполняется откат транзакции и всех изменений, произведенных в ходе примера. Это сделано для того, чтобы сохранить данные таблицы authors в первоначальном виде. Чтобы подтвердить изменения, вместо ROLLBACK нужно написать COMMIT TRAN .

Удаление данных

Удаление данных из курсора производится с помощью команды DELETE, имеющей синтаксис:

```
DELETE [FROM] table_name WHERE CURRENT OF cursor_name
```

Закрытие курсора

После того, как все необходимые, операции с использованием курсора были выполнены можно закрыть курсор. Закрытие курсора подразумевает освобождение всех областей памяти (как оперативной, так и в базе данных Tempdb), используемых для хранения данных курсора. Однако при этом не происходит удаления курсора как объекта. Закрытый курсор может быть снова открыт. При этом происходит повторное выполнение запроса SELECT, ассоциированного с курсором.

```
CLOSE {{[GLOBAL] cursor_name} | @cursor_var_name}
```

Освобождение курсора

Операция закрытия курсора приводит к освобождению областей памяти, используемых для хранения результирующего набора курсора. Для полного удаления всей информации, ассоциированной с курсором, включая его имя и код запроса SELECT, необходимо выполнить освобождение курсора.

Если провести аналогию с таблицами, то закрытие курсора сравнимо с полным удалением всех строк таблицы. Однако таблица продолжает существовать как объект базы данных, имеющий определенную структуру и имя. Освобождение курсора сравнивается с удалением самой таблицы.

Для освобождения курсора используется команда DEALLOCATE, имеющая синтаксис:

```
DEALLOCATE {{[GLOBAL] cursor_name} | @cursor_var_name}
```

Дополнительные средства

Количество строк, имеющихся в последнем открытом в соединении курсоре, может быть получено с помощью функции @@CURSOR_ROWS

Если функция @@CURSOR_ROWS возвращает значение 0, то либо в соединении не было открыто ни одного курсора, либо последний открытый курсор не содержит ни одной строки.

В Transact-SQL имеется специальная функция @@FETCH_STATUS, возвращающая результат выполнения последней команды выборки данных из курсора. Если функция возвращает значение 0, то последняя операция была выполнена успешно. Если возвращается значение —1, то это означает, что была предпринята попытка выборки строки, находящейся за пределами результирующего набора.

Замечание:

При работе с несколькими курсорами следует учитывать, что функция @@FETCH_STATUS возвращает результат выполнения последней команды FETCH.

В качестве примера рассмотрим использование функции @@FETCH_STATUS для обнаружения границы курсора:

```
DECLARE curs3 CURSOR LOCAL STATIC FOR SELECT au_lname FROM authors
OPEN curs3
DECLARE @LName char(30), @Rs varchar(40)
SET @Rs=''
FETCH FIRST FROM curs3 INTO @LName
WHILE @@FETCH_STATUS=0
BEGIN
SET @Rs=@Rs+LEFT(@LName,l) FETCH FROM curs3 INTO @LNaroe
END
SELECT @Rs
CLOSE curs3
DEALLOCATE curs3
```

Хранимые процедуры.

Выполнение операций INSERT, SELECT, UPDATE и DELETE не вызывает особых затруднений. Пользователь может написать запрос непосредственно в окне Query Analyzer и выполнить его. Тем не менее, основными клиентами баз данных являются приложения. Эти приложения часто реализуют сложные операции.

Использование хранимых процедур позволяет снизить стоимость сопровождения системы и дает возможность избавиться от необходимости изменять клиентские приложения.

Системные хранимые процедуры

Информация обо всех аспектах работы сервера хранится в системных таблицах. Самая важная из них, база данных Master, содержит информацию о

параметрах работы ядра SQL Server, обладает данными об учетных записях пользователей, сведениями о созданных на сервере базах данных и т. д.

Изменение значений в таблицах базы данных Master приводит к изменению тех или иных аспектов работы сервера. Пользователи могут изменять значения в этих таблицах напрямую с помощью команд UPDATE, INSERT, SELECT и DELETE. Однако такие изменения требуют специальных знаний. Чтобы избавить пользователя от рутины и обеспечить высокую безопасность выполняемых изменений, в SQL Server 2000 используются специальные системные хранимые процедуры (system stored procedure).

Расширенные хранимые процедуры

Вызов этих процедур выполняется подобно другим хранимым процедурам, но они представляют собой динамически подключаемые библиотеки (файлы dll).

В SQL Server 2000 имеется набор встроенных расширенных хранимых процедур (например, для отправки сообщений по электронной почте). Пользователи могут создавать новые хранимые процедуры с помощью любого языка, допускающего создание библиотек. При этом они должны использовать интерфейс программирования SQL Server Open Data Services API.

Создание хранимой процедуры

Создание хранимой процедуры производится с помощью команды CREATE PROCEDURE, имеющей следующий синтаксис:

CREATE PROCEDURE] procedure_name [;number]

[{@parameter data_type} [VARYING] [= default] [OUTPUT]]

[,..n]

AS sql_statement [...n]

- *number* предназначен для создания группы одноименных процедур. Такой подход позволяет управлять несколькими процедурами как одним целым. Номера процедур не обязательно должны следовать друг за другом. При работе с группой процедур с помощью Enterprise Manager все процедуры будут сохранены под одним именем.
- *VARYING* ключевое слово, используется только для параметров, имеющих тип данных cursor. Использование VARYING необходимо, когда в хранимой процедуре происходит создание динамического курсора и полученные данные нужно возвратить из процедуры для дальнейшего использования.
- *OUTPUT* ключевое слово, приводится, когда необходимо возвратить измененное значение параметра хранимой процедуры. При указании этого параметра аргумент процедуры

может быть использован не только как выходной, но и как входной.

Использование параметров

```
CREATE PROCEDURE get_list_authors;6
@state char(2) = 'UT'
AS SELECT au_id, au_lname, state, phone
FROM pubs..authors
WHERE state=@state
EXEC get_list_authors;6 'CA'
```

Возвращение значений из процедуры

```
CREATE PROCEDURE get_author_info
@phone chart (20)=NULL OUTPUT,
@au_id char(l0)=NULL OUTPUT,
@LName char(40)=NULL OUTPUT,
@FName char (20) =NULL OUTPUT
AS IF @phone IS NOT NULL
SELECT @au_id=au_id, @LName=au_lname, @FName=au_fname
FROM authors WHERE phone=@phone ELSE
IF @au_id IS NOT NULL
 SELECT @phone=phone FROM authors WHERE au_id=@au_id
ELSE
 SELECT @phone=phone FROM authors WHERE au_lname=@LName AND au_fname=@FName
```

Запуск процедуры:

```
DECLARE @phone char(20), @ID char(10), @LastName char(40), @FirstName char(20)
SET @phone='1234567'
EXEC get_author_info @phone, @ID OUTPUT, @LastName OUTPUT, @FirstName OUTPUT
SELECT @ID, @LastName, FirstName
```

Использование кода завершения

Когда встречается команда RETURN, выполнение хранимой процедуры прерывается и управление передается в то место, откуда произошел вызов процедуры. В команде RETURN допускается указание кода завершения. Если значение не указано явно, то будет возвращено значение 0.

Для получения кода завершения используется конструкция:

EXEC <variable> = <stored procedures>

Изменение хранимых процедур

При выполнении изменений средствами Transact-SQL нет возможности модифицировать часть команд, оставив другие без изменения. Команда ALTER PROCEDURE выполняет исправление всего кода процедуры.

Удаление хранимых процедур

Для удаления хранимых процедур используется следующая команда:

DROP PROCEDURE

Триггеры.

Часто разработчикам приходится реализовывать сложные алгоритмы поддержки целостности данных. Ранее рассматривалось использование ограничений целостности Primary Key, Foreign Key, Unique, правил, умолчаний и т. д. Однако все же их часто бывает недостаточно. Например, с помощью упомянутых механизмов нельзя разрешить изменение данных в том случае, если в одном из столбцов находится определенное значение.

Триггеры (triggers) SQL Server представляют собой набор команд Transact-SQL, выполняемых автоматически при осуществлении тех или иных модификаций данных в таблице. Каждый триггер связан с конкретной таблицей и запускается сервером автоматически каждый раз, когда пользователи пытаются произвести вставку, изменение или удаление данных. Триггер получает всю информацию о выполняемых пользователем изменениях в таблице. Разработчик реализовывает в триггере необходимые проверки и изменения данных в других таблицах базы данных.

Когда пользователь начинает изменение данных, сервер автоматически начинает транзакцию, в которой и выполняется триггер. В теле транзакции разработчик может реализовывать произвольные алгоритмы, которые могут выполнять как проверку, так и изменения данных. В конце концов, работа триггера сводится либо к фиксации, либо к откату транзакции.

В SQL Server триггеры могут быть созданы не только для таблиц, но и для представлений.

Существует несколько типов триггеров, каждый из которых реагирует на определенный тип изменений данных. В SQL Server существует три типа триггеров, классифицирующихся по названию команд, на которые они реагируют:

- **INSERT TRIGGER**. Триггеры этого типа вызываются при попытке пользователя добавить данные в таблицу, например, с помощью команды INSERT.
- **UPDATE TRIGGER**. Этот тип триггеров выполняется при изменении данных с помощью команды UPDATE.
- **DELETE TRIGGER**. Этот тип триггеров выполняется при удалении данных с помощью команды DELETE.

Помимо классификации триггеров по типу операции, на которую они реагируют, они также различаются по поведению:

- **AFTER**. Это стандартный тип триггеров, активно используемый в предыдущих версиях SQL Server. По умолчанию триггер SQL Server имеет именно этот тип.
- **INSTEAD OF**. Триггеры этого типа выполняются взамен пользовательских действий. То есть запрос пользователя не выполняется, а вносятся лишь изменения, осуществляемые в теле триггера.

Для одной таблицы допускается создавать множество триггеров AFTER и только один триггер INSTEAD OF для каждой операции.
Однако выполнение триггеров является довольно "тяжелой" задачей и по возможности нужно избегать их использования. Эффективнее решить задачу на уровне целостности.

Сервер на все время выполнения триггера удерживает блокировки на данные, к которым обращается этот триггер.

Создание триггера

При написании триггера следует тщательно продумать алгоритм выполнения изменений и проверок данных, а также согласовать последовательность этих действий с другими триггерами и хранимыми процедурами. Дело в том, что триггеры способны длительное время блокировать значительную часть ресурсов, и неправильно написанные триггеры могут создавать мертвые блокировки (deathlocks).

Если к одним и тем же данным обращается множество триггеров и хранимых процедур, то во всех них следует использовать одинаковую последовательность доступа к данным.
Для создания триггера используется команда CREATE TRIGGER, имеющая синтаксис:

```
CREATE TRIGGER trigger_name
ON (table | view) [WITH ENCRYPTION]
{ { FOR [{ AFTER | INSTEAD OF }]
( [DELETE] [,] [INSERT] [,] [UPDATE] }
[ WITH APPEND ] [ NOT FOR REPLICATION ] AS
[ ( IF UPDATE (column)
[ ( AND I OR } UPDATE (column) ]
[ ...n ] | IF (COLUMNS_UPDATED()
{ bitwise_operator } updated_bitmask )
( comparison_operator } column_bitmask [ ...n ] } ]
sql_statement [ ...n ] }
}
```

Создание триггера должно выполняться как отдельная команда, т.е. триггер не может быть создан внутри хранимой процедуры. Как видно из синтаксиса, триггер может быть создан только в текущей базе данных.

Однако в его теле могут производиться обращения к таблицам различных баз данных, возможно расположенных на различных серверах.

Триггеры не могут быть созданы для системных таблиц. Кроме того не допускается создание триггеров в системных баз данных.

- IF UPDATE (column): с помощью этой конструкции можно разрешить выполнение триггера только при осуществлении изменений в определенном столбце таблицы. Имя нужного столбца указывается с помощью параметра column. Однако использование этой возможности допускается только для триггеров типа UPDATE и INSERT. Если необходимо разрешить вызов триггера при изменении более чем одного столбца, то можно указать имена дополнительных столбов с помощью параметра UPDATE (column), объединив множество таких параметров с помощью логических операторов OR или AND.
- IF (COLUMNS_UPDATED()...): эта конструкция является вторым способом выполнения проверки изменений определенных столбцов. Функция COLUMNS_UPDATED() возвращает двоичное значение, каждый бит которого соответствует одному столбцу. Самый правый бит соответствует самому левому столбцу таблицы. Вся же конструкция IF (COLUMNS_UPDATED() ...) позволяет выполнять триггер только тогда, когда изменяются определенные столбцы.

Модификация триггера

Для изменения параметров и кода триггера предназначена команда ALTER TRIGGER.

При выполнении изменений следует учитывать, что нужно указывать все необходимые опции.

Удаление триггера

Для этого используется команда Transact-SQL:

DROP TRIGGER (trigger) [,...n].

Программирование триггеров

Хорошим тоном является написание триггеров, которые не обращаются ни к каким объектам базы данных кроме таблицы, с которой они ассоциированы. Если триггер обращается к внешней таблице, представлению или хранимой процедуре, то изменение этих объектов может привести к нарушению функционирования триггера.

Также не рекомендуется написание триггеров, возвращающих данные с помощью команды SELECT.

Для отслеживания изменений, производимых пользователем, сервер автоматически создает при вызове триггера две специальных таблицы — inserted и deleted.

