

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по выполнению лабораторных работ
по дисциплине «Тюнинг баз данных / Database tuning»
для магистрантов направления 09.04.03 «Прикладная
информатика» направленность (профиль) «Управление
знаниями»

Ставрополь
2026

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

ЛАБОРАТОРНАЯ РАБОТА №1 Индексные структуры

ЛАБОРАТОРНАЯ РАБОТА №2 Соединение таблиц данных

ЛАБОРАТОРНАЯ РАБОТА №3 Алгоритмы соединения таблиц

ЛАБОРАТОРНАЯ РАБОТА №4 Запросы с подзапросами

ЛАБОРАТОРНАЯ РАБОТА №5 Планы выполнения запросов

ЛАБОРАТОРНАЯ РАБОТА №6 Транзакции

ЛАБОРАТОРНАЯ РАБОТА №7 Работа с XML. Часть 1

ЛАБОРАТОРНАЯ РАБОТА №8 Работа с XML. Часть 2

ЛАБОРАТОРНАЯ РАБОТА №9 Иерархически организованные данные

ЛАБОРАТОРНАЯ РАБОТА №10 Секционирование таблиц

ЛАБОРАТОРНАЯ РАБОТА №11 Принципы построения статистической и динамической модели

ЛАБОРАТОРНАЯ РАБОТА №12 Конструкции языка XML

ЛАБОРАТОРНАЯ РАБОТА №13 Способы создания Web-сервисов

ЛАБОРАТОРНАЯ РАБОТА №14 Технологии создания и валидации XML-документов

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

ВВЕДЕНИЕ

Целью настоящего курса является формирование у магистрантов понимания основных тенденций развития ИС, связанных с изменениями условий в области применения, развитие профессиональных способностей в области прогнозирования, моделирования и создания информационных процессов в конкретной предметной области. Данный курс должен формировать у магистров знания и умения необходимые для правильного выбора инструментальных средств создания информационных систем, определения подходящей модели данных, организации запросов к хранимым данным и других моментов от которых во многом зависит эффективность разрабатываемых систем. В ходе изучения курса у магистра должно формироваться представление о перспективных информационных технологиях создания, анализа и сопровождения профессионально-ориентированных ИС.

В ходе достижения цели решаются следующие задачи:

- развитие логического и алгоритмического мышления;
- изучение принципов работы программно-технических средств и организации данных в информационных системах, использующих БД;
- освоение работы с современными СУБД;
- выработка умения самостоятельного решения задач обработки текстовой и нетекстовой информации в БД;
- получение навыков в алгоритмизации задач, программировании на алгоритмическом языке, отладке и выполнении задач на персональном компьютере;
- изучение перспектив развития информационных технологий в информационных системах в предметной области;
- изучение рынков информационных ресурсов и особенностей их использования.

ЛАБОРАТОРНАЯ РАБОТА №1 Индексные структуры

Тема - Индексные структуры

Цель работы - Освоить индексные структуры.

Формируемые компетенции или их части - ОПК-5

Теоретическая часть.

Индексы позволяют максимально эффективно находить информацию в огромных базах данных.

SQL Server 2008 поддерживает два базовых типа индексов: кластеризованные и некластеризованные. Индексы обоих типов реализуются как сбалансированное дерево (B-дерево), в котором уровень листьев находится на нижнем уровне структуры. Разница между индексами двух типов состоит в том, что кластеризованный индекс обеспечивает физическое упорядочивание данных на диске. Кластерный индекс является разреженным – указатели в листьях B-дерева ссылаются на страницу данных.

Некластеризованный индекс является плотным и содержит только столбцы, включенные в ключ индекса. В плотных индексах указатели в листьях B-дерева ссылаются на строки реальных данных. Если для таблицы не определен кластеризованный индекс, она называется кучей (heap) или неотсортированной таблицей. В последнем случае таблица физически организуется (отсортирована) в порядке добавления новых записей в отличие от таблиц с кластеризованными индексами, которые упорядочиваются по значениям ключа сортировки. Можно сказать, что таблица может быть представлена в одной из двух форм, в виде кучи или в виде кластеризованного индекса [1].

Кластеризованные индексы

Кластеризованные индексы можно создавать на основе одного или нескольких столбцов таблицы – такой индекс называется индексным ключом и у него есть ряд ограничений:

- индекс не может охватывать не более 16 столбцов;
- максимальный размер индексного ключа – 900 байт.

Столбцы кластеризованного индекса называются ключом кластеризации (clustering key). Кластеризованный индекс оказывает особое влияние на SQL Server, так как заставляет его упорядочивать данные в таблице согласно ключу кластеризации. Поскольку таблица может упорядочиваться лишь одним способом, в ней можно задать лишь один кластеризованный индекс.

Кластеризованные индексы задают порядок сортировки данных в таблице. Однако кластеризованные индексы не обеспечивают порядок физической сортировки. Кластеризованный индекс не приводит к физическому упорядочиванию данных на диске,

потому что это привело бы к большому числу операций дискового ввода-вывода при разбиении страниц. Он лишь гарантирует, что индексируемая цепочка страниц упорядочена логически, что позволяет SQL Server при поиске данных переходить прямо по цепочке страниц. В процессе движения сервера SQL Server по индексируемой цепочке страниц строки данных считываются в порядке ключа кластеризации [2].

Некластеризованный индекс

Некластеризованный индекс не накладывает никаких ограничений на упорядочивание записей в таблице, поэтому в одной таблице можно создать много некластеризованных индексов, но у этих индексов такие же ограничения, как и у кластеризованных индексов:

- индекс не может охватывать не более 16 столбцов;
- максимальный размер индексного ключа – 900 байт.

Конечный уровень некластеризованного индекса содержит указатель на нужные данные. Если в таблице есть кластеризованный индекс, конечный уровень некластеризованного индекса указывает на ключ кластеризации. Если же кластеризованного индекса нет, страницы конечного уровня указывают на строки данных в таблице [2].

Общий синтаксис создания реляционного индекса таков:

```
CREATE [UNIQUE] [CLUSTERED | NONCLUSTERED] INDEX имя_индекса
ON <объект> (column [ASC | DESC] [, ... n])
[INCLUDE (имя_столбца [, ... n] )]
[WHERE <предикат_фильтра> ]
[WITH (<параметры_реляционного_индекса> [, ... n] )]
[ON {имя_схемы_секции ( имя_столбца ) | имя_файловой_группы |
default }]
[FILESTREAM_ON { имя_файловой_группы_filestream |
имя_схемы_секции | «NULL»}][ ; ] [3]
```

Составной индекс

Составной индекс может быть создан на основании нескольких полей. В этом случае справедливы ограничения описанные ранее. Если индекс построен по полям с фиксированным размером, сумма длин этих полей должна не превышать эти 900 байт, если индекс построен по полям с переменной длиной, сумма максимальных размеров полей может превышать 900 байт, но само значение сумм по каждой записи не может быть больше 900 байт. Например, в таблице есть два поля переменной длины по 500 байт. SQL Server позволяет создать составной ключ на базе этих двух полей, если нет записей, сумма длин по обоим полям которых превышает 900 байт. Стоит обратить внимание на тот момент, что составной индекс для (Column1, Column2) является отличным от (Column2, Column1), а так же от индексов, созданных по двум этим полям в отдельности.

Фрагментация индексов

Файлы операционной системы обычно со временем фрагментируются из-за многократных операций записи. Индексы тоже могут становиться фрагментированными, но фрагментация индексов отличается от фрагментации файлов.

При создании индекса все значения ключа индекса записываются в упорядоченном виде на страницах индекса. При удалении строки из таблицы SQL Server должен удалить соответствующую запись в индексе, что создает "дыры" на странице индекса. SQL Server не возвращает освобожденное пространство из-за слишком высокой стоимости операции обнаружения и повторного использования "дыр" в индексе. Если значение в базовой таблице изменяется, SQL Server перемещает запись с указателем в другое место, что создает еще одну "дыру". При переполнении страниц индексов и потребности разбиения страниц снова происходит фрагментация индекса. Со временем индексы таблицы, в которых происходит изменение данных, становятся фрагментированными [2].

Для управления степенью фрагментации индекса обычно используют параметр, который называется коэффициентом заполнения (fill factor). Для устранения фрагментации можно так же задействовать инструкцию ALTER INDEX. Параметр fill factor - это параметр индекса, который определяет долю свободного пространства, которое резервируется на каждой странице конечного уровня при создании или перестроении индекса. Зарезервированное пространство позволяет в дальнейшем размещать дополнительные значения, снижая таким образом число разбиений страниц. Коэффициент заполнения измеряется в целых процентах, например значение 75 означает, что каждая создаваемая страница конечного уровня должно содержать 25% свободного пространства для размещения будущих значений [1].

Дефрагментация индексов

Поскольку SQL Server не возвращает пространство в систему, надо периодически освобождать пустое пространство в индексе, чтобы сохранить тот выигрыш в производительности, из-за которого индекс изначально создавался. Для дефрагментации индексов используют инструкцию ALTER INDEX [2].

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

```

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

```

При дефрагментации индексов можно выбрать параметры REBUILD или REORGANIZE.

Первый параметр перестраивает все уровни индекса и заполняет страницы в соответствии с параметром fill factor. При перестроении кластеризованного индекса перестраивается только он, однако если задать параметр ALL, будет перестроен как кластеризованный, так и все некластеризованные индексы таблицы. Перестроение индекса обновляет всю структуру сбалансированного дерева, поэтому, если не задан параметр ONLINE, таблица блокируется до завершения перестроения [2]. Например, для того, чтобы перестроить индекс IX_BillID, таблицы BillItem, необходимо выполнить следующий запрос:

```

ALTER INDEX IX_BillID
ON BillItem
REBUILD

```

Параметр REORGANIZE устраняет дефрагментацию только на конечном уровне. Страницы промежуточного уровня и корневая страница не дефрагментируются. Операция REORGANIZE всегда выполняется в оперативном режиме, поэтому не вызывает долгосрочной блокировки таблицы. [2] Например, чтобы реорганизовать индекс IX_BillID, таблицы BillItem, необходимо выполнить следующий запрос:

```

ALTER INDEX IX_BillID
ON BillItem
REORGANIZE

```

Работа с индексами в MS SQL Server Management Studio

Для того что бы посмотреть какие индексы созданы нужно, открыть вкладку Index таблицы Bill на панели Object Explorer. Полный путь до вкладки: Databases → EducationDatabase → Tables → [имя таблицы] → Indexes показан на рисунке 1.1. Согласно рисунку для данной таблицы создан один кластеризованный индекс PK_Bill.

Проверьте наличие кластеризованных индексов во всех таблицах базы данных самостоятельно.

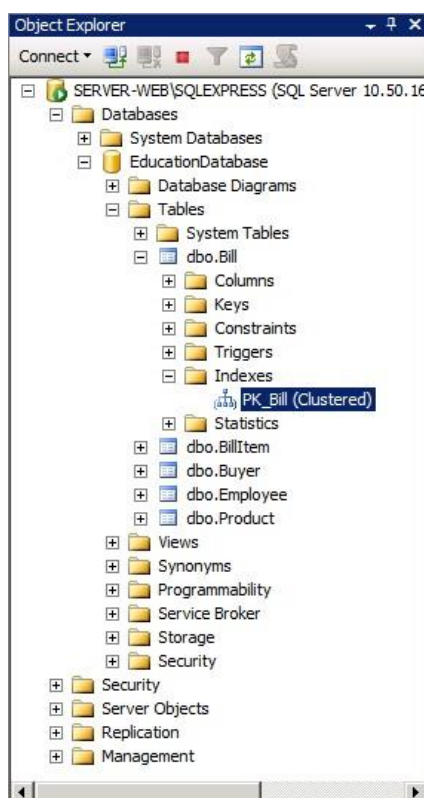


Рисунок 1.1 – Object Explorer, раскрытая вкладка Indexes

Создадим дополнительный индекс по полю внешнего ключа BillID таблицы BillItem. Создать индекс можно двумя способами:

Выполнение запроса CREATE INDEX. Создадим запрос в новой вкладке, нажав кнопку New Query стандартной панели инструментов. Панель инструментов показана на рисунке 1.2.



Рисунок 1.2 – Панель инструментов

После открытия новой вкладки, выполним запрос, показанный на рисунке 1.3. Для того что бы выполнить запрос, необходимо нажать кнопку Execute на панели инструментов (рисунок 1.2), или нажать клавишу F5 на клавиатуре.

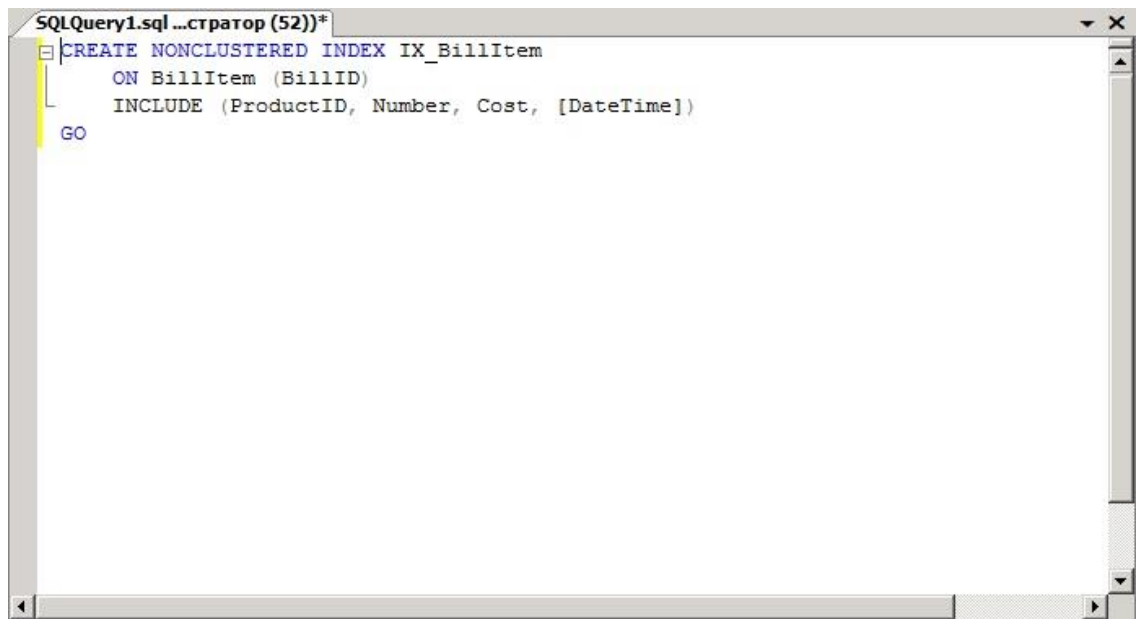


Рисунок 1.3 – Запрос CREATE INDEX

С помощью графического интерфейса Microsoft SQL Server Management Studio. В контекстном меню, вкладки Indexes выбираем пункт New Index, как показано на рисунке 1.4.

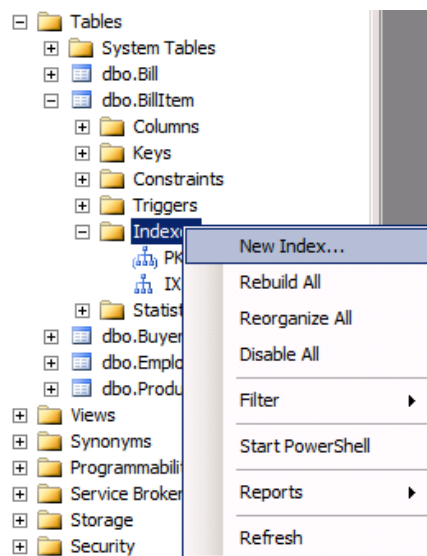


Рисунок 1.4 – Контекстное меню вкладки Indexes

В открывшемся окне необходимо указать имя индекса, атрибуты сортировки и тип индекса (кластерный, некластерный или первичный индекс XML). Если в таблице уже существует кластерный индекс, то при попытке создать новый кластерный индекс система выдаст предупреждение о возможности удаления существующего индекса и создания нового. При создании кластерного индекса происходит перестроение всех некластерных индексов.

Кроме того, в окне создания индекса можно указать признак поддержки уникальности значений в индексируемых полях. Наличие такого индекса будет препятствовать добавлению дублированных значений в индексируемые поля.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Проверьте наличие индексов по ключевым полям таблицы. При необходимости создайте кластеризованные индексы. Для создания нового индекса воспользуйтесь командой CREATE INDEX, или в среде Microsoft SQL Management Studio в разделе Tables/имя_таблицы/Indexes используйте команду New Index...
2. Создайте некластеризованные индексы по полям внешних ключей таблиц базы данных. Объясните, для чего нужны такие индексы?
3. Создайте некластеризованные индексы по информационным полям: Name и Date во всех таблицах базы данных. Объясните, для чего нужны такие индексы?
4. Для кластерного индекса и индекса по полю Date таблицы записей в чеке получите сведения о расширенных свойствах индексов. Объяснить значение информации, представленной в разделе «Fragmentation» на странице «Properties». Объясните, как вычислена глубина дерева индекса, число листьев, коэффициент фрагментации.
5. Перестройте кластеризованный индекс таблицы BillItem, используя команду ALTER INDEX или с помощью команды Rebuild в контекстном меню индекса.

6. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Контрольные вопросы:

1. Создайте некластеризованные индексы по полям внешних ключей таблиц базы данных. Объясните, для чего нужны такие индексы?
2. Создайте некластеризованные индексы по информационным полям: Name и Date во всех таблицах базы данных. Объясните, для чего нужны такие индексы?
3. Для кластерного индекса и индекса по полю Date таблицы записей в чеке получите сведения о расширенных свойствах индексов. Объяснить значение информации, представленной в разделе «Fragmentation» на странице «Properties». Объясните, как вычислена глубина дерева индекса, число листьев, коэффициент фрагментации.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.
2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №2. Соединение таблиц данных

Тема - Соединение таблиц данных

Цель работы - освоить соединение таблиц данных

Формируемые компетенции или их части - ОПК-5

Теоретическая часть. Оператор JOIN

Оператор JOIN реализует реляционную операцию соединения на основе предикатного правила. Само правило может иметь простой или сложный вид на основе композиции простых предикатных правил. Наиболее распространенным и практически значимым является, так называемое, эквисоединение, реализуемое бинарной операцией равенства атрибутов «=». Эквисоединение используется для восстановления без потери информации данных таблиц, подвергшихся процедуре нормализации и применения к ним операции проекции. Другие возможные варианты простых предикатов реализуются через операции: «<», «<=», «<>», «>=», «>». В общем случае не накладывается ограничение на именование столбцов соединяемых таблиц. Использование одинаковых имен столбцов первичного и внешнего ключа в связанных таблицах делает процесс управления данными более удобным и является рекомендуемой практикой. Несмотря на то, что фактические связи, реализованные с помощью создания ограничений PRIMARY KEY и FOREIGN KEY, не обязательны, для того чтобы результаты были значимыми, необходимо, чтобы по крайней мере один столбец в каждой из таблиц имел одно и то же значение (имеется в виду наиболее распространенный случай – эквисоединение) [1].

Если применяемый в запросе столбец с одним и тем же именем встречается в нескольких таблицах, в списке выбора или при перечислении столбцов в предложении WHERE и других предложения инструкции SELECT, следует указывать имя столбца вместе с именем таблицы. Например, столбец Name есть в таблицах Product, Employee и Buyer. Если вы пишете запрос, соединяющий эти таблицы, и хотите включить столбец Name в список выбора или в другую часть запроса, необходимо задавать столбцы как Employee.Name и т.д. Для того чтобы избавиться от длинного программного кода из-за длинных имен схемы и объектов, можно применять псевдонимы имен таблиц. В следующем примере показано использование псевдонимов.

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

Результатом запроса является выборка фамилий покупателя и работника магазина для каждой сделки. При этом фамилия покупателя выводится только в случае, когда она

была указана в сделке (соединение LEFT JOIN). В противном случае выводится значение NULL.

При определении условия оператора JOIN необходимо задать таблицы, которые будут соединяться, тип соединения и условие соединения, формируемого из столбцов

Существуют несколько типов соединения.

Внутреннее соединение (INNER JOIN)

Внутренние соединения возвращают только те строки, которые удовлетворяют условию соединения. Несмотря на то, что внутреннее соединение может быть приведено как в предложении FROM, так и предложении WHERE, рекомендуется задавать оператор JOIN в предложении FROM [1].

Следующий пример возвращает дату покупки и идентификатор чека из таблицы Bill (чек), идентификатор записи чека, идентификатор товара, количество купленных товаров, дату записи чека и стоимость из таблицы BillItem (записи чека). В этом примере псевдонимы применяются не только для имен таблиц, но и для имен столбцов, связано это с тем, что в обеих таблицах имеются столбцы DateTime и BillID. Обратите внимание на то, для остальных столбцов не требуются полные имена, поскольку их имена уникальны в обеих таблицах.

```
SELECT  B.BillID,  
        B.Date AS BillDT,  
        BillItemID,  
        ProductID,  
        Number,  
        Cost,  
        BI.Date AS BillItemDT  
FROM Bill AS B INNER JOIN BillItem AS BI ON B.BillID = BI.BillID
```

Поскольку заданно внутреннее соединение, строки из таблицы Bill, не имеющие соответствия в таблице BillItem, не возвращаются. Для преодоления такого поведения следует применять оператор OUTER JOIN.

Внешнее соединение (OUTER JOIN)

Внешнее соединение может применяться для возврата всех строк из одной таблицы и данных из строк второй таблицы, имеющих соответствие строкам первой, или для возврата всех строк из всех таблиц, заданных в предложении JOIN. Слово OUTER может быть пропущено в синтаксической записи, но обязательно следует указать LEFT, RIGHT или FULL [1]. Следует помнить, что использование внешних соединений определяет однозначный порядок соединения таблиц. Если аналогичный результат может быть получен с использованием внутренних соединений, рекомендуется отказаться от использования внешних.

Ключевые слова LEFT и RIGHT можно использовать для задания таблицы, из которой будут возвращены все строки. Когда вы задаете LEFT OUTER JOIN, возвращаются все строки из таблицы, указанной слева от ключевого слова JOIN. Эта таблица называется внешней. Вы можете добиться одного и того же результата как с помощью LEFT, так и с помощью RIGHT, изменив порядок указания имен таблиц. Например, следующие два запроса возвращают идентичный набор данных:

```
SELECT  B.BillID,  
        B.Date,  
        BU.Name  
FROM Bill AS B LEFT JOIN Buyer AS BU ON B.BuyerID = BU.BuyerID
```

```
SELECT  B.BillID,  
        B.Date,  
        BU.Name  
FROM Buyer AS BU RIGHT JOIN Bill AS B ON BU.BuyerID = B.BuyerID
```

Оператор FULL OUTER JOIN отображает каждую строку обеих таблиц из предложения JOIN. Это может быть полезно для поиска не имеющих соответствия строк, если в таблицах не реализована реляционная (ссылочная) целостность. Если в столбце соединения есть ограничения в виде внешнего ключа, оператор FULL OUTER JOIN обеспечивает тот же результат, что и оператор LEFT OUTER JOIN с таблицей, в которой внешний ключ определен слева от ключевого слова JOIN. Это объясняется тем, что для соблюдения ограничения у столбца внешнего ключа должна быть соответствующая строка для столбца первичного ключа. Кроме того, когда заданы ограничения внешнего ключа, оператор OUTER JOIN, определенный для таблицы с первичным ключом, заданной как внешняя таблица, обеспечивает те же результаты, что и оператор INNER JOIN.

Перекрестные соединения (CROSS JOIN)

Перекрестное соединение формирует результат, называемый декартовым произведением двух таблиц. В предложении JOIN каждая строка из первой таблицы соединяется с каждой строкой из второй таблицы. Этот тип соединения применяется лишь в исключительных ситуациях. Синтаксическая запись оператора CROSS JOIN такая же, как у других типов соединений за исключением того, что при этом соединении не требуется указывать условия соединения. Для того, чтобы получить выборку всех возможных комбинаций покупателей и продавцов, необходимо выполнить следующий запрос: [2]

```
SELECT  B.Name AS BuyerName,  
        E.Name AS EmployeeName  
FROM Buyer AS B CROSS JOIN Employee AS E
```

Представления

Некоторые из создаваемых вами инструкций SELECT используются для решения конкретной одноразовой задачи, а другие инструкции SELECT применяются в вашей рабочей среде снова и снова. Некоторые запросы, многократно используемые в вашей рабочей среде, содержат сложную бизнес-логику и сложный программный код на языке T-SQL, которые вы не хотите воспроизводить каждый раз, когда понадобится запрос. SQL сервер позволяет сохранять инструкцию SELECT в базе данных с помощью объекта, называемого представлением [1].

Представление – это простая инструкция SELECT с заданным именем, хранящаяся в базе данных. Главное преимущество представления состоит в том, что после создания оно действует как таблица в отношении всех остальных инструкций SELECT, которые вы захотите написать [1].

Синтаксис создания представления имеет следующий вид:

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

Определяемая для представления инструкция SELECT может ссылаться на таблицы, другие представления и функции, но не может делать следующего:

- содержать предложение COMPUTE или COMPUTE BY;
- создавать постоянную или временную таблицу с помощью ключевого слова INTO;
- применять предложение OPTION;
- ссылаться на временную таблицу;
- ссылаться на переменную любого типа;
- содержать переложение ORDER BY, если не задан параметр TOP.

Представление может содержать множественные инструкции SELECT, пока вы применяете операторы UNION или UNION ALL. [3]

Хранимые процедуры

Хранимые процедуры представляют уровень абстракции, ограждающий приложение от низкоуровневой структуры базы данных. Будучи главной опорой любого приложения SQL Server, хранимые процедуры позволяют изменять структуру баз данных и управлять производительностью без переписывания приложений и распространения их обновлений [2].

Хранимая процедура – это одна или несколько инструкций, которым дано имя и которые хранятся в базе данных. Почти любая команда языка T-SQL может быть

включена в хранимую процедуру, что делает процедуры подходящими для приложений и для решения административных задач. В хранимых процедурах не могут применяться лишь следующие команды:

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

При первом обращении к хранимой процедуре SQL Server генерирует планы компиляции и выполнения, которые хранятся в кэше запроса и повторно используются последующим выполнением. Следовательно, применяя хранимые процедуры, вы можете получить небольшой выигрыш в производительности, т.к. при последующих выполнениях хранимой процедуры исчезнет необходимость в синтаксическом анализе, компиляции и генерации плана запроса. Но главное назначение хранимой процедуры – создание защищенного слоя и прикладного интерфейса для ваших баз данных, которые отделяют приложение от изменения в структуре базы данных.

Далее приведена синтаксическая запись общего вида для создания хранимой процедуры [2].

```
CREATE {PROC | PROCEDURE} [schema_name.] procedure_name [;  
number]  
[ {@parameter [type_schema_name.]data_type}  
  [VARYING] [= default] [OUT | OUTPUT] [READONLY]  
] [, ... n]  
[ WITH <procedure_option> [, ... n]]  
[ FOR REPLICATION]  
AS {<sql_statement> [;][, ... n] | <method_specifier>} [;]
```

Содержание работы.

1. Создайте запрос, который будет формировать выборку данных о записях чека с включением данных о проданных товарах. В выборку включить поля: BillID, BillItemID,

ProductID, Product.Name, Number, Price, Cost, Bill.Date, BillItem.Date. Используйте внутреннее соединение таблиц (INNER JOIN). Примените условие фильтрации согласно своему варианту.

2. Создайте запрос, который будет формировать выборку данных с информацией о чеках, проданных товарах и покупателях со следующими полями: BillID, BillItemID, ProductID, BuyerID, Product.Name, Number, Price, Cost, Bill.Date, BillItem.Date, Buyer.Name, Photo. Используйте внешнее левое соединение от таблицы Bill к таблице Buyer (LEFT JOIN).

Объясните, что делает полученный запрос? Почему в этом случае используется внешнее соединение, а не внутреннее, как в предыдущем примере?