Рассмотрим, какие данные содержатся в этих таблицах при выполнении различных операций:

- INSERT. При вызове триггера этого типа таблица inserted содержит список строк, вставляемых пользователем. При успешном завершении триггера (фиксации транзакции) все строки из таблицы inserted переносятся в пользовательскую таблицу базы данных.
- DELETE. Для триггеров этого типа в таблице deleted приводится список всех строк, которые будут удалены.
- UPDATE. При выполнении триггеров этого типа данные имеются как в таблице inserted, так и в таблице deleted. Последняя из них — список всех строк таблицы, которые пытается изменить пользователь. В таблице inserted указываются строки, которые будут внесены в таблицу вместо соответствующих строк таблицы deleted.

Таблицы inserted и deleted не существуют физически, а представляют логические структуры, создаваемые сервером индивидуально для каждого триггера. В эти таблицы запрещено вносить любые изменения.

Триггер может анализировать содержимое таблиц inserted и deleted для принятия решения о фиксировании или отмене транзакции и внесении различных изменений в таблицу базах данных.

Функции SQL Server.

Встроенные функции

- функции просмотра конфигурации;

Большинство из них являются недетерминированными, т. к. могут возвращать различные результаты при каждом новом вызове. Простейшим примером такой функции является @@DBTS (возвращает текущее значение счетчика TIMESTAMP для текущей базы данных), возвращаемое значение которой постоянно меняется с течением времени. Отметим, что функции не изменяют никаких настроек сервера, а лишь позволяют просмотреть их.

- функции для работы с курсорами;
- функции работы с датой и временем;

Эти функции оперируют типами datetime и smalldatetime.

- математические функции;
- функции метаданных;

Предназначены для получения информации о различных объектов SQL Server .

- функции подсистемы безопасности;

Используются для получения информации о пользователях, учетных записях, о членстве их в фиксированных и пользовательских ролях базы данных и др.

- строковые функции;
- системные функции;
- статистические функции;

Возвращают информацию о работе сервера – об операциях чтения диска, количестве переданной по сети информации, затратах времени и т.д.

- функции для работы с типами данных image, text и ntext.

Многие системные функции начинаются с символов @@. Полноценная функция должна работать с параметрами, значения которых указываются в скобках. Некоторые же функции Transact-SQL не используют скобки при их вызове. Такие функции скорее можно назвать глобальными переменными.

Функции, определяемые пользователем

Пользовательские функции являются обычными объектами базы данных, такими же, как и хранимые процедуры, триггеры, представления и т. д. Каждый из пользователей, имеющий соответствующие права доступа, может создать произвольную функцию. Таким образом, определяемая пользователем функция, в отличие от встроенной функции, имеет своего владельца, который может предоставлять другим пользователям право на ее вызов.

В SQL Server имеется несколько типов определяемых пользователем функций:

- **Scalar**. Функция Scalar может содержать множество команд, объединяемых конструкцией BEGIN…END в одно целое, и возвращает скалярное значение любого из типов данных, поддерживаемого SQL Server, за исключением timestamp, text, ntext, image, table и cursor.
- **Inline**. Функции этого типа всегда возвращают значения типа данных table, представляющие собой сложный набор данных. Кроме того, функция Inline может состоять всего из одной команды SELECT. Их особенностью является то, что код функции при выполнении программы вставляется непосредственно в исполняемый набор команд, т.е. происходит не вызов функции, а встраивание.
- **Multi-Scalar.** Multi-Scalar возвращают значение типа данных table. Однако в отличие от первых, функции рассматриваемого типа могут состоять более чем из одной команды, что дает возможность использовать в теле функции транзакции, курсоры, вызывать хранимые процедуры и т. д.

Замечание:

Функции, возвращающие значения типа данных table (Inline и Multi-Scalar), могут вызываться непосредственно в разделе from при работе с запросами SELECT, INSERT, DELETE и UPDATE.

Входные параметры могут иметь любой из поддерживаемых SQL Server 2000 типов данных за исключением image, text, ntext, cursor и table. Входные параметры определяемой пользователем функции, в отличие от параметров хранимой процедуры, нельзя применять для возврата значений, т. е. указание ключевого слова OUTPUT не допускается. Тем не менее, можно определять значения по умолчанию.

В отношении пользовательских функций существует множество ограничений. Если от хранимой процедуры не требуется указывать

возвращаемое значение (код возврата), то функция должна в обязательном порядке явно вернуть какое-то значение. При программировании функций не разрешается использование команд, которые могут вернуть значения непосредственно в соединение, а не как результат вычисления функции. Таким образом, в теле функции не допускается вызов команды PRINT, SELECT, FETCH. Однако применение этой команды для помещения данных из курсора в локальные переменные допускается.

В функции не разрешается изменений данных в таблицах базы данных, создание, модификация и удаление объектов базы данных. Тем не менее, разрешается использование команд UPDATE, INSERT и DELETE, изменяющих данные в переменной table, которая является возвращаемым функцией значением.

Создание пользовательских функций

Создание пользовательских функций выполняется с помощью команды CREATE FUNCTION.

Для создания функций типа Scalar предназначен следующий синтаксис команды CREATE FUNCTION:

```
CREATE FUNCTION [owner_name.]function_name
([{@param scalar_data_type [= default ]} [,...n]])
RETURNS scalar_data_type
[WITH <function_option> [,...n]] [AS]
BEGIN
    function_body
    RETURN scalar_expr
END
```

Рассмотрим назначение и использование параметров команды:

- [owner_name.]function_name

Данный параметр предназначен для указания имени (function_name), а при необходимости дополнительно и владельца (owner_name) создаваемой функции. Указание имени базы данных или сервера не разрешается, т. е. функция может быть создана только в текущей базе данных.

- @param scalar_data_type [=default]

Скалярный тип данных. То есть, как уже было сказано, не допускается применение типов данных timestamp, text, ntext, image, table и cursor.

- RETURNS scalar_data_type

Ключевое слово RETURNS свидетельствует, что далее следует имя типа данных, значение которого будет возвращать функция.

Ключевое слово ENCRYPTION предписывает серверу выполнить шифрование кода команды CREATE FUNCTION, с помощью которой была создана функция. Это позволяет скрыть от пользователей принципы работы

функции. Если же указывается ключевое слово SCHEMABINDING, то сервер выполняет связывание создаваемой функции со всеми объектами, на которые ссылаются команды функции. Это позволяет избежать модификации этих объектов, которая может привести к нарушению работы функции. Однако в этом случае для обращения к объектам необходимо использовать имена, включающие имя владельца.

Для создания функций Inline используется следующий вариант команды CREATE FUNCTION:

```
CREATE FUNCTION [owner_name.]function_name
([{@param scalar_data_type [= default ]} [,..n]])
RETURNS TABLE
[WITH <function_option> [,...n]]
[AS]
RETURN [ ( ] select_stmt [ ) ]
```

После ключевого слова AS не допускается указание конструкции BEGIN...END, а разрешается лишь использование запроса SELECT, с помощью которого будет формироваться набор данных, возвращаемый функцией.

Для создания функций этого типа применяется следующий вариант команды CREATE FUNCTION:

```
CREATE FUNCTION [owner_name.]function_name
( [ { @param scalar_data_type [ = default ] }
[,...n]])
RETURNS @return_var TABLE <table_definition>
[WITH <function_option> [,...n]]
[AS]
BEGIN
    function_body
    RETURN
END
```

В отличие от функций Inline и Scalar здесь необходимо указать имя локальной переменной, содержимое которой будет возвращено как результат выполнения функции. Необходимо явно определить набор столбцов, которые будут применяться для хранения данных. Дополнительно можно определить индексы, ограничения целостности и т. д.

Завершение работы функции Multi-statement происходит при выполнении команды return. Однако не требуется указание значения, которое будет возвращено как результат выполнения функции. Всегда возвращается

содержимое переменной типа table, указанной после ключевого слова RETURNS.

Изменение функций

Можно удалить функцию и повторно ее создать с внесенными изменениями. Однако минусом такого подхода является потеря всех разрешений, выданных пользователям базы данных на доступ к этой функции. Более того, если с функцией был связан некоторый объект базы данных (т. е. ссылающийся на функцию и созданный с указанием опции schemabinding), то удалить ее не удастся.

Синтаксис команды ALTER FUNCTION отличается для каждого типа функций.

Для просмотра функции можно использовать системную хранимую процедуру sp_helptext, позволяющая получить код команды CREATE FUNCTION, с помощью которого была создана функция.

Замечание:

Код функции не удастся просмотреть, если она была создана с указанием параметра ENCRYPTION.

Удаление функций

Для удаления функции служит команда DROP FUNCTION, имеющая синтаксис:

```
DROP FUNCTION {[owner_name.]function_name} [,...n]
```

ПРАКТИЧЕСКИЙ РАЗДЕЛ

Лабораторная работа №1

СОЗДАНИЕ БАЗЫ ДАННЫХ (БД) В MICROSOFT SQL SERVER

Цель работы –с помощью операторов языка Transact SQL научиться создавать базу данных и таблицы, принадлежащие указанной базе данных.

Содержание работы:

1. Ознакомиться с назначением и синтаксисом команд USE, CREATE DATABASE, CREATE TABLE.
2. По выданным вариантам спроектировать и создать персональную базу данных.
3. По итогам работы подготовить отчет стандартной формы.

Пояснения к выполнению работы.

По описанию предметной области, указанном в задании**,** создать инфологическую модель базы данных: определить таблицы, атрибуты, их свойства, связи между таблицами.

Запустить **SQL Server Management Studio.** Для написания программного кода в **SQL Server Management Studio** нужно нажать кнопку «Создать запрос» («New Query») на панели инструментов «Стандартная» («Standart»).

Создать новую базу данных с именем **Фамилия студента_номер варианта** с помощью команды CREATE DATABASE имя_ базы данных.

В этой базе данных создать таблицы с помощью команды CREATE TABLE имя_таблицы, предварительно указав USE имя_используемой _базы данных. При создании таблиц задать декларативные ограничения целостности: PRIMARY KEY, FOREIGN KEY...REFERENCES, NOT NULL, CHECK, DEFAULT.

В разделе диаграмм создать новую диаграмму, в которую добавить все созданные таблицы, проверить связи между таблицами.

Пример выполнения работы.

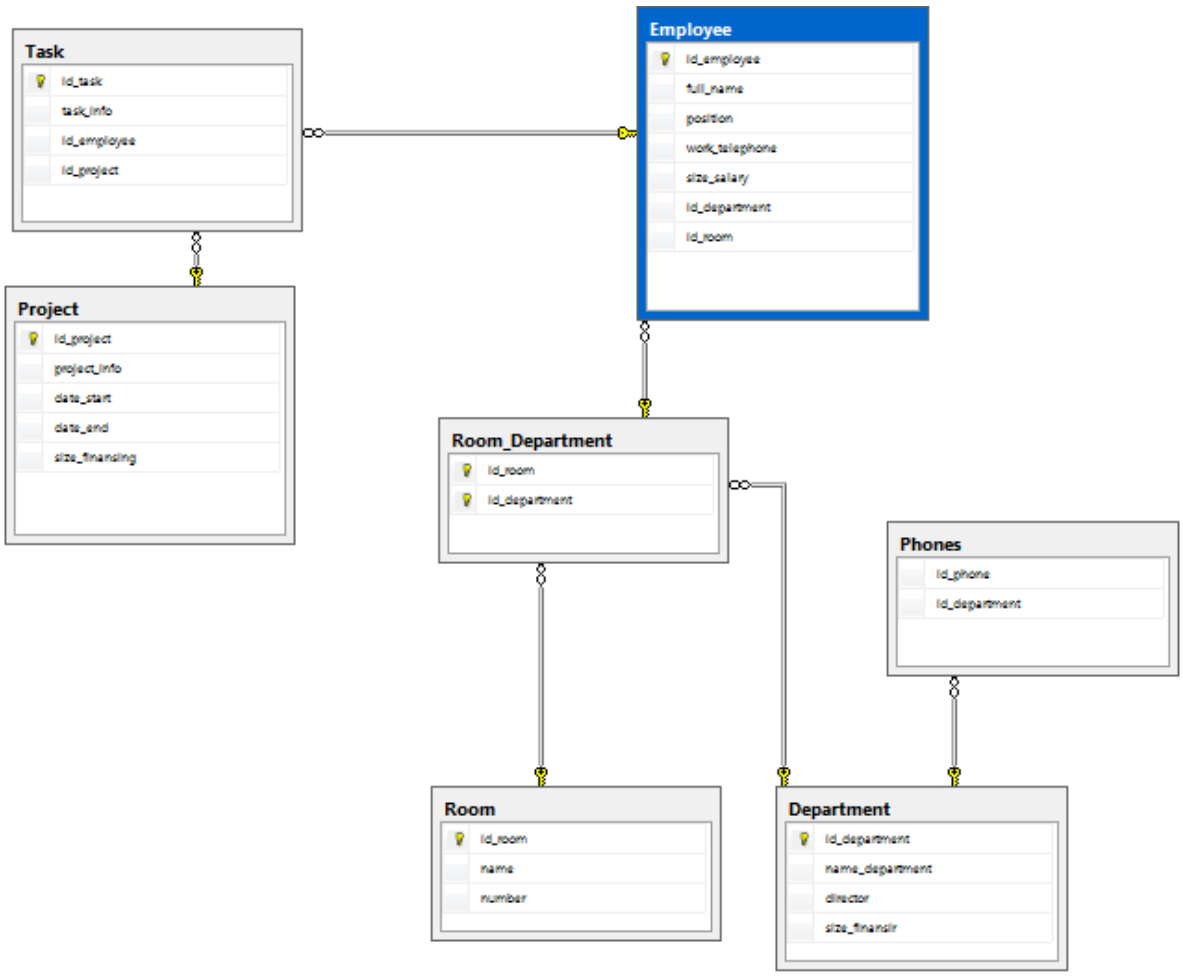
Задача – управление проектами.

В компании несколько отделов. В каждом отделе есть некоторое количество сотрудников, занятых в нескольких проектах и размещающихся в нескольких офисах. Каждый сотрудник имеет план работы, т.е. несколько заданий, которые он должен выполнить. Для каждого такого задания существует ведомость, содержащая перечень денежных сумм, полученных сотрудником за выполнение этого задания. В каждом офисе установлено несколько телефонов.

Для каждого проекта : номер проекта (уникальный) и его бюджет.

Для каждого офиса : номер офиса (уникальный), площадь в квадратных футах,
номера всех установленных в нем телефонов.

Схема базы данных.



Создание базы данных.

```
CREATE DATABASE Department
ON PRIMARY (NAME = Department,
            filename = 'd:\GaLLeRy\Styding\Programming\Data Base\Files\Department.mdf',
            SIZE = 10 MB,
            MAXSIZE = 1000,
            FILEGROWTH = 5)
LOG ON (NAME = Department_log,
        filename = 'd:\GaLLeRy\Styding\Programming\Data Base\Files\Department.ldf',
        SIZE = 10 MB,
        MAXSIZE = 1000,
        FILEGROWTH = 5)
```

Создание таблиц и заполнение их данными.

```
CREATE TABLE Department
    (
        [id_department] [int] NOT NULL,
        [name_department] [nvarchar](50) NOT NULL,
        [director] [nvarchar](50) NOT NULL,
        [size_finansir] [int] NULL,
        primary key(id_department)
    )
INSERT INTO Department
VALUES ('100101','Финансовый отдел','Давыдова Анна Викторовна','157000');

INSERT INTO Department
VALUES ('100102','Юридический отдел','Кутузов Тимофей Борисович','118000');

INSERT INTO Department
VALUES ('100103','Маркетинговый отдел','Иванов Роман Александрович','118000');

CREATE TABLE Employee
    (
        [id_employee] [int] NOT NULL,
        [full_name] [nvarchar](50) NOT NULL,
        [position] [nvarchar](50) NOT NULL,
        [work_telephone] [int] NULL,
        [size_salary] [nvarchar](50) NOT NULL,
        [id_department] [int] NOT NULL,
        [id_room] [int] NOT NULL,
        primary key(id_employee)
    )

INSERT INTO Employee
VALUES ('300101','Борисов Алексей Евгеньевич','Юрист','8469774','20100','100102','200102');

INSERT INTO Employee
VALUES ('300102','Степанова Елена Юрьевна','Юрист','8197774','17600','100102','200101');

INSERT INTO Employee
VALUES ('300103','Щербаков Алексей Николаевич','Юрист','3499774','21400','100102','200102');

INSERT INTO Employee
VALUES ('300104','Денисов Максим Романович','Маркетолог','7954458','25700','100103','200103');

INSERT INTO Employee
VALUES ('300105','Евдокимов Иван Александрович','Маркетолог','7912458','27500','100103','200103');
```

```
CREATE TABLE Phones
    (
        [id_phone] [int] NOT NULL,
        [id_department] [int] NOT NULL
    )

INSERT INTO Phones
VALUES ('1578642','100101');

INSERT INTO Phones
VALUES ('7849135','100101');

INSERT INTO Phones
VALUES ('1548642','100102');

INSERT INTO Phones
VALUES ('1748642','100103');
```

```
CREATE TABLE Project
    (
        [id_project] [int] NOT NULL,
        [project_info] [nvarchar](50) NOT NULL,
        [date_start] [date] NOT NULL,
        [date_end] [date] NOT NULL,
        [size_finansing] [int] NOT NULL,
        primary key(id_project)
    )

INSERT INTO Project
VALUES ('400101','Проект номер 1','2012-01-02','2014-11-03','30000');

INSERT INTO Project
VALUES ('400102','Проект номер 2','2013-07-09','2014-04-10','13000');

INSERT INTO Project
VALUES ('400103','Проект номер 3','2014.06.01','2015.12.10','18000');
```

```
CREATE TABLE Room
    (
        [id_room] [int] NOT NULL,
        [name] [int],
        [number] [nvarchar](50) NOT NULL,
        primary key(id_room)
    )

INSERT INTO Room
VALUES ('200101','','110');

INSERT INTO Room
VALUES ('200102','','115');

INSERT INTO Room
VALUES ('200103','','120');
```

```
CREATE TABLE Room_Department
    (
        [id_room] [int] NOT NULL,
        [id_department] [int] NOT NULL,
        primary key(id_room,id_department)
    )

INSERT INTO Room_Department
VALUES ('200101','100101');

INSERT INTO Room_Department
VALUES ('200102','100101');

INSERT INTO Room_Department
VALUES ('200102','100102');

INSERT INTO Room_Department
VALUES ('200103','100103');
CREATE TABLE Task
    (
        [id_task] [int] NOT NULL,
        [task_info] [nvarchar](250) NULL,
        [id_employee] [int] NOT NULL,
        [id_project] [int] NOT NULL,
        primary key(id_task)
    )

INSERT INTO Task
VALUES ('500101','Задание 1','300101','400102');

INSERT INTO Task
VALUES ('500102','Задание 1','300101','400102');

INSERT INTO Task
VALUES ('500103','Задание 2','300105','400101');

INSERT INTO Task
VALUES ('500104','Задание 3','300104','400103');
```

Варианты заданий к лабораторной работе №1

Вариант1.

Задача – организация учебного процесса в вузе:

* Студенты: паспортные данные, адрес, дата зачисления, номер приказа, факультет, группа, является ли старостой, кафедра (специализация), изучаемые (изученные) предметы, оценки, задолженности, стипендия.

* Учебные курсы: название, факультет(ы), групп(ы), кафедра, семестр(ы), форма отчётности, число часов.

* Преподаватели: паспортные данные, адрес, телефон, фотография, кафедра, должность, учёная степень, начальник (зав. кафедрой), предмет(ы), число ставок, зарплата.

Вариант2.

Учет и выдача книг в библиотеке вуза:

* Книги: авторы, название, раздел УДК, раздел (техническая, общественно-политическая и т.п.), место и год издания, издательство, количество страниц, иллюстрированность, цена, дата покупки, номер сопроводительного документа (чек, счёт/накладная), вид издания (книги, учебники, брошюры, периодические издания), инвентарный номер (есть только для книг и некоторых учебников), длительность использования читателями (год, две недели, день), электронная версия книги или ее реферата (отсканированный текст).
* Читатели: номер читательского билета, ФИО, год рождения, адрес, дата записи, вид (студент, аспирант, преподаватель, сотрудник), курс, номер группы, названия взятых книг и даты их выдачи.

Вариант3.

Отдел кадров некоторой компании.
* Сотрудники: ФИО, паспортные данные, фотография, дом. и моб. телефоны, отдел, комната, раб. телефоны (в т.ч. местный), подчинённые сотрудники, должность, тип(ы) работы, задание(я), проект(ы), размер зарплаты, форма зарплаты (почасовая, фиксированная).
* Отделы: название, комната, телефон(ы), начальник, размер финансирования, число сотрудников.
* Проекты: название, дата начала, дата окончания, размер финансирования, тип финансирования (периодический, разовый), задачи и их исполнители, структура затрат и статьи расходов.

Вариант4.

Отдел поставок некоторого предприятия:
* Поставщики: название компании, ФИО контактного лица, расчётный счёт в банке, телефон, факс, поставляемое оборудование (материалы), даты поставок (по договорам и реальные), метод и стоимость доставки.

* Сырьё: тип, марка, минимальный запас на складе, время задержки, цена, продукты, при производстве которых используется, потребляемые объемы (необходимый, реальный, на единицу продукции).

Вариант5.

Пункт проката видеозаписей (внутренний учет).
* Видеокассеты: идентификационный номер видеокассеты, тип видеокассет, дата его создания, компания-поставщик, число штук данного типа (общее, в магазине, выдано в настоящее время, выдано всего, выдано в среднем за месяц), общая длительность записей; записи видеокассет: название, длительность, категория, год выпуска и производитель (оригинала).
* Клиенты: ФИО, паспортные данные, адрес, телефон; заказы, т.е. взятые видеокассеты (сейчас и в прошлом): номер, дата выдачи, дата возвращения, общая стоимость заказа.

Вариант6.

Пункт проката видеозаписей (информация для клиентов).
* Видеокассеты: краткое описание, внешний вид (этикетка), марка (пустой) видеокассеты, цена за единицу прокатного времени (например: 1 день, 3 дня, неделя), есть ли в наличии, общая длительность записей; записи на видеокассете: название, длительность, жанр (категория), тема, год и страна выпуска (оригинала), кинокомпания, описание, актеры, режиссер.
* Заказы: идентификационные номера и названия выданных видеокассет, дата выдачи, дата возвращения (продления), общая стоимость заказа, возвращены ли кассеты заказа.

Вариант7.

Кинотеатры (информация для зрителей).
* Фильмы: название, описание, жанр (категория), длительность, популярность (рейтинг, число проданных билетов в России и в мире), показывается ли сейчас (сегодня, на текущей неделе), в каких кинотеатрах показывается, цены на билеты (в т.ч. средние).

* Кинотеатры: название, адрес, схема проезда, описание, число мест (в разных залах, если их несколько), акустическая система, широкоэкранность, фильмы и цены на них: детские и взрослые билеты в зависимости от сеанса (дневной, вечерний и т.п.) и от категории мест (передние, задние и т.п.); сеансы показа фильмов (дата и время начала).

Вариант8.

Ресторан (информация для посетителей).

* Меню: дневное или вечернее, список блюд по категориям.
* Блюда: цена, название, вид кухни, категории (первое, второе и т.п.; мясное, рыбное, салат и т.п.), является ли вегетарианским, компоненты блюда, время приготовления, есть ли в наличии.
* Компоненты блюд: тип (гарнир, соус, мясо и т.п.), калорийность, цена, рецепт, время приготовления, есть ли в наличии, ингредиенты (продукты) и их расходы на порцию.

Вариант9.

Задача- информационная поддержка деятельности склада.

База данных должна содержать информацию о наименовании товара, его поставщике, количестве, цене товара, конечном сроке реализации, сроке хранения на складе. Торговый склад производит уценку хранящейся продукции. Если продукция хранится на складе дольше 10 месяцев, то она уценивается в 2 раза, а если срок хранения превысил 6 месяцев, но не достиг 10, то в 1,5 раза. Ведомость уценки товаров должна содержать информацию: наименование товара, количество товара(шт.), цена товара до уценки, срок хранения товара, цена товара после уценки, общая стоимость товаров после уценки.

Вариант10.

Задача – информационная поддержка деятельности адвокатской конторы. БД должна осуществлять:

ведение списка адвокатов;

ведение списка клиентов;

ведение архива законченных дел.

Необходимо предусмотреть:

получение списка текущих клиентов для конкретного адвоката;

определение эффективности защиты (максимальный срок минус полученный срок) с учётом оправданий, условных сроков и штрафов;

определение неэффективности защиты (полученный срок минус минимальный срок);

подсчёт суммы гонораров (по отдельных делам) в текущем году;

получение для конкретного адвоката списка текущих клиентов, которых он защищал ранее (из архива, с указанием полученных сроков и статей).

Вариант11.

Задача – информационная поддержка деятельности гостиницы.

БД должна осуществлять:

ведение списка постояльцев;

учёт забронированных мест;

ведение архива выбывших постояльцев за последний год.

Необходимо предусмотреть:

получение списка свободных номеров (по количеству мест и классу);

получение списка номеров (мест), освобождающихся сегодня и завтра;

выдачу информации по конкретному номеру;

автоматизацию выдачи счетов на оплату номера и услуг;

получение списка забронированных номеров;

проверку наличия брони по имени клиента и/или названию организации

Вариант12.

Задача – управление проектами.

В компании несколько отделов. В каждом отделе есть некоторое количество сотрудников, занятых в нескольких проектах и размещающихся в нескольких офисах. Каждый сотрудник имеет план работы, т.е. несколько заданий, которые он должен выполнить. Для каждого такого задания существует ведомость, содержащая перечень денежных сумм, полученных сотрудником за выполнение этого задания. В каждом офисе установлено несколько телефонов.
Для каждого проекта : номер проекта (уникальный) и его бюджет.
Для каждого офиса : номер офиса (уникальный), площадь в квадратных футах, номера всех установленных в нем телефонов.

Вариант13.

Задача – информационная поддержка деятельности спортивного клуба. БД должна осуществлять:

ведение списков спортсменов и тренеров;

учёт проводимых соревнований (с ведением их архива);

учёт травм, полученных спортсменами.

Необходимо предусмотреть:

возможность перехода спортсмена от одного тренера к другому;

составление рейтингов спортсменов;

составление рейтингов тренеров;

выдачу информации по соревнованиям;

выдачу информации по конкретному спортсмену;

подбор возможных кандидатур на участие в соревнованиях (соответствующего уровня мастерства, возраста и без травм).

Вариант14.

Задача – информационная поддержка деятельности аптечного склада.