3. Создайте запрос, который будет формировать выборку данных с информацией о чеках, проданных товарах, покупателях и сотрудниках, расширив перечень полей в запросе из п.2. Используйте внешнее правое соединение от таблицы Bill к таблице Employee (RIGHT JOIN).

Объясните, что делает данный запрос?

4. Создайте запрос, который будет формировать выборку с данными по чекам и покупателям со следующими полями: BuyerID, BillID, Name, Date. Ограничьте выборку данных, используя фильтр по дате в таблице чеков (используйте фильтр в разделе WHERE). Используйте внешнее полное соединение таблиц Buyer и Bill (FULL JOIN).

Объясните, что делает этот запрос?

5. Создайте запрос, который будет формировать выборку, содержащую все возможные пары (сотрудник, покупатель) со следующими полями: EmployeeID, BuyerID, Buyer.Name, Employee.Name, Employee.Post. Используйте перекрестное соединение таблиц Employee и Buyer (CROSS JOIN).

6. Сохраните запрос из пункта 1 в виде представления. В среде Microsoft SQL Management Studio в разделе Views используйте команду New View... Назовите представление BillAndProduct. Проверьте результат командой: `SELECT * FROM BillAndProduct`.

7. Сохраните запросы из пунктов 2 и 3 аналогичным образом.

8. Запрос из пункта 4 сохраните в виде хранимой процедуры с одним параметром – датой. Для создания процедуры используйте команду CREATE PROCEDURE или воспользуйтесь командой New Stored Procedure в разделе Programmability/Stored Procedures. Назовите хранимую процедуру BillAndBuyer. Проверьте работоспособность процедуры, используя ее вызов: `exec BillAndProduct @date='2011-07-05'`.

9. Сохраните запрос из пункта 5 в хранимой процедуре.

10. Подготовьте материал для включения в отчетную презентацию по курсу Базы данных: специальный курс.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Фильтр по отдельному названию товара. Используйте одиночное условие фильтрации «=». Название товара выберите самостоятельно.
2. Фильтр по множеству названий товаров. Используйте условие фильтрации in. Перечень товаров выберите самостоятельно.
3. Фильтр по отдельной дате. Используйте одиночное условие фильтрации «=». Дату чека выберите самостоятельно.
4. Фильтр с ограничением диапазона по датам чека. Используйте фильтрацию по диапазону between или совместно «<=», «>». Диапазон дат выберите самостоятельно.
5. Фильтр по полуоткрытому интервалу дат чека. Используйте фильтрацию по условию «<=>» или «>=<». Диапазон дат выберите самостоятельно.

Контрольные вопросы:

1. Фильтр по множеству названий товаров. Используйте условие фильтрации in. Перечень товаров выберите самостоятельно.
2. Фильтр по отдельной дате. Используйте одиночное условие фильтрации «=». Дату чека выберите самостоятельно.

Фильтр с ограничением диапазона по датам чека. Используйте фильтрацию по диапазону between или совместно «<=», «=>». Диапазон дат выберите самостоятельно.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №3. Алгоритмы соединения таблиц

Тема - Алгоритмы соединения таблиц

Цель работы - освоить алгоритмы соединения таблиц

Формируемые компетенции или их части - ОПК-5

Теоретическая часть.

Один из наиболее важных аспектов настройки запроса – измерение или количественная оценка производительности. При измерении производительности нужно знать, что на самом деле измерять, т.е. какую систему показателей применять. В SQL Server есть три основных учитываемых показателя: затраты на запрос или стоимость запроса (query cost), число считываний страницы (page reads) и время выполнения запроса (query execution time).

Стоимость запроса

Стоимость выполнения запроса – это внутренняя характеристика, применяемая в SQL Server и учитывающая ресурсы ЦП и ввода/вывода, потребляемые запросом. Теоретически, чем меньше стоимость запроса, тем выше его производительность. На стоимость не влияют такие проблемы, как конфликт ресурсов или ожидание снятия блокировок. В большинстве случаев стоимость запроса – хорошая мера производительности, но если в запросе применяются особые элементы, например, скалярные пользовательские функции или программы общезыковой среды выполнения (Common Language Runtime, CLR), затраты на них не учитываются, что делает стоимость запроса ниже реальной. Поэтому стоимость выполнения запроса называют ориентировочной или приблизительной стоимостью запроса (estimated query cost).

Считывания страниц

Количество считываний страниц представляет количество 8-килобайтовых страниц данных, к которым обращался механизм запоминания SQL Server во время выполнения запроса. Извлечь эту характеристику можно с помощью команды SET STATISTICS IO ON. Эта команда приводит к выводу на вкладке окна запроса Message сведений, подобных приведенным далее:

```
Table 'Buyer'. Scan count 0, logical reads 526, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.  
Table 'Bill'. Scan count 1, logical reads 713, physical reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob read-ahead reads 0.
```

Общее количество считанных страниц в этом примере равно $526 + 713$, т.е. сумме значений, обозначенных как logical reads. Логические считывания - это количество

считанных из памяти страниц. Логические считывания представляют количества страниц данных, прочитанных из любого индекса таблицы *Bill*. Другие показатели сообщают о том, сколько логических считываний произведено с жесткого диска (*physical* и *read-ahead read*), количество выполненных сканирований индекса или кучи в ответ на запрос и количество считываний, потребовавшихся для извлечения данных типа *Large Object (LOB)*. Данные типа *LOB* хранятся вне строки с типами данных *varchar(max)*, *nvarchar(max)*, *varbinary(max)*, *text*, *ntext*, *image* и *XML*. Показатель считывания страниц не учитывает ресурсы ЦП, потребляемые во время выполнения запроса. Поэтому количество считываний страниц - обычно не столь подходящая мера производительности, как стоимость запроса. У этих показателей существует та же проблема, что и у стоимости запроса, связанная со скалярными пользовательскими функциями и подпрограммами на CLR и заключающаяся в том, что считывания, вызываемые этими объектами, не включаются в вывод параметра *STATISTICS IO*.

Время выполнения запроса

Время выполнения запроса – самая изменчивая характеристика. На нее влияют блокировки и конфликты ресурсов на сервере. С учетом этого особенно важно всегда включать в сравнение производительностей время выполнения запроса, потому что эта мера может помочь обнаружить проблемы, пропущенные другими показателями производительности. Если выполнить команду *SET STATISTICS TIME ON*, SQL Server вернет время выполнения всех запросов в миллисекундах.

Алгоритмы соединения

Оптимизатор запросов SQL Server обычно автоматически выбирает наилучший план выполнения запроса. Поэтому подсказки рекомендуется использовать только опытным пользователям и администраторам базы данных в случае крайней необходимости.

Без явного указания аргумента (*LOOP | HASH | MERGE*) оптимизатор выбирает, на его взгляд, самый оптимальный план. Но мы всегда можем повлиять на него, если явно укажем подсказку [3].

Соединения при помощи вложенных циклов

Соединение *LOOP JOIN*, называемое также *nested iteration*, использует одну таблицу в качестве внешней (на графическом плане она является верхней), а второй в качестве внутренней (нижней). *LOOP JOIN* построчно сравнивает внешнюю таблицу с внутренней. В цикле для каждой внешней строки производится сканирование внутренней таблицы и выводятся совпадающие строки.

В простейшем случае во время поиска целиком сканируется таблица или индекс (naive nested loops join). Если при поиске используется индекс, то такой поиск называется index nested loops join. Если индекс создается в качестве части плана запроса (и уничтожается после завершения запроса), то он называется temporary index nested loops join. Оптимизатор сам выбирает один из этих поисков.

LOOP JOIN является особенно эффективным в случае, когда внешняя таблица сравнительно невелика, а внутренняя гораздо больше и для неё существуют индексы. Планы соединений с вложенными циклами являются надежными, так как не требуют много памяти и не требуют места на диске. В большинстве запросов соединения с вложенными циклами будут наиболее эффективными и обеспечат предсказуемое поведение в случае изменения размеров соединяемых таблиц.

Пример запроса использующего алгоритм соединения вложенных циклов представлен далее:

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

Соединения слиянием

MERGE JOIN требует сортировки обоих наборов входных данных по столбцам слияния, которые определены предложениями равенства (ON) предиката соединения. Если мы имеем предикат соединения «B.EmployeeID = E.EmployeeID», то таблица B должна быть отсортирована по B.EmployeeID, а таблица E должна быть сортирована по E.EmployeeID.

Так как каждый набор входных данных сортируется, оператор Merge Join получает строку из каждого набора входных данных и сравнивает их. Например, для операций INNER JOIN строки возвращаются в том случае, если они равны. Если они не равны, строка с меньшим значением не учитывается, и из этого набора входных данных берется другая строка. Этот процесс повторяется, пока не будет выполнена обработка всех строк.

MERGE JOIN может поддерживать слияние «многие ко многим». В этом случае, при каждом соединении двух строк нужно сохранять копию каждой строки второго входного потока. Это позволяет, при последующем обнаружении в первом входном потоке дубликатов строк, воспроизвести сохраненные строки. С другой стороны, если будет ясно, что следующая строка первого входного потока не является дубликатом, от сохраненных строк можно отказаться. Такие строки сохраняются во временной таблице

базы tempdb. Размер дискового пространства, который для этого необходим, зависит от числа дубликатов во втором входном потоке.

MERGE JOIN «один ко многим» всегда будет эффективнее слияния «многие ко многим», поскольку для него не требуется временная таблица. Для того чтобы задействовать слиянием «один ко многим», оптимизатор должен иметь возможность определить, что один из входных потоков состоит из уникальных строк. Как правило, это означает, что у такого входного потока существует уникальный индекс или в плане запроса присутствует явным образом оператор (например, сортировка при DISTINCT или группировка), который гарантирует, что строки на входе будут уникальны.

MERGE JOIN — очень быстрая операция, но она может оказаться ресурсоемкой, если требуется выполнение операций сортировки. Однако на больших объёмах при наличии индексов и предварительной сортировке, соединение слиянием является самым быстрым из доступных алгоритмов соединения [4].

Пример запроса использующего алгоритм соединения слиянием:

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

Соединение хешированием

HASH JOIN – более эффективен при работе с большими наборами данных и даже тогда, когда таблицы не отсортированы по столбцам, по которым производится соединение. HASH JOIN распараллеливается и масштабируется лучше любого другого соединения и сильно выигрывает при большой производительности информационных хранилищ. Всегда адекватно оценивайте возможности распараллеливания, так как довольно часто издержки на поддержку процедуры распараллеливания соизмеримы со временем, которое экономится при параллельном выполнении операций.

Соединение происходит с использованием хеширования, вычисляя хеш записей из меньшей таблицы (Build-таблица) и вставляя их в хеш-таблицу, затем обрабатывается большая таблица (Probe-таблица) по одной записи, сканируя хеш-таблицу для поиска совпадений [4].

Пример запроса использующего алгоритм соединения слиянием:

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

Следует учитывать, что, не смотря на высокую производительность соединения слиянием и соединения хэшированием, они очень чувствительны к доступным аппаратным ресурсам и размерам соединяемых таблиц. В этом смысле наиболее предсказуемое и надежное поведение демонстрирует соединение с использованием вложенных циклов.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Запросы, созданные в пунктах 1,2 предыдущей лабораторной работы переделайте таким образом, чтобы для соединения таблиц использовался алгоритмы соединения с использованием вложенных циклов, соединения хэшированием и соединения слиянием.
2. Оцениваем время выполнения каждого из полученных запросов.
3. Полученные запросы сохраните в представлениях, добавляя к имени исходного представления имя используемого алгоритма.
4. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Контрольные вопросы:

1. Оцениваем время выполнения каждого из полученных запросов.
2. Полученные запросы сохраните в представлениях, добавляя к имени исходного представления имя используемого алгоритма.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №4. Запросы с подзапросами

Тема - Запросы с подзапросами

Цель работы - освоить запросы с подзапросами

Формируемые компетенции или их части - ОПК-5

Теоретическая часть. Вложенные запросы

Вложенным запросом называется запрос, помещаемый в инструкцию SELECT, INSERT, UPDATE или DELETE или в другой вложенный запрос. Подзапрос может быть использован везде, где разрешены выражения. В данном примере вложенный запрос используется в качестве выражения для столбца с именем MaxUnitPrice в инструкции SELECT.

Вложенный запрос по-другому называют внутренним запросом или внутренней операцией выбора, в то время как инструкцию, содержащую вложенный запрос, называют внешним запросом или внешней операцией выбора.

Многие инструкции языка Transact-SQL, включающие подзапросы, можно записать в виде соединений. Другие запросы могут быть осуществлены только с помощью подзапросов. В языке Transact-SQL обычно не бывает разницы в производительности между инструкцией, включающей вложенный запрос, и семантически эквивалентной версией без вложенного запроса. Однако в некоторых случаях, когда проверяется существование, соединения показывают лучшую производительность. В противном случае для устранения дубликатов вложенный запрос должен обрабатываться для получения каждого результата внешнего запроса. В таких случаях метод работы соединений дает лучшие результаты [1].

Несвязанные вложенные запросы

Основное назначение несвязанного вложенного запроса — дать возможность написать более динамичный код, не требующий от пользователя знания всех промежуточных значений, существующих в базе данных в текущий момент. Например, если вы хотите получить список сотрудников, совершивших хотя бы одну сделку, вам необходимо будет создать вложенный запрос, который будет возвращать идентификаторы сотрудников, отображенные в таблице Bill [3].

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

Подзапросы с ключевым словом EXISTS

Если вложенный запрос вводится с помощью ключевого слова EXISTS, то он выступает в роли проверки на существование. В предложении WHERE внешнего запроса проверяется факт существования строк, возвращенных вложенным запросом. Вложенный запрос не выдает никаких данных, а возвращает значение TRUE или FALSE.

Вложенный запрос, созданный с помощью ключевого слова EXISTS, имеет следующий синтаксис.

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

По представленному ниже запросу осуществляется поиск всех сотрудников, не совершивших ни одной продажи.

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

Чтобы понять результаты данного запроса, необходимо рассмотреть наименование каждого продукта. Это позволит узнать, возвращается ли в результате выполнения данного вложенного запроса хотя бы одна строка. Иными словами, выдается ли в результате проверки на существование значение TRUE.

Обратите внимание на то, что вложенные запросы, введенные с помощью ключевого слова EXISTS, отличаются от других вложенных запросов следующим образом.

Перед ключевым словом EXISTS не ставится имя столбца, константы или другого выражения.

Список выборки для вложенного запроса, введенного с помощью ключевого слова EXISTS, практически всегда состоит из знаков «звездочка» (*). Так как производится проверка только на предмет существования строк, удовлетворяющих заданным во вложенном запросе условиям, то нет необходимости выводить имена найденных столбцов.

Важность ключевого слова EXISTS обусловлена тем, что часто без него невозможно найти иное решение без вложенных запросов. Хотя некоторые запросы, созданные с помощью ключевого слова EXISTS, не могут быть выражены по-другому,

многие запросы для достижения подобных результатов используют IN или оператор сравнения, измененный с помощью ANY либо ALL.

Например, предыдущий запрос может быть задан с использованием оператора IN [3].

Подзапросы с оператором NOT EXISTS

Оператор NOT EXISTS работает так же, как и оператор EXISTS, за исключением того, что предложение WHERE, в котором используется этот оператор, выполняется, если вложенный запрос не возвращает ни одной строки [3].

Табличные переменные

Особый тип данных, который можно использовать для хранения результирующего набора с целью последующей его обработки. Тип table применяется, главным образом, для временного хранения набора строк, возвращаемого в качестве результирующего набора возвращающей табличное значение функции.

Функции и переменные могут быть объявлены как переменные типа table. Переменные table могут использоваться в функциях, хранимых процедурах и в пакетах.

Запросы, изменяющие переменные table, не создают параллельных планов выполнения запроса. При изменении очень больших переменных table или переменных table в сложных запросах может снизиться производительность. В подобных случаях целесообразно рассмотреть возможность использования временных таблиц. Запросы, которые считывают переменные table, не изменяя их, могут выполняться параллельно.

Использование переменных table дает следующие преимущества.

- Переменная table ведет себя как локальная переменная. Она имеет точно определенную область применения. Это функция, хранимая процедура или пакет, в котором она объявлена.

- Внутри этой области переменная table может использоваться как обычная таблица. Она может быть применена в любом месте, где используется таблица или табличное выражение в инструкциях SELECT, INSERT, UPDATE и DELETE. Однако переменная table не может быть использована в следующей инструкции:

```
SELECT select_list INTO table_variable
```

Переменные table автоматически очищаются в конце функции, хранимой процедуры или пакета, где они были определены.

- Ограничения CHECK, значения DEFAULT и вычисляемые столбцы в декларации типа table не могут вызывать пользовательские функции.

- При использовании в хранимых процедурах переменных `table` приходится прибегать к повторным компиляциям режее, чем при использовании временных таблиц.
- Транзакции с использованием переменных `table` продолжаются только во время процесса обновления соответствующей переменной `table`. Поэтому переменные `table` режее подвергаются блокировке и требуют меньших ресурсов для ведения журналов регистрации.

На переменных `table` нельзя создавать индексы и статистику. В некоторых случаях можно добиться повышения производительности за счет использования вместо табличных переменных временных таблиц, которые позволяют создавать индексы и вести статистический учет.

Можно создавать локальные и глобальные временные таблицы. Локальные временные таблицы видимы только во время текущего сеанса, а глобальные – во всех сеансах.

Имени локальной временной таблицы должен предшествовать префикс знака номера (`#table_name`), а имени глобальной временной таблицы — двойной знак номера (`##table_name`).

Инструкции SQL могут обращаться к временной таблице по заданному в инструкции `CREATE TABLE` значению аргумента `table_name`.

Если локальная временная таблица создается хранимой процедурой или приложением, которые одновременно могут выполняться несколькими пользователями, компонент Database Engine должен иметь возможность различать таблицы, созданные разными пользователями. Компонент Database Engine делает это путем внутреннего присоединения числового суффикса к имени каждой локальной временной таблицы. Полное имя временной таблицы, хранящееся в таблице **sysobjects** базы данных **tempdb**, состоит из имени таблицы, заданного инструкцией `CREATE TABLE`, и сформированного системой числового суффикса. Для обеспечения возможности добавления суффикса значение параметра `table_name`, определенного как имя локальной временной таблицы, не должно содержать более 116 символов.

Пример использования локальной временной таблицы:

```
CREATE TABLE #t (BuyerID INT PRIMARY KEY)
INSERT INTO #t
select top 1 BuyerID FROM Buyer

SELECT  B.BillID,
        B.Date,
        BU.Name
FROM #t AS T INNER JOIN Buyer AS BU ON T.BuyerID=BU.BuyerID
        INNER JOIN Bill AS B ON BU.BuyerID = B.BuyerID
```

DROP TABLE #t

В последнем запросе создается временная таблица и в нее добавляется идентификатор одного покупателя. В основном запросе производится выборка всех покупок указанного покупателя.

Временные таблицы автоматически удаляются при выходе за пределы области определения, если не удалять их явно с помощью инструкции DROP TABLE.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Выберите все чеки на определенную дату, которые не единственны и существуют другие чеки на данную дату. Для формирования запроса используйте оператор EXISTS.
2. Выберите всех покупателей, которые не делали покупки на определенную дату. Для формирования запроса используйте операторы NOT EXISTS.
3. Сохраните полученные запросы.
4. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Контрольные вопросы:

1. Создайте временную таблицу (команда CREATE TABLE) или переменную типа таблица (команда DECLARE TABLE) со структурой как у таблицы покупатель, в

которую добавьте сведения о пяти произвольных покупателях (используйте оператор INSERT). Выберите сведения о 5-ти добавленных покупателях и даты их покупок. Используйте соединение временной таблицы (переменной типа таблица) с таблицей чеков Bill. Для формирования запроса используйте оператор IN.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №5. Планы выполнения запросов

Тема - Планы выполнения запросов

Цель работы - освоить планы выполнения запросов

Формируемые компетенции или их части - ОПК-5

Теоретическая часть.

Среда SQL Server Management Studio является интерактивным графическим инструментом, который позволяет администратору или разработчику базы данных создавать запросы, выполнять несколько запросов одновременно, просматривать результаты, анализировать план запроса, а также предоставляет помощь, необходимую для улучшения производительности запросов. Параметры плана выполнения графически отображают способы получения данных, выбранные с помощью оптимизатора запросов SQL Server. Для графического отображения плана выполнения используются значки, представляющие выполнение определенных инструкций и запросов SQL Server вместо их табличного отображения, получаемого с помощью параметров SHOWPLAN_ALL или SHOWPLAN_TEXT инструкции SET языка Transact-SQL или с помощью XML-представления, получаемого при помощи инструкции SET SHOWPLAN_XML. Графическое отображение очень полезно для понимания показателей производительности запроса. Среда SQL Server Management Studio показывает, какой статистики не хватает, тем самым принуждая к оптимизации запросов за счет оценок качества выборки, а затем позволяет легко создать пропущенную статистику [3].

Использование параметров плана выполнения

Для того чтобы увидеть план выполнения запроса откройте или введите в редакторе запросов среды Management Studio сценарий на языке Transact-SQL, содержащий запросы, которые нужно проанализировать. После того как сценарий будет загружен в редактор запросов среды Management Studio, выберите, какой план выполнения показывать – расчетный или фактический, нажав кнопку Показать предполагаемый план выполнения (Display Estimated Execution Plan) или кнопку Включить действительный план выполнения (Include Actual Execution Plan) на панели инструментов редактора запросов. Панель инструментов редактора запросов показана на рисунке 5.1. Если нажать кнопку «Показать предполагаемый план выполнения», выполняется анализ сценария и формируется расчетный план выполнения. Если нажать кнопку «Включить действительный план выполнения», перед формированием плана выполнения необходимо выполнить сценарий. После того, как сценарий проанализирован и выполнен, выберите вкладку «План выполнения», чтобы просмотреть графическое представление результатов формирования плана.



Рисунок 5.1 - Панель инструментов редактора запросов.

Чтобы воспользоваться графическими возможностями плана выполнения в среде Management Studio и параметрами SHOWPLAN инструкции SET языка Transact-SQL, пользователи должны обладать достаточными разрешениями на выполнение инструкций и запросов языка Transact-SQL. Также пользователи должны обладать разрешением SHOWPLAN на все базы данных, в которых содержатся упомянутые в запросах объекты. Дополнительные сведения см. в разделе Безопасность инструкции Showplan [3].

Чтение результатов работы плана выполнения

Для просмотра плана выполнения перейдите на вкладку «План выполнения» (Execution Plan) в области результатов. Графическое представление результатов работы плана выполнения в среде SQL Server Management Studio читается справа налево и сверху вниз. Каждый проанализированный запрос в пакете отображается вместе с затратами на его выполнение в процентном отношении от общих затрат на выполнение пакета. На рисунке 5.2 показан план выполнения запроса:

```

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

```

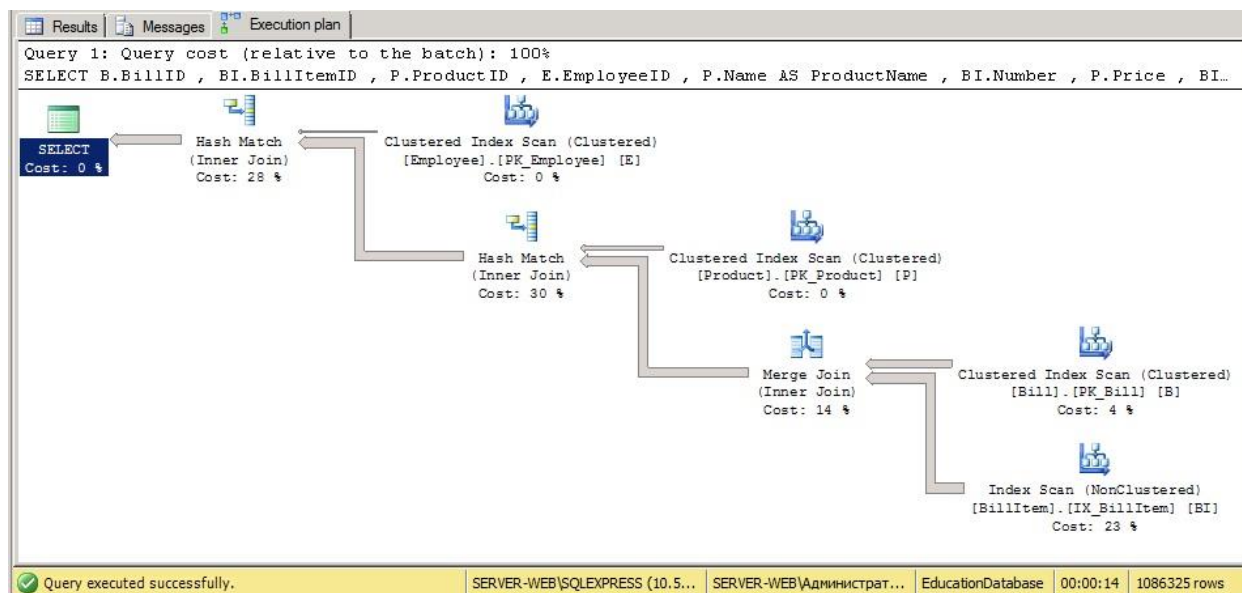


Рисунок 5.2 - Действительный план выполнения запроса.

Следующие описания являются руководством по интерпретации результатов графического отображения плана выполнения в среде Management Studio:

1. Каждый узел древовидной структуры представлен в виде значка, указывающего логический и физический оператор, используемый для выполнения этой части запроса или инструкции.
2. Каждый узел связан со своим родительским узлом. Дочерние узлы одного родительского узла отображаются в одном столбце. Однако все узлы в одном столбце не обязательно имеют общий родительский узел. Правила со стрелками на конце соединяют каждый узел с его родителем.
3. Операторы показаны в виде символов, связанных с определенным родительским узлом.
4. Ширина стрелки пропорциональна количеству строк. Если имеются данные о фактическом количестве строк, используются эти данные. В противном случае используется ориентировочное количество строк.
5. Если запрос содержит несколько инструкций, показывается несколько планов выполнения запроса.
6. Части древовидных структур определены типом выполняемой инструкции.
7. Для параллельных запросов, которые задействуют несколько процессоров, пункт меню Свойства для каждого узла в графическом плане выполнения отображает сведения об используемых потоках операционной системы. Чтобы просмотреть свойства узла, щелкните узел правой кнопкой мыши и выберите Свойства [2].

Всплывающие подсказки узлов графического плана выполнения

Каждый узел отображает во всплывающей подсказке, возникающей при наведении курсора на этот узел, сведения в соответствии с содержимым следующей таблицы. Не все узлы в графическом плане выполнения имеют всплывающие подсказки, которые описаны здесь.

Таблица 5.1 - Основные параметры узла графического плана выполнения запроса

Элемент	Описание
1	2
Физическая операция (Physical Operation)	Используемый оператор, такой как Hash Join или Nested Loops. Физические операторы, отображаемые красным цветом, показывают, что оптимизатор запросов выдал предупреждение, например предупреждение о нехватке статистики столбцов или отсутствии предикатов соединения. Это может привести к выбору оптимизатором плана запроса менее эффективного, чем тот, который ожидался. Дополнительные сведения о статистике столбцов см. в разделе Использование статистики для повышения производительности запросов . Если графический план выполнения предлагает создание или обновление статистики либо создание индекса, отсутствующая статистика или индексы могут быть немедленно созданы или изменены с использованием контекстных меню обозревателя объектов среды SQL Server Management Studio. Дополнительные сведения см. в разделе Инструкции по индексам .
Логическая операция (Logical Operation)	Логический оператор, который соответствует физическому оператору, такому как оператор Inner Join. Имя логического оператора приводится после физического оператора сверху всплывающей подсказки.
Предполагаемый размер строки (Estimated Row Size)	Предполагаемый размер строки, получаемой на выходе оператора (в байтах).
Предполагаемая стоимость операций ввода-вывода (Estimated I/O Cost)	Приблизительные затраты на выполнение действий ввода-вывода для данной операции. Это значение должно быть минимально возможным.
Предполагаемая стоимость процессного ресурса (Estimated CPU Cost)	Приблизительные затраты на произведение ЦП всех вычислений для данной операции.
Предполагаемая стоимость оператора (Estimated Operator Cost)	Затраты оптимизатора запросов на выполнение этой операции. Затраты на выполнение этой операции в процентном отношении к общим затратам на выполнение запроса отображаются в скобках. Так как ядро запросов выбирает наиболее эффективную операцию для выполнения запроса или инструкции, это значение должно быть минимально возможным.
Предполагаемая стоимость поддерева	Общие затраты оптимизатора запросов на выполнение этой и всех предшествующих операций в данном поддереве.

(Estimated Subtree Cost)	
Предполагаемое количество строк (Estimated Number of Rows)	Количество строк, выдаваемых оператором.

Значки графических планов выполнения

Значки, отображаемые в графическом плане выполнения среды SQL Server Management Studio, представляют собой операторы, используемые в SQL Server для выполнения инструкций. Перечень основных операторов представлен в таблице 5.2 [3].

Таблица 5.2 - Основные операторы графического плана выполнения запроса

Значок	Оператор	Описание
	Clustered Index Delete	Оператор Clustered Index Delete удаляет строки из кластеризованного индекса, заданного в столбце Argument плана выполнения запроса. Оператор Clustered Index Delete является физическим оператором.
	Clustered Index Insert	Оператор Clustered Index Insert инструкции Showplan вставляет строки со своего входа в кластеризованный индекс, указанный в столбце Argument. Оператор Clustered Index Insert является физическим оператором.
	Clustered Index Scan	Оператор Clustered Index Scan сканирует кластеризованный индекс, заданный в столбце Argument плана выполнения запроса. Clustered Index Scan является логическим и физическим оператором.
	Clustered Index Seek	Оператор Clustered Index Seek использует возможности поиска по индексам для получения строк из кластеризованного индекса. Столбец Argument содержит имя используемого кластеризованного индекса и предикат SEEK:(). Clustered Index Seek является логическим и физическим оператором.
	Clustered Index Update	Оператор Clustered Index Update обновляет входные строки кластеризованного индекса, указанные в столбце Argument. Clustered Index Update является логическим и физическим оператором.
	Hash Match	Оператор Hash Match строит хэш-таблицу при помощи вычисления хэш-значения для каждой строки из своих входных данных. Hash Match является физическим оператором.
	Merge Join	Оператор Merge Join выполняет внутреннее соединение, левое внешнее соединение, левое полусоединение, левое антиполусоединение, правое внешнее соединение, правое полусоединение, правое антиполусоединение, а также логические операции соединения. Оператор Merge Join является физическим.

	Nested Loops	Оператор Nested Loops выполняет логические операции внутреннего соединения, левого внешнего соединения, левого полусоединения и антилевого полусоединения. Nested Loops является физическим оператором.
	Nonclustered Index Delete	Оператор Nonclustered Index Delete удаляет входные строки из некластеризованного индекса, указанного в столбце Argument. Оператор Nonclustered Index Delete является физическим.
	Nonclustered Index Insert	Оператор Index Insert вставляет строки из входного потока в некластеризованный индекс, заданный в столбце Argument. Кроме того, столбец Argument содержит предикат SET:(), который указывает значение, заданное для каждого столбца. Index Insert — это физический оператор.
	Nonclustered Index Scan	Оператор Index Scan получает все записи некластеризованного индекса, указанного в столбце Argument. Если необязательный предикат WHERE:() появится в столбце Argument, то возвращаются только те строки, которые удовлетворяют условию, указанному в этом предикате. Оператор Index Scan является логическим и физическим оператором.
	Nonclustered Index Seek	Оператор Index Seek использует возможности поиска по индексам для получения строк из некластеризованного индекса. Index Seek является логическим и физическим оператором.
	Nonclustered Index Update	Физический оператор Nonclustered Index Update обновляет строки, указанные во входных параметрах, в некластеризованном индексе, заданном в столбце Argument. Если присутствует предикат SET:() , каждому обновленному столбцу присваивается указанное значение. Nonclustered Index Update является физическим оператором.
	Sort	Оператор Sort сортирует входящие строки. Столбец Аргумент содержит или предикат DISTINCT ORDER BY:(), если в этой операции удаляются дубликаты, или предикат ORDER BY:() с разделенным запятыми списком сортируемых столбцов. Столбцы имеют префикс со значением ASC, если они сортируются по возрастанию, или значением DESC, если сортируются по убыванию. Sort – это логический и физический оператор.

Текстовое представление плана выполнения запроса

Установка значения параметра SET SHOWPLAN_TEXT выполняется во время выполнения или запуска, а не во время синтаксического анализа.

Если выполнена инструкция SET SHOWPLAN_TEXT ON, SQL Server возвращает сведения для каждой инструкции Transact-SQL, не выполняя ее. Когда параметру присвоено значение ON, возвращаются сведения по планам выполнения всех последующих инструкций SQL Server, пока параметру не будет снова присвоено значение

OFF. Например, если инструкция CREATE TABLE будет выполнена при выполненной инструкции SET SHOWPLAN_TEXT ON, SQL Server вернет сообщение об ошибке в последующей инструкции SELECT, относящейся к той же таблице, сообщая пользователю, что указанная таблица не существует. Таким образом, последующие обращения к этой таблице вызовут ошибки. Если выполнена инструкция SET SHOWPLAN_TEXT OFF, то SQL Server выполняет инструкции, не создавая отчет с информацией по плану выполнения.

Инструкция SET SHOWPLAN_TEXT выводит данные в читаемом формате, предназначенном для приложений командной строки платформы Microsoft Win32. Инструкция SET SHOWPLAN_ALL возвращает более подробные данные, предназначенные для обработки программами, специально ориентированными на этот формат вывода.

Инструкции SET SHOWPLAN_TEXT и SET SHOWPLAN_ALL нельзя указывать в хранимой процедуре. Их можно указывать только как инструкции в пакете.

Инструкция SET SHOWPLAN_TEXT возвращает данные в виде набора строк, формирующего иерархическое дерево, которое представляет последовательность шагов, выполняемых обработчиком запросов SQL Server по ходу выполнения каждой инструкции. Каждая инструкция, отраженная в выходных данных, содержит одну строку с текстом инструкции, за которой следует несколько строк с подробностями этапов выполнения. На рисунке 5.3 показан текстовый план выполнения запроса рассмотренного ранее [2].

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

Рисунок 5.3 – Текстовый план выполнения запроса

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не

производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Для запросов, созданных в пунктах 1 и 2 лабораторной работы 2 и запросов, созданных в пункте 1, лабораторной работы 3 сформируйте графический и текстовый план выполнения запросов. Изучите планы выполнения и объясните порядок соединения таблиц.

2. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Контрольные вопросы:

1. Оцените трудоемкость отдельных узлов в дереве плана запроса. Выберите самые трудоемкие операции, изучите их и объясните, почему указанные операции самые трудоемкие по сравнению с остальными.

2. Оцените объем данных, передаваемых между операторами плана выполнения запроса.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №6. Транзакции

Тема - Транзакции

Цель работы - освоить транзакции

Формируемые компетенции или их части - ОПК-5

Теоретическая часть

Поскольку обработка данных – важнейшая функция в базах данных оперативной обработки транзакций (OLTP), в процессе управления и сохранения непротиворечивых данных транзакции играют важную роль.

Транзакции часто определяются как набор действий или операций, которые выполняются успешно или завершаются аварийно как единое целое. Более того, транзакции могут обеспечивать четыре основные характеристики процессов обработки данных, обращенных к базе данных.

- Атомарность – если в транзакцию вовлечены две или несколько порций данных, либо фиксируются все, либо ни одна.

- Согласованность – в конце транзакции либо существует новое адекватное представления данных, либо данные возвращаются в исходное состояние. Возвращение данных в исходное состояние – часть механизма отката, обеспечиваемого транзакциями SQL Server.

- Изоляция – во время выполнения транзакции (до того, как она будет зафиксирована или произойдет ее откат) данные должны быть изолированными и недоступными для других транзакций. Уровень изоляции можно контролировать в каждой транзакции.

- Длительность или долговечность – после фиксации транзакции окончательное состояние данных сохраняется, даже если сервер выходит из строя или перезагружается. Это свойство транзакции обеспечивается контрольными точками и процессом восстановления базы данных, выполняемым при запуске SQL Server.

Для обозначения этих четырех свойств используется обозначение ACID (Atomicity, Consistency, Isolation, Durability).

По умолчанию в SQL Server каждая инструкция INSERT, UPDATE и DELETE – это отдельная транзакция, которая фиксируется автоматически и не предоставляет возможности отката.

Вы можете активировать неявные транзакции в параметрах вашего соединения и механизм управления Database Engine автоматически начнет транзакцию при выполнении следующих команд: ALTER TABLE, CREATE, DELETE, DENY, DROP, FETCH, GRANT, INSERT, OPEN, REVOKE, SELECT, TRUNCATE TABLE или UPDATE.

Транзакция активна до тех пор, пока вы в ручную не введете инструкцию COMMIT или ROLLBACK. Активировать неявные транзакции можно применив инструкции SET IMPLICIT_TRANSACTIONS ON через прикладные программные интерфейсы (API) Object Linking and Embedding Database (OLE DB) или Open Database Connectivity (ODBC) на странице ANSI окна Query Options (Параметры запроса) в среде SSMS, или модифицировать свойства сервера, изменив его поведение по умолчанию на активацию неявных транзакций во всех соединениях, если в конкретном соединении они явно не установлены в положение OFF [1].

Определение явных транзакций

Явные транзакции обычно определяются в хранимых процедурах. Явная транзакция запускается, когда выполняется инструкция BEGIN TRANSACTION. Транзакция завершается при вводе инструкции COMMIT TRANSACTION или инструкции ROLLBACK TRANSACTION. После того как транзакция зафиксирована, SQL Server даже в аварийных ситуациях гарантирует, что данные записаны в базу данных. Инструкция ROLLBACK возвращает данные в состояние, предшествующее запуску транзакции. Несмотря на то, что инструкция ROLLBACK возвращает данные в прежнее состояние, некоторые компоненты, например, такие, как начальные значения для идентифицирующих столбцов, не восстанавливаются [1].

Особые сценарии ROLLBACK

Если транзакции вложены благодаря наличию в сеансе множественных инструкций BEGIN TRANSACTION, инструкция ROLLBACK выполняет откат для самой внешней из вложенных транзакций. Это справедливо даже если перед командой ROLLBACK, относящейся к внешней транзакции, встречаются инструкции COMMIT для транзакций на более низких уровнях вложенности. В следующем примере выполняется откат данных вплоть до транзакции, запущенной в строке 1, и вставляемая строка полностью удаляется из таблицы.

```
1. BEGIN TRANSACTION
2. INSERT INTO Employee
3. VALUES ("Новое имя", "Должность");
4.     BEGIN TRANSACTION
5.     UPDATE Buyer
6.     SET Name = "Егоров Анатолий Васильевич"
7.     WHERE BuyerID = 26;
8.     COMMIT TRANSACTION;
9. ROLLBACK;
```

Если вы хотите выполнить откат только части транзакции, можно задать точки сохранения с помощью инструкции SAVE TRANSACTION savepoint_name и затем

сослаться на имя точки сохранения в инструкции ROLLBACK. Делая это, вы сообщаете Database Engine об отмене изменений только до точки, в которой вы привели инструкцию SAVE TRANSACTION с тем же именем.

Если задается несколько точек сохранения с одним и тем же именем, инструкция ROLLBACK выполняет откат до самой последней инструкции ROLLBACK. Если вы хотите отменить транзакцию целиком, введите инструкцию ROLLBACK TRANSACTION с именем транзакции или вообще без имени. Помните о том, что использование инструкций ROLLBACK TRANSACTION без имени приводит к откату всех вложенных транзакций [1].

Установка уровней изоляции

С помощью инструкции SET TRANSACTION ISOLATION LEVEL можно установить следующие уровни изоляции транзакции.

READ UNCOMMITTED – разрешает инструкциям читать строки, обновленные любой транзакцией и еще не зафиксированные в базе данных. Этот уровень изоляции минимизирует вероятность конфликтов, но допускает «грязные» считывания (dirty reads) и неповторимые считывания.

READ COMMITTED – разрешает инструкциям в текущих соединениях и транзакциях производить неповторимые считывания, но запрещает «грязные» считывания (данные, обновленные транзакцией, открытой в другом соединении). Это уровень изоляции, принятый по умолчанию в SQL Server.

REPEATABLE READ – не разрешает транзакциям считывать незафиксированные измененные данные («грязные» считывания) и гарантирует сохранение совмещаемых блокировок до завершения текущей транзакции.

SNAPSHOT – требует, чтобы параметр базы данных ALLOW_SNAPSHOT_ISOLATION был равен значению ON. Этот уровень изоляции формирует моментальный снимок данных в момент их считывания в транзакцию, но не поддерживает блокировку данных. Могут выполняться обновления из другой транзакции, но текущая транзакция не видит этих изменений, отраженных в последующих считываниях исходных данных. Если текущая транзакция модифицирует данные, эти модификации видны только в текущей транзакции.

SERIALIZABLE – не разрешает считывать данные, которые были обновлены, но не зафиксированы другими транзакциями. Кроме того, никакая другая транзакция не может до завершения текущей транзакции обновлять данные, которые были считаны текущей транзакцией. Уровень изоляции SERIALIZABLE защищает от фантомных считываний, но создает наивысший уровень конкуренции и блокирования [1].

Синтаксическая запись команды определения уровня изоляции транзакции:

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

После того как в сеансе выполнена инструкция SET TRANSACTION ISOLATION LEVEL, все транзакции в пределах этого соединения используют заданный уровень изоляции [1].

Функции IDENT_CURRENT, SCOPE_IDENTITY, @@IDENTITY

Для того, что бы узнать последний добавленный идентификатор (IDENTITY), используются функции IDENT_CURRENT, SCOPE_IDENTITY и @@IDENTITY.

Функция IDENT_CURRENT практически аналогична функциям удостоверений SQL Server SCOPE_IDENTITY и @@IDENTITY. Все три функции возвращают созданные последними значения идентификаторов. Однако эти функции различаются областью действия и сеансом.

IDENT_CURRENT возвращает последнее значение идентифицирующего столбца, созданное для конкретной таблицы в любом сеансе и области поиска.

@@IDENTITY возвращает последнее значение идентификатора, созданное для любой таблицы в текущем сеансе по всем областям поиска.

SCOPE_IDENTITY возвращает последнее значение идентификатора, созданное для любой таблицы в текущем сеансе по текущей области поиска.

Функция IDENT_CURRENT возвращает значение NULL, если вызвана для пустой таблицы или для таблицы без столбца идентификаторов.

Инструкции и транзакции, приведшие к ошибке, могут изменять текущее удостоверение для таблицы и вызывать появление пропусков в значениях столбца идентификаторов. Откат удостоверения не производится никогда, несмотря на то, что транзакция, пытавшаяся вставить в таблицу значение, не была зафиксирована. Например, если инструкция INSERT привела к ошибке из-за нарушения ограничения IGNORE_DUP_KEY, текущее удостоверение для таблицы все равно увеличивается.

Будьте внимательны при использовании функции IDENT_CURRENT для прогнозирования следующего создаваемого значения идентификатора. В действительности создаваемое значение может отличаться от полученного с помощью функции IDENT_CURRENT плюс IDENTITY_SEED, потому что в других сеансах могут выполняться операции вставки [3].

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Добавьте в таблицу товар (Product) поле количество (Quantity). Заполните поле произвольными значениями от 0 до 100.
2. Создайте хранимую процедуру AddBill, которая будет формировать чек, а затем возвращать его идентификатор. Входные параметры процедуры: BuyerID и EmployeeID. Все инструкции данной хранимой процедуры должны выполняться в транзакции.
3. Сделайте выборку содержимого чека с использованием запроса №1 из лабораторной работы №2.
4. Проверьте работоспособность написанных хранимых процедур. Попробуйте создать чек и добавить в него 2 товара с достаточным в магазине количеством и 1 товар с недостаточным количеством. Убедитесь, что в чеке присутствуют только 2 проданных товара.
5. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Контрольные вопросы:

1. Создайте хранимую процедуру добавления товара в чек. Назовите ее AddBillItem. Входные параметры процедуры: BillID, ProductID, Quantity. Установить

уровень изоляции транзакции в режим `SERIALIZABLE`. После открытия транзакции (`BEGIN TRANSACTION`) необходимо проверить количество товара в магазине. В случае если товара в магазине недостаточно, то процедура должна завершить выполнение транзакции (`ROLLBACK TRANSACTION`). В противном случае хранимая процедура уменьшает доступное количество товара в магазине и добавляет запись о покупке в чек (добавляются записи в таблицу `BillItem`). Закрываем транзакцию (`COMMIT TRANSACTION`). Объясните выбор уровня изоляции транзакции.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №7. Работа с XML. Часть 1

Тема - Работа с XML

Цель работы - освоить работу с XML

Формируемые компетенции или их части - ОПК-5

Теоретическая часть. Структура XML–документа

XML-документ состоит из деклараций, элементов, комментариев, специальных символов и директив.

XML – это теговый язык разметки документов. Иными словами, любой документ на языке XML представляет собой набор элементов, причем начало и конец каждого элемента обозначается специальными пометками, называемыми тегами.

Элемент состоит из трех частей: начального тега, содержимого и конечного тега. Тег – это текст, заключенный в угловые скобки "<" и ">". Конечный тег имеет то же имя, что начальный тег, но начинается с косой черты "/". Пример XML–элемента:

```
<buyer>Сергей Довлатов</buyer>
```

Имена элементов зависят от регистра, т. е. <author>, <Author> и <AUTHOR> – это имена различных элементов. Наличие закрывающего тега всегда обязательно. Если тег является пустым, т. е. не имеет содержимого и закрывающего тега, то он имеет специальную форму:

```
<tag/>
```

Любой элемент может иметь атрибуты, содержащие дополнительную информацию об элементе. Атрибуты всегда включаются в начальный тег элемента и имеют вид:

```
имя_атрибута="значение_атрибута"
```

Атрибут обязан иметь значение, которое всегда должно быть заключено в одинарные или двойные кавычки. Имена атрибутов также зависят от регистра. Пример элемента, имеющего атрибут:

```
<employee post="Продавец">Сергей Довлатов</employee >
```

Элементы должны либо следовать друг за другом, либо быть вложены один в другой:

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

```

        <title>Часть речи</title>
        <author>Бродский, Иосиф</author>
        <present/>
    </book>
    <book isbn="0345374827">
        <title>Марш одиноких</title>
        <author>Довлатов, Сергей</author>
        <present/>
    </book>
</books>

```

Здесь элемент books (книги) содержит два вложенных элемента book (книга), которые, в свою очередь, имеют атрибут isbn и содержат три последовательных элемента: title (название), author (автор) и present (есть в наличии), причем последний пуст, т. к. в данном случае соответствует логическому флажку.

Из приведенного описания видно, что синтаксис XML напоминает синтаксис HTML (что естественно, т. к. оба они являются диалектами одного языка SGML), но требования к оформлению правильных XML-документов выше. Еще одним очень важным отличием XML от HTML является то, что содержимое элементов, т. е. все, что содержится между начальным и конечным тегами, считается данными. Это означает, что XML не игнорирует символы пробела и разрыва строк, как это делает HTML.

Любой XML-документ состоит из пролога и корневого элемента, например:

```

<?xml version="1.0"?>
<books>
    <book isbn="0345374827">
        <title>Марш одиноких</title>
        <author>Довлатов, Сергей</author>
        <present/>
    </book>
</books>

```

В этом примере пролог сводится к единственной директиве (первая строка документа), указывающей версию XML. За ней следует XML-элемент с уникальным именем, который содержит в себе все остальные элементы и называется корневым. Директива (processing instruction) – это выражение, заключенное в специальные теги "<?" и "?>", которое содержит указания программе, обрабатывающей XML-документ.

Стандарт XML резервирует только одну директиву <?xml version="1.0"?>, указывающую на версию языка XML, которой соответствует данный документ (второй версии XML пока нет). В действительности, эта директива несколько богаче и в самом общем виде выглядит так:

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

Здесь атрибут `encoding` задает кодировку символов документа. По умолчанию считается, что XML-документы должны создаваться в формате UTF-8 или UTF-16. Атрибут `standalone` говорит о том, содержит ли данный документ внешние разделы. Значение `yes` означает, что таких разделов нет, значение `no` – что они есть.

В общем случае, пролог может содержать также декларации типа документа.

XML-документы могут содержать комментарии, которые игнорируются приложением, обрабатывающим документ. Комментарии строятся по тем же правилам, что и в HTML:

- начинайте комментарий с символов "`<!--`",
- завершайте комментарий символами "`-->`",
- не используйте внутри комментария символов "`--`".
- Пример комментариев:

```
<!-- это комментарий -->  
<!-- а вот еще комментарий,  
занимающий более одной строки -->
```

Все имена элементов, атрибутов и разделов должны начинаться с буквы Unicode и состоять из букв, цифр, символов точки (.), подчеркивания (`_`) и дефиса (`-`). Единственное ограничение состоит в том, что они не должны начинаться с комбинации букв `xml` в любом регистре; подобные имена зарезервированы для будущих расширений языка. Существенно, что стандарт допускает использование в именах не только английских букв, но и любых других, хотя существующие XML-процессоры часто ограничены теми системами кодировок, которые в них заложены создателями.

XML документы могут быть хорошо сформированными (*well-formed*) и допустимыми (*valid*). [5]

Хорошо сформированные документы XML

Для того чтобы быть хорошо сформированным, документ должен следовать правилам синтаксиса, установленным для XML консорциумом W3C в рекомендации XML 1.0. Неформально «хорошо сформированный» означает главным образом то, что документ должен содержать один или более элементов, и один из них, корневой, обязан включать в себя все остальные элементы. Кроме того, для каждого элемента должны соблюдаться правила вложенности. Например, следующий документ не будет хорошо

сформированным, потому что закрывающийся тег `</Bill>` встречается после открывающегося тега `<Buyer>` для следующего элемента:

```
<?xml version="1.0" encoding="UTF-8"?>
< Document >
    <Bill>
        Чек
        <Buyer>
        </Bill>
        Покупатель
    </Buyer>
</Document> [6]
```

Валидные документы XML

Большинство браузеров проверяют XML документы на то, являются ли они хорошо сформированными. Некоторые браузеры могут так же проверить, валиден (допустим) ли документ. Документ XML валиден (допустим), если с ним связаны объявление типа документа (DTD – Document Type Declaration) или схема XML, и если документ удовлетворяет этим DTD или схеме. То есть DTD или схема задает набор правил для внутренней целостности самого документа, и если браузер может подтвердить, что документ удовлетворяет этим правилам, он является валидным [6].

Функция синтаксического анализа XML – документа

Процедура `sp_xml_preparedocument` считывает входной XML–документ, проводит его синтаксический анализ при помощи синтаксического анализатора MSXML (`Msxmlsql.dll`) и выдает проанализированный документ, готовый к потреблению. Проанализированный документ является древовидным представлением различных узлов в XML-документе: элементов, атрибутов, текста, комментариев и т. д.

Процедура `sp_xml_preparedocument` возвращает дескриптор, по которому можно обратиться к вновь созданному внутреннему представлению XML-документа. Дескриптор действует только в течение сеанса либо до тех пор, пока не будет аннулирован путем выполнения процедуры `sp_xml_removedocument`.

Проанализированный документ хранится во внутреннем кэше SQL Server. Средство синтаксического анализа MSXML использует одну восьмую всей памяти, доступной SQL Server. Во избежание проблем с недостатком памяти освободите память с помощью процедуры `sp_xml_removedocument`.

В целях обратной совместимости процедура `sp_xml_preparedocument` удаляет символы возврата каретки (`char(13)`) и перевода строки (`char(10)`) из атрибутов, даже если эти символы существенны.

Средство синтаксического анализа XML, вызываемое процедурой `sp_xml_preparedocument`, позволяет анализировать внутренние DTD и объявления сущностей. Так как злонамеренно созданные DTD и объявления сущностей могут быть использованы для проведения атак отказа в обслуживании (DoS), настоятельно рекомендуется не передавать напрямую XML-документы из сомнительных источников в процедуру `sp_xml_preparedocument`.

Для избежания атак методом расширения рекурсивных сущностей, процедура `sp_xml_preparedocument` ограничивает до 10 000 число сущностей, которые можно поместить в корневую сущность документа. Это ограничение не применяется к символьным и числовым сущностям. Ограничение позволяет хранить документы с большим количеством ссылок на сущности, но предотвращает рекурсивное расширение отдельной сущности в цепочку длиннее 10 000 расширений.

Процедура `sp_xml_preparedocument` ограничивает число одновременно открываемых элементов до 256.

Процедура `sp_xml_removedocument` удаляет встроенное представление XML-документа, заданного дескриптором документа и делает недействительным дескриптор документа [3].

Преобразование XML – документа в таблицу

Для преобразования XML – документа в плоскую таблицу необходимо использовать оператор OPENXML.

OPENXML предоставляет представление набора строк XML-документа. Так как OPENXML является поставщиком наборов строк, OPENXML применяется в инструкциях языка Transact-SQL, в которых могут быть использованы поставщики наборов строк, такие как таблицы, представления или функция OPENROWSET.

Синтаксис представлен ниже:

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

Парсинг XML – документов

Входной документ имеет следующий вид:

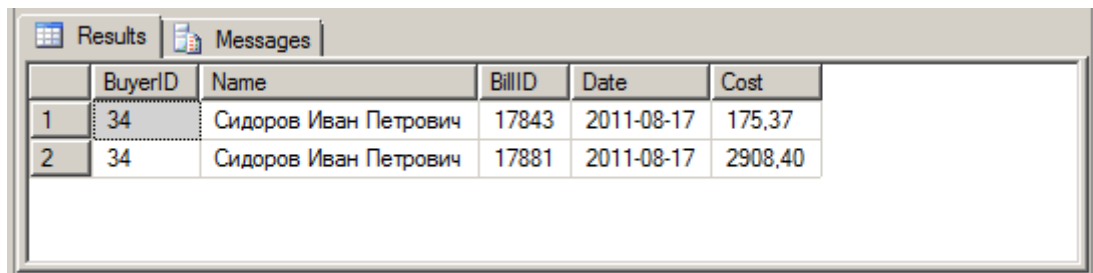
```
<root>  
  <buyer id="34" name="Сидоров Иван Петрович">  
    <bill id="17843" date="17.08.2011 14:47:19"  
cost="175.37"/>  
    <bill id="17881" date="17.08.2011 15:12:58"  
cost="2908.4"/>  
  </buyer>  
</root>
```

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

Для того что бы преобразовать данный документ в плоскую таблицу, содержащую поля: BuyerID, Name, BillID, Date, Cost, выполним следующий запрос:

```
DECLARE @idoc int
DECLARE @doc XML = '<root><buyer id="34" name="Сидоров Иван
Петрович"><bill id="17843" date="17.08.2011 14:47:19"
cost="175.37"/><bill id="17881" date="17.08.2011 15:12:58"
cost="2908.4"/></buyer></root>'
EXEC sp_xml_preparedocument @idoc OUTPUT, @doc
SELECT      *
FROM        OPENXML (@idoc, '/root/buyer/bill', 2)
            WITH (BuyerID int ' ../@id',
                  Name nvarchar(100) ' ../@name',
                  BillID int '@id',
                  [Date] Date '@date',
                  Cost Money '@cost')
EXEC sp_xml_removedocument @idoc;
```

Результат выполнения данного запроса показан на рисунке 7.1.