В аптечном складе хранятся лекарства. Сведения о лекарствах содержатся в специальной ведомости: наименование лекарственного препарата; количество (в шт.); цена; срок хранения на складе (в месяцах). Лекарства поступают на склад ежедневно от разных поставщиков, отпускаются два раза в неделю по предварительным заказам аптек. Выяснить, сколько стоит самый дорогой и самый дешевый препарат; сколько препаратов хранится на складе более 3 месяцев; сколько стоят все препараты, хранящиеся на складе, отыскать препараты, остаток которых равен нулю , ниже требуемого по заказам.

Вариант15.

“Электронный журнал посещаемости"

Предметная область представлена следующими документами:

Список студентов

Журнал посещаемости

Расписание занятий

Предусмотреть учет пропусков по уважительным, неуважительным причинам. Подсчет пропусков по каждому студенту, за неделю, месяц, заданный период, по конкретному предмету.

Лабораторная работа №2

Программирование с помощью встроенного языка TRANSACT- SQL В MICROSOFT SQL SERVER

Цель работы **–**знакомство с основными принципами программирования в MS SQL Server средствами Transact-SQL.

Содержание работы:

1. Изучение правил написания программ средствами Transact-SQL.

2. Изучение правил построения идентификаторов, правил объявления переменных и их типов.
3. Изучение работы с циклами и ветвлениями.
4. Выполнение индивидуальных заданий по вариантам

Пояснения к выполнению работы.

Работа выполняется для базы данных, созданной в предыдущей работе. В MS SQL Server , как и во многих других СУБД, разрешается создание временных объектов. Назначение и синтаксис команд по созданию и работе с временными объектами описан в теоретической части в соответствующем разделе.

Пример выполнения работы.

```
--Создать временную таблицу, содержащую информацию о проекте и числе сотрудников, работающих на нем.
USE Department
GO
DECLARE @Projects TABLE (name_project nvarchar(50), count_employee int)
INSERT INTO @Projects
    SELECT P.project_info, COUNT(E.id_employee)
    FROM Project P, Employee E, Task T
    WHERE P.id_project = T.id_project and T.id_employee = E.id_employee
GROUP BY P.project_info

SELECT * FROM @Projects
```

```
--Создать временную таблицу, содержащую информацию о проекте и числе сотрудников, работающих на нем.
USE Department
GO
DECLARE @Projects TABLE (name_project nvarchar(50), count_employee int)
INSERT INTO @Projects
    SELECT P.project_info, COUNT(E.id_employee)
    FROM Project P, Employee E, Task T
    WHERE P.id_project = T.id_project and T.id_employee = E.id_employee
GROUP BY P.project_info

SELECT * FROM @Projects
```

```
-- Объявить временную переменную, содержащую id проекта, с наибольшим числом сотрудников.
DECLARE @maxCount int
SELECT @maxCount = Project.id_project FROM Task INNER JOIN Project ON Task.id_project = Project.id_project
GROUP BY Project.id_project HAVING COUNT(Task.id_employee) = (SELECT MAX(x.Kolsotrudnikov) as 'MAX' FROM
(SELECT Project.id_project as 'ID Project', COUNT(Task.id_employee) as 'Kolsotrudnikov' FROM Task INNER
JOIN Project on
Task.id_project = Project.id_project GROUP BY Project.id_project) as x)

select @maxCount as 'id dep'
```

```
-- Создать табличную переменную, содержащую информацию о номере отдела и числе сотрудников в нем.

DECLARE @Otd TABLE (count_employee int, number_departmnent int)
INSERT INTO @Otd
SELECT COUNT(Employee.id_employee)
FROM Employee
GROUP BY Department.id_department

SELECT * FROM @Otd
```

```sql
--  Добавить в таблицу сотрудники поле рейтинг и установить его значение равное числу проектов, в
которых задействован сотрудник.
ALTER TABLE Employee ADD Rank VARCHAR(50) NULL

DECLARE @count int
SELECT @count = COUNT(*) FROM Employee

DECLARE @i int
SET @i = 1

WHILE @i<@count
BEGIN
UPDATE Employee SET Rank = (SELECT COUNT(Task.id_employee) as 'Количество'
FROM Employee INNER JOIN Task ON Employee.id_employee = Task.id_employee WHERE Employee.id_employee = @i
GROUP BY Employee.id_employee) WHERE Employee.id_employee = @i
SET @i = @i + 1
END

SELECT * FROM Employee
```

Варианты заданий к лабораторной работе №2.

Вариант 1.

1. Создать временную таблицу, содержащую информацию о числе сотрудников на каждой кафедре, имеющих ученую степень
2. Создать временную переменную, содержащую название кафедры с наибольшим числом сотрудников со степенью
3. Создать табличную переменную, содержащую информацию о среднем балле на каждом факультете
4. Создать табличную переменную рейтинг студентов, содержащую информацию о фамилии студента, номере группы и его рейтинге. Поле рейтинг установить равным 1, средний балл студента выше 8, 2, если вше 6, 3 в остальных случаях.

Вариант 2

1. Создать временную таблицу, содержащую информацию о читателях и количестве взятых их книг. Создать временную переменную, сод инф о максимальном количестве книг.
2. Создать табличную переменную, сод инф о фамилии автора и количестве его изданий.
3. Добавить в таблицу читатель поле рейтинг и установить его значение равным 1, если число книг взятых им выше среднего и 0 в противном случае.
4. Создать глобальную временную таблицу, содержащую информацию о книгах, которые заняты и срок пользования просрочен.

Вариант 3

1. Создать временную таблицу, содержащую информацию о проекте и числе сотрудников, работающих на нем.
2. Объявить временную переменную, содержащую информацию о проекте, с наибольшим числом сотрудников.
3. Добавить в таблицу сотрудники поле рейтинг и установить его значение равное числу проектов, в которых задействован сотрудник.
4. Создать табличную переменную, содержащую информацию о номере отдела и числе сотрудников в нем.

Вариант 4

1. создать временную локальную таблицу содержащую информацию о поставщиках текущего месяца
2. создать переменную, содержащую код материала, имеющий максимальную стоимость
3. создать табличную переменную, содержащую инфу о поставщике и количестве его поставок
4. для поставок, у которых дата реальная и договорная различаются более , чем на 10 дней, снижать стоимость на 10 % (при помощи конструкции if)

Вариант 5

1. Создать временную таблицу, содержащую информацию о клиентах и общей сумме их заказов.
2. Создать переменную, содержащую информацию о максимальной сумме заказов.
3. Создать табличную переменную ,содержащую информацию о видеокассетах и числе их заказов(сколько раз бралась кассета)
4. Добавить в таблицу «видеокассеты» поле «рейтинг» и установить его значение равным числу раз, которое бралась кассета. Если видеокассета ни разу не бралась, удалить ее из таблицы.

Вариант 6

1. создать временную таблицу, содержащую информацию о жанре и числе кассет этого жанра,
2. создать табличную переменную, содержащую информацию о кассетах, выданных в текущем месяце
3. создать глобальную временную таблицу, которая содержит информацию о клиентах,не возвративших в срок кассеты
4. добавить в таблицу видеокассеты поле рейтинг и установить его значение равным 10, если данная видеокассета заказана была выше среднего числа заказов, в противном случае рейтинг равен нулю.

Вариант 7

1. создать временную таблицу, содержащую полную информацию о кинотеатрах, в которых сегодня имеются льготные сеансы на заданный фильм
2. создать табличную переменную, содержащую информацию о льготных билетах в каждом кинотеатре,
3. создать временную переменную содержащую информацию о средней стоимости билетов в кинотеатрах,
4. создать временную таблицу рейтинг кинотеатров, содержащую информацию о кинотеатре, числе залов в нем и его рейтинге. Поле рейтинг установить 1, если число залов в кинотеатре 3 и более и средняя цена на билеты , выше средней во всех кинотеатрах, 2 ,если число залов в кинотеатре 3 и более и средняя цена на билеты ,ниже средней во всех кинотеатрах, 3 во всех остальных случаях..

Вариант 8

1. создать временную таблицу, содержащую информацию о калорийности каждого блюда,
2. создать временную переменную, содержащую информацию о средней цене блюд вегетарианского меню
3. создать табличную переменную, содержащую информацию, название меню, средняя стоимость блюд, количество блюд,

4. создать временную таблицу, содержащую информацию о названии блюда, его стоимости и рейтинге. Поле рейтинг установить равным 1, если калорийность блюда выше среднего значения для данного типа меню, 2 , если равна, 3, если выше среднего значения для данного типа меню.

Вариант 9.

1. Создать временную таблицу , содержащую информацию о товаре и общем количестве его поставок.

2. Создать переменную содержащую информацию о поставщике, выполнившем наибольшее количество поставок.

3. Добавить в таблицу «Поставщики» поле «Рейтинг» и установить его значение равным 0, если число поставок ниже среднего значения и 1, если его число поставок выше среднего значения.

4. Создать временную таблицу содержащую информацию о номере поставки, номере товара и его приоритете. Значение поля «Приоритет» установить равным 0, если срок хранения его просрочен, равным 1, если срок его хранения не более месяца, 2 не более 2-х и т.д.

Вариант 10

1. Создать временную таблицу, содержащую информацию об адвокатах и числе дел, которые они ведут.
2. Создать переменную, содержащую информацию о номере адвоката, имеющего максимальное количество дел.
3. Создать табличную переменную, содержащую информацию о номере дела, номере адвоката, если полученный срок равен минимальному.
4. Добавить в таблицу адвокаты поле «рейтинг» и установить его значение равным нулю, если число дел ниже среднего, единиц - если выше среднего.

Вариант 11.

1. Создать временную таблицу, содержащую информацию о количестве постояльцев за прошедший месяц.
2. Создать временную переменную, содержащую фамилию постояльца, проживавшего наибольший период времени в гостинице.
3. Создать табличную переменную, содержащую информацию о постояльцах, которые пребывали в гостинице более одного раза.
4. Добавить в таблицу постояльцы поле статус и установить его значение равным нулю, если постоялец впервые в гостинице и 1, если уже был.

Вариант 12

1. Создать временную таблицу, содержащую информацию о номере отдела и числе сотрудников в нём.
2. Создать переменную, содержащую информацию о номере отдела, имеющем максимальное число проектов.
3. Добавить в таблицу "Отделы" поле "Рейтинг" и установить его значение равным 1, если число проектов отдела выше среднего, 0, если число проектов ниже среднего.
4. Установить поле "Бюджет отдела" равным сумме бюджетов проектов, которые выполняются в отделе.

Вариант 13.

1. Создать временную таблицу содержащую информацию о фамилии тренера и количестве спортсменов, которые он тренирует.

2. Создать временную переменную, содержащую информацию о среднем количестве травм у спортсменов

3. Создать переменную табличного типа которая содержит информацию об участии спортсменов в соревнованиях за прошедший год.
4. Создать временную таблицу содержащую информацию о статусе спортсменов. Поле статус установить 1, если число участий в соревнованиях выше среднего значения, о в противном случае..

Вариант 14

1. Создать переменную табличного типа, содержащую информацию об аптеках, сделавших заказы в текущем месяце ; табличная переменная содержит номер аптеки, количество заказов
2. Создать локальную переменную, содержащую информацию о наиболее востребованном лекарстве
3. Создать глобальную временную таблицу содержащую информацию о названии лекарства и сроках его хранения
4. В таблицу поставщик добавить поле статус, и установить его равным 1, если поставщик поставляет заказы только в текущем году, 2 - более двух лет 3 - более 5 лет.

Вариант 15

1. Создать временную таблицу содержащую информацию о фамилии студента и количестве пропусков по уважительным и неуважительным причинам.
2. Создать временную переменную сод информацию о среднем количестве пропусков по неуважительным причинам.
3. Создать переменную табличного типа, которая содержит информацию о пропусках студентов по предметам за прошедший месяц.
4. Создать временную таблицу содержащую информацию о статусе студентов. Поле статус установить равным: прогульщик. Если число пропусков по неуважительным причинам выше среднего. Установить равным: слабое здоровье, если выше среднего.

Лабораторная работа №3

Использование курсоров в Transact SQL.

Цель работы –знакомство с основными принципами программирования в MS SQL Server средствами Transact SQL.

Содержание работы:

1. Изучение правил работы с курсорами.
2. Выполнение индивидуальных заданий по вариантам.
3. Составление отчета по работе

Пояснения к выполнению работы.

Работа выполняется для базы данных, созданной в лабораторной работе 1. Часто бывает просто необходимо иметь возможность обратиться к конкретной строке выборки по ее номеру, однако с помощью команды SELECT сделать это довольно трудно. Механизмом, обеспечивающим хранение результата выборки на сервере и преоставляющим пользователю возможность доступа к любой строке выборки по ее номеру, являются курсоры (cursors). Пользователь может работать в каждый момент времени только с одной строкой, но, перемещая окно, он способен получить доступ к любой строке выборки.

Назначение и синтаксис команд по созданию и работе с курсором описан в теоретической части в соответствующем разделе.

Пример выполнения работы.