	BuyerID	Name	BillID	Date	Cost
1	34	Сидоров Иван Петрович	17843	2011-08-17	175.37
2	34	Сидоров Иван Петрович	17881	2011-08-17	2908.40

Рисунок 7.1 – Результат разбора XML – документа

Применение инструкций INSERT...SELECT

Применение инструкции INSERT...SELECT позволяет добавлять в существующую таблицу строки, основанные на данных, извлеченных из другой таблицы. В следующей команде выбираются строки из таблицы Employee и добавляются в таблицу Buyer. Поскольку количество и тип данных столбцов в результирующем наборе должны соответствовать целевой таблице, для получения корректного числа столбцов применяется сцепление.

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

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный

(x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Создайте новую хранимую процедуру AddBillXML. Входным параметром данной хранимой процедуры будет XML документ, пример которого показан ниже.

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

2. С помощью sp_xml_preparedocument, sp_xml_removedocument и OPENXML проведите разбор XML-документа, выделяя для записи чека поля: productId и quantity. Сформируйте плоскую временную таблицу с содержимым XML-документа.

3. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Контрольные вопросы:

1. После разбора XML-документа и формирования плоской таблицы откройте транзакцию для формирования нового чека. Создайте новый чек (используйте хранимую процедуру AddBill из л.р.№6). Добавьте в созданный чек с помощью инструкции INSERT...SELECT только те товары, которых имеется достаточное количество. С помощью инструкции UPDATE обновляем поле Quantity таблицы Product, вычитая товары, которые были куплены. Самостоятельно выберите уровень изоляции транзакции и прокомментируйте свой выбор.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №8. Работа с XML. Часть 2

Тема - Работа с XML

Цель работы - освоить работу с XML

Формируемые компетенции или их части - ОПК-5

Теоретическая часть. Предложение FOR XML

Запрос SELECT возвращает результаты в виде набора строк. При необходимости можно получать результаты SQL-запроса в формате XML. Для этого в запросе необходимо указать предложение FOR XML. Предложение FOR XML может использоваться в запросах верхнего уровня и во вложенных запросах. Предложение FOR XML верхнего уровня можно использовать только в инструкции SELECT. Во вложенных запросах предложение FOR XML можно использовать в инструкциях INSERT, UPDATE и DELETE. Оно также может использоваться в инструкциях присваивания.

В предложении FOR XML можно указать один из следующих режимов:

- RAW
- AUTO
- EXPLICIT
- PATH

В режиме RAW создается одиночный элемент <row> для каждой строки набора строк, возвращенного инструкцией SELECT.

--Простая выборка в формате XML из одной таблицы
SELECT EmployeeID, [Name] FROM dbo.Employee FOR XML RAW

Можно переименовывать элемент <row>, добавлять корневой элемент к выборке, переопределять структуру.

--Выборка данных из двух связанных таблиц с заданием корневого --элемента и переименовыванием <row>
SELECT e1.[Name], b1.Date, b1.BillID
FROM dbo.Employee e1
INNER JOIN dbo.Bill b1 ON b1.EmployeeID=e1.EmployeeID
FOR XML RAW('Empl'), ROOT('Employers')

--Значения столбцов форматируются как вложенные узлы при выборке из одной таблицы
SELECT ProdID, EmployeeID, Qty FROM Orders
FOR XML RAW('Ord'), ROOT('Orders'), ELEMENTS

--Значения столбцов дочерней таблицы выводятся как вложенные элементы
SELECT e1.[Name], e1.Post, b1.BillID, b1.Date
FROM dbo.Employee e1

```
INNER JOIN dbo.Bill b1(NOLOCK) ON b1.EmployeeID=e1.EmployeeID
WHERE b1.Date<'01.01.2011'
FOR XML RAW('Empl'), ROOT('Employers'), ELEMENTS
```

Режим FOR XML AUTO позволяет сформировать простейший XML фрагмент с вложенными элементами. Вложенность XML создается эвристически, в зависимости от метода определения инструкции SELECT. Управление формой создаваемой XML структуры минимально. Имена элементов и уровень вложенности генерируется на основе именования и отношений между объектами в БД.

```
--При выборке из одной таблицы с использование FOR XML AUTO
--Каждый узел в отличие от режима FOR XML RAW по умолчанию носит
--имя таблицы <Bill>
SELECT * FROM Bill FOR XML AUTO
```

```
--Автоматическое формирование уровней вложенности при выборке из
--двух связанных таблиц
SELECT Employee.[Name], Employee.Post, Bill.BillID, Bill.Date
FROM Employee
INNER JOIN Bill ON Bill.EmployeeID=Employee.EmployeeID
FOR XML AUTO--Если включить опцию ELEMENTS, то значения столбцов
--формируются как вложенные подэлементы
```

Для формирования простых вложенных выборок используется опция TYPE. Эта опция позволяет сгруппировать данные внутри одного элемента и сформировать их в соответствии с правилами режима FOR XML AUTO

```
--Данный запрос для каждого OrderID возвращает ProdID и Qty в
--виде вложенных элементов узла <Orders>
SELECT Bill.BillID, Bill.Date,
      (SELECT BillItem.Number, BillItem.Cost FROM BillItem
      WHERE BillItem.BillID=Bill.BillID FOR XML AUTO, TYPE)
FROM Bill
WHERE Bill.Date<'02.01.2010' AND Bill.EmployeeID=1
FOR XML AUTO, TYPE
```

Режим EXPLICIT предоставляет больше возможностей по управлению формой XML-структуры. В XML-структуре могут быть использованы смешанные атрибуты и элементы. Это требует особого формата для результирующего набора строк, создаваемого в результате выполнения запроса. Формат этого набора строк затем сопоставляется с формой XML-структуры. Преимущества режима EXPLICIT состоят в возможности использования смешанных атрибутов и элементов, в возможности создания упаковщиков и вложенных составных свойств, создания значений, разделенных пробелами (например, атрибут BillID может содержать список значений идентификаторов порядка), и смешанного содержимого.

Чтобы логика обработки запроса в режиме EXPLICIT могла создать желаемый XML необходимы следующие два столбца метаданных:

- первый столбец должен предоставлять номер тега текущего элемента (целочисленного типа), с именем столбца **Tag**. В запросе должен быть указан уникальный номер тега для каждого элемента, который будет создан из набора строк;
- второй столбец должен задавать номер тега для родительского элемента, и этот столбец должен иметь имя **Parent**.

Таким образом, столбцы **Tag** и **Parent** предоставляют сведения об иерархии.

Рассмотрим пример создания XML-структуры с помощью режима EXPLICIT.

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

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

Данный запрос создает XML-структуру с тремя уровнями иерархии. Для каждого уровня иерархии пишется свой запрос, а все вместе они объединяются с помощью UNION ALL.

Первый запрос извлекает значения для элемента <Employee> и его атрибутов. Запрос присваивает полю **Tag** значение 1 для элемента <Employee>, а полю **Parent** значение NULL, так как это элемент верхнего уровня.

Также в первом запросе указываются имена для столбцов в результирующем наборе строк. Эти имена образуют три группы столбцов. Группа со значением **Tag**, равным 1, в имени столбца задает <Employee> в качестве элемента, а EmpID и Name в качестве атрибутов. Вторая группа столбцов со значением Tag, равным 2, в имени столбца задает <Bill> в качестве элемента и Date в качестве атрибута. И третья группа соответственно задает элемент <Product> с атрибутами Name и Cost.

Второй запрос извлекает значения для элемента <Bill>. Столбцу **Tag** присваивается значение 2 для элемента <Bill>, а столбцу **Parent** — значение 1, в результате чего <Employee> используется в качестве родителя.

Третий запрос извлекает значения для элемента <Product>. Столбцу **Tag** присваивается значение 3 для элемента <Product>, а столбцу **Parent** — значение 3, в результате чего <Bill> используется в качестве родителя.

В итоге, значения в метастолбцах **Tag** и **Parent**, сведения, указанные в именах столбцов, и правильный порядок строк создают желаемый XML при использовании режима EXPLICIT.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного

обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Контрольные вопросы:

1. Написать запрос, возвращающий номер чека, общую сумму по чеку, ФИО сотрудника, дату чека; информацию о каждом товаре, название, цену, количество, стоимость. Результат вернуть в виде XML, представленной в предыдущей лабораторной работе.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №9. Иерархически организованные данные

Тема - Иерархически организованные данные

Цель работы - освоить иерархически организованные данные

Формируемые компетенции или их части - ОПК-5

Теоретическая часть. Подходы к организации иерархических структур

Любая достаточно сложная естественная или искусственная система характеризуется иерархической структурой – многоуровневой формой организации объектов с соответствием объектов нижнего уровня одному или нескольким объектам верхнего уровня. Примеры таких структур могут быть найдены в совершенно разных предметных областях: классификация товаров, контрагентов, комплектация изделия, иерархия должностей, административно-территориальное деление, генеалогическое древо, наконец, просто дерево перебора вариантов или дерево классов.

В связи с этим разработчикам программных продуктов все чаще приходится работать с иерархически организованными данными. В общем случае все сводится к моделированию многоуровневой связи «главный-подчиненный», «предок-потомок», «общий-конкретный», т.е. моделируется граф без циклов.

Существует 3 основных подхода к организации иерархических данных:

- родитель-потомок;
- метод вложенных множеств;
- модель материализованного пути.

Родитель-потомок

Метод родитель-потомок или модель списка смежных вершин (adjacency list model) – является одним из первых методов, разработанных для хранения иерархий в реляционных БД.

Это интуитивно понятный способ организации иерархической структуры, который представляет собой рефлексивную связь – замыкание связи таблицы БД на саму себя (Рисунок 1).

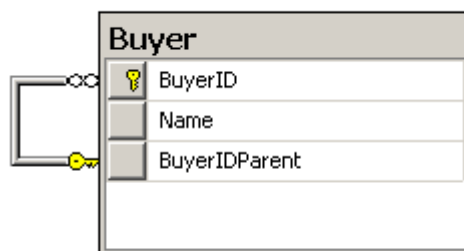


Рисунок 1. Пример рефлексивной связи

Как известно из теории, граф можно представить в виде матрицы, где на пересечении *i*-й строки и *j*-го столбца стоит 1, если между узлами (вершинами) графа с номерами *i* и *j* есть связь (ребро, дуга), или 0 в противном случае. Такая абстракция называется матрицей смежности.

Матрица смежности может быть также представлена в виде списка (множества) пар с номерами (идентификаторами, кодами) вершин по принципу: есть пара – есть связь, нет пары – нет связи.

Корневые вершины отличаются от других пар пустой (NULL) ссылкой на предка, в приведенном примере это поле «BuyerIDParent».

Выборка непосредственных родителей или потомков при такой организации иерархических данных представляет собой простой SQL-запрос, в то время как для выборки поддерева по заданному узлу потребуется рекурсивный запрос

```
--Выборка непосредственных потомков некоторого узла
SELECT * FROM dbo.Buyer where BuyerIDParent=1
--Выборка всех потомков некоторого узла
WITH BuyerChild(BuyerID, BuyerIDParent, [Name], [Level]) AS
(
    SELECT BuyerID, BuyerIDParent, [Name], 1
    FROM dbo.Buyer
    WHERE BuyerID=1
    UNION ALL
    SELECT dbo.Buyer.BuyerID, dbo.Buyer.BuyerIDParent,
    dbo.Buyer.[Name], [Level]+1
    FROM dbo.Buyer INNER JOIN BuyerChild ON
    BuyerChild.BuyerID=dbo.Buyer.BuyerIDParent
)
SELECT * FROM BuyerChild ORDER BY [Level], BuyerIDParent
```

Преимущества и недостатки модели списка смежных вершин приведены в Таблица 1.

Таблица 1. Преимущества и недостатки подхода родитель-потомок

Характеристика	Модель списка смежных вершин
Простота структуры (количество таблиц/ссылок/минимальное количество полей)	1/1/3
Прямая выборка детей узла	Да
Прямая выборка поддерева (всех потомков узла)	Нет, рекурсия
Прямая выборка пути от узла до корня (всех предков узла)	Нет, рекурсия
Порядок следования узлов при сортировке	Нет
Быстрая вставка новых узлов	Да
Быстрое перемещение поддерева	Да
Быстрое удаление поддерева	Да, рекурсия
Избыточность хранения	Нет
Количество уровней дерева	Не ограничено
Дополнительная поддержка целостности (кроме ссылочной)	Не нужна

Модель вложенных множеств

Модель вложенных множеств или множественная модель дерева (nested-set model of trees) была предложена Джо Селко и представляет собой хранение маршрута обхода графа в префиксном порядке (Рисунок 2).

Квадратик на рисунке обозначает узел, цифра в левом его углу является порядковым номером этапа маршрута при входе в узел, а цифра справа — при выходе. При префиксном порядке обхода сначала посещается корень графа, вершина «Федоров Егор Иванович», а затем выполняются префиксный обход поддеревьев с корнями «Кравцов Евгений Николаевич», ..., «Петько Татьяна Филиповна» в указанной последовательности.

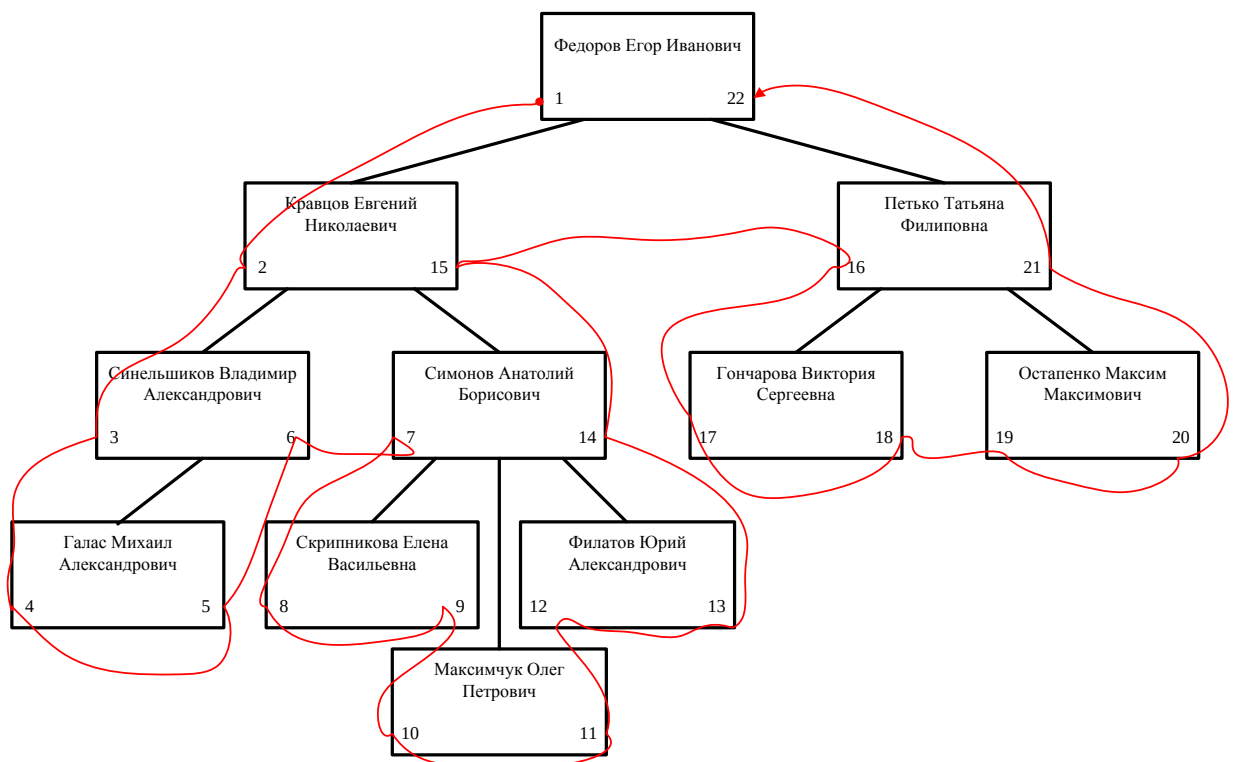


Рисунок 2. Пример префиксного маршрута обхода

Как нетрудно заметить, номера потомков всегда располагаются в интервале между соответствующими номерами предка, сколь угодно дальнего. Храня такой порядок обхода дерева в БД (Рисунок 3), при выборке потомков/предков некоторого узла можно избежать рекурсии

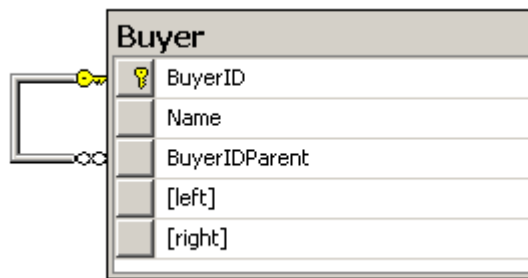


Рисунок 3. Пример таблицы с полями для хранения префиксного порядка обхода

```
--Выборка всех потомков некоторого узла
SELECT b2.* FROM dbo.Buyer b1 INNER JOIN dbo.Buyer b2 ON
b2.[left] BETWEEN b1.[left] AND b1.[right]
WHERE b1.BuyerID=1
```

Очевидная сложность модели вложенных множеств – это пересчет порядка обхода при добавлении новых или перемещении имеющихся узлов.

Для вычисления значений *left* и *right* вершин используется следующий алгоритм:

1. Обход начинается с корневой вершины. Значению *left* корня присваивается 1. Счетчику *p* присваивается 2.
2. Если для текущей вершины не существует ни одного прямого потомка или среди потомков нет таких, для которых значение *left* не присвоено, то значение *right* этой вершины равно p . $p = p + 1$. Переход на 5.
3. Прямому потомку текущей вершины, для которого значение *left* еще не было присвоено, назначается $left = p$. $p = p + 1$.
4. Текущей вершиной становится прямой потомок, которому было присвоено значение *left*. Переход на 2.
5. Если для текущей вершины существует родитель, то текущей вершиной становится этот родитель. Переход на 2. Иначе конец алгоритма.

Преимущества и недостатки множественной модели дерева приведены в Таблица 2. Таблица 2. Преимущества и недостатки множественной модели дерева

Характеристика	Модель списка смежных вершин
Простота структуры (количество таблиц/ссылок/минимальное количество полей)	1/0/4
Прямая выборка детей узла	Да
Прямая выборка поддерев (всех потомков узла)	Да
Прямая выборка пути от узла до корня (всех предков узла)	Да
Порядок следования узлов при сортировке	Да
Быстрая вставка новых узлов	Нет
Быстрое перемещение поддерев	Нет

Быстрое удаление поддерева	Да
Избыточность хранения	Нет
Количество уровней дерева	Не ограничено
Дополнительная поддержка целостности (кроме ссылочной)	Нужна, сложная

Модель материализованного пути

Модель материализованного пути (materialized path) – относительно молодое направление, при котором используется специальный ключ, сопоставляемый каждой вершине дерева и содержащий в себе информацию обо всех вершинах по пути (последовательности вершин, где каждая из вершин соединена с соседней дугой) из корня дерева в саму вершину (Рисунок 4).

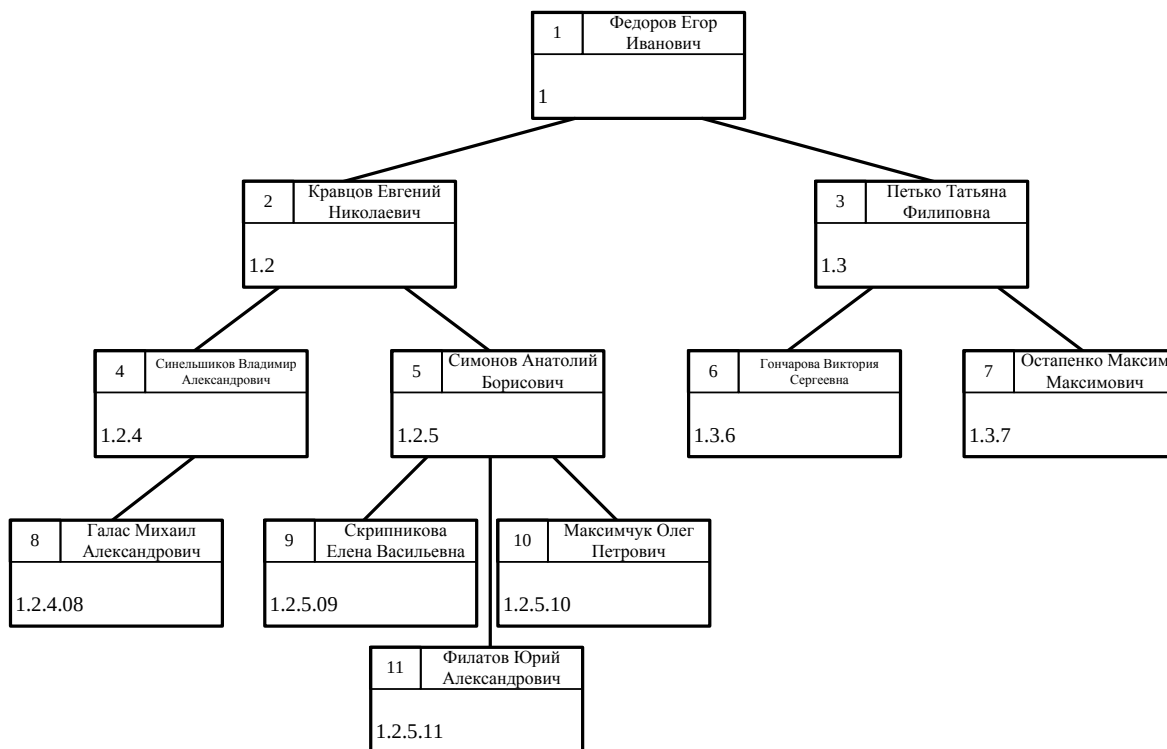


Рисунок 4. Пример префиксного маршрута обхода

Квадратик на рисунке обозначает узел, цифра в левом верхнем углу идентификатор вершины, в левом нижнем – материализованный путь, который содержит список номеров всех значений, которые нужно пройти, от корня иерархии до заданного значения. Каждый уровень иерархии разделяется символом ".".

Данный подход является наиболее наглядным с точки зрения кодификации элементов: каждый узел получает интуитивно понятное значение, сам код и его части несут смысловую нагрузку.

Запрос на выборку всех потомков некоторого узла при использовании модели материализованного пути будет выглядеть следующим образом

--Выборка всех потомков некоторого узла

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

В приведенном выше запросе поле «treeKey» хранит материализованный путь.

Преимущества и недостатки множественной модели материализованного пути приведены в Таблица 3.

Таблица 3. Преимущества и недостатки модели материализованного пути

Характеристика	Модель списка смежных вершин
Простота структуры (количество таблиц/ссылок/минимальное количество полей)	1/0/2
Прямая выборка детей узла	Да
Прямая выборка поддерева (всех потомков узла)	Да
Прямая выборка пути от узла до корня (всех предков узла)	Да
Порядок следования узлов при сортировке	Да
Быстрая вставка новых узлов	Да
Быстрое перемещение поддерева	Нет
Быстрое удаление поддерева	Да
Избыточность хранения	Нет
Количество уровней дерева	Ограничено
Дополнительная поддержка целостности (кроме ссылочной)	Нужна, не сложная

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Добавить в таблицу сотрудники столбец менеджер (внешний ключ к коду сотрудника). Добавить данные в таблицу так, чтобы все существующие сотрудники были

подчинены 2-м новым менеджерам (каждый продавец только одному своему менеджеру), а 2 новых менеджера в свою очередь подчинены директору магазина.

2. Добавить в таблицу 2 новых столбца (правый ключ, левый ключ). Написать запрос на вычисление ключей и ввод ключей по методу Джо Селко.

3. Добавить в таблицу 1 новый столбец (материализованный путь). Написать запрос на вычисление и обновление материализованного пути.

4. Написать запрос на выборку всех подчиненных (на всех уровнях) некоторого сотрудника для каждого из трех способов представления иерархии.

Контрольные вопросы:

1. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №10. Секционирование таблиц

Тема - Секционирование таблиц

Цель работы - освоить секционирование таблиц

Формируемые компетенции или их части - ОПК-5

Теоретическая часть.

Секционирование таблиц (горизонтальное секционирование таблиц) и индексов применяется с целью повышения производительности, масштабируемости и управляемости в больших базах данных (VLDB – very large databases). VLDB принято называть базы данных, совокупный размер которых измеряется сотнями гигабайт, а возможно терабайтами и больше. Большая таблица – это такая таблица, показатели производительности или временные затраты на обслуживание которой выходят за допустимые рамки. Кроме того, таблицу можно считать большой, если действия, производимые одним пользователем, оказывают значительное влияние на другого пользователя, или если операции обслуживания базы данных влияют на возможности остальных пользователей.

В многопроцессорных системах разделение больших таблиц приводит к улучшению производительности за счет параллелизма. Крупномасштабные операции над чрезвычайно большими наборами данных (в миллионы строк) могут извлечь выгоду из параллельной обработки независимых подмножеств данных. В качестве простого примера улучшения производительности при использовании секционирования может выступать агрегирование (группировка). Вместо того чтобы группировать данные в одной большой таблице, SQL Server может группировать их в нескольких секциях независимо друг от друга, и затем объединить агрегаты.

В SQL Server связанные таблицы (к примеру, Bill и BillItem), которые разделены по одному и тому же ключу секционирования (в нашем случае BillId) и одной и той же функции секционирования, называются выровненными. Если оптимизатор SQL Server обнаруживает, что соединяются (join) две секционированные и выровненные таблицы, он может предпочесть соединить сначала данные, располагающиеся в одних и тех же секциях, а затем объединить результаты. Это позволяет SQL Server более эффективно использовать многопроцессорные системы.

Секционирование делает большие таблицы и индексы более управляемыми, так как позволяет быстро и эффективно получать доступ к поднаборам данных и управлять ими, при этом сохраняя целостность всей коллекции. При использовании секционирования загрузка данных из системы OLTP в систему OLAP занимают значительно меньше времени по сравнению с использованием несекционированных таблиц и индексов.

Данные секционированных таблиц и индексов подразделяются на блоки (секциям), которые могут быть распределены по нескольким файловым группам в базе данных. Данные секционируются горизонтально, поэтому группы строк сопоставляются с отдельными секциями. Таблица или индекс рассматриваются как единая логическая сущность при выполнении над данными запросов или обновлений. Все секции одного индекса или таблицы должны находиться в одной и той же базе данных.

Секционированные таблицы и индексы поддерживают все свойства и возможности, связываемые с разработкой и опрашиванием стандартных таблиц и индексов, включая ограничения, значения по умолчанию, значения идентификации и отметок времени, а также триггеры.

Перед секционированием таблицы или индекса необходимо запланировать создание следующих объектов базы данных:

- функции секционирования;
- схемы секционирования.

Функция секционирования на основе значений некоторых столбцов, называемых столбцами секционирования, определяет, каким образом строки в таблице или индексе распределяются по секциям.

Схема секционирования сопоставляет каждую из секций, определенных функцией секционирования, с файловой группой.

При планировании функции секционирования следует исходить из двух условий: столбцов, значения которых определяют то, каким образом секционируется таблица, то есть так называемых столбцов секционирования, и диапазона значений столбца секционирования для каждой из секций. Этот диапазон определяет число секций, на которые разбивается таблица. В MS SQL Server у таблиц может быть не более 1000 секций. Например, в качестве функции секционирования таблицы Bill может быть выбрано разбиение по месяцам продаж по столбцу Bill.Date.

При частом выполнении запросов на эквивалентное соединение двух и более секционированных таблиц, их секционированные столбцы должны совпадать со столбцами, по которым производится соединение. Дополнительно: таблицы или их индексы должны быть упорядочены. Это означает, что в них используется либо общая именованная функция секционирования, либо разные, но дающие одинаковый результат. То есть:

1. Указанные таблицы секционированы по одинаковому количеству параметров, имеющих одинаковый тип данных.
2. В указанных таблицах имеется одинаковое количество секций.

3. В указанных таблицах секции имеют одинаковые граничные значения.

При выполнении указанных условий оптимизатор запросов SQL Server проводит операцию соединения гораздо быстрее, так как в этом случае могут быть соединены сами секции. Если запрос проводит соединение двух неупорядоченных таблиц, либо таблиц, не секционированных для соединения, наличие секционирования в таблице может замедлить выполнение запроса вместо его ускорения.

Для повышения производительности операций ввода-вывода может понадобиться объединить секции в файловые группы, размещенные на разных физических дисках. В процессе сортировки данных для операций ввода-вывода в SQL Server сначала проводится сортировка данных по секциям. При такой схеме работы SQL Server может осуществлять одновременный доступ только к одному диску, что может привести к снижению производительности. Для повышения производительности рекомендуется распределить файлы данных разных секций по нескольким жестким дискам, установив при этом RAID. Таким образом, несмотря на сортировку данных по секциям, SQL Server сможет одновременно осуществлять доступ ко всем жестким дискам каждой секции. Данная конфигурация может быть применена как к секциям внутри одной файловой группы, так и к секциям, размещенным в нескольких файловых группах.

Алгоритм секционирования

Чтобы создать секционированную таблицу или индекс, необходимо выполнить следующие шаги:

1. Создайте функцию секционирования, чтобы указать, каким образом таблица или индекс, где используется эта функция, могут быть секционированы.
2. Создайте схему секционирования, чтобы указать размещение секций функцией секционирования для файловых групп.
3. Создайте таблицу или индекс с использованием схемы секционирования.

Функция секционирования указывает, как секционируется таблица или индекс. Посредством этой функции домен преобразуется в набор секций. Чтобы создать функцию секционирования, необходимо указать число секций, столбец секционирования и диапазон значений столбца секционирования для каждой секции. Обратите внимание на то, что можно указать только один столбец секционирования. Все типы данных, допустимые для использования в качестве индексных столбцов, могут использоваться как столбцы секционирования, за исключением timestamp. Не могут быть указаны типы данных ntext, text, image, xml, varchar(max), nvarchar(max) и varbinary(max). Кроме того, нельзя указывать столбцы с данными пользовательского типа и типа alias среды CLR Microsoft.NET Framework.

Функция секционирования имеет следующий синтаксис:

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

где,

`input_parameter_type` – тип данных столбца, используемого для секционирования;

`boundary_value` – граничные значения для каждой секции;

`LEFT | RIGHT` – указывает, к какой области интервала значений принадлежит аргумент *boundary_value* [,...n] (к левой или правой).

Схема секционирования сопоставляет секции, полученные в результате использования функции секционирования, с определенным набором файловых групп.

При создании схемы секционирования пользователь определяет файловые группы, с которыми сопоставляются табличные секции, в соответствии с аргументами функции секционирования. Необходимо указать достаточно файловых групп для размещения соответствующего количества секций. Можно указать сопоставление всех секций с различными файловыми группами, сопоставление некоторых секций с одной группой файлов или сопоставление всех секций с одной файловой группой. Можно также указать дополнительные, «неназначенные» файловые группы в случае, если в дальнейшем предполагается добавить еще секции. В этом случае SQL Server помечает одну из файловых групп свойством NEXT USED. Это означает, что файловая группа будет содержать следующую добавленную секцию.

Для схемы секционирования может быть использована только одна функция секционирования. Однако функция секционирования может участвовать в двух и более схемах секционирования.

Функция создания схемы секционирования имеет синтаксис:

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

где

`partition_function_name` – имя функции секционирования, использующей схему секционирования;

ALL – указывает, что все секции сопоставляются с файловой группой, определяемой аргументом *file_group_name*, или с первичной файловой группой, если указывается [PRIMARY];

file_group_name – указывает имена файловых групп, содержащих секции, указываемые аргументом *partition_function_name*. Если указывается [PRIMARY], секция сохраняется в первичной файловой группе.

Создание секционированных таблиц

В учебной базе данных EducationDatabase накапливаются данные по продажам; текущим (последним обрабатываемым) годом продаж является 2011. Проведем секционирование таблицы Bill. Атрибутом секционирования выберем дату продажи товара Date. Каждая секция будет содержать сведения о продажах за определенный месяц 2011 года, за исключением последней секции, которая будет содержать сведения о продажах в последнем месяце 2011 года и позже. Дополнительно одна секция будет содержать продажи, совершенные до 2011 года.

Порядок секционирования таблиц:

1. Для таблицы Bill в контекстном меню выберите раздел Storage -> Create Partition. Откроется окно мастера секционирования.
2. На первом шаге работы мастера выберите столбец таблицы, на основе которого будет производиться секционирование. Выберем столбец Date (рисунок 1).

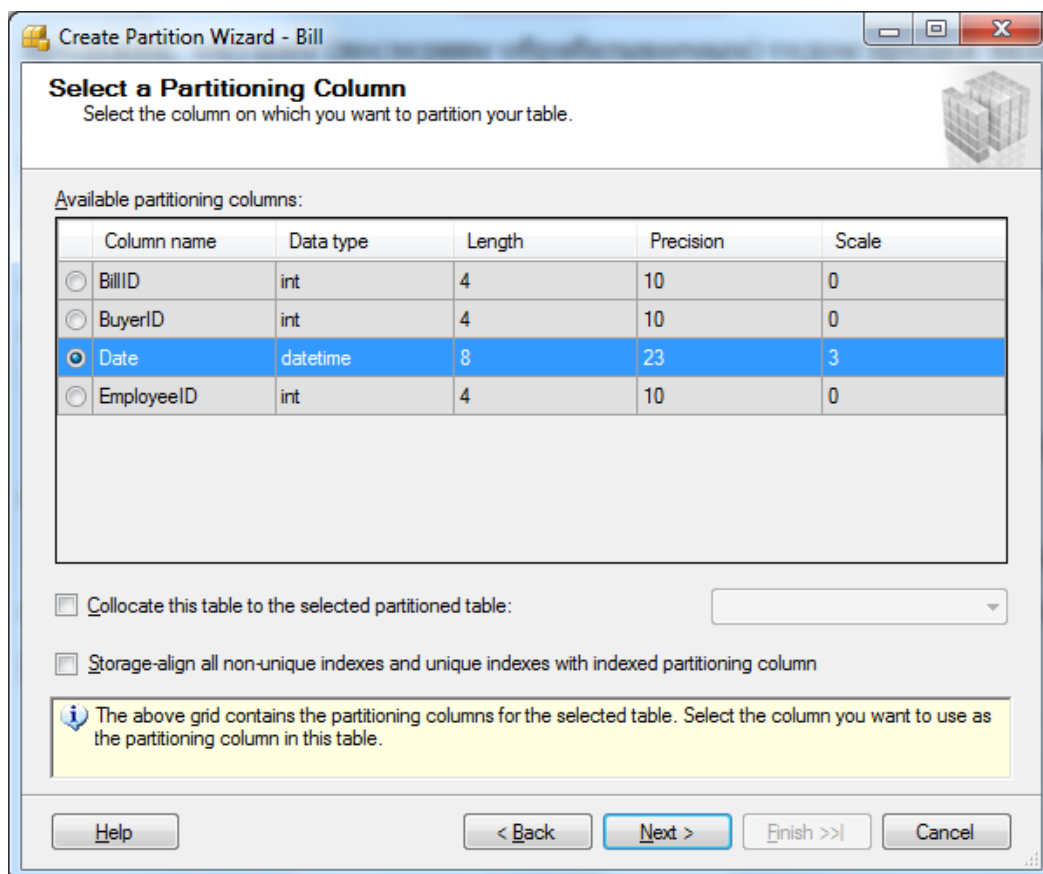


Рисунок 1 – Выбор столбца секционирования таблицы

3. На шаге Select a Partition Function укажите имя функции секционирования (рисунок 2).

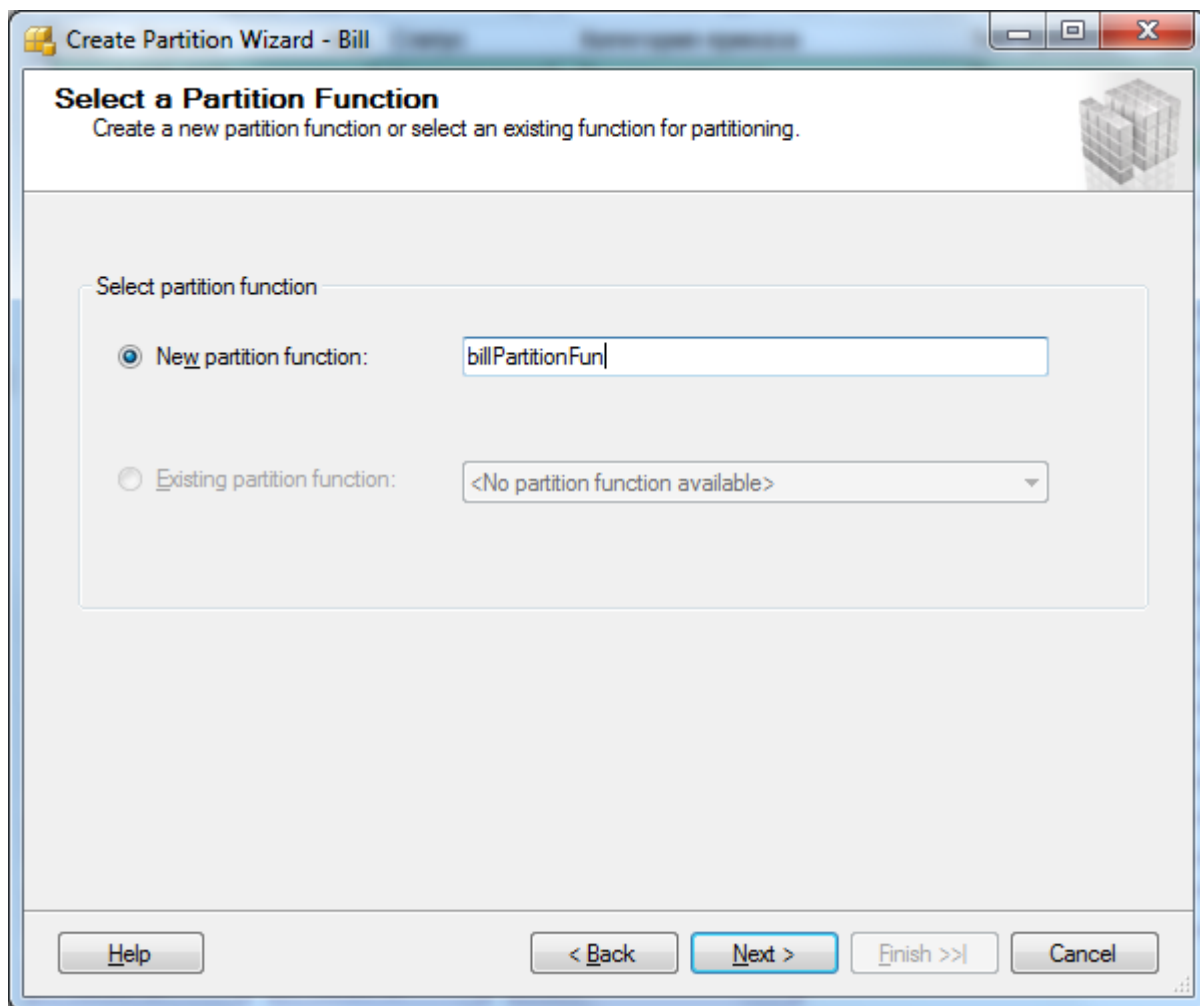


Рисунок 2 – Создание функции секционирования

4. На шаге Select a Partition Scheme укажите имя схемы секционирования (рисунок 3).

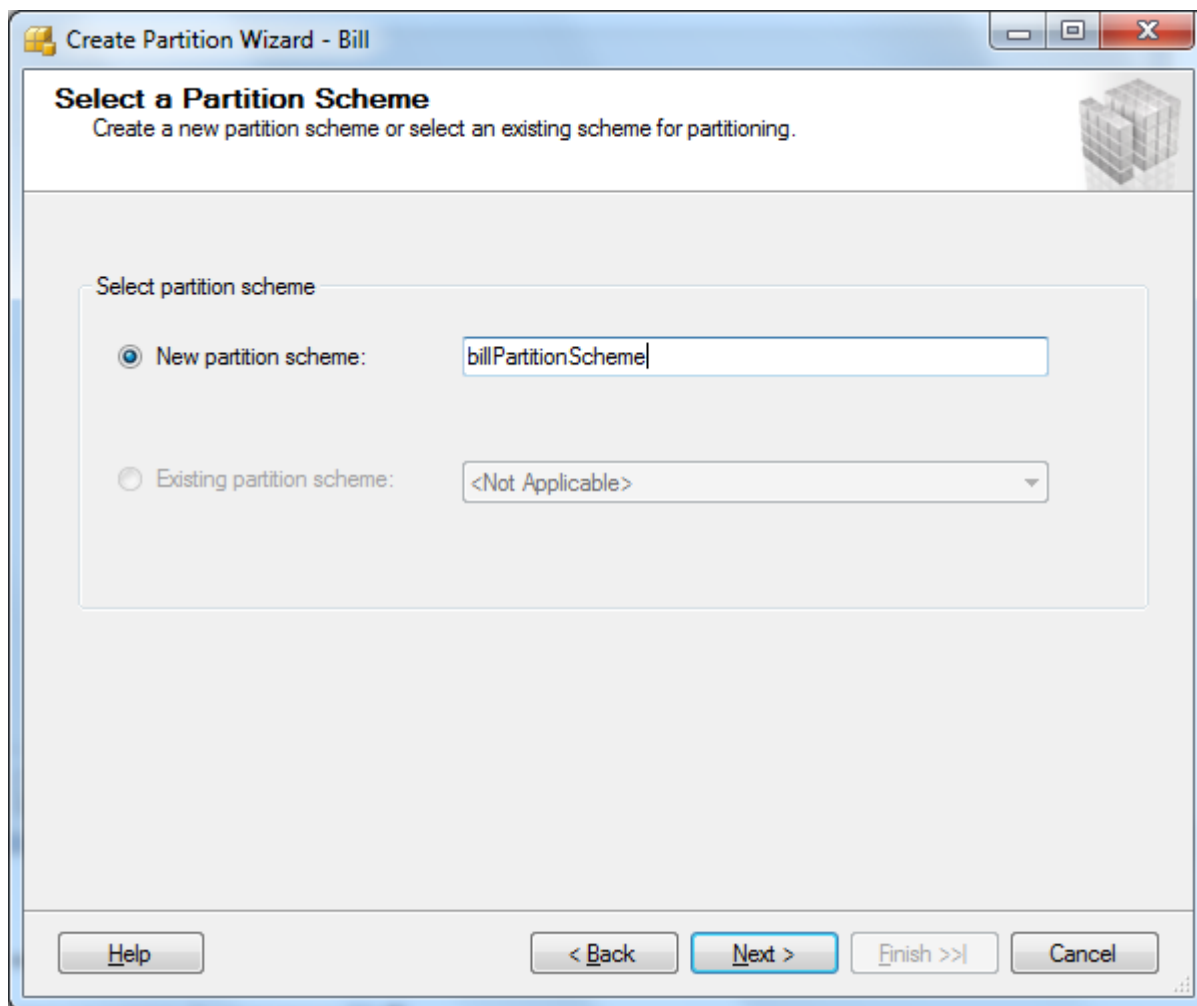


Рисунок 3 – Создание схемы секционирования

5. На шаге Map Partitions требуется определить границы секций и связать их с файловыми группами. Сначала необходимо определиться с видом интервалов секционирования. Если выбрана опция Left Boundary, будут формироваться интервалы вида (предыдущее значение, текущее значение]. Если выбрана опция Right Boundary, будут формироваться интервалы вида [предыдущее значение, текущее значение). Для каждой секции может быть назначена своя файловая группа, либо несколько секций (возможно все) будут связаны с одной файловой группой. Выберем файловую группу PRIMARY для всех секций. Следует учитывать, что значительного выигрыша в производительности и пропускной способности возможно достичь при размещении различных секций в разных файловых группах на различных физических дисках. При этом возможно обеспечить, для определенных запросов, параллельный доступ к массивам дисков. Правые границы диапазонов секций указываются в колонке Boundary, при этом необходимо указать границы каждой секции. Возможен автоматический расчет границ

диапазонов с использованием кнопки Set Boundaries... В последнем случае СУБД автоматически определит минимальное и максимальное значение в колонке секционирования и предложит размер секции (по месяцам, по кварталам и т.п.). Пользователь должен самостоятельно определить требуемые верхнюю и нижнюю границы секционирования (нас интересует 2011 год) и определиться с размером секции (выберем по месяцам). Кнопка Estimate Storage используется для расчета характеристик секций: количество записей, необходимый объем памяти, занимаемый объемом памяти. В итоге карта секционирования примет вид, представленный на рисунке 4.

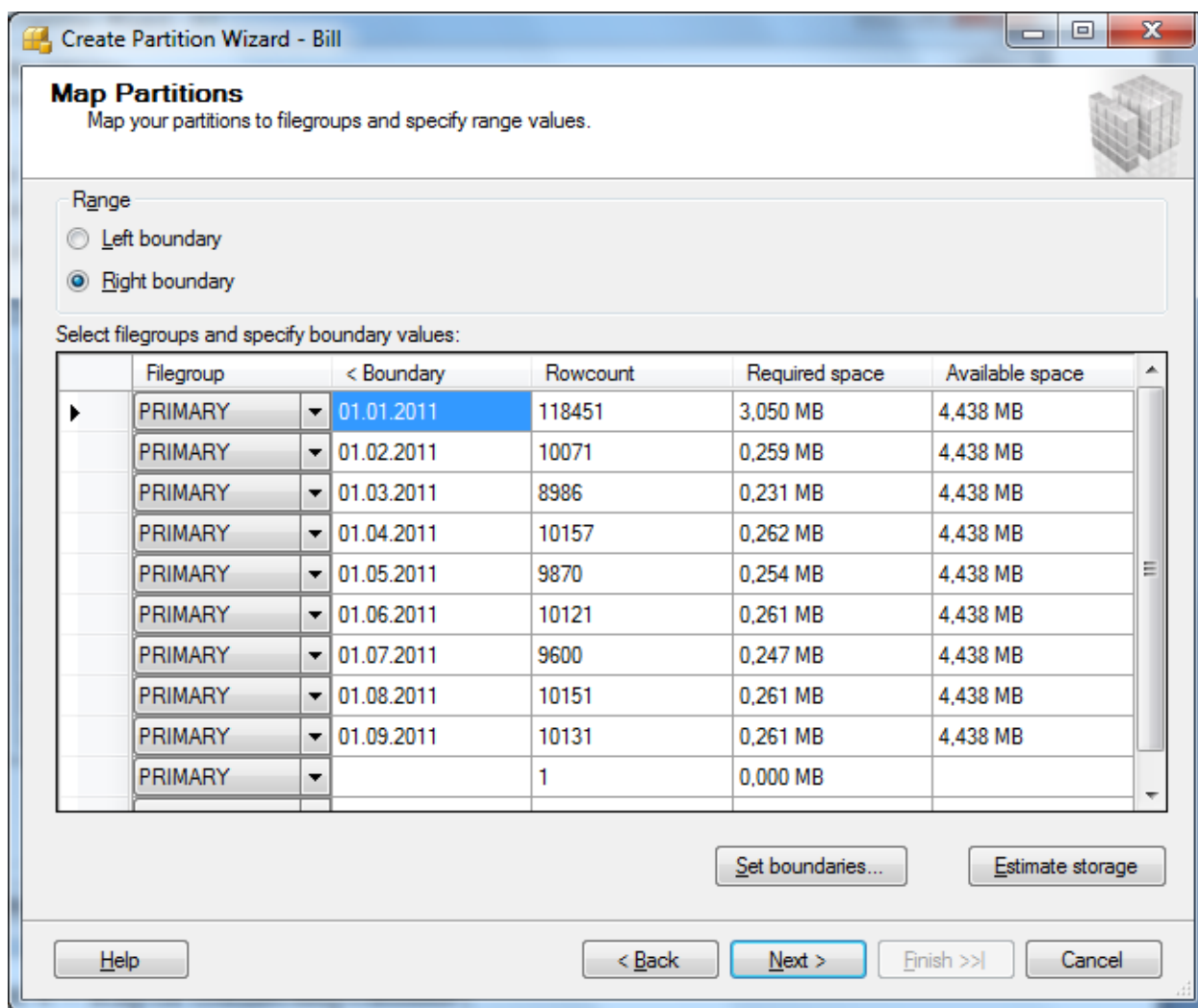


Рисунок 4 – Карта секционирования

6. На шаге Select an Output Option выберите опцию Create Script, открыть скрипт в новом окне редактирования запросов Script to New Query Window (рисунок 5). Другими возможными опциями являются: выполнить немедленно Run Immediately, выполнить по расписанию Schedule.

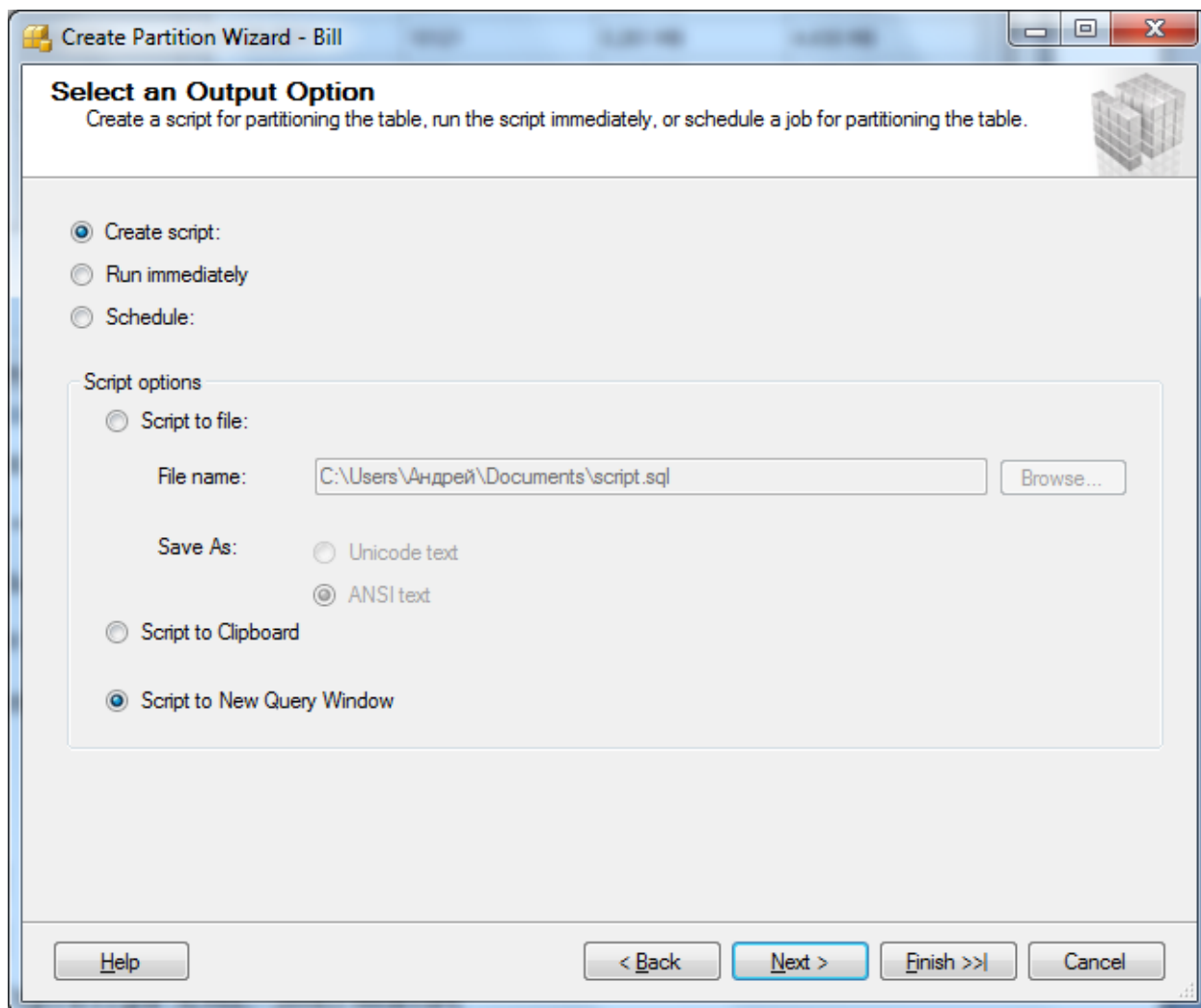


Рисунок 5 – Опции выгрузки

7. На заключительном шаге Review Summary проверьте итоговую информацию о секционировании и нажмите кнопку Finish.

Выполните сформированный скрипт:

```
USE [EducationDatabase]
GO
BEGIN TRANSACTION
```

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

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

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

COMMIT TRANSACTION

Обратите внимание, что СУБД создает новый кластерный индекс по столбцу Date взамен кластерного индекса по первичному ключу-счетчику BillId, что предполагает реорганизацию данных в порядке сортировки по датам покупок и разделения отсортированных данных по секциям.

Рассмотрим, как изменились планы выполнения запросов на выборку данных из таблицы Bill.

Выполним запрос:

```
select * from bill
where date between '05.02.2011' and '20.02.2011'
```

В данном случае все результирующие записи располагаются в одной секции, связанной с месяцем «февраль 2011 г.». Поиск записей осуществляется в одной секции (рисунок 6) с использованием кластерного индекса по полю date (рисунок 7).

Clustered Index Seek	
Scanning a particular range of rows from a clustered index.	
Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Estimated I/O Cost	0,0157176
Estimated CPU Cost	0,0055426
Estimated Number of Executions	1
Estimated Operator Cost	0,0212602 (100%)
Estimated Subtree Cost	0,0212602
Estimated Number of Rows	4895,99
Estimated Row Size	27 B
Partition ID	ConstExpr1009
Ordered	True
Node ID	0
Object	
[EducationDatabase].[dbo].[Bill].	
[ClusteredIndex_on_billPartitionScheme_634648357407858521]	
Output List	
[EducationDatabase].[dbo].[Bill].BillID; [EducationDatabase].[dbo].[Bill].EmployeeID; [EducationDatabase].[dbo].[Bill].BuyerID; [EducationDatabase].[dbo].[Bill].Date	
Seek Predicates	
Start: [EducationDatabase].[dbo].[Bill].Date >= Scalar Operator('2011-02-05 00:00:00.000'); End: [EducationDatabase].[dbo].[Bill].Date <= Scalar Operator('2011-02-20 00:00:00.000')	

Рисунок 6 – Описание операции поиска по кластерному индексу

Query 1: Query cost (relative to the batch): 100%

```
select * from bill where date between '05.02.2011' and '20.02.2011'
```

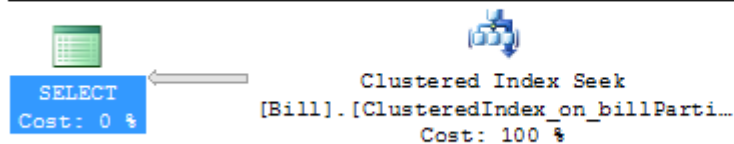


Рисунок 7 – План выполнения запроса

Выполним следующий запрос:

```
select * from bill
where date between '05.02.2011' and '20.03.2011'
```

Теперь результирующие записи располагаются в двух секциях, связанных с месяцами «февраль 2011 г.» и «март 2011 г.». План запроса изменился за счет появления дополнительной операции чтения внутренней таблицы констант с номерами секций (рисунок 8). Операция поиска по кластерному индексу выполняется два раза по числу секций, участвующих в запросе (рисунок 9). Новый план выполнения запроса представлен на рисунке 10.