1. Создать курсор, который бы формировал список сотрудников , которые участвуют в каждом из проектов .Информацию выводить в следующем виде: Название проекта, Список сотрудников.

```
USE Department

DECLARE @project_info nvarchar(30)

DECLARE proj_CURSOR SCROLL CURSOR FOR
SELECT DISTINCT project_info FROM Employee E, Task T, Project P
    WHERE E.id_employee = T.id_employee AND P.id_project = T.id_project

OPEN proj_CURSOR

FETCH FIRST FROM proj_CURSOR
INTO @project_info

WHILE @@FETCH_STATUS = 0
BEGIN
    SELECT DISTINCT @project_info as 'Name project'
    DECLARE @name_employee nvarchar(30)
    CREATE TABLE #sometable (name_employee nvarchar(30))
    DECLARE employee_cursor CURSOR FOR
    SELECT full_name
    FROM Employee E, Task T, Project P
           WHERE E.id_employee = T.id_employee AND P.id_project = T.id_project AND
P.project_info = @project_info

    OPEN employee_cursor

    FETCH NEXT FROM employee_cursor
    INTO @name_employee

    WHILE @@FETCH_STATUS = 0
    BEGIN
           INSERT INTO #sometable VALUES(@name_employee)
           FETCH NEXT FROM employee_cursor
           INTO @name_employee
    END

    SELECT name_employee as 'full name'
    FROM #sometable

    DROP TABLE #sometable

    CLOSE employee_cursor
    DEALLOCATE employee_cursor

    FETCH NEXT FROM proj_CURSOR
    INTO @project_info
END
CLOSE proj_CURSOR
DEALLOCATE proj_CURSOR
```

2 Создать курсор, который бы отображал информацию о проектах , по которым имеется в настоящие время финансирование .

```
USE Department
DROP TABLE #t
CREATE TABLE #t (sotrID int, full_name nvarchar(30), count_project int, salary
nvarchar(30) )
INSERT INTO #t
```

```
SELECT E.id_employee, E.full_name, COUNT(T.id_employee) as 'Count projects',
E.size_salary
FROM Employee E, Task T WHERE E.id_employee = T.id_employee
GROUP BY T.id_employee, E.full_name, E.id_employee, E.size_salary
having COUNT(T.id_employee)>0

DECLARE some_CURSOR CURSOR
FOR
SELECT * FROM #t

OPEN some_CURSOR

DECLARE @counter int
DECLARE @sotrID int, @full_name nvarchar(30), @countproject int,
@salary nvarchar(30)
SET @counter = 0

FETCH NEXT FROM some_CURSOR INTO
@sotrID, @full_name, @countproject, @salary

WHILE @@FETCH_STATUS = 0
BEGIN
   update #t SET #t.salary = CAST(@salary as int)*1.1
   WHERE current of some_CURSOR

   FETCH NEXT FROM some_CURSOR INTO
   @sotrID, @full_name, @countproject, @salary
END

CLOSE some_CURSOR
DEALLOCATE some_CURSOR
```

3 Создать курсор, который бы выводил информации о сотрудниках участвующих в двух и более проектах. С помощью курсора увеличить этим сотрудникам выплаты на 10%.

```
USE Department
DROP TABLE #t
CREATE TABLE #t (sotrID int, full_name nvarchar(30), count_project int, salary
nvarchar(30) )
INSERT INTO #t
SELECT E.id_employee, E.full_name, COUNT(T.id_employee) as 'Count projects',
E.size_salary
FROM Employee E, Task T WHERE E.id_employee = T.id_employee
GROUP BY T.id_employee, E.full_name, E.id_employee, E.size_salary
having COUNT(T.id_employee)>0

DECLARE some_CURSOR CURSOR
FOR
SELECT * FROM #t

OPEN some_CURSOR

DECLARE @counter int
DECLARE @sotrID int, @full_name nvarchar(30), @countproject int,
@salary nvarchar(30)
SET @counter = 0

FETCH NEXT FROM some_CURSOR INTO
@sotrID, @full_name, @countproject, @salary

WHILE @@FETCH_STATUS = 0
BEGIN
   update #t SET #t.salary = CAST(@salary as int)*1.1
   WHERE current of some_CURSOR
```

```
    FETCH NEXT FROM some_CURSOR INTO
    @sotrID, @full_name, @countproject, @salary
END

CLOSE some_CURSOR
DEALLOCATE some_CURSOR

SELECT * FROM #t
```

4 Создать курсор, который отыскивал бы сотрудников , не участвующих ни в одном из проектов и с помощью курсора удалить этих сотрудников.

```
 USE Department
CREATE TABLE #s_table (some_sotrud nvarchar(30))
INSERT INTO #s_table SELECT full_name FROM Employee

CREATE TABLE #d_table (some_sotrud nvarchar(30))
INSERT INTO #d_table SELECT E.full_name FROM Employee E except (SELECT E.full_name
FROM Employee E INNER JOIN Task T ON
E.id_employee = T.id_employee)

DECLARE udalit_CURSOR CURSOR FOR SELECT * FROM #d_table FOR UPDATE


DECLARE @some_sotrud nvarchar(30)
OPEN  udalit_CURSOR


FETCH NEXT FROM udalit_CURSOR
INTO @some_sotrud
WHILE @@FETCH_STATUS = 0
    BEGIN
           DELETE FROM #s_table
           WHERE #s_table.some_sotrud = @some_sotrud
           FETCH NEXT FROM udalit_CURSOR
           INTO @some_sotrud
    END

SELECT * FROM #s_table

CLOSE udalit_CURSOR

DEALLOCATE udalit_CURSOR
DROP TABLE #s_table
DROP TABLE #d_table
```

Варианты заданий к лабораторной работе №3

Вариант 1.

1. Создать курсор, который бы формировал список преподавателей заданной кафедры.
2. Создать курсор, который бы формировал успеваемость студентов по дисциплинам (название дисциплины - шапка, список студентов).
3. Отыскать с помощью курсора преподавателей заданной кафедры и перевести всех их на другую кафедру.
4. Отыскать преподавателей, читающих дисциплину «история», изменить название дисциплины на «История ВОВ»

Вариант 2.

1. Создать курсор, который бы формировал инфу о название раздела и числе книг данного раздела. Инфу с курсора выводить в обратном порядке, используя скролирующий курсор

2. Создать курсор, который выводит инф о читателе и списке книг, которые за ним числятся.

Информацию выводить в виде: Номер читательского билета, ФИО, - как шапка

Список книг

3. Создать курсор, который бы отыскивал книги, год издания которых ранее 1962 г., с помощью курсора удалить эти книги.

4. Создать курсов, который бы отыскивал книги , которые были выданы более месяца назад. Для этих книг изменить дату выдачи на текущую

Вариант 3.

1. Создать курсор , который бы формировал список сотрудников ,которые учувствуют в каждом из проектов .Информацию выводить в следующем виде: Название проекта, - как шапка, Список сотрудников.
2. Создать курсор, который бы отображал информацию о проектах , по которым имеется в настоящие время финансирование
3. Создать курсор , который бы выводил информации о сотрудниках участвующих в двух и более проектах. С помощью курсора увеличить этим сотрудникам выплаты на 10%
4. Создать курсор , который отыскивал бы сотрудников не участвующих ни в одном проекте и с помощью курсора удалить этих сотрудников

Вариант 4.

1. Создать курсор, который бы формировал поставки, выполненные в прошлом месяце. Результат выводить в обратном порядке.
2. Создать курсор, который бы выводил поставщиков и товары, которые они поставили.
3. Создать курсор, который бы искал поставки, в которые входит заданный вид сырья. С помощью его увеличить объём поставок в 2 раза.
4. Создать курсор, который бы искал поставщиков, не имеющих поставок. Удалить их с помощью курсора.

Вариант 5.

1. Создать курсор, который формирует список видеокассет по заданному жанру.в обратном порядке.
2. Создать курсор, который выводит список кассет, которые числятся за каждым клиентом. Фамилия клиента как шапка, потом список кассет.
3. Увеличить стоимость проката кассет на 10% для кассет текущего года.
4. Удалить кассеты, которые не пользуются спросом.

Вариант 6.

1. Создать курсор который формирует список кассет выданных в текущем месяце. Список выводить по датам текущего месяца в обратном порядке
2. Создать курсор, который формирует список фильмов по жанрам. Верхняя строчка – название жанра, затем список фильмов
3. Обновление поля рейтинг с помощью курсора.
4. Удалить кассеты заданного типа.

Вариант 7.

1. Создать курсор , который бы формировал список фильмов ,которые демонстрируются в нескольких кинотеатрах и выводил информацию: название фильма – шапка, а список кинотеатров в столбец внизу.
2. Создать курсор, который бы отображал информацию по текущей дате о льготных сеансах в кинотеатрах и с помощью его уменьшить цену на льготные сеансы на 10%
3. Создать курсор , который бы выводил кинотеатры и фильмы, которые демонстрируются в настоящее время.
4. Создать курсор , который отыскивал фильмы показ которых закончен и с помощью него удалить информацию об этих фильмах.

Вариант 8.

1. Создать курсор, который бы формировал информацию о названии меню и числе блюд данного меню. Информацию выводить в обратном порядке с помощью скроллирующего курсора.
2. Создать курсов который бы выводил информацию о блюдах в которые входит каждый компонент. Информацию выводить в следующем виде: название компонента в отдельной строке затем список блюд.
3. Создать курсор который бы отыскивал блюда в которые входит заданный компонент, затем с помощью курсора удалить эти блюда.
4. Создать курсор который бы увеличивал рейтинг блюд , в которые входит заданный компонент.

Вариант 9.

1. Создать курсор, который бы формировал список товаров , поступивших в каждом месяце текущего года. Информацию из курсора извлекать в обратном порядке.
2. Создать курсор, который бы выводил поставщиков и товары, которые они поставили.
3. Создать курсор, который бы искал товары с истекшим сроком хранения . С помощью его удалить эти товары.
4. Создать курсор, который бы искал товары, которые пролежали на складе более года, с помощью курсора уменьшить цену на 30%.

Вариант 10.

1. Создать курсор который бы выводил информацию о делах , которые ведет заданный адвокат в обратном порядке
2. Создать курсор который выводит фамилии адвокатов и даты завершенных дел.
3. Создать курсор который бы отыскивал дела с условным срокам наказания и увеличить штраф на 10%(с помощью курсора).
4. Отыскать адвокатов с нулевым рейтингом и удалить их с помощью курсора.

Вариант 11.

1. Создать курсор, который бы формировал инфу о проживающих в данный момент постояльцев с указанием места проживания. Информацию из курсора выводить в обратном порядке, используя скроллирующий курсор.
2. Создать курсор, который выводит о выручке гостиницы по месяцам - Название месяца и общая сумма.
3. Создать курсор, который бы отыскивал постояльцев, которые за прошедшие два года не пользовались гостиницей. С помощью курсора удалить их.
4. Создать курсов, который бы отыскивал постояльцев, пользующихся услугами гостиницы более двух раз с помощью курсора для этих постояльцев снизить оплату на 10 процентов.

Вариант 13.

1. Создать курсор, который бы формировал информацию о соревнованиях текущего года и их участниках. Информацию из курсора выводить в обратном порядке, используя скроллирующий курсор.
2. Создать курсор, который бы формировал список тренеров и их спортсменов. Информацию из курсора выводить в следующем порядке : 1 строка- фамилия тренера, затем в столбик фамилии спортсменов.
3. Отыскать с помощью курсора спортсменов заданного тренера и перевести всех их другому тренеру.
4. Удалить тренера у которого нет спортсменов.

Вариант 14.

1. Создать курсор, который бы формировал поставки, выполненные в прошлом месяце. Результат из курсора выводить в обратном порядке.
2. Создать курсор, который бы выводил заказы и лекарства, которые заказаны.
3. Создать курсор, который бы искал лекарства, которые не заказывали . С помо-щью курсора удалить их.
4. Создать курсор, который бы искал лекарства, которые заказывались в текущем месяце. Увеличить цену на них на 10%.

Вариант 15.

1. Создать курсор, который бы формировал список студентов заданной специальности.
2. Создать курсор, который бы формировал успеваемость студентов по дисциплинам (название дисциплины - шапка, список студентов).
3. Отыскать с помощью курсора студентов заданной специальности и перевести всех их на другую специальность.
4. С помощью курсора отыскать специальности, по которым нет наборов студентов и с помощью курсора удалить все записи.

Лабораторная работа №4

Создание хранимых процедур в MICROSOFT SQL SERVER.

Цель работы **–**создавать и использовать хранимые процедуры на сервере БД.

Содержание работы:

1. Изучение основных команд для создания, вызова, корректировки и выполнения хранимых процедур.
2. Выполнение индивидуальных заданий по вариантам.
3. Составление отчета по работе.

Пояснения к выполнению работы.

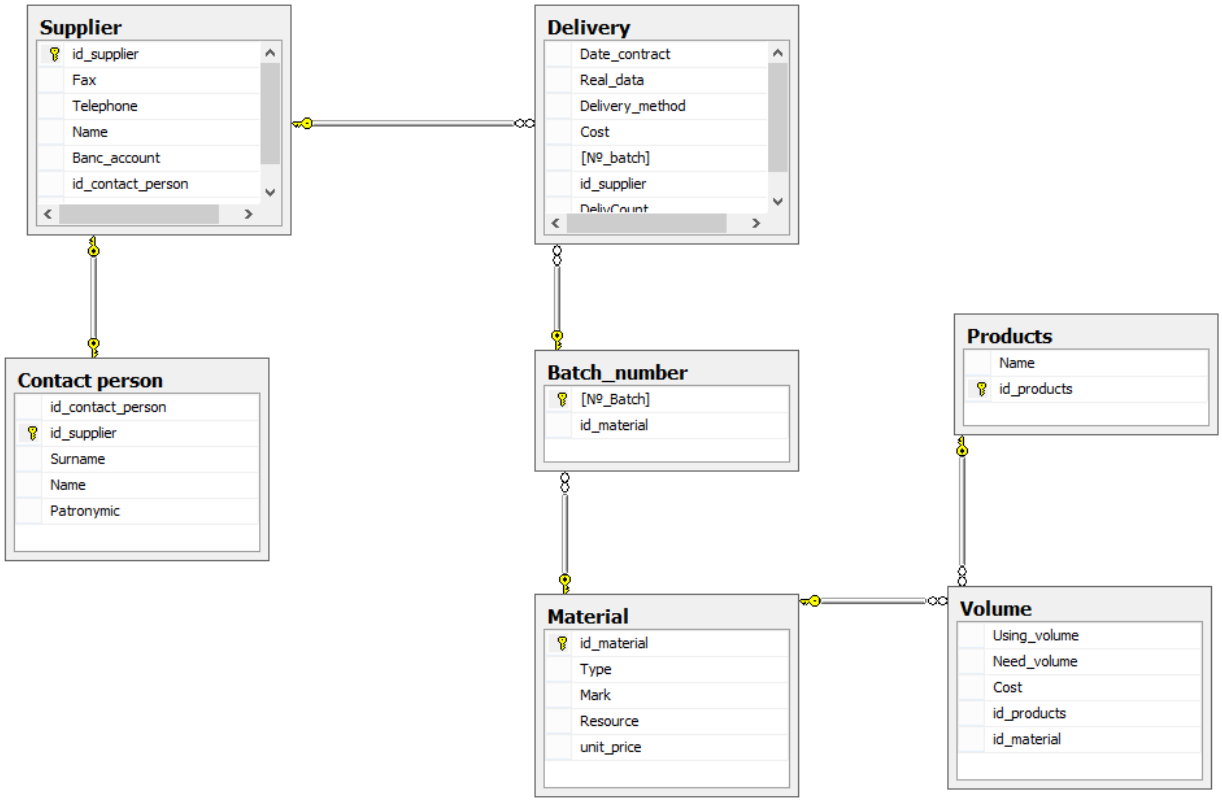
Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняются в базе данных в откомпилированном виде. При создании хранимой процедуры следует решить следующие задачи:

- Планирование прав доступа: хранимая процедура будет иметь те же права доступа к объектам базы данных, что и создавший ее пользователь.
- Определение параметров хранимой процедуры: хранимая процедура может не иметь параметров, может иметь входные параметры, может иметь выходные параметры.
- Код хранимой процедуры может содержать последовательность любых команд SQL, включая вызов других хранимых процедур.

Назначение и синтаксис команд по работе с хранимыми процедурами описан в теоретической части в соответствующем разделе. Работа выполняется для базы данных, созданной в лабораторной работе 1.

Пример выполнения работы.

Схема базы данных.



Задания для приведенной базы данных.

1.Создать хранимую процедуру, предназначенную для удаления поставщика. Значение внешнего ключа в подчиненных таблицах установить равным NULL.

Создание хранимой процедуры.

```
create procedure delSupplier;1
      @supplier char(20)
as
      begin
      declare @id int
      set @id = (select Supplier.id_supplier from Supplier where Supplier.Name
= @supplier)
            update Delivery
            set id_supplier = NULL
            where Delivery.id_supplier = @id
            delete from Supplier
            where Supplier.Name = @supplier
```

```
      end
```
Выполнение хранимой процедуры.
```
exec delSupplier MTZ
```
Удаление хранимой процедуры.
```
drop procedure delSupplier
```

2. Создать хранимую процедуру для добавления нового материала.

Создание хранимой процедуры.
```
create procedure AddMaterial;1
      @id int,
      @type nvarchar(30),
      @mark nvarchar(30),
      @resource int,
      @price int
AS
      insert into Material (id_material, Type, Mark, Resource, unit_price)
      values (@id, @type, @mark, @resource, @price)
```
Выполнение хранимой процедуры.
```
EXEC AddMaterial 122, adsaasf, sfasf, 1231, 213
```
3. Создать хранимую процедуру, которая для заданного поставщика определяет общую сумму , выполненных им поставок.

Создание хранимой процедуры.
```
create procedure SupplierSumm;1
      @supplier nvarchar(30)
as
BEGIN
      select distinct sum(temp.Resource * temp.unit_price) as price,
Supplier.Name
            from supplier, (select Material.Resource, Supplier.id_supplier,
Material.unit_price
                  from Material,Delivery,Supplier,Batch_number
                  where Supplier.id_supplier = Delivery.id_supplier
                  and Delivery.[¹_batch] = Batch_number.[¹_Batch]
                  and Batch_number.id_material = Material.id_material
                  and Supplier.Name = @supplier
                  group by Supplier.id_supplier, Material.Resource,
Material.unit_price) temp
            where Supplier.Name = @supplier
            group by Supplier.Name
   end
```
Выполнение хранимой процедуры.
```
   exec SupplierSumm 'Belaz'
```

Варианты заданий к лабораторной работе №4

Вариант 1.

1. Создать хранимую процедуру, предназначенную для увольнения сотрудника по его номеру паспорта. Дисциплины этого сотрудника перевести другому сотруднику этой же кафедры, имеющем менее трех дисциплин..
2. Создать хранимую процедуру, которая определяет число дисциплин для каждого сотрудника заданной кафедры. Название кафедры – входной параметр.
3. Создать хранимую процедуру, которая бы определяла средний балл по каждой дисциплине.

Вариант 2.

1. Создать хранимую процедуру для выдачи читателю книг по его фамилии. Если читатель записан в библиотеку, то выбирать номер его читательского билета, если не записан, то записать его в библиотеку и выдать книги.
2. Создать хранимую процедуру, которая бы по фамилии, имени, отчеству подсчитывала число книг, которые за ним числятся
3. Создать хранимую процедуру, которая бы отыскивала должников, т.е.читателей, за которыми числятся книги более 1 месяца.

Вариант 3.

1. Создать хранимую процедуру, предназначенную для приема на работу нового сотрудника и назначения ему задания. Задание выдавать по проекту, на котором работает наименьшее число сотрудников..
2. Создать хранимую процедуру, которая определяет число заданий для каждого сотрудника заданного отдела. Номер отдела – входной параметр.
3. Создать хранимую процедуру, которая бы определяла исполнителей и сумму выплат по каждому проекту.

Вариант 4.

1. Создать хранимую процедуру, предназначенную для удаления поставщика. Значение внешнего ключа в подчиненных таблицах установить равным NULL.
2. Создать хранимую процедуру для добавления нового материала.
3. Создать хранимую процедуру, которая для заданного поставщика определяя общую сумму , выполненных им поставок.

Вариант 5.

1. Создать хранимую процедуру, которая бы удаляла из базы клиентов, не бравщих кассеты более года.
2. Создать хранимую процедуру для оформления нового заказа. Если такой клиент уже есть в базе, то выбрать его номер, если нет добавить в базу.
3. Создать хранимую процедуру, которая бы позволяла определять выручку за заданный месяц. Месяц передается как параметр.

Вариант 6.

1. Создать хранимую процедуру для удаления кассеты. Значения внешнего ключа заменить на NULL.
2. Создать хранимую процедуру, которая бы позволяла изменять стоимость проката кассет.
3. Создать хранимую процедуру, которая бы позволяла получать информацию обо всех кассетах-новинках, либо о кассетах-новинках заданного жанра.

Вариант 7.

1. Создать хранимую процедуру, которая бы обновляла афишу по заданной дате
2. Создать хранимую процедуру, которая бы добавляла в БД новый фильм, при этом проверяла,что бы фильм не был введён дважды под разными id
3. Создать хранимую процедуру, которая бы по текущей дате формировала афишу для всех к/т.

Вариант 8.

1. Создать хранимую процедуру для добавления нового блюда. Если все продукты, необходимые для него имеются, выбрать их. Если нет, добавить их в базу.
2. Создать хранимую процедуру, которая бы по заданной калорийности выводила список блюд.
3. Создать хранимую процедуру, которая бы позволяла составить обеденное меню заданной калорийности.

Вариант 9.

1. Создать хранимую процедуру, предназначенную для оформления новой поставки. Если поставщик есть в базе, то выбрать его, иначе добавить в базу.

2. Создать хранимую процедуру, которая бы по заданной дате определяла общее число поставок.
3. Создать хранимую процедуру, которая бы позволяла определять самый востребованный товар прошедшего месяца.

Вариант 10.

1. Создать хранимую процедуру добавления нового дела. Если клиент присутствует, то выбрать его, если нет , то добавить в базу. Если адвокат присутствует, то выбрать его, иначе добавить в базу.
2. Создать хранимую процедуру, которая бы позволяла определять гонорар заданного адвоката.
3. Создать хранимую процедуру, которая бы позволяла определять эффективность защиты.

Вариант 11.

1. Создать хранимую процедуру, которая бы выполняла заселение в гостиницу. Если постоялец уже был в гостинице, то выбирать его из архива, иначе добавлять в базу данных. Заселять в свободный номер.
2. Создать хранимую процедуру, которая бы определяла количество свободных номеров заданного типа.
3. Создать хранимую процедуру, которая бы выводила список выбывших постояльцев на заданную дату.

Вариант 12.

1. Создать хранимую процедуру, предназначенную для увольнения сотрудника по его номеру паспорта. Задания этого сотрудника перевести другому сотруднику этого отдела, имеющего менее трех заданий.
2. Создать хранимую процедуру, которая определяет число заданий для каждого сотрудника заданного отдела. Номер отдела – входной параметр.
3. Создать хранимую процедуру, которая бы определяла сумму выплат по каждому проекту.

Вариант 13

1. Создать хранимую процедуру которая бы добавляла информацию о новом соревновании и его участниках если они уже есть выбирать номер иначе- добавлять спортсмена в базу
2. Создать хранимую процедуру которая бы удаляла тренера, его спортсменов перевести к тренеру с наименьшим числом спортсменов.
3. Создать хранимую процедуру которая по фамилии спортсмена выведет информацию о рейтинге спортсмена числе соревнований и полученных им травм.

Вариант 14.

1.Создать хранимую процедуру, которая бы определяла список лекарств со сроком хранения более года.
2. Создать хранимую процедуру, которая удаляет препараты, остаток которых ноль.
3. Создать процедуру для формирования нового заказа и проверки нужного количества лекарства на складе.

Вариант 15.

1. Создать хранимую процедуру, предназначенную для отчисления студента по его среднему баллу меньше 4.
2. Создать хранимую процедуру, которая определяет число дисциплин для каждой специальности. Название специальности – входной параметр.
3. Создать хранимую процедуру, которая бы определяла средний балл по каждой специальности.

Лабораторная работа №5

Создание и использование триггеров в MICROSOFT SQL SERVER.

Цель работы –создавать и использовать триггеры с целью обеспечения целостности базы данных.

Содержание работы:

1. Изучение правил работы с триггерами.
2. Выполнение индивидуальных заданий по вариантам.
3. Составление отчета по работе

Пояснения к выполнению работы.

Триггеры (triggers) SQL Server представляют собой набор команд Transact-SQL, выполняемых автоматически при осуществлении тех или иных модификаций данных в таблице. Каждый триггер связан с конкретной таблицей и запускается сервером автоматически каждый раз, когда пользователи пытаются произвести вставку, изменение или удаление данных. Назначение и синтаксис команд по работе с триггерами описан в теоретической части в соответствующем разделе. Работа выполняется для базы данных, созданной в лабораторной работе 1.

Пример выполнения работы.

1. Создать триггер, который при добавлении нового сотрудника контролировал число сотрудников в отделе .

```
USE Department
GO
ALTER TRIGGER [dbo].[EmployeControl]
ON [dbo].[Employee]
AFTER INSERT
AS
	BEGIN

	DECLARE @id_department int
	DECLARE @count_employee int

	DECLARE @Ot2 TABLE (count_employee int, number_departmnent int)
	INSERT INTO @Ot2 SELECT COUNT(id_employee) as 'count',id_department FROM Employee
GROUP BY id_department

	SELECT @id_department = (SELECT id_department FROM inserted)
	SELECT @count_employee = (SELECT count_employee FROM @Ot2 WHERE number_departmnent
= @id_department)

	IF(@count_employee >= 4)
		BEGIN
			RAISERROR(N'Число сотрудеиков в данном отделе больше установленного
лимита', 16,1)
			ROLLBACK TRANSACTION
		END
	END
```

2. Создать триггер, который бы при увольнении сотрудника переводил его проекты другому сотруднику этого же отдела ,у которого число заданий меньше 3 .

```
USE Department
```

```
GO
DROP TRIGGER Delete_Sotrud
CREATE TRIGGER Delete_Sotrud
ON Employee
instead of DELETE
AS
BEGIN
   DECLARE @id_employee int, @id_department int, @id_count_task int;

   (SELECT top 1 @id_employee = x.id_employee FROM
   (SELECT COUNT(T.id_employee) AS 'count', E.id_employee
   FROM Employee E, Task T WHERE E.id_employee = T.id_employee
   GROUP BY E.id_employee) AS x WHERE x.count < 3)

   DECLARE @id_sotrud_delete int, @id_project int, @task_sotrud
nvarchar(20);

   SELECT @id_sotrud_delete = id_employee FROM deleted

   UPDATE Task SET id_employee = @id_employee WHERE id_employee =
@id_sotrud_delete

   DELETE Employee WHERE id_employee = @id_sotrud_delete
END
USE Department
```

3.Создать таблицу, контролирующую деятельность всех триггеров.

```
IF OBJECT_ID('LogTable') IS NULL
    CREATE TABLE LogTable
        (
          id INT PRIMARY KEY IDENTITY ,
          tableName CHAR(20) ,
          tableAction CHAR(20) ,
          date DATE ,
          rowsAffected INT
        );


CREATE PROCEDURE logTriggerFunc
       @tableName CHAR(20),
       @deletedCount INT,
       @insertedCount INT
AS
BEGIN
       DECLARE @count INT, @action CHAR(20);

       SET @count = @insertedCount;
       IF (@count = 0) SET @count = @deletedCount;

       IF (@insertedCount <> 0 AND @deletedCount = 0)
              SET @action = 'INSERT data';
       ELSE IF (@insertedCount = 0 AND @deletedCount <> 0)
              SET @action = 'DELETE data';
       ELSE
              SET @action = 'UPDATE data';

       INSERT INTO LogTable VALUES (@tableName, @action, GETDATE(), @count);
END;
```

4.Создать триггер, контролирующий изменения в таблице Employee.

```
CREATE TRIGGER LogWarehouse
ON Employee
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
```

```
        DECLARE @deletedCount INT, @insertedCount INT;

        SELECT @deletedCount = COUNT(*) FROM DELETED;
        SELECT @insertedCount = COUNT(*) FROM INSERTED;

        EXEC logTriggerFunc 'Employee', @deletedCount, @insertedCount;
END;
```

Варианты заданий к лабораторной работе №5

Вариант 1.