Constant Scan	
Scan an internal table of constants.	
Physical Operation	Constant Scan
Logical Operation	Constant Scan
Estimated I/O Cost	0
Estimated CPU Cost	0,0000022
Estimated Number of Executions	1
Estimated Operator Cost	0,0000022 (0%)
Estimated Subtree Cost	0,0000022
Estimated Number of Rows	2
Estimated Row Size	11 B
Node ID	1
Values	
(Scalar Operator((3))); (Scalar Operator((4)))	
Output List	
PtnIds1006	

Рисунок 8 – Операция сканирования внутренней таблицы констант

Clustered Index Seek	
Scanning a particular range of rows from a clustered index.	
Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Estimated I/O Cost	0,0216435
Estimated CPU Cost	0,0078606
Estimated Number of Executions	2
Estimated Operator Cost	0,0481751 (41%)
Estimated Subtree Cost	0,0481751
Estimated Number of Rows	7003,26
Estimated Row Size	27 B
Partition ID	PtnIds1006
Ordered	True
Node ID	4
Object	
[EducationDatabase].[dbo].[Bill].	
[ClusteredIndex_on_billPartitionScheme_634648357407858521]	
Output List	
[EducationDatabase].[dbo].[Bill].BillID; [EducationDatabase].[dbo].[Bill].EmployeeID; [EducationDatabase].[dbo].[Bill].BuyerID; [EducationDatabase].[dbo].[Bill].Date	
Seek Predicates	
Start: [EducationDatabase].[dbo].[Bill].Date >= Scalar Operator('2011-02-05 00:00:00.000'); End: [EducationDatabase].[dbo].[Bill].Date <= Scalar Operator('2011-03-20 00:00:00.000')	

Рисунок 9 – Операция поиска по кластерному индексу

Query 1: Query cost (relative to the batch): 100%
 select * from bill where date between '05.02.2011' and '20.03.2011'

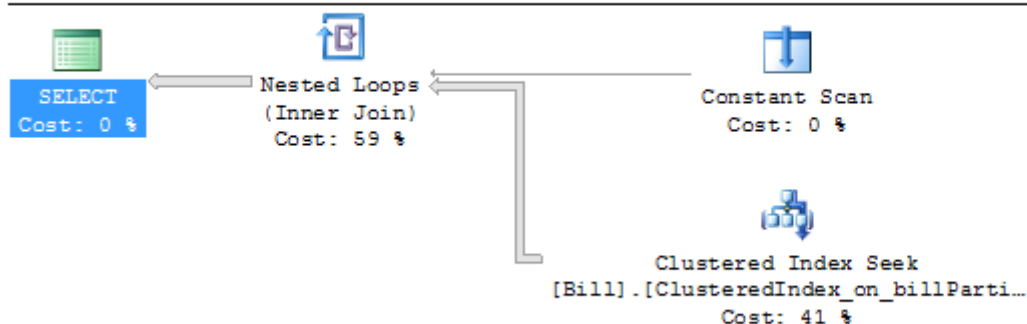


Рисунок 10 – План выполнения запроса

Представленные два примера наглядно демонстрируют преимущества секционирования данных – интерфейс доступа к данным не меняется (используется одинаковый синтаксис SQL-запросов), но оптимизатор запросов прозрачным для разработчика и приложения способом меняется способ доступа к данным с возможностью параллельного выполнения отдельных операторов (в последнем случае – поиск по кластерному индексу).

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Напишите 2 запроса соединения таблиц Bill и BillItem в первом случае с фильтрацией Bill.date between '05.02.2011' and '20.02.2011', во втором случае с фильтрацией Bill.date between '05.02.2011' and '20.03.2011'. Изучите планы выполнения запросов, объясните, почему они различны. Сохраните планы выполнения запросов.
2. Проведите секционирование таблицы BillItem по сценарию, описанному в данной лабораторной работе. В качестве ключа секционирования используйте BillItem.Date. Объясните, являются ли секционированные таблицы Bill и BillItem выровненными?
3. Выполните запросы из п.1 и изучите планы их выполнения. Объясните, что изменилось в планах выполнения запросов.

Контрольные вопросы:

1. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее

профессиональное образование. Информатика и вычислительная техника. Бакалавриат) -
Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224
с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL:
<http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №11. Принципы построения статистической и динамической модели.

Тема - Принципы построения статистической и динамической модели.

Цель работы - освоить принципы построения статистической и динамической модели.

Формируемые компетенции или их части - ОПК-5

Теоретическая часть.

Существуют два основных типа информационной модели: **статическая и динамическая.**

Статическая информационная модель

В статической модели описываются допустимые состояния системы. В статических моделях описываются типы объектов в системе, их свойства и связи. Описывая объекты, они определяют, разумеется, и их имена. Достичь соглашения по именованию всех объектов означает выиграть половину битвы, именно поэтому информационные модели XML иногда называют словарями.

При построении статической информационной модели необходимо пройти следующие этапы:

Этап 1. Идентификация понятий, присвоение им имен и их определение.

Этап 2. Организация понятий в иерархию классов

Этап 3. Определение связей, множественности и ограничений

Этап 4. Добавление свойств для конкретизации деталей значений, связанных с объектами

Этап 1. Именование понятий

Для начала нужно составим список понятий, относящихся к системе. Иногда предлагают даже описать систему на бумаге и выделить все существительные. В любом случае этот этап, как правило, не представляет затруднений.

Далее, и это иногда требует больше времени, надо описать типы объектов. Описание термина должно быть точным, чтобы не возникало разногласий по существу определения. Ценность моделирования в том и заключается, что оно предотвращает появление потенциальных источников непонимания.

По завершении работы, вероятно, получится длинный список типов объектов, и у некоторых будут длинные имена. Следует выбирать такие имена, которые занятые в этом бизнесе люди смогут корректно понять и интерпретировать: дело в том, что они не всегда будут сверяться с написанными определениями.

Помимо именования типов объектов стоит определить также, каким образом можно идентифицировать индивидуальные экземпляры. Возможно, в этом виде бизнеса существует код, который следует знать, или написать его. Надо принимать во внимание, что в схемах кодирования в бизнесе часто обнаруживаются проблемы.

В конце этого этапа мы получим список типов объектов с именами и определениями, по поводу которых достигнуто соглашение.

Этап 2: Таксономия

Таксономия — это термин, используемый в биологии для обозначения системы классификации; в информационном моделировании ее также называют иерархией типов (иногда ее называют еще *онтологией*). Перечислив и назвав типы объектов, их надо организовать в иерархическую схему классификации. Часто эти иерархические отношения возникают уже на этапе определения типов объектов.

Ключевой здесь является фраза, определяющая принадлежность (в английском языке — *is* или *is kind of*). Написав предложение вида “А есть разновидность В” или “Каждое А есть В”, вы определили отношения подтипов в вашей таксономии.

Иногда эти действия называются тестом “*is a*” (“*есть*”, “*представляет собой*”). Однако будьте внимательны, поскольку эта конструкция используется также и для описания отношений между конкретным экземпляром и его типом, безопаснее писать этот тест в форме “*is a kind of*” (“*представляет собой разновидность*”).

Идентификация подтипов бывает, полезна при проектировании документа, что более важно, она помогает лучше понять определения типов объектов.

Если вы занимались объектно-ориентированным программированием, то понимаете, как определять иерархии типов. Но программисты часто рассматривают классы объектов, прежде всего в терминах модулей функциональности внутри системы, а не понятий, которые они представляют в окружающем мире. Тогда для обозначения типов объектов используются глаголы, а не существительные - что неверно.

Итак, этап 2 сводится к организации типов объектов в иерархию типов.

Этап 3: Поиск связей

После того как объекты названы, в статическом информационном моделировании надо определить связи, существующие между ними. Связи (на языке UML они называются ассоциациями) можно показать просто сформулировав их в виде обычных предложений или их можно показать графически в виде диаграммы. Для диаграмм, описывающих связи между объектами, существует большое количество нотаций, каждый может выбрать более предпочтительную для себя. Диаграммы следует делать предельно

простыми и интуитивно понятными, сосредоточившись на ключевых сообщениях и оставив подробности текстовым документам, которые проще обслуживать.

Существует некая информация, которую надо знать о каждой связи:

1. Множественность связи показывает сколько объектов каждого типа принимать в ней участие.

а. Связи типа “один–ко-многим”: одна глава содержит много параграфов, один человек покупает много туристических поездок.

б. Связи типа “многие-ко-многим” также часто встречаются: один автор может написать несколько книг, но у книги также может быть несколько авторов.

с. Связи типа “один-к-одному”.

2. При моделировании информации для окончательного представления XML особенно важным типом связей являются **связи включения**. Множественность этих связей всегда бывает "один-ко-многим" и "один-к-одному". Хотя четкого правила по поводу того, какие объекты образуют связи включения, не существует, можно иногда использовать правила обычного языка: глава содержит параграфы, курорт содержит отели, а отель содержит посетителей. В языке UML определено две формы связей включения. Первая — это агрегации, относительно свободное объединение объектов, позволяющее рассматривать их группу в течение некоторого времени как целое (например, туристическая группа, одни и те же люди могут в разное время входить в разные группы). Вторая форма — композиция. Это более строгая форма; отдельные части целого не могут существовать независимо от него (например, комнаты в отеле не могут существовать независимо от отеля).

Найти подходящие имена для связей иногда бывает нелегко. При записи связей полезно использовать полные фразы типа “отель расположен на курорте” или “человек — это автор книги”. Нам не надо использовать эти имена в тегах разметки XML, они присутствуют только в документации на систему.

Итак, в конце этапа 3 мы определили связи существующие между типами объектов в нашей модели.

Этап 4. Описание свойств

Типы объектов и связи формируют скелет статической информационной модели; свойства можно сравнить с плотью на костях. Свойства представляют собой простые значения, ассоциированные с объектами. У человека можно определить рост, вес, национальность и род занятий; отель имеет определенное количество комнат, этажность и ценовую категорию.

Не следует снова включать связи в список свойств объекта: расположение отеля не является его свойством, если мы уже промоделировали его как связь с курортом.

Главное, что нам надо знать о свойствах, — это тип их данных. Определен ли для них фиксированный диапазон значений, являются ли они числовыми, в каких единицах выражаются? Является ли свойство обязательным и есть ли у него значение по умолчанию?

В конце этапа 4 мы завершили формирование статической информационной модели: получили полное описание типов объектов в системе, их связей друг с другом и их свойств.

Динамическая информационная модель

Динамические модели описывают, что происходит с информацией: примерами таких моделей являются диаграммы рабочих процессов, потоков данных и жизненных циклов объектов. Динамические модели состоят примерно из таких утверждений: “Отделение патологии отправит результаты теста консультанту, отвечающему за пациента”. Динамические модели описывают процесс обмена информацией: данные отправляются из одного места в другое с конкретной целью.

Для построения динамической модели можно воспользоваться различными программными системами (например, диаграммы рабочих процессов и диаграммы потоков данных можно построить, воспользовавшись программой BPWin).

Существует несколько подходов к динамическому моделированию:

- Модели рабочих процессов
- Модели потоков данных
- Объектные модели
- Жизненные циклы объектов
- Варианты использования
- Диаграммы взаимодействия объектов

Модели рабочих процессов

Модели рабочих процессов заостряют внимание на роли людей и организаций в выполнении работы, хранение и обработка информации играют в них вторичную роль. Модель процесса описывает, например, что будет с путешественником, если на отдыхе с ним произойдет несчастный случай. Она определяет, за какие действия отвечает локальный представитель на курорте, агент в стране, где находит курорт и главный офис. В результате становится ясно, кто должен отвечать за организацию медицинской помощи, за перевозку туриста домой и за информирование родственников.

Она может описывать различные формы, заполняемые и пересылаемые между участниками, и вообще не привлекать компьютерные системы. Модель процесса обычно фокусирует внимание на ролях, обязанностях и задачах каждого действующего лица системы (actor), а workflow-модель имеет дело с документами, передаваемыми между действующими лицами. Модели потоков данных.

Модели потоков данных

Модели потоков данных очень напоминают предыдущий тип, но здесь основное внимание уделяется информационным, а не бизнес-систем. Эта модель описывает *хранилища данных* (data stores), где информация находится постоянно (это может быть база данных в компьютере или просто кабинет с папками), *процессоры*, манипулирующие с этими данными, и *потоки данных*, передающие данные от одного процессора другому.

Она активно использует статическую информационную модель: последняя описывает, что означают такие концепции, как путешественник или отель, но ничего не сообщает о том, где содержится информация. Напротив, из модели потоков данных становится ясно, что информация о туристической поездке находится в базе данных покупок до завершения поездки и оплаты всех счетов, после чего резюме этой информации передается в маркетинговую информационную систему, а все остальное — в архив.

Объектные модели

Объектные модели содержат как динамический, так и статический компоненты. Динамическая или поведенческая часть определения объекта сосредоточена на том, что может делать или делал каждый объект, представляя для этого набор операций или методов, описывающих его действия.

Жизненные циклы объекта

Жизненные циклы объекта (на языке UML это называется линиями жизни объекта) также заостряют внимание на индивидуальных объектах, но придерживаются более целостного подхода. Они описывают, что происходит с объектом на протяжении его жизни: как он создается, какие события с ним происходят, как он реагирует на эти события и какие условия приводят в конце к его разрушению.

Жизненные циклы объекта очень полезны для тестирования завершенности модели. Часто наблюдается тенденция к акцентированию внимания только на некоторых событиях за счет остальных. Пока мы не определим, каким образом каждый объект попадает в систему и как он удаляется из нее, полного понимания не будет.

Варианты использования

Варианты использования (use cases) анализируют выполнение специфических задач пользователя (например, человек, купивший туристическую поездку, отменяет свой заказ). Вариант использования напоминает модель процесса, но в общем случае заостряет внимание на деятельности одного конкретного пользователя.

Варианты использования могут быть полезны как на этапе моделирования деловой активности, так и при описании внутреннего поведения информационных систем. Одна из опасностей заключается в смешении двух уровней. Лучше этого не делать, поскольку они представляют интерес для различных аудиторий.

Представленный в виде варианта использования диалог пользовательского интерфейса описывает, какой информацией пользователь обмениваются с системой, а не то, как она представлена на экране. Это естественным образом приводит к реализации XML, в которой информационное содержание отделено от особенностей представления.

Диаграммы взаимодействия объектов

Диаграммы взаимодействия объектов позволяют проанализировать обмен сообщениями между объектами на более тонком уровне детализации, чем модель потока данных.

Диаграммы взаимодействия объектов неоценимы, если требуется описать взаимодействие между различными системами. Они позволяют определить, какая информация содержится в каком сообщении. Поскольку эти сообщения написаны на языке XML, диаграммы взаимодействия объектов дают нам контекст, требуемый для начала проектирования структуры XML каждого индивидуального сообщения.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя

персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Изучить основные принципы построения статической и динамической модели.
2. Построить статистическую и динамическую модель для заданного варианта.
3. Оформить отчет, включающий постановку задачи, статическую и динамическую модели.
4. Защитить лабораторную работу.

Контрольные вопросы:

1. Дать определение статической информационной модели.
2. Какие этапы необходимо пройти при построении статической информационной модели.
3. В чем заключается этап «идентификация понятий» при построении статической информационной модели.
4. В чем заключается этап «организация понятий в иерархию классов» при построении статической информационной модели.
5. В чем заключается этап «определение связей» при построении статической информационной модели.
6. В чем заключается этап «описания свойств» при построении статической информационной модели.
7. Дать определение динамической информационной модели.
8. Что представляют собой модели рабочих процессов.
9. Что представляют собой модели потоков данных.
10. Что представляют собой объектные модели.
11. Что описывают жизненные циклы объектов.
12. Что описывают варианты использования.
13. Что описывают диаграммы взаимодействия объектов.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.
2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №12. Конструкции языка XML.

Тема - Конструкции языка XML.

Цель работы - освоить конструкции языка XML.

Формируемые компетенции или их части - ОПК-5

Теоретическая часть. Синтаксис XML

Любой XML-документ состоит из следующих частей:

- Необязательный пролог.
- Тело документа.
- Необязательный эпилог, следующего за деревом элементов.

Рассмотрим каждую из частей более подробно.

Пролог

Пролог состоит из нескольких частей:

1. необязательное объявление XML (XML Declaration) которое заключено между символами `<?...?>`. Объявление содержит:

- пометку `xml` и номер версии (*version*) спецификации XML;
- указание на кодировку символов (*encoding*), в которой написан документ (по умолчанию `encoding="UTF-8"`);
- параметр *standalone* который может принимать значения `"yes"` или `"no"` (по умолчанию `standalone="yes"`). Значение `"yes"` показывает, что в документе содержатся все требуемые декларации элементов, а `"no"` — что нужны внешние определения DTD.

Все это вместе может выглядеть следующим образом:

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

Важно отметить, что в объявлении XML только атрибут *version* является обязательным, все остальные атрибуты могут быть опущены и, следовательно, принимать значения по умолчанию. Так же нужно помнить, что все эти атрибуты следует указывать только в приведенном выше порядке.

2. комментарии.

3. команды обработки.

4. символы пустых пространств.

необязательное объявление типа документа, DTD (Document Type Declaration) которое заключено между символами `<!DOCTYPE...>` и может занимать несколько строк.

В этой части объявляются теги, использованные в документе, или приводится ссылка на файл, в котором записаны такие объявления.

После объявления типа документа так же могут следовать комментарии, команды обработки и символы пустых пространств.

Поскольку все эти части необязательны, пролог может быть опущен.

Тело документа

Тело документа, состоит из одного или больше элементов. В правильно оформленном XML документе элементы формируют простое иерархическое дерево, в котором обязательно присутствует *корневой элемент* (root element) в который вложены все остальные элементы документа. Имена элементов должны быть уникальны в пределах документа. Имя корневого элемента считается именем всего документа и указывается во второй части пролога после слова Doctype.

Элемент начинается открывающим тегом, затем идет необязательное содержимое элемента, после чего записывается закрывающий тег (в отличие от HTML наличие закрывающего тега обязательно, исключением являются элементы без содержания, так называемые пустые элементы, которые могут быть записаны в сокращенной форме:

<имя_элемента/>). В качестве содержимого элемента могут выступать:

1. другие элементы
2. символьные данные
3. ссылки на символы

Для того, что бы вставить в текст документа некоторый символ, который, например не присутствует в раскладке клавиатуры либо может быть неправильно истолкован анализатором, используют ссылки на символы. Ссылка на символ обязательно начинается со знака «амперсанта» и заканчивается точкой с запятой.

Ссылки на символы записываются в следующем виде:

&# код_символа_в_Unicode;.

Код символа можно записать и в шестнадцатеричном виде. В этом случае перед ним ставится символ “x”:

&#xШестнадцатеричный_код_символа;.

4. ссылки на сущности

Ссылки на сущности позволяют включать любые строковые константы в содержание элементов или значение атрибутов. Ссылки на сущности, как и ссылки на символы, начинающиеся с амперсанта, после которого идет имя сущности и заканчивающиеся точкой с запятой:

&имя_сущности;.

Ссылки на сущности указывают программе-анализатору подставить вместо них строку символов заранее заданную в определении типа документа.

5. комментарии

Если надо вставить в текст документа комментарий либо сделать какой-то фрагмент "невидимым" для программы-анализатора, то его оформляют следующим образом:

`<!--...текст комментария...-->`

6. разделы CDATA

Секция CDATA используется, для того чтобы задать область документа, которую при разборе анализатор будет рассматривать как простой текст, игнорируя любые инструкции и специальные символы. Программа-анализатор не разбивает секцию CDATA на элементы, а считает ее просто набором символов. В отличие от комментариев, содержание данной секции не игнорируется, а передается без изменений на выход программы анализатора, благодаря чему его можно использовать в приложении.

Секция CDATA начинается со строки `<![CDATA[` после которой записывается содержимое секции. Завершается секция двумя закрывающими квадратными скобками и знаком "меньше":

`<![CDATA[ содержание секции ]]>`

7. инструкции по обработки.

Инструкции по обработке содержат указания программе анализатору документа XML. Инструкции по обработке заключаются между символами `<? и ?>`.

Сразу за начальным вопросительным знаком записывается имя программного модуля, которому предназначена инструкция. Затем, через пробел, идет сама инструкция, передаваемая программному модулю. Сама инструкция это обычная строка, которая не должна содержать набор символов `«?>»`, означающий конец инструкции. Примером инструкции по обработке может служить строка объявления XML:

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

Эта инструкция предназначена программе, обрабатывающей документ XML. Инструкция передает ей номер версии и кодировку, в которой записан документ.

Открывающие теги либо теги пустых элементов в XML могут содержать атрибуты, представляющие собой пару имя=значение. В одном открывающем теге разрешается использовать только один экземпляр имени атрибута. Атрибуты могут содержать ссылки на объекты, ссылки на символы, текстовые символы. В отличие от языка HTML, в XML значения атрибутов обязательно надо заключать в апострофы (‘), либо в кавычки (“).

Таким образом, атрибут может быть записан в одном из двух форматов:

имя_атрибута=’значение_атрибута’

имя_атрибут= 'значение_атрибута'.

Описание структуры документа средствами DTD

Для связывания декларации DTD с экземпляром документа в версии XML 1.0 предлагается специальная декларация DOCTYPE. Декларация DOCTYPE содержит ключевое слово DOCTYPE, за которым следует имя корневого элемента документа, а затем конструкция с декларациями содержания. Можно написать внешнее подмножество деклараций в отдельном файле DTD, включить внутреннее подмножество в тело декларации DOCTYPE или сделать то и другое. В последнем случае (смешение внутренних и внешних DTD) во внутренних DTD могут быть заданы новые декларации или переписаны те, что содержатся во внешних (по определению спецификации XML анализаторы сначала читают внутреннее подмножество, и потому содержащиеся там декларации пользуются приоритетом).

Блок внутренней декларации разметки тега DOCTYPE состоит из левой квадратной скобки, списка деклараций и правой квадратной скобки:

```
<! DOCTYPE root_element_name [...здесь находятся декларации внутреннего подмножества ... ]>
```

Для внешних DTD декларация DOCTYPE состоит из обычного ключевого слова и имени корневого элемента, за которым следует еще одно ключевое слово SYSTEM либо PUBLIC, обозначающее источник внешнего определения DTD, а за ним — локализация этого определения.

Допустимое в документе XML содержание определяется с помощью четырех типов декларации разметки в DTD. В следующей далее таблице показаны связанные с этими декларациями ключевые слова и их значения. Первые два типа связаны с информацией, которую мы рассчитываем найти в документе XML, — элементами и атрибутами.

Последние два типа используются для поддержки. Особенно облегчают жизнь разработчика словаря XML сущности. Как правило, они состоят из содержания, которое настолько часто используется в DTD или документе, что оправдывает создание специальной декларации. Применение этой декларации напоминает оператор include в языках C/C++, когда в качестве замены для содержания используется имя. Нотации описывают содержание, разработанное не на языке XML. Используются они для того, чтобы объявить конкретный класс данных и связать его с внешней программой.

Конструкция DTD Значение ELEMENT Декларация типа элемента XML ATTLIST Декларация атрибутов, которые могут быть назначены конкретным типам элементов, а также разрешенных значений этих атрибутов

ENTITY Декларация повторно используемого содержания NOTATION Декларация форматирования для внешнего содержания, которое не должно быть проанализировано

(например, двоичные данные), а также для внешних приложений, обрабатывающих содержание

Рассмотрим более подробно первые два типа декларации разметки в DTD. В документе XML должен быть описан каждый элемент. Объявление элемента начинается с символов <!ELEMENT, после которых через пробел идет имя элемента и его содержимое. Заканчивается объявление символом "больше". По своему содержанию элементы делятся на четыре группы.

1. Пустой элемент - может иметь атрибуты, но не содержит текст или порожденные элементы. Объявляется следующим образом: после имени элемента указывается ключевое слово EMPTY.

Пример: <!ELEMENT element_name EMPTY>

2. Элемент содержит только порожденные элементы, но не текст. Объявляется следующим образом: после имени элемента в скобках через запятую перечисляются все вложенные элементы. Причем вложенные элементы должны следовать в XML документе в том порядке, в каком они перечислены в объявлении.

Пример: <ELEMENT element_name (elem_1,elem_2)>

3. Элемент содержит не только порожденные элементы, но и текст. Объявляется следующим образом: после имени элемента в скобках указывается ключевое слово #PCDATA, после которого через запятую, как и в предыдущем случае, перечисляются все вложенные элементы (если они имеются).

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

<ELEMENT element_name (#PCDATA)>

4. Элемент, открытый для любого содержания. Объявляется следующим образом: после имени элемента указывается ключевое слово ANY.

Пример: ELEMENT element_name ANY>

Иногда из нескольких вложенных элементов или списков (список элементов указанных в скобках) может встретиться только один. В таком случае их имена перечисляются через вертикальную черту(|). Например: <!ELEMENT element_name (elem_1,(elem_2|elem_3))> - элемент element_name должен содержать элемент elem_1, а затем либо elem_2, либо elem_3. Элементы появляются именно в таком порядке.

Если вложенный элемент можно записать в объявляемом элементе несколько раз, то необходимо это указать используя звездочку, плюс или вопросительным знак.

? — элемент или список может встретиться нуль или один раз;

* — элемент или список может встретиться нуль или несколько раз;

+ — элемент или список может встретиться один или несколько раз.

После объявления элемента объявляются его атрибуты. Все атрибуты одного элемента объявляются сразу, одним списком. Список начинается с символов <!ATTLIST, после них через пробел следует имя элемента, к которому относятся атрибуты. Затем идет имя атрибута, его тип или список значений, которые он может принимать (все значения перечисляются через вертикальную черту, в скобках), признак обязательности присутствия атрибута в элементе или значение по умолчанию (это значение будет использовано, если атрибут не записан явно в XML документе).

Тип атрибута записывается одним из ключевых слов:

1. CDATA — строка символов.

2. ID — уникальный идентификатор, однозначно определяющий элемент, в котором встретился этот атрибут; значения такого атрибута не должны повторяться в документе.

Они играют ту же роль, что и первичные ключи в таблицах баз данных.

3. IDREF — идентификатор, содержащий одно из значений атрибутов типа id, используется в качестве ссылки на другие элементы.

4. IDREFS — идентификатор, содержащий набор значений атрибутов типа id, перечисленных через пробелы; тоже используется в качестве ссылки сразу на несколько элементов.

5. ENTITY — имя непроверяемой анализатором сущности объявленных в этом же описании DTD.

6. ENTITIES — имена непроверяемых сущностей.

NMTOKEN — слово, содержащее только символы, применяемые в именах.

Атрибуты этого типа могут содержать имена других элементов или атрибутов, например, для того чтобы ссылаться на них.

7. NMTOKENS — слова, перечисленные через пробелы.

8. NOTATION — обозначение расшифрованное в описании DTD.

Признак обязательности записывается с использованием ключевых слов:

1. #REQUIRED — атрибут надо обязательно записывать в элементе;

2. #IMPLIED — атрибут необязателен, у него нет значения по умолчанию;

3. #FIXED — у атрибута есть только одно значение, которое записывается тут же, через пробел.

Пример: <!ATTLIST city type (город | поселок | деревня) "город">

<!ATTLIST pre xml:lang NMTOKEN "ru_RU">

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

Описание схемы документа на языке XSD

Связать XML документ с его XSD схемой можно разными способами:

1. подать файлы со схемой на вход анализатора.
2. задать файлы со схемой как свойство анализатора.
3. указать прямо в документе XML.

Рассмотрим 3 способ более подробно.

Если элементы документа не принадлежат никакому пространству имен и записаны без префикса, то в корневом элементе документа записывается атрибут `noNamespaceSchemaLocation`, указывающий расположение файла со схемой в форме URI:

```
< root_element_name xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance  
xsi:noNamespaceSchemaLocation="file_name.xsd">
```

В этом случае в схеме не должно быть целевого пространства имен.

Если же элементы документа относятся к некоторому пространству имен, то применяется атрибут `schemaLocation`, в котором через пробел перечисляются пространство имен и расположение файла со схемой, описывающей это пространство имен.

Элементы, из которых будет состоять документ XML, объявляются в схеме компонентом `element`:

```
<xsd:element_name="имя элемента" type="тип элемента" minOccurs="Наменьшее  
число появлений элемента в документе" maxOccurs="наибольшее число появлений" />
```

Значение по умолчанию необязательных атрибутов `minOccurs` и `maxOccurs` равно 1.

Это означает, что если эти атрибуты отсутствуют, то элемент должен появиться в документе XML ровно один раз.