1. Создать триггер, который бы контролировал, чтобы за преподавателем числилось не более трех дисциплин.
2. Создать триггер, который бы при увольнении преподавателя переводил его дисциплины другому преподавателю. После чего увольнял преподавателя.
3. создать триггер, контролирующий изменения в таблице "студент"
4. создать таблицу, контролирующую деятельность всех триггеров.

Вариант 2.

1. Создать триггер, не позволяющий читателю выдавать более 5-ти книг.
2. Создать триггер, который бы на 30 июня каждого года выдавал список студентов-выпускников, за которыми числятся книги.
3. создать триггер, контролирующий изменения в таблице "выданные книги"
4. создать таблицу, контролирующую деятельность всех триггеров.

Вариант 3.

1. Создать триггер, который при добавлении нового проекта увеличивал бюджет отдела, который выполняет этот проект.
2. Создать триггер, который бы при увольнении сотрудника переводил его проекты другому сотруднику. После чего увольнял сотрудника.
3. создать триггер, контролирующий изменения в таблице "сотрудники"
4. создать таблицу, контролирующую деятельность всех триггеров

Вариант 4.

1. Создать триггер, который при поставке материала его количество увеличивал на соответствующую величину.
2. Создать триггер, который при исключении материала , заменял его другим.
3. Создать триггер, контролирующий изменения в таблице Материалы.
4. Создать таблицу, контролирующую деятельность всех триггеров.

Вариант 5.

1. Создать триггер, не позволяющий клиенту выдавать более 3-трех кассет.
2. Создать триггер, который контролирует, чтобы фамилия вводилась с заглавной буквы.
3. Создать триггер, контролирующий изменения в таблице Заказы.
4. Создать таблицу, контролирующую деятельность всех триггеров.

ВАРИАНТ 6

1. Создать триггер, который бы при удалении клиента контролировал, чтобы за ним не числилось кассет, в случае если есть кассеты, выдавало бы предупреждающее сообщение.
2. Создать триггер, который бы не позволял выдать клиенту более 5 кассет.
3. Создать триггер, который бы контролировал изменения в таблице заказы.
4. Создать таблицу, контролирующую действий всех триггеров.

ВАРИАНТ 7

1. Создать триггер , который ежедневно контролирует афишу и обновляет её.

2. Создать триггер, который при удалении информации о кинотеатре, удаляет информацию об афише этого кинотеатра.
3. Создать триггер, контролирующий изменения в таблице Афиша.
4. Создать таблицу, контролирующую деятельность всех триггеров.

ВАРИАНТ 8.

1. Создать триггер, который контролирует, чтобы название блюда вводилось с заглавной буквы.
2. Создать триггер, который добавляет новое блюдо при условии наличия всех необходимых продуктов для него.
3. Создать триггер, который контролирует изменения в таблице Блюда.
4. Создать таблицу, контролирующую таблицу действий всех триггеров.

ВАРИАНТ 9

1. Создать триггер, который контролирует остаток товаров на складе. Для товара с нулевым остатком товара выводит сообщение: требуется поставка этого товара.
2. Создать триггер, который бы при удалении информации о поставщике внешний ключ задавал как NULL -значение.
3. Создать триггер, который контролирует изменения в таблице Склад
4. Создать таблицу, контролирующую таблицу действий всех триггеров.

ВАРИАНТ 10.

1. Создать триггер, который при увольнении адвоката, переводит его дела другому адвокату.
2. Создать триггер, который при назначении адвокату нового дела контролирует, чтобы за ним числилось не более 3 дел.
3. Создать триггер, контролирующий изменения в таблице Дела.
4. создать таблицу, контролирующую деятельность всех триггеров.

Вариант11.

1. Создать триггер, который контролирует, чтобы фамилия вводилась с заглавной буквы
2. Создать триггер, который контролирует, чтобы не заселили клиента в уже занятый номер.
3. Создать триггер, контролирующий изменения в таблице Номера.
4. Создать таблицу, контролирующую деятельность всех триггеров.

Вариант 12.

1. Создать триггер, который при добавлении нового проекта увеличивает бюджет отдела, который выполняет этот проект.

2. Создать триггер, который при увольнении сотрудника переводит его проекты другому сотруднику.
3. Создать триггер, контролирующий изменения в таблице Проекты.
4. Создать таблицу, контролирующую деятельность всех триггеров

Вариант 13

1. Создать триггер, который не позволяет назначать тренеру более 5 спортсменов.
2. Создать триггер , который бы при увольнении тренера переводит спортсменов другому тренеру.
3. Создать триггер, который контролирует изменения в таблице Тренеры.
4. Создать таблицу, контролирующую действий всех триггеров.

Вариант 14

1. Создать триггер, который не позволяет формировать заказы для лекарств с просроченным сроком.
2. Создать триггер, который не позволял дважды ввести поставщика под разными id.

3. Создать триггер, который бы контролировал изменения в таблице Заказы.
4. Создать таблицу, контролирующую действий всех триггеров.

Вариант 15.

1. Создать триггер, который контролирует, чтобы фамилия студента вводилась с заглавной буквы.
2. Создать триггер, который при добавлении нового пропуска по неуважительной причине заносит фамилию студента в таблицу «Прогульщик», если эта фамилия в этой таблице еще не присутствует.
3. Создать триггер, который бы контролировал изменения в таблице «Прогульщик».
4. Создать таблицу, контролирующую действий всех триггеров.

Лабораторная работа №6

Создание и использование функций в MICROSOFT SQL SERVER.

Цель работы: приобретение практических навыков использования встроенных функций и создания пользовательских функций.

Содержание работы:

1. Изучение правил работы с встроенными и пользовательскими функциями.
2. Выполнение индивидуальных заданий по вариантам.
3. Составление отчета по работе

Пояснения к выполнению работы.

Необходимо ознакомиться со встроенными и пользовательскими функциями. Назначение и синтаксис команд по работе с функциями описан в теоретической части в соответствующем разделе. Работа выполняется для базы данных, созданной в лабораторной работе 1. Первые три задания работы выполнять с использованием встроенных функций. Пользовательские функции являются обычными объектами базы данных, такими же, как и хранимые процедуры, триггеры, представления и т. д. Каждый из пользователей, имеющий соответствующие права доступа, может создать произвольную функцию. Таким образом, определяемая пользователем функция, в отличие от встроенной функции, имеет своего владельца, который может предоставлять другим пользователям право на ее вызов.

Пример выполнения работы.

1.Подсчитать число сотрудников в каждом отделе.

```
CREATE FUNCTION funcFirst()
RETURNS @tb1 TABLE (id_department int, numbers int)
AS
BEGIN
 INSERT INTO @tb1 SELECT id_department, COUNT(*) as 'number_employee' FROM Employee
GROUP BY id_department
RETURN
END

SELECT * FROM funcFirst()
```

2.Отыскать сотрудников, получающих зарплату выше 25000.

```
CREATE FUNCTION funcSecond()
RETURNS @tb2 TABLE (id_department int, numbers int)
AS
BEGIN
 INSERT INTO @tb1 SELECT full_name FROM Employee
WHERE size_salary > 25000
RETURN
END

SELECT * FROM funcSecond()
```

3.Отыскать полное имя сотрудника наибольшей длины.

```
CREATE FUNCTION funcThird()
RETURNS @tb3 TABLE (full_name nvarchar(50))
AS
BEGIN
 INSERT INTO @tb3 SELECT full_name FROM Employee Where full_name = (SELECT MAX(full_name)
FROM Employee)
RETURN
END

SELECT * FROM funcThird()
```

4.Создать функцию, которая бы для заданного отдела по каждому проекту выводила общую сумму сделанных выплат.

```
CREATE FUNCTION funcSix(@id_dep int)
RETURNS int
AS
BEGIN
       declare @summa int
 SELECT @summa=SUM(sum_pay) FROM Task WHERE id_employee IN
(SELECT id_employee FROM Employee WHERE id_department = @id_dep)
RETURN
END

SELECT dbo.funcSix(11) as общая_сумма
```

Варианты заданий к лабораторной работе №6

Вариант 1.

1. Подсчитать число сотрудников на каждой кафедре**.**
2. Отыскать сотрудников, родившихся в текущем месяце.(если нет полня ДАТАРОЖДЕНИЯ. то добавить ее).
3. Отыскать фамилию наибольшей длины.
4. Создать функцию, которая для заданной группы, возвращает ее средний балл.
5. Создать функцию, которая по номеру группы формирует список студентов.
6. Создать функцию, которая для заданной группы по каждому предмету выводит список студентов, имеющих оценки «8» и выше.

Вариант 2.

1. Подсчитать число читателей, которые брали книги в прошлом месяце.
2. Отыскать имя наибольшей длины.

3. Вывести имя текущей базы данных.
4. Создать функцию, которая бы выводила список читателей, у которых книги более месяца на руках.
5. Создать функцию, которая бы для заданного читателя выводила число книг, которые за ним числятся.
6. Создать функцию, которая бы выводила число книг каждого издательства за заданное число лет.

Вариант 3.

1. Подсчитать число сотрудников в каждом отделе.
2. Отыскать сотрудников, родившихся в текущем месяце.(если нет поля ДАТАРОЖДЕНИЯ то добавить такое поле).
3. Отыскать фамилию наибольшей длины.
4. Создать функцию, которая для заданного отдела , возвращает его бюджет.
5. Создать функцию, которая по номеру проекта формирует список сотрудников, работающих над ним.
6. Создать функцию, которая для заданного отдела по каждому проекту выводит общую сумму сделанных выплат.

Вариант 4.

1.Подсчитать число поставщиков, которые в текущем году не делали поставок.

2.Подсчитать число символов в наименовании материала с кодом 10.

3.Вывести имя текущей базы данных.

4. Создать функцию, которая бы для заданного материала выводила список его поставщиков.

5.Создать функцию, которая бы по заданной дате выводила число выполненных поставок.

6.Создать функцию, которая бы выводила поставщиков, которые не делали поставок в текущем месяце. Если все поставщики работали, то вывести список материалов, которые не поставлялись в текущем месяце. Иначе вывести поставщиков и материалы, которые они поставляли.

Вариант 5.

1. Подсчитать число клиентов за каждый месяц текущего года.
2. Отыскать клиентов, у которых не указан телефон.
3. Сделать все символы фамилии клиента заглавными буквами.
4. Создать функцию, которая по заданной дате выводит общий список заказов.
5. Создать функцию, которая для заданного клиента подсчитывает число кассет, которые за ним числятся.
6. Создать функцию, которая для каждого фильма выводит список клиентов, которые его брали. Если фильм никто не брал, информацию о нем выводить не надо.

Вариант 6.

1.Подсчитать число кассет каждого жанра.

2. Отыскать кассеты, которые поступили в прокат в последние три года.

3.Определить какой язык сейчас используется.

4.Создать функцию, которая по заданному году выводила число кассет, которые поступили.

5.Создать функцию, которая бы возвращала список кассет, которые на руках.

6.Создать функцию, которая бы для каждого фильма выводила список клиентов, которые его брали. Если фильм никто не брал, информацию о нем выводить не надо.

Вариант 7.

1. Отыскать среднюю стоимость билетов в заданном кинотеатре.
2. На текущую дату показать афишу кинотеатров.
3. Все символы в названии кинотеатра преобразовать в заглавные буквы.
4. Создать функцию, которая бы подсчитывало число фильмов, которые демонстрировались в заданном кинотеатре.
5. Создать функцию, которая бы показывала, где демонстрируется заданный фильм.
6. Создать функцию, которая бы отыскивала кинотеатры, в которых есть льготные сеансы, если таких сеансов нет, то выводить название кинотеатра и сообщение «льготных сеансов нет».

Вариант 8.

1. Подсчитать число блюд по заданному типу меню.
2. Отыскать блюдо, название которого состоит более чем из 7 символов.
3. Определить какой язык сейчас используется.
4. Создать функцию, которая бы определяла среднюю калорийность блюд заданного типа.
5. Создать функцию, которая бы определяла список блюд, в которые входит данный продукт
6. Создать функцию, которая бы выводила для каждого типа меню список блюд заданной калорийности.