Указание типа элемента в атрибуте `type` удобно, если это встроенный простой тип или тип, определенный заранее. Тогда в атрибуте `type` можно записать только имя типа.

Если же тип элемента определяется здесь же, то определение типа элемента лучше вынести:

```
<xsd:element name="имя элемента">
```

Определение типа элемента

```
</xsd:element>
```

Объявление атрибута элемента тоже несложно:

```
<xsd:attribute name=" имя атрибута" type="тип атрибута" use="обязательность  
! _____ атрибута" default="значение по умолчанию" />
```

Необязательный атрибут `use` принимает три значения:

- optional — описываемый атрибут необязателен (это значение по умолчанию);
- required — описываемый атрибут обязателен;

-prohibited — описываемый атрибут неприменим. Это значение полезно при определении подтипа, чтобы отменить некоторые атрибуты базового типа.

Определение типа атрибута, — а это должен быть простой тип, — можно вынести в содержимое элемента attribute:

```
<xsd:attribute name="имя атрибута">
```

Тип атрибута

```
</xsd:attribute>
```

Определение простых типов

Простой тип в схемах XML определяется компонентом схемы simpleType, имеющим вид `<xsd:simpleType name="имя типа">Определение типа</xsd:simpleType>`

Определение новых типов простых элементов

Кроме встроенных типов в схемах XML можно определить новые типы простых элементов. Они вводятся как

1. сужение (restriction) встроенного или ранее определенного простого типа,
2. список (list) простых типов
3. объединение (union) простых типов.

Сужение простого типа определяется компонентом restriction, в котором атрибут base указывает сужаемый простой тип, а в содержимом задаются ограничения, выделяющие определяемый простой тип.

Теги задающие ограничения, называются фасетками (facets). Вот их список:

1. `<maxExclusive>` — наибольшее значение, которое уже не входит в определяемый тип;
2. `<maxInclusive>` — наибольшей значение определяемого типа;
3. `<minExclusive>` — наименьшее значение, уже не входящее в определяемый тип;
4. `<minInclusive>` — наименьшее значение определяемого типа;
5. `<totalDigits>` — общее количество цифр в определяемом числовом типе — сужении типа decimal;
6. `<fractionDigits>` — количество цифр в дробной части числа;
7. `<length>` — длина значений определяемого типа;
8. `<maxLength>` — наибольшая длина значений определяемого типа;
9. `<minLength>` — наименьшая длина значений определяемого типа;
10. `<enumeration>` — одно из перечислимых значений;
11. `<pattern>` — регулярное выражение;

12. <whiteSpace> — применяется при сужении типа string и определяет способ преобразования пробельных символов '\n', '\r', '\t'. Атрибут value этого тега принимает одно из трех значений:

- preserve — не убирать пробельные символы;
- replace — заменить пробельные символы пробелами;
- collapse - после замены пробельных символов пробелами убрать начальные и конечные пробелы, а из нескольких подряд идущих пробелов оставить только один.

В тегах-фасетках можно записывать следующие атрибуты, называемые базисными фасетками (fundamental facets):

1. ordered— задает упорядоченность определяемого типа, принимает одно из трех значений:

- false — тип неупорядочен;
- partial — тип частично упорядочен;
- total — тип полностью упорядочен;

2. bounded — задает ограниченность или неограниченность типа значением true или false;

3. cardinality — задает конечность или бесконечность типа значением finite или countably infinite;

4. numeric — показывает, числовой этот тип или нет, значением true или false.

Список определяется компонентом list, в котором атрибутом itemType указывается тип элементов определяемого списка. Тип элементов списка можно определить и в содержимом элемента list.

При определении списка можно применять фасетки <length>, <minLength>, <maxLength>, <enumeration>, <pattern>.

Простой тип-объединение определяется компонентом union, в котором атрибутом memberTypes можно указать имена объединяемых типов. Например:

```
<xsd: union memberTypes="xsd: string xsd: integer lisstOfInteger" />
```

Другой способ — записать в содержимом компонента union определения простых типов, входящих в объединение.

Определение сложных типов

Тип элемента называется сложным, если в элемент вложены другие элементы и/или в открывающем теге элемента есть атрибуты.

Сложный тип определяется компонентом complexType, имеющим вид:

```
<xsd:complexType name="имя типа" >
```

Определение типа

</xsd:complexType>

Необязательный атрибут name задает имя типа, а в содержимом компонента complexType описываются элементы, входящие в сложный тип, и/или атрибуты открывающего тега.

Определение сложного типа можно разделить на три группы:

1. определение типа пустого элемента (элемента, не содержащего тела, а содержащего только атрибуты в открывающем теге).

Каждый атрибут объявляется одним компонентом attribute, например:

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

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

```
</xsd:complexType>
```

2. определение типа элемента с простым телом (элемента, содержащего тело простого типа и атрибуты в открывающем теге).

Этот тип отличается от простого типа только наличием атрибутов и определяется компонентом simpleContent. В теле этого компонента должен быть либо компонент restriction, либо компонент extension. Атрибутом base задают тип (простой) тела описываемого элемента.

В компоненте extension указываются атрибуты открывающего тега описываемого элемента.

В компоненте restriction, кроме атрибутов, описывается простой тип содержимого элемента и/или фасетки, ограничивающие тип, заданный атрибутом base.

3. определение типа элемента, содержащего вложенные элементы.

Если значениями определяемого сложного типа будут элементы, содержащие вложенные элементы, то перед тем, как перечислять их описания, надо выбрать модель группы (model group) вложенных элементов. Дело в том, что вложенные элементы, составляющие определяемый тип, могут появляться или в определенном порядке, или в произвольном порядке, кроме того, можно выбирать только один из перечисленных элементов. Эта возможность и называется моделью группы элементов. Она определяется одним из трех компонентов: sequence, all или choice.

Компонент sequence применяется в том случае, когда перечисляемые элементы должны записываться в документе в определенном порядке.

Если же вместо компонента xsd:sequence записать компонент xsd:all, то элементы можно перечислять в любом порядке. Компонент choice применяется в том случае, когда надо выбрать один из нескольких элементов.

4. Определение типа элемента со сложным телом

При определении сложного типа можно воспользоваться уже определенным, базовым, сложным типом, расширив его дополнительными элементами, или, наоборот, удалив из него некоторые элементы. Для этого необходимо применить компонент `complexContent`. В этом компоненте, так же как и в компоненте `simpleContent`, записывается либо компонент `extension`, если надо расширить базовый тип, либо компонент `restriction`, если нужно его сузить. Базовый тип указывается атрибутом `base`, так же как и при записи компонента `simpleContent`, но теперь это должен быть сложный, а не простой тип. При сужении базового типа компонентом `restriction` надо перечислить те элементы, которые останутся после сужения.

Преобразование документов средствами XSLT

Язык преобразований XSLT — это одна из реализации XML. Все элементы XML, объявленные в языке XSLT, относятся к пространству имен `http://www.w3.org/1999/XSL/Transform`. Обычно они записываются с префиксом `xsl`. Если принят этот префикс, то корневой элемент документе XSLT— таблицы стилей— будет называться `xsl:stylesheet` у которого есть один обязательный атрибут `version`, указывающий версию языка.

Таблицу стилей можно записать в файл с расширением `xsl`. Ссылку на таблицу стилей можно поместить в документ XML как одну из инструкций по обработке, а именно инструкцию `xml-stylesheet`. Пример такой ссылки приведен ниже.

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

“Увидев” инструкцию по обработке `xml-stylesheet`, программа-обработчик XML, если она, кроме того, является процессором XSLT, выполнит преобразование, заданное в таблице стилей.

Так же таблицу стилей можно записывать не в отдельный файл, а непосредственно в преобразуемый документ XML. Для этого в элементе `xsl:stylesheet` предусмотрен атрибут-идентификатор `id`, на который можно сослаться обычным образом из инструкции по обработке `xml-stylesheet`.

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

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<!-- преобразования -->
</xsl:stylesheet>
<!-- содержание -->
</book>
```

Рассмотрим некоторые элементы языка XSLT.

Декларация xsl:import

Элемент xsl:import записывается очень просто:

```
<xsl:import href="адрес URI таблицы стилей" />
```

Его можно записать только непосредственно в корневом элементе xsl:stylesheet и только в самом начале таблицы стилей. Элементов xsl:import может быть несколько.

Процессор XSLT отыскивает таблицу стилей по указанному атрибуту href адресу и подставляет ее на место элемента xsl:import перед преобразованием.

Некоторые правила преобразования из таблиц, импортируемых элементами xsl:import, могут конфликтовать с правилами, импортированными из других таблиц или определенными в самой таблице стилей. В таком случае чаще всего применяются те правила, которые записаны последними. Поэтому порядок записи элементов xsl:import в таблицу стилей имеет большое значение.

Декларация xsl:include

Второй элемент, включающий внешние таблицы стилей в данную таблицу, — это элемент xsl:include. Он записывается точно так же, как элемент xsl:import, и оказывает такое же действие:

```
<xsl:include href="адрес URI таблицы стилей" />
```

Его можно записать не только непосредственно в корневом элементе xsl:stylesheet, но, в отличие от элемента xsl:import, в любом месте таблицы стилей. Второе отличие от элемента xsl:import заключается в том, что порядок записи элементов xsl:include в таблице стилей не имеет значения.

Декларация xsl:variable

Элемент xsl:variable определяет имя объекта. Оно записывается обязательным атрибутом name. Имя объекта должно быть уточненным именем XML типа QName. Кроме того, атрибутом select переменной можно задать сам объект, а атрибутом as определить тип объекта. Например:

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

Имя var1 будет хранить число элементов person. Объект может быть получен из содержимого элемента xsl:variable:

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

или создан конструктором последовательности:

```
<xsl:variable name="var3">
```

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

```
<xsl:variable>
```

Если объект не получен из атрибута `select` или содержимого элемента `xsl:variable`, то по умолчанию имя связывается с пустой строкой.

Для того чтобы получить объект, связанный с именем, определенным элементом `xsl:variable`, перед именем надо поставить знак доллара: `$var1`, `$var2`. При этом следует учитывать область действия имени.

Область действия имени простирается на весь элемент, в котором оно определено, начиная с места определения и включая вложенные элементы, если только в них не определено то же самое имя.

Имена, определенные непосредственно в корневом элементе `xsl:stylesheet`, называются глобальными именами, остальные — локальными именами.

Хотя слово "variable" и переводится с английского языка как "переменная", имя, созданное элементом `xsl:variable` — это не имя переменной, его значение нельзя изменить. Это только название некоторого объекта, которое удобно использовать в тех случаях, когда объект надо использовать во многих местах таблицы стилей, а его вычисление сложно или громоздко.

Декларация `xsl:param`

Элемент `xsl:param` записывается или непосредственно в элементе `xsl:stylesheet`, чтобы задать параметр преобразования, или в элементе `xsl:template`, чтобы задать параметр правила, или в элементе `xsl:function` как аргумент функции. У него один обязательный атрибут `name`, определяющий имя параметра. Кроме него, часто присутствует необязательный атрибут `select`, в котором записывается выражение для получения значения параметра:

```
<xsl:param name="p1" select="10 + 20" />
```

Если атрибут `select` отсутствует, то значение параметра берется из содержимого элемента, которым может быть конструктор последовательности узлов и атомарных значений:

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

Если отсутствует и атрибут `select`, и содержимое элемента, то параметр получает значение пустой строки.

Для получения значения параметра надо записывать его имя со знаком доллара:

&p1,&p2. Например:

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

Правила, определяющие область видимости параметров, такие же, как и у имен объектов, определенных декларацией `xsl:variable`.

Еще один необязательный атрибут `as` содержит желательный тип, к которому будет приведено значение параметра. Наконец, последний атрибут `required`, принимающий значения `yes` или `no` (по умолчанию), указывает обязательность параметра. Если параметр обязателен, `required="yes"`, то элемент `xsl:param` должен быть пустым и не содержать атрибут `select`. В таком случае он получит определенное значение при вызове функции или элементом `xsl:with-param` при вызове шаблона.

Элемент `xsl:with-param`

Элемент `xsl:with-param` ссылается на некоторый параметр, имя которого записано в обязательном атрибуте `name`. Необязательным атрибутом `select` можно задать выражение, результат вычисления которого будет новым значением параметра:

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

Новое значение можно задать и в содержимом элемента:

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

Элемент `xsl:with-param` используется только в инструкциях `xsl:apply-templates`, `xsl:apply-imports`, `xsl:call-template`.

Инструкция `xsl:value-of`

Элемент `xsl:value-of` вычисляет выражение, записанное в его обязательном атрибуте `select`, и преобразует его в строку. Например, выше мы определили имя объекта `var2`. Чтобы получить значение объекта `var2`, надо записать:

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

Если в результате вычисления выражения получается последовательность, то процессор XSLT версии 1.0 выберет из нее только первый элемент, преобразованный в строку.

Инструкции управления `xsl:if`, `xsl:for-each`, `xsl:choose`

Элемент `xsl:if` запускает конструктор последовательности, содержащийся в его теле, только если истинно выражение, записанное в обязательном и единственном атрибуте `test`:

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

```
<xsl:value-of select="$x"•/> больше <xsl:value-of select="$y" />
```

```
</xsl:if> '
```

У элемента `xsl:for-each` в обязательном и единственном атрибуте `select` записывается выражение, дающее в результате последовательность. Для каждого члена этой последовательности выполняется конструктор, содержащийся в теле элемента `xsl:for-each`. В результате получается цикл, выполняющийся столько раз, сколько элементов у последовательности, полученной в результате вычисления выражения атрибута `select`.

У элемента `xsl:choose` нет ни одного атрибута, но в его теле записывается один или несколько элементов `xsl:when` и один необязательный элемент `xsl:otherwise`:

```
<xsl:choose>
  <xsl when test="$day=1">Понедельник</xsl:when>
  <xsl when test="$day=2">Вторник</xsl:when>
  <xsl when test="$day=3">Среда</xsl:when>
  <xsl when test="$day=4">Четверг</xsl:when>
  <xsl when test="$day=5">Пятница</xsl:when>
  <xsl when test="$day=6">Суббота</xsl:when>
  <xsl when test="$day=7">Воскресенье</xsl:when>
  <xsl otherwise>Ошибка определения дня недели</xsl:otherwise>
</xsl:choose>
```

У элемента `xsl:when` есть только один обязательный параметр `test`, содержащий логическое выражение, и тело, содержащее конструктор последовательности.

Элемент `xsl:otherwise` не имеет атрибутов, у него есть только тело, содержащее конструктор последовательности.

В инструкции `xsl:choose` всегда выполняется не больше одного варианта, Варианты `xsl:when` просматриваются в порядке их написания в инструкции `xsl:choose`. Как только будет найден вариант с истинным значением выражения `test`, он будет выполнен, и результат этого варианта будет результатом всей инструкции. Варианты, следующие за ним по порядку, не рассматриваются. Если ни один вариант не подойдет, то результатом инструкции будет результат конструктора, записанного в элементе `xsl:otherwise`. Если элемента `xsl:otherwise` в инструкции нет, то результатом будет пустая строка.

Декларация `xsl:function`

Элемент `xsl:function` позволяет описывать самые настоящие пользовательские функции. Имя функции записывается в обязательном атрибуте `name`, (аргументы функции задаются элементами `xsl:param`, а тело функции — это конструктор последовательности, записанный в содержимом элементе `xsl:function`. Результатом функции будет последовательность, созданная конструктором. Тип функции можно указать необязательным атрибутом `as`.

Имя функции — это уточненное имя типа QName, причем оно должно обязательно записываться с префиксом.

Все аргументы функции позиционные, следовательно, все элементы `xsl:param` должны быть записаны в начале тела функции и порядок их записи имеет значение при вызове функции. У аргументов функции не может быть значений по умолчанию, следовательно, элементы `xsl:param` должны быть пустыми и у них не должно быть атрибутов `select`. Например:

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

вызвать функцию можно в любом выражении подходящего типа, записав ее имя и аргументы в скобках. Например:

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

Функция может вызываться рекурсивно.

Декларация `xsl:template`

Элемент `xsl:template` определяет шаблонное правило преобразования. Своим атрибутом `match` он задает образец для отбора узлов, подлежащих преобразованию, а в теле содержит конструктор последовательности узлов и атомарных значений, которая и будет результатом преобразования отобранных по образцу узлов.

```
<xsl: template match=" Образец" name="Имя">Конструктор</xsl:template>
```

Каждый из атрибутов `match` и `name` не обязателен, но хотя бы один из них должен присутствовать.

Атрибут `match` содержит образец для отбора преобразуемых узлов.

Атрибут `name` определяет имя шаблона. Имя шаблона — это обычное уточненное имя XML типа QName. Шаблон можно вызвать по имени элементом `xsl:call-template`, а если он не содержит атрибута `match`, то такой вызов обязателен, поскольку неизвестны узлы, к которым его надо применить, и он не будут применяться автоматически. Очень часто именованный шаблон содержит параметры, заданные элементами `xsl:param`, и вызывается с различными параметрами совсем как обычная функция.

У элемента `xsl:template` могут быть дополнительные атрибуты `mode`, `as`, `priority`.

Атрибут `mode` определяет режим обработки.

Атрибут `as` указывает желаемый тип результата (полученная последовательность будет приведена к этому типу).

Атрибут `priority` назначает правилу приоритет, который будет учитываться при отборе правил, применимых к некоторому узлу.

При вызове именованного шаблона элементов `xsl:call-template` атрибуты `match`, `mode` и `priority` игнорируются.

Если у именованного шаблона нет атрибута `match`, то у него не должно быть и атрибутов `mode` и `priority`, в них просто нет никакого смысла.

Инструкция `xsl:apply-templates`

Элемент `xsl:apply-templates`, записываемый чаще всего внутри элемента `xsl:template`, в простейшем виде пуст:

```
<xsl:apply-templates />
```

Он предписывает обработать рекурсивно все узлы-потомки узлов, отобранных родительским элементом `xsl:template`.

У элемента `xsl:apply-templates` есть необязательные атрибуты `select` и `mode`.

Атрибут `select`, значением которого должно быть выражение, дающее последовательность узлов, ограничивает обработку только указанными в нем узлами.

Атрибут `mode` выбирает режим обработки из режимов, уже определенных в элементах `xsl:template`. Режим — это любое имя типа `QName`, но два режима предопределены. Это текущий режим, отмечаемый словом `#current`, и режим по умолчанию, принимаемый при отсутствии атрибута `mode`, или отмечаемый явно словом `#default`.

Содержимым элемента `xsl:apply-templates` могут служить элементы `xsl:sort` и `xsl:with-param`.

Инструкция `xsl:for-each-group`

Элемент `xsl:for-each-group` отбирает последовательность узлов и атомарных значений своим обязательным атрибутом `select` и группирует ее элементы по признаку, заданному выражением, записанным в атрибуте `group-by`. Этот признак называется ключом группы. Ключ группы вычисляется заново для каждого элемента последовательности. В результате исходная последовательность, отобранная атрибутом `select`, разбивается на несколько последовательностей. В следующем примере в одну группу собираются все узлы-элементы `name` с одинаковым значением атрибута `surname`:

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

Выражение, являющееся значением атрибута `group-by`, может давать несколько ключей группы, и один узел может попасть сразу в несколько групп. Например: `<xsl:for-each-group select="name" group-by="(@second, @surname)">`

Второй способ разбить последовательность на группы дает атрибут `group-adjacent`.

Он собирает в группу все подряд идущие элементы последовательности с одинаковым значением ключа. Такой отбор возможен, если в атрибуте `group-adjacent` содержится только один ключ группы. Процессор XSLT следит за тем, чтобы значением атрибута `group-adjacent` был один и только один ключ группы.

Третий и четвертый способы применимы к последовательностям, состоящим только из узлов, без атомарных значений. Эти способы применяют атрибут `group-starting-with` или атрибут `group-ending-with`. Значением этих атрибутов может быть не любое выражение, а только образец. В одну группу собираются все подряд идущие узлы, первый (последний) из которых удовлетворяет образцу. Остальные узлы из этой группы не будут удовлетворять образцу. Если несколько подряд идущих элементов последовательности удовлетворяют образцу, то они попадут в разные группы.

Итак, группы узлов создаются одним из четырех атрибутов элемента `xsl:for-each-group`: `group-by`, `group-adjacent`, `group-starting-with`, `group-ending-with`. В каждом элементе `xsl:for-each-group` должен быть один и только один из этих атрибутов. Полученные группы существуют и могут использоваться только в содержимом элемента `xsl:for-each-group`. В теле элемента `xsl:for-each-group` записывается конструктор последовательности, который выполняется по одному разу для каждой группы.

Для удобства работы с группами в язык XSLT введены функции `current-group()` и `current-group-key()`. Их можно использовать только в теле элемента `xsl:for-each-group`. У обеих функций нет аргументов. Результатом функции `current-group()` будет последовательность — текущая группа, а результатом функции `current-group-key()` — значение ключа текущей группы.

Последовательность преобразований

Теперь, после краткого описания основных элементов языка XSLT, можно понять, каким образом они выполняют преобразование документа XML.

Документ XSLT — таблица стилей — состоит из шаблонных правил преобразования, записанных элементами `xsl:template`. В каждом правиле атрибутом `match` перечисляется последовательность исходных узлов, для преобразования которых предназначено данное правило, а в содержимом элемента `xml:template` задается конструктор последовательности преобразованных узлов. Из каждого узла исходного дерева получается один или несколько узлов преобразованного дерева. Порядок записи правил не имеет значения,

поскольку для каждого узла исходного дерева просматривается вся таблица стилей в поисках подходящего правила.

Применение правил преобразования

Обычно просмотр исходного документа XML начинается с его корневого узла. Для него подбирается правило преобразования, конструктор которого явно или неявно создает корневой узел преобразованного документа. Затем конструктор создает последовательность узлов, которые будут потомками корневого узла нового документа.

Если для какого-то узла дерева документа в таблице стилей не оказалось соответствующего ему правила, то к нему применяется встроенное в процессор XSLT правило по умолчанию. Правила по умолчанию опираются на вид узла.

Для корневого узла и узла-элемента правило по умолчанию означает просмотр его потомков.

Для узла-атрибута и текстового узла встроенное правило по умолчанию создает текстовый узел, содержащий их значения. Если значение атрибута или текстового узла пусто, то правило не делает ничего, точнее говоря, создает пустую последовательность узлов.

Для узла-комментария, инструкций по обработке и узла пространства имен по умолчанию создается пустая последовательность узлов.

С другой стороны, для какого-то исходного узла в таблице стилей может оказаться несколько подходящих правил преобразования. Одним из них всегда будет правило по умолчанию. Как бы то ни было, к узлу всегда применяется только одно правило. Оно отбирается так:

- во-первых, из всех импортированных правил отбирается то, которое записано в таблицу стилей в последнюю очередь;
- во-вторых, импортированное правило отбрасывается, если есть подходящее правило в самой таблице стилей;
- в-третьих, отбирается правило с наибольшим значением атрибута `priority`. Если атрибут `priority` отсутствует, то правилу назначается приоритет по умолчанию, вычисляемый по виду образца, записанного в атрибуте `match`.

Создание преобразованных узлов

После того как к текущему узлу подобрано правило его преобразования, выполняется конструктор последовательности узлов и атомарных значений, записанный в этом правиле. Определение конструктора носит чрезвычайно общий характер.

Конструктор последовательности — это последовательность узлов таблицы стилей, имеющих общий родительский узел, применение которой дает последовательность узлов

и атомарных значений. Общий родительский узел, о котором идет речь в определении, — это чаще всего узел-элемент `xsl:template`, хотя им могут служить элементы `xsl:variable`, `xsl:param` и многие другие элементы XSLT.

В конструкторе последовательности могут встречаться четыре вида узлов:

- текстовые узлы, которые будут просто скопированы в конструируемую последовательность.
- узлы-инструкции XSLT, генерирующие последовательности узлов и атомарных значений;
- внешние, не XSLT-инструкции с именами из пространства имен, отличного от пространства имен XSLT. Они тоже генерируют последовательности узлов и атомарных значений;
- элементы из пространства имен, отличного от пространства имен XSLT, не являющиеся внешними инструкциями. Они становятся узлами-элементами конструируемой последовательности.

Узлы-инструкции, входящие в конструктор, выполняются и вместе с остальными узлами конструктора образуют одну последовательность узлов и/или атомарных значений. Все подряд идущие атомарные значения из созданной конструктором последовательности преобразуются в строки и собираются в один текстовый узел, в котором они записываются через пробел. Затем в полученной последовательности узлов все подряд идущие текстовые узлы собираются в один текстовый узел без всяких разделителей.

Итак, после применения каждого правила получается последовательность узлов, а после использования всех правил таблицы стилей ко всем выбранным узлам исходного дерева получается набор таких последовательностей. Осталось собрать из этого набора последовательностей одно или несколько деревьев, которые и будут результатом преобразования. Они создаются или явно элементами `xsl:result-document`, или неявно начальным правилом преобразования.

Очень часто преобразованное дерево выводится на какое-то устройство в виде документа HTML, XHTML, XML или просто в виде плоского ASCII-текста. В самом простом случае выполняется сериализация, для которой в языке XSLT предусмотрен элемент `xsl:output`.

Преобразование документов с помощью DOM

W3C DOM представляет собой не зависящее от языка или платформы определение.

W3C DOM модель устанавливает стандартную функциональность для навигации по документу и манипулирования содержанием и структурой документов, написанных на языках XML и HTML.

При использовании DOM для работы с текстовым файлом в формате XML она анализирует файл, разбивает его на индивидуальные элементы, атрибуты, комментарии и т.д. Затем в памяти создается представление файла XML в виде дерева узлов в котором каждый объект в документе рассматривается в виде узла: элементы, атрибуты, комментарии, команды обработки и даже составляющий атрибуты обыкновенный текст.

После этого разработчик может обращаться к содержанию документа, используя дерево узлов, и при необходимости вносить в него изменения, например чтобы добавить новый элемент(для этого достаточно просто создать новый узел и прикрепить его в качестве потомка к нужному узлу).

В W3C DOM для различных составляющих DOM объектов определены интерфейсы облегчающие манипулирование с деревом узлов, но для этих интерфейсов не предлагается никакой специфической реализации, и ее можно осуществить на любом языке программирования.

Основные типы интерфейсов которые поддерживает объектная модель:

Интерфейс Document

Узел Document содержит следующие дочерние узлы: Element(не более чем один), ProcessingInstruction (команда приложения), Comment (комментарий), DocumentType (тип документа).

Документ представляет собой единое целое. Методы создания документа не предоставляются, эта задача возлагается на агента пользователя. Объектная модель документа не предоставляет способа сохранения сформированного документа, эта задача отводится агенту пользователя. Большинство документов построенных с помощью скрипта исчезает при закрытии окна браузера клиента вследствие того что скрипты клиентов используемые в браузерах не имеют доступа к жесткому диску сервера.

Значение свойства Name интерфейса Document равно #document, а значение свойства Value интерфейса Document равно null.

Документ может иметь только один узел Element, а именно корневой узел.

Интерфейс Node

Все типы узлов объектной модели документа наследуют основной набор свойств от интерфейса Node.

Интерфейс NodeList

Доступ к узлам NodeList можно получить с помощью метода `item`, который в качестве аргумента использует `index`. Длину списка можно определить методом `length`.

Интерфейс `NodeList` хранит запись дочерних узлов конкретного узла и их позиции. Он является основой структуры дерева документа в памяти компьютера.

DOM в браузерах Internet Explorer 5

В браузерах Internet Explorer 5 и выше находятся встроенные библиотеки DOM и поддержка XSL. Для сценариев на стороне клиента доступно множество объектов для работы с XML-документом, самые важные из них, объекты **XMLDOMDocument**, **XMLDOMNode**, **XMLDOMNodeList**, **XMLDOMParseError** представляющие интерфейс для доступа ко всему документу, отдельным его узлам и поддеревьям, предоставляющие необходимую для отладки информацию о произошедших ошибках анализатора соответственно.

Объект `XMLDOMNode` реализует базовый DOM интерфейс **Node**, предназначен для манипулирования с отдельным узлом дерева документа. Его свойства и методы позволяют получать и изменять полную информацию о текущем узле - его тип (является ли текущий узел элементом, комментарием, текстом и т.д.), название, полное название (вместе с Namespace префиксом), его содержимое, список дочерних элементов и т.д.

Объект `XMLDOMDocument` представляет верхний уровень объектной иерархии и содержит методы для работы с документом: его загрузки, анализа, создания в нем элементов, атрибутов, комментариев и т.д. Многие свойства и методы этого объекта реализованы также в класса `Node`, т.к. документ может быть рассмотрен как корневой узел с вложенными в него поддеревьями.

Объект `XMLDOMNodeList` представляет собой список узлов - поддеревья и содержит методы, при помощи которых можно организовать процедуру обхода дерева.

Объект `XMLDOMParserError` объект позволяет получить всю необходимую информацию об ошибке, произошедшей в ходе разбора документа. Все свойства этого объекта доступны только для чтения.

Создание XML документа с использованием Microsoft Visual Studio .Net

Microsoft Visual Studio .Net позволяет достаточно легко создавать XML документы и XSD схемы к ним. Для того чтобы создать XML документ необходимо в среде разработки Visual Studio .Net выполнить команду меню **File | New | File...** в появившемся диалоговом окне **New File** в наборе **Templates** выделить значок **XML File**. В результате на основном рабочем поле появится следующего вида страница:

На данной странице можно набрать XML документ. Причем при написании открывающегося тега закрывающийся тег будет появляться автоматически. Заполнить

XML документ данными можно с помощью раздела Data. Для этого в низу окна нужно найти кнопку “Data”. В результате страница приобретет примерно следующий вид:

Введенные таким образом данные будут добавлены в XML документ.

Для того чтобы для созданного XML документа написать XSD схему необходимо на рабочем поле щелкнуть правой кнопкой мыши и в выпадающем списке выбрать **Create Schema**. В результате появится:

Тип элементов и атрибутов можно выбрать из выпадающего списка.

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Изучить конструкции языка XML, способы описания структуры документа средствами DTD, описания схемы документа на языке XSD. Изучить способы обработки документов XML.
2. Написать по созданной в лабораторной работе №1 статистической и динамической модели XML документ.
3. Описать структуру документа средствами DTD.
4. Описать схему документа на языке XSD
5. Преобразовать документ средствами XSLT
6. Преобразовать документ с помощью DOM
7. Оформить отчет, включающий постановку задачи, тексты XML, DTD, XSD, XSLT документов, демонстрацию работы программ.

8. Защитить лабораторную работу.

Контрольные вопросы:

1. Что такое языки разметки. Стилистическая, структурная и семантическая разметки.
2. История развития языков разметки: SGML, HTML, XML.
3. Что такое XML. Как появился XML. XML в качестве данных
4. Структура XML документа. Правильно оформленные и верные документы.
5. Анализатор XML. Событийно управляемые анализаторы. Древовидные анализаторы.
6. Зачем нужно определение типа документа (DTD). Общие принципы написания определений DTD:
7. Недостатки и особенности определений DTD.
8. Моделирование данных с использованием XML. Информационная модель. Отображение на XML. Схемы.
9. Объектная модель документа (DOM).
10. Пространство имен и схемы.
11. Связи и запросы.
12. Трансформация документа XML.
13. Дизайн документов XML.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.
2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №13. Способы создания Web-сервисов.

Тема - Способы создания Web-сервисов.

Цель работы - освоить способы создания Web-сервисов.

Формируемые компетенции или их части - ОПК-1

Теоретическая часть.

Для того чтобы создать самый простой Web-сервис, который по запросу пользователя будет выдавать текущую дату или комбинацию даты и времени в среде разработки Visual Studio .NET следует выполнить команду меню **File | New | Project** и в появившемся диалоговом окне **New Project** в наборе **Templates** выделить значок **ASP.NET Web Service**. В поле **Name** следует указать наименование создаваемого проекта. После того, как все необходимые приготовления средой разработки будут сделаны, на основном рабочем поле появятся две новые страницы. Одна будет предназначена для разработки визуального дизайна, а на второй будет располагаться код нашего сервиса. Естественно, у создаваемого сервиса не может быть внешнего вида как такового, ему нечего отображать, поэтому страницу для дизайна можно спокойно закрыть.

В состав одного проекта может входить несколько отдельных сервисов. Для нашего примера потребуется всего один Web-сервис, заготовка для которого создается средой разработки Visual Studio .NET автоматически, поэтому ничего изменять не потребуется, и мы можем сразу перейти на страницу с наименованием `Service1.aspx.vb`. На этой странице нужно разместим код нашего Web-сервиса:

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