Вариант 9.

1.Подсчитать число поставщиков, которые в текущем году не делали поставок.

2.Подсчитать число символов в наименовании товара с заданным кодом.

3.Вывести имя текущей базы данных.

4. Создать функцию, которая бы для заданного товара выводила список его поставщиков.

5.Создать функцию, которая бы по заданной дате выводит число выполненных поставок.

6.Создать функцию, которая выводит поставщиков, которые не делали поставок в текущем месяце. Если все поставщики работали, то вывести список товаров, которые не поставлялись в текущем месяце. Иначе вывести поставщиков и материалы, которые они поставляли.

Вариант 10.

1. Подсчитать число работающих адвокатов.

2. Все символы в фамилии адвоката преобразовать в заглавные буквы. преобразовать в заглавные буквы.
3. Определить имеет ли заданное поле допустимый числовой тип данных.
4. Создать функцию, которая бы для заданного адвоката подсчитывала число дел.
5. Создать функцию, которая бы выводила список завершенных дел.
6. Создать функцию, которая бы позволяла оценить эффективность защиты.

Вариант 11.

1. Используя функции даты и времени подсчитать среднее количество постояльцев за месяц.
2. Получить статистику по базе данных
3. Создать временную таблицу, в которой сохранить объединённые поля фио и дату последнего пребывания в гостинице.
4. Создать функцию, возвращающую свободные номера в гостинице (номер и его тип).
5. Создать функцию, которая по фамилии постояльца определяет число его пребываний в гостинице (скалярная функция).
6. Создать функцию, которая формирует бы на заданную дату счета на оплату.

Вариант 12.

1.Подсчитать число проектов в каждом отделе.
2. Отыскать сотрудников, которые имели выплаты в текущем месяце
3. Отыскать фамилию руководителя, имеющую наибольшую длину.
4.Создать функцию, которая для заданного проекта , возвращает его бюджет.
5.Создать функцию, которая по номеру отдела формирует список проектов.
6.Создать функцию, которая для заданного сотрудника по каждому проекту выводит сумму сделанных выплат за каждый указанный месяц.

Вариант13.

1. Подсчитать число спортсменов у каждого тренера.
2.Подсчитать число символов в фамилии тренера, имеющего наибольшее число спортсменов.
3.Создать функцию, которая бы по заданному месяцу выводила список соревнований
4. Создать функцию, которая для заданного тренера выводит список его спортсменов.
5. Создать функцию, которая возвращает число соревнований в заданном месяце.
6. Создать функцию, которая выводит список спортсменов, которые в текущем месяце участвовали в соревнованиях. Если соревнований в этом месяце не было, то выводить сообщение: в текущем месяце не было соревнований.

Вариант 14.

1. Подсчитать число лекарств заданной категории.
2.Подсчитать количество лекарства, поступившего в текущем месяце.
3.Отыскать лекарство, в названии которого наибольшее число символов.
4. Создать функцию, которая для заданного производителя выводит список поставляемых им лекарств.
5. Создать функцию, которая возвращает число поставок лекарств в заданном месяце.
6. Создать функцию, которая выводит список лекарств, которые поступили в текущем месяце. Если поставок в этом месяце не было, то выводить сообщение: в текущем месяце не было поставок.

Вариант 15.

1. Подсчитать число студентов на каждом факультете**.**
2. Отыскать студентов, родившихся в текущем месяце.
3. Отыскать фамилию наибольшей длины.
4.Создать функцию, которая для заданной группы возвращает число пропусков по неуважительным причинам.
5.Создать функцию, которая по номеру группы формирует список студентов.
6.Создать функцию, которая для заданной группы по каждому предмету выводит список студентов, имеющих число пропусков выше заданного.

КОНТРОЛЬ ЗНАНИЙ

Контрольные вопросы и тестовые задания к лабораторной работе №1.

1. Основные задачи этапа физического проектирования базы данных.
2. Для чего нужны вторичные индексы?
3. Что называют оптимизацией исполнения?
4. Атрибут в описании базы данных в физической реализации соответствует:

столбцу
таблице
экземпляру строки

5. В SQL Server существует множество типов данных. Тип данных smallint отличается от tinyint тем, что:

smallint имеет диапазон допустимых значений 0..255, a tinyint -2^15 .. 2^15-1
tinyint имеет диапазон допустимых значений 0..255, a smallint -2^15 .. 2^15-1
tinyint - десятичные числа с фиксированным количеством знаков до и после запятой, a smallint - целые числа 0..255

6. Свойство identity может быть установлено:

только для одного столбца в таблице
для нескольких столбцов в таблице
для всех целочисленных столбцов одновременно

7. Концепция сущностей и атрибутов в описании базы данных в физической реализации соответствуют:

сущности реализуются в виде экземпляров полей, а атрибуты – в виде экземпляров строк
сущности реализуются в виде столбцов, а атрибуты – в виде таблиц
сущности реализуются в виде таблиц, а атрибуты – в виде столбцов

8. При создании уникальных индексов для столбцов, которые допускают использование NULL-значений, SQL Server:

разрешит использование только одной строки со значением NULL
разрешит использование множества строк со значением NULL
не разрешает создавать уникальных индексов

9.Диаграмма базы данных может использоваться для:

создания новых баз данных
добавления новых таблиц
управления отношениями
управления доступом к базе данных
изменения существующих таблиц

10. Для обеспечения целостности базы данных следует

а) задать первичные ключи;
б) задать внешние ключи;
в) задать условия каскадного удаления данных;
г) обеспечить выполнение декларативных и процедурных ограничений целостности.

11.Основные правила целостности данных в реляционной модели.

Контрольные вопросы и тестовые задания к лабораторной работе № 2.

1.Для чего используются временные объекты?
2.Какие типы временных объектов вы знаете?
3.Как используется и создается временная переменная?
4.Каких типов данных могут быть временные переменные?
5.Как создается переменная табличного типа?
6.Каково назначение временных таблиц?
7.Каких типов бывают временные таблицы?
8.Как в SQL Server создаются циклы?
9.Как объединить команды в единый блок?
10.Определение временной переменной табличного типа идентично описанию
а) определению CREATE TABLE$
б) запрещается использовать NULL, CHECK,
в) разрешается использовать PRIMARY KEY, UNIQUE KEY, NULL, CHECK,
г) можно лишь задавать тип полей;

11. Какие типы данных допустимы для переменных?

12. Переменная табличного типа и временная таблица: общее и отличие.

13. Какие типы циклов используются в T-SQL
- с постусловием,
- с предусловием,
- с постусловием и предусловием
14. Данный сценарий

SELECT OilName, LatinName

INTO ##TempTable

FROM Oil
выполняет действия:

определяют новую виртуальную таблицу TempTable
создает локальную временную таблицу ##TempTable
создает глобальную временную таблицу ##TempTable

15. DECLARE var1 int, @var2 text, @var3 ntext.
В данном сценарии допущена ошибка:

все переменные обозначаются префиксом @(у переменной var1 нет префикса)
все переменные обозначаются БЕЗ каких-либо префиксов (у переменных var2 и var3 есть префикс @)
не допускается использование типа text
не допускается использование типа ntext
не допускается использование более двух переменных

16. Временные таблицы всегда создаются в системной базе данных:

tempdb
msdb
model

Контрольные вопросы и тестовые задания к лабораторной работе №3.

1. Что такое курсор? В чем отличие последовательного и скроллирующего курсора по описанию и по использованию?
2. Каково назначение и синтаксис оператора Declare?
3. Каково назначение и синтаксис оператора Open?
4. Каково назначение и синтаксис оператора Fetch?
5. Каково назначение и синтаксис оператора Close?
6. По какой из команд сервер выделяет память под курсор?
7. По какой из команд сервер начинает поиск строк запроса?
8. Чем заканчивается работа оператора Open?
9. Чем заканчивается работа оператора Fetch?
10. Характеризуйте состав и назначение спецификаций, связанных с перемещением скроллирующего курсора.
11. Что такое уровень изоляции? Какой уровень изоляции может использоваться при работе с последовательным и скроллирующим курсором?
12. Работа с курсором

а) аналогична работе с представлением;
б) в каждый момент времени пользователь работает только с одной строкой;
в) как с обычной таблицей, используя команды SELECT, INSERT, UPDATE, DELETE.

13.По какой из команд сервер выделяет память под курсор?
14. Чем отличаются операторы CLOSE и DEALLOCATE?

15. Какие опции команды FETCH используются при работе с последовательным курсором:: NEXT, LAST,PRIOR?
16.Команда OPEN

А) выделяет память под курсор,
Б) формирует результирующий набор,
В) извлекает данные из курсора.

7. Опишите последовательность работы с курсором. .

Контрольные вопросы и тестовые задания к лабораторной работе 4.

1. Хранимая процедура

а) всегда должна иметь параметры;
б) может не иметь параметров;
в) должна всегда иметь входные параметры;
г) должна всегда иметь выходные параметры.

2. Хранимая процедура может иметь дополнительные опции

а) OUTPUT,

б) VARYING,

в) RECOMPILE, ENCRYPTION, FOR REPLICATION,

г) OUTPUT, ENCRYPTION.

3. Входные параметры хранимой процедуры задаются

А) @ parameter data type,

б) VARYING,

в) OUTPUT VARYING.

4. Преимущества использования хранимых процедур в сравнении со сценариями.

5. Типы хранимых процедур.

Контрольные вопросы и тестовые задания к лабораторной работе № 5

1. Существует следующее число триггеров, каждый из которых реагирует на определенный тип операции:
а) 5,
б)2,
в) 3,
г) 4.
2.Для одной таблицы разрешается создавать
А) только один триггер каждого типа,
Б) множество триггеров каждого типа,
В) только один триггер одного типа
Г) множество триггеров типа AFTER и один типа INSTEAD OF.
3.Как выполнить триггер?
4. Триггер предназначен для
А) обеспечения целостности базы данных,
Б) управления доступом данных у базе данных,
В)выполнения операций модификации данных.

Контрольные вопросы и тестовые задания к лабораторной работе № 6

1. Пользовательская функция имеет следующие ограничения:

А) обязательно должна вернуть какое-то значение,
Б) разрешается использование команд PRINT, FETCH, UPDATE, INSERT. DELETE,
Г) запрещается использование команд UPDATE, INSERT. DELETE,

Д) разрешается использование команд , которые могут вернуть значение непосредственно в соединение, а не как результат вычисления функции.

2. Типы пользовательских функций.

3. Скалярная функция может использоваться

А) в предложении FROM,
Б) в выражениях,
В) в конструкции WHERE.

4. Хранимые процедуры и функции: общее и отличие.

ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

ЛИТЕРАТУРА

1. К. Дж. Дейт Введение в системы баз данных/ К. Дж. Дейт - 7 издание. – 2003- 674с.
2. Мейер Д. Теория реляционных баз данных. Мейер Д. - М.: Мир, 1987.- 608 с
3. Артре Ш. Структурный подход к организации баз данных./ Артре Ш. - Пер. с англ. - М.: Финансы и статистика, 1989.
4. Кузнецов С.Д. Базы данных: языки и модели: учебник / Кузнецов С.Д. - ISBN: 978-5-9518-023 - 2008 - 720 с.
5. Грофф Дж., Вайнберг П. SQL: полное руководство / Грофф Дж., Вайнберг П. - Пер. с англ. – К.: Издательская группа BHV, 2000. – 608 с.
6. Карпова Т. Базы данных. Модели, разработка, реализация/ Карпова Т. – СПб.: Питер, 2001. – 304 с.
7. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика/ Конноли Т., Бэгг К., Страчан А.- 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
8. Крёнке Д. Теория и практика построения баз данных/ Крёнке Д. - 8-е изд. – СПб.: Питер, 2003. – 800 с.
9. Мартин Грабер. SQL/ Мартин Грабер.- Издательство: Лори, 2007 - 672 с.
10. Ицик Бен Ган . Microsoft SQL 2008. Основы T-SQL / Ицик Бен Ган – СПб.: БХВ Петербург, 2009 – 432с:ил.
11. Браст Эндрю Дж., Форте Стивен. Разработка приложений но основе Microsoft SQL Server™ 2005.Мастер-класс/ Браст Эндрю Дж., Форте Стивен :Пер. с англ. – М.: Издательство «Русская Редакция», 2007. – 880 с.
12. Ульман Дж. Д., Уидом Дж. Введение в системы баз данных/ Ульман Дж. Д., Уидом Дж.: Пер. с англ. – М.: Лори, 2000. – 374 с.
13. Советов Б. Я., Цехановский В. В., Чертовский В. Д.. Базы данных. Теория и практика/ Советов Б. Я., Цехановский В. В., Чертовский В. Д - Издательство: М. Высшая школа, 2005 г., 464 с.
14. Грофф Дж., Вайнберг П. Энциклопедия SQL/ Грофф Дж., Вайнберг П. 3-е изд. СПб.: Питер, 2003
15. Джен Л. Харрингтон. Проектирование реляционных баз данных /Джен Л. Харрингтон. - Издательство: Лори, 2006 г., 240 с.
16. Системы управления базами данных: лабораторный практикум для студентов специальностей 1 40 01 01 «Программное обеспечение информационных технологий»/ сост. И. А. Бухвалова – Минск, БНТУ, 2006-56с.
17. Базы данных: методические указания к выполнению курсовой работы для студентов специальности 1 40 01 01 «Программное обеспечение информационных технологий»/ сост. И. А. Бухвалова – Минск, БНТУ, 2013-36с.