```

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

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
'CODEGEN: This procedure is required by the Web Services Designer
'Do not modify it using the code editor.
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub
#End Region

<WebMethod()> Public Function MyDate(ByVal ShowTime As Boolean
) As String
Dim MD As DateTime
If ShowTime Then
MyDate = MD.Now
Else
MyDate = MD.Today
End If
End Function
End Class

```

Из всего этого кода лишь малая часть написана разработчиком. Это функция MyDate, расположенная в конце листинга.

После компиляции к созданному Web сервису можно обратиться даже из браузера.

На рис. приведен внешний вид Web-страницы, которая генерируется Web-сервером при попытке обращения к Web сервису.

На этой Web-странице расположена ссылка на формальное описание структуры Web-сервиса, а также перечислены все функции, поддерживаемые Web-сервисом. В нашем случае, естественно, указана лишь одна функция MyDate. Наименование функции также является гиперссылкой, нажав на которую можно перейти к Web-странице, позволяющей воспользоваться этой функцией. После щелчка мышью на ссылке в браузере

будет отображена Web-страница следующего вида На этой странице расположено поле ввода, в котором пользователь может указать значение параметра, передаваемого функции.

В результате работы функции сервиса получим:

```
<?xml version="1.0" encoding="utf-8" ?>
```

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

Клиентская часть

Для создания клиентского приложения нужно для начала создать новый проект.

Для этого следует выполнить команду меню **File | New | Project**. В появившемся диалоговом окне **New Project** в наборе **Templates** выбрать шаблон **Windows**

Application и в поле **Name** указать наименование создаваемого приложения.

После этого Visual Studio .NET создаст форму с именем Form1, устанавливаемым по умолчанию. В окне этой формы нам потребуется разместить один элемент Checkbox, одно текстовое поле Label, и одну кнопку Button. При помощи независимого переключателя CheckBox1 мы будем задавать параметр, передаваемый функции MyDate, входящей в состав сервиса. Текстовое поле будет содержать результат, возвращаемый этой функцией, а кнопка — запускать процесс установки связи с Web-сервисом и получения от него требуемых данных.

После этого следует установить связь с искомым Web-сервисом. Для этого надо в окне **Solution Explorer** выбрать наименование создаваемого приложения и правым щелчком мыши вызвать для него контекстное меню. В этом контекстном меню необходимо выполнить команду **Add Web Reference**, после чего будет активизировано одноименное диалоговое окно, позволяющее установить ссылки на используемые сервисы.

Для того чтобы получить ссылки на функции Web-сервиса и перенести их в проект, нужно в текстовом поле **Address** указать URL необходимого сервиса и загрузить ресурс, располагающийся по этому адресу. После этого в левой части искомого окна **Add Web Reference** будет отображено содержимое стартовой Web-страницы.

После того как искомые ссылки на функции Web-сервиса будут найдены, при помощи кнопки **Add Reference** их необходимо добавить к разрабатываемому проекту.

После чего можно обращаться к функциям Web-сервиса в приложении (функция Button_Click).

```
Public Class Form1
```

```
Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

```

Public Sub New()
MyBase.New()
'This call is required by the Windows Form Designer.
InitializeComponent()
'Add any initialization after the InitializeComponent() call
End Sub

'Form overrides dispose to clean up the component list.
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Friend WithEvents CheckBox1 As System.Windows.Forms.CheckBox
Friend WithEvents Label1 As System.Windows.Forms.Label
Friend WithEvents Button1 As System.Windows.Forms.Button
<System.Diagnostics.DebuggerStepThrough> Private Sub InitializeComponent()
Me.CheckBox1 = New System.Windows.Forms.CheckBox()
Me.Label1 = New System.Windows.Forms.Label()
Me.Button1 = New System.Windows.Forms.Button()
Me.SuspendLayout()
'CheckBox1
,

Me.CheckBox1.Location = New System.Drawing.Point(16, 16)
Me.CheckBox1.Name = "CheckBox1"
Me.CheckBox1.Size = New System.Drawing.Size(168, 24)
Me.CheckBox1.TabIndex = 0
Me.CheckBox1.Text = "Отображать время"

```

```

    ,

    'Label1
    ,

    Me.Label1.Location = New System.Drawing.Point(16, 64)
    Me.Label1.Name = "Label1"
    Me.Label1.Size = New System.Drawing.Size(264, 23)
    Me.Label1.TabIndex = 1
    ,

    'Button1
    ,

    Me.Button1.Location = New System.Drawing.Point(16, 128)
    Me.Button1.Name = "Button1"
    Me.Button1.Size = New System.Drawing.Size(256, 32)
    Me.Button1.TabIndex = 2
    Me.Button1.Text = "Запуск Web-сервиса"
    ,

    40
    'Form1
    ,

    Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
    Me.ClientSize = New System.Drawing.Size(292, 273)
    Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.Button1,
Me.Label1, Me.CheckBox1})
    Me.Name = "Form1"
    Me.Text = "Form1"
    Me.ResumeLayout(False)
    End Sub
    #End Region

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
Button1.Click
    Dim t1 = New WindowsApplication1.localhost.Service1()
    Label1.Text = t1.MyDate(CheckBox1.Checked)
End Sub
End Class

```

После запуска приложение будет выглядеть следующим образом:

XML-документ в качестве источника данных

Создадим Web-сервис, который будет включать единственную функцию, возвращающую обычный набор данных (DataSet), содержащий информацию, взятую из XML-файла.

Для этого создадим новый проект Web-сервиса. На его страницу отбуксируем один компонент DataSet с вкладки Data. При этом будет активировано диалоговое окно **AddDataset**, в котором разработчику предлагается указать тип создаваемого набора данных. Так как в нашем случае XML-файл может содержать данные любой структуры, то для набора данных нельзя заранее указать структуру. Следовательно, в этом диалоговом окне разработчик должен выбрать переключатель **Untyped dataset**.

Код Web-сервиса приведен ниже.

```
Imports System.Web.Services
<WebService(Namespace := "http://tempuri.org")> _
Public Class Service1
Inherits System.Web.Services.WebService
#Region " Web Services Designer Generated Code "
Public Sub New()
MyBase.New()
'This call is required by the Web Services Designer.
InitializeComponent()
'Add your own initialization code after the InitializeComponent() call
End Sub
'Required by the Web Services Designer
Private components As System.ComponentModel.IContainer
'NOTE: The following procedure is required by the Web Services Designer
'It can be modified using the Web Services Designer.
'Do not modify it using the code editor.
Friend WithEvents DataSet1 As System.Data.DataSet
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
Me.DataSet1 = New System.Data.DataSet()
CType(Me.DataSet1, System.ComponentModel.ISupportInitialize).BeginInit()
'
'DataSet1
'
```

```

Me.DataSet1.DataSetName = "NewDataSet"
Me.DataSet1.Locale = New System.Globalization.CultureInfo("ru-RU")
CType(Me.DataSet1, System.ComponentModel.ISupportInitialize).EndInit()
End Sub

Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
'CODEGEN: This procedure is required by the Web Services Designer
'Do not modify it using the code editor.
If disposing Then
If Not (components Is Nothing) Then
components.Dispose()
End If
End If
MyBase.Dispose(disposing)
End Sub

#End Region

<WebMethod> Public Function XMLData(ByVal file As String) As DataSet
Dim FS As IO.FileStream
Dim Reader As IO.StreamReader
FS = New IO.FileStream(Server.MapPath(file), IO.FileMode.Open,
IO.FileAccess.Read)
Reader = New IO.StreamReader(FS)
DataSet1.ReadXml(Reader)
FS.Close()
XMLData = DataSet1
End Function
End Class

```

Если содержимое XML-файла, используемого в нашем примере, будет выглядеть следующим образом:

```

<Document>
<Row>
<Column1>1</Column1>
<Column2>2</Column2>
<Column3>Text1</Column3>
<Column4>Text2</Column4>
</Row>

```

```

<Row>
<Column1>3</Column1>
<Column2>4</Column2>
<Column3>Text3</Column3>
<Column4>Text4</Column4>
</Row>
</Document>,

```

то в результате работы данного Web- сервиса мы получим:

```

<?xml version="1.0" encoding="utf-8" ?>
- <DataSet xmlns="http://tempuri.org/">
-
    <xs:schema id="Document" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
- <xs:element name="Document" msdata:IsDataSet="true" msdata:Locale="ru-RU">
- <xs:complexType>
- <xs:choice maxOccurs="unbounded">
- <xs:element name="Row">
- <xs:complexType>
- <xs:sequence>
<xs:element name="Column1" type="xs:string"
minOccurs="0" />
<xs:element name="Column2" type="xs:string"
minOccurs="0" />
<xs:element name="Column3" type="xs:string"
minOccurs="0" />
<xs:element name="Column4" type="xs:string"
minOccurs="0" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:choice>
</xs:complexType>
</xs:element>
</xs:schema>
- <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata"

```

```

xmlns:diffgr="urn:schemas-microsoft-com:xml-diffgram-v1">
- <Document xmlns="">
- <Row diffgr:id="Row1" msdata:rowOrder="0" diffgr:hasChanges="inserted">
<Column1>1</Column1>
<Column2>2</Column2>
<Column3>Text1</Column3>
<Column4>Text2</Column4>
</Row>
- <Row diffgr:id="Row2" msdata:rowOrder="1" diffgr:hasChanges="inserted">
<Column1>3</Column1>
<Column2>4</Column2>
<Column3>Text3</Column3>
<Column4>Text4</Column4>
</Row>
</Document>
</diffgr:diffgram>
</DataSet>

```

Для создания клиентского приложения нужно для начала создать новый проект.

Для этого как и в предыдущем примере следует выполнить команду меню **File | New | Project**. В появившемся диалоговом окне **New Project** в наборе **Templates** выбрать аблон **Windows Application** и в поле **Name** указать наименование создаваемого приложения.

После этого Visual Studio .NET создаст форму с именем Form1, устанавливаемым по умолчанию. В окне этой формы нам потребуется разместить один элемент TextBox, одно текстовое поле Label, одну кнопку Button, один компонент DataGrid, и один компонент DataSet с вкладки Data. Пример формы приведен ниже.

После этого следует установить связь с искомым Web-сервисом. Для этого надо в окне **Solution Explorer** выбрать наименование создаваемого приложения и правым щелчком мыши вызвать для него контекстное меню. В этом контекстном меню необходимо выполнить команду **Add Web Reference**, после чего будет активизировано одноименное диалоговое окно, позволяющее установить ссылки на используемые сервисы.

Для того чтобы получить ссылки на функции Web-сервиса и перенести их в проект, нужно в текстовом поле **Address** указать URL необходимого сервиса и загрузить ресурс,

располагающийся по этому адресу. После этого в левой части искомого окна **Add Web Reference** будет отображено содержимое стартовой Web-страницы.

После того как искомые ссылки на функции Web-сервиса будут найдены, при помощи кнопки **Add Reference** их необходимо добавить к разрабатываемому проекту.

После чего можно обращаться к функциям Web-сервиса в приложении (функция Button_Click).

```
Public Class Form1
```

```
Inherits System.Windows.Forms.Form
```

```
#Region " Windows Form Designer generated code "
```

```
Public Sub New()
```

```
MyBase.New()
```

```
'This call is required by the Windows Form Designer.
```

```
InitializeComponent()
```

```
'Add any initialization after the InitializeComponent() call
```

```
End Sub
```

```
'Form overrides dispose to clean up the component list.
```

```
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
```

```
If disposing Then
```

```
If Not (components Is Nothing) Then
```

```
components.Dispose()
```

```
End If
```

```
End If
```

```
MyBase.Dispose(disposing)
```

```
End Sub
```

```
'Required by the Windows Form Designer
```

```
Private components As System.ComponentModel.IContainer
```

```
'NOTE: The following procedure is required by the Windows Form Designer
```

```
'It can be modified using the Windows Form Designer.
```

```
'Do not modify it using the code editor.
```

```
Friend WithEvents TextBox1 As System.Windows.Forms.TextBox
```

```
Friend WithEvents Label1 As System.Windows.Forms.Label
```

```
Friend WithEvents Button1 As System.Windows.Forms.Button
```

```
Friend WithEvents DataGridView1 As System.Windows.Forms.DataGridView
```

```
Friend WithEvents DataSet1 As System.Data.DataSet
```

```
<System.Diagnostics.DebuggerStepThrough> Private Sub InitializeComponent()
```



```

Me.DataGrid1.DataMember = ""
Me.DataGrid1.DataSource = Me.DataSet1
Me.DataGrid1.HeaderForeColor = System.Drawing.SystemColors.ControlText
Me.DataGrid1.Location = New System.Drawing.Point(16, 88)
Me.DataGrid1.Name = "DataGrid1"
Me.DataGrid1.Size = New System.Drawing.Size(264, 80)
Me.DataGrid1.TabIndex = 3
'
'DataSet1
'

Me.DataSet1.DataSetName = "NewDataSet"
Me.DataSet1.Locale = New System.Globalization.CultureInfo("ru-RU")
'
'Form1
'

Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(292, 273)
Me.Controls.AddRange(New System.Windows.Forms.Control() { Me.DataGrid1,
Me.Button1,
Me.Label1, Me.TextBox1})
Me.Name = "Form1"
Me.Text = "Form1"
CType(Me.DataGrid1, System.ComponentModel.ISupportInitialize).EndInit()
CType(Me.DataSet1, System.ComponentModel.ISupportInitialize).EndInit()
Me.ResumeLayout(False)
End Sub
#End Region

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
Button1.Click
Dim xml = New WindowsApplication2.localhost.Service1()
DataSet1.Merge(xml.XMLData(TextBox1.Text))
End Sub
End Class

```

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Изучить основные принципы создания Web-сервисов.
2. Создать web-сервис, который будет использовать XML документ в качестве источника данных. То есть создать Web- сервис, возвращающий обычный набор данных содержащий информацию, взятую из XML-файла.
3. Дополнить созданный web- сервис процедурой позволяющей сохранять данные в XML файле.
4. Создать клиентское приложение.
5. Оформить отчет, включающий постановку задачи, текст программ, демонстрацию работы программ.
6. Защитить лабораторную работу

Контрольные вопросы:

1. Web- службы. Основные понятия. Назначение.
2. История развития Web-служб. Модель COM/DCOM. .Стандарт CORBA/IIOP. Недостатки перечисленных технологий.
3. Web-службы .Net. Достоинства. Архитектура.
4. Базовые технологии web-служб: SOAP, UDDI, WSDL.

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>

ЛАБОРАТОРНАЯ РАБОТА №14. Технологии создания и валидации XML-документов

Тема - Технологии создания и валидации XML-документов

Цель работы - освоить технологии создания и валидации XML-документов.

Формируемые компетенции или их части - ОПК-5

Теоретическая часть.

1. Создание XML-документов с помощью среды Microsoft Visual Web Developer

Методика выполнения

1.1. Создать новый XML-документ

Пуск | Программы | Microsoft Visual Web Developer 2008 Express Edition File → New File... →ф. New File | выбр. Xml File →кн. Open

1.2. Определить инструкцию по обработке (если инструкция по обработке уже задана, то пропустить этот пункт):

ф. New File.xml | Ввести строку кода:

```
<?xml version="1.0" encoding="utf-8"?>
```

перейти на новую строку

1.3. Задать корневой элемент root

ф. New File.xml | Ввести открывающую угловую скобку (<); Указать имя элемента (root); Ввести закрывающую угловую скобку (>) → будет автоматически сгенерирован закрывающий тег элемента root

1.4. Создать вложенный элемент первого уровня test1

ф. New File.xml | между открывающим и закрывающим тегами элемента root задать соответствующие теги элемента test1 (форматирование документа будет выполнено автоматически)

1.5. Задать атрибут attr1 со значением value1

ф. New File.xml | установить курсор в открывающий тег элемента test1 после имени элемента до закрывающей угловой скобки; ввести пробел; ввести имя атрибута attr1; ввести символ «=» → будут автоматически сгенерированы двойные кавычки для представления значения атрибута; в текущей позиции курсора ввести текст «value1».

1.6. Аналогичным образом ввести вложенный элемент второго уровня test1-1 с атрибутом attr1-1 со значением value1-1 (test1-1 – дочерний элемент для элемента test1)

1.7. Сохранить документ.

Задание для выполнения

По заданной схеме (рис. 1) и варианту построить XML-документ. Для каждого элемента первого уровня предусмотреть не менее 3 экземпляров, для каждого элемента последующих уровней – не менее двух экземпляров.

Вариант А. Успеваемость студентов

Вариант Б. Книжный магазин

Вариант В. Кинотеатр

Вариант Г. Аэропорт

Вариант Д. Отдел кадров

2. Проверка корректности XML-документов с помощью браузера

Методика выполнения

2.1. Открыть XML-документ в браузере. Если документ корректный, в окне браузера будет отображена иерархическая XML-структура.

2.2. Сделать документ некорректным по основным требованиям корректности.

2.2.1. Скопировать корневой элемент root вместе с его содержимым в конец документа (в итоге должно получиться два следующих друг за другом элемента root). После открытия документа в браузере, будет получено сообщение об ошибке: **«В документах XML допускается только один элемент верхнего уровня.»**

2.2.2. Заменить имя элемента root только в открывающем теге на Root.

После открытия документа в браузере, будет получено сообщение об ошибке:

«Тег конца "root" не соответствует тегу начала "Root".»

2.2.3. Перенести закрывающий тег элемента test1-1 после закрывающего тега элемента test1. После открытия документа в браузере, будет получено сообщение об ошибке: **«Тег конца "test1" не соответствует тегу начала "test1- 1".»**

2.3. Отменить все выполнения изменения, чтобы документ снова стал корректным.

Задание для выполнения

Включить в отчет сообщения о некорректности документа по аналогии с методикой выполнения.

3. Создание XSD-схемы XML-документа с помощью среды Microsoft Visual Web Developer

Методика выполнения

3.1. Создать новую XSD-схему

Пуск | Программы | Microsoft Visual Web Developer 2008 Express Edition File → New File... →ф. New File | выбр. Xml Schema →кн. Open

3.2. Изменить префикс с xs на xsd:

ф. XMLSchema1.xsd | в открывающем и закрывающем тегах элемента schema заменить xs:schema на xsd:schema; в строке

xmlns:xs="http://www.w3.org/2001/XMLSchema" заменить xs на xsd

Удалить атрибуты targetNamespace, elementFormDefault, xmlns, xmlns:mstns вместе с их значениями.

3.3. Определить описание корневого элемента root

ф. XMLSchema1.xsd | после закрывающей угловой скобки открывающего тега xsd:schema выполнить переход на новую строку и ввести открывающую угловую скобку («<») → из открывшегося списка выбрать значение «xsd:element» и нажать на [Enter] → поставить пробел → из открывшегося списка выбрать значение «name» и нажать на [Enter] → автоматически будет создан атрибут name с двойными кавычками для значения → в текущей позиции курсора (в кавычках) ввести значение «root» → Ввести закрывающую угловую скобку (>) → будет автоматически сгенерирован закрывающий тег элемента xsd:element

3.4. Определить структуру элемента root (сложный / complexType с элементами / sequence)

ф. XMLSchema1.xsd | между открывающим и закрывающим тегами элемента xsd:element ввести открывающую угловую скобку («<») → из открывшегося списка выбрать значение «xsd:complexType» и нажать на [Enter] → Ввести закрывающую угловую скобку (>) → будет автоматически сгенерирован закрывающий тег элемента xsd:complexType. Аналогичным образом в составе элемента xsd:complexType создать элемент xsd:sequence.

3.4. Создать описание вложенного элемента первого уровня test1 (сложный / complexType с элементами / sequence)

ф. New File.xml | между открывающим и закрывающим тегами элемента xsd:sequence создать элемент xsd:element со значением свойства name, равным «test1». В составе нового элемента xsd:element определить элементы xsd:complexType и xsd:sequence.

3.5. По аналогии с п. 3.4 определить элемент (xsd:element) с именем test1-1 между открывающим и закрывающим тегами элемента xsd:sequence.

3.6. Создать описание атрибута attr1

ф. New File.xml | установить курсор после закрывающего тега

xsd:sequence до закрывающего тега xsd:complexType из п. 3.4; ввести открывающую угловую скобку («<») → из открывшегося списка выбрать значение «xsd:attribute» и нажать на [Enter] → поставить пробел → из открывшегося списка

выбрать значение «name» и нажать на [Enter] → автоматически будет создан атрибут name с двойными кавычками для значения → в текущей позиции курсора (в кавычках) ввести значение «attr1» → Аналогичным образом указать параметр type со значением xsd:string → Ввести закрывающую угловую скобку (>) → будет автоматически сгенерирован закрывающий тег элемента xsd:attribute

3.7. Аналогичным образом создать описание атрибута attr1-1.

3.8. Сохранить XSD-схему.

Задание для выполнения

Создать XSD-схему в соответствии со своим вариантом XML-документа.

4. Проверка валидности XML-документа с помощью JavaScript

4.1. Открыть XML-документ, созданный в п.1.

4.2. Ввести ссылку на схему в корневом элементе:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="Имя схемы.xsd"
```

4.3. Создать текстовый документ.

4.4. Внести в документ следующий код:

```
var sOutput = validateFile("Имя XML-файла.xml");
WScript.Echo(sOutput);
function validateFile(strFile)
{
var x = new ActiveXObject("MSXML2.DOMDocument.4.0");
x.async = false;
x.validateOnParse = true;
x.resolveExternals = true;
x.load(strFile);
if (x.parseError.errorCode != 0)
{
return("НЕ ВАЛИДНЫЙ документ " + strFile +
"\n===== " +
"\nПричина: " + x.parseError.reason +
"\nИсточник: " + x.parseError.srcText +
"\nСтрока: " + x.parseError.line + "\n");
}
else
return("Валидный документ " + strFile +
```

```
"\n=====\\n" +  
x.xml + "\\n");  
}
```

4.5. Сохранить документ как validate.js.

4.6. Запустить скрипт validate.js.____

Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система WINDOWS 7 и выше, MongoDB 2.4.3.

Указания по технике безопасности

Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров. Самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Содержание отчета

1. Изучить основные принципы создания Web-сервисов.
2. Создать web-сервис, который будет использовать XML документ в качестве источника данных. То есть создать Web- сервис, возвращающий обычный набор данных содержащий информацию, взятую из XML-файла.
3. Дополнить созданный web- сервис процедурой позволяющей сохранять данные в XML файле.
4. Создать клиентское приложение.
5. Оформить отчет, включающий постановку задачи, текст программ, демонстрацию работы программ.
6. Защитить лабораторную работу

Список литературы, рекомендуемый к использованию по данной теме

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее

профессиональное образование. Информатика и вычислительная техника. Бакалавриат) -
Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.

2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224
с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL:
<http://biblioclub.ru/index.php?page=book&id=231519>

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

1. Кузин, Александр Владимирович. Базы данных : учеб. пособие для вузов / А.В. Кузин, С.В. Левонисова. - 5-е изд., испр. - Москва : Академия, 2012. - 314, [1] с. : ил. ; 22. - (Высшее профессиональное образование. Информатика и вычислительная техника. Бакалавриат) - Гриф: Доп. УМО. - 1000 экз. - ISBN 978-5-7695-9308-6.
2. Шнырев, С.Л. Базы данных : учебное пособие / С.Л. Шнырев. - М. : МИФИ, 2011. - 224 с. - ISBN 978-5-7262-1483-2 ; То же [Электронный ресурс]. - URL: <http://biblioclub.ru/index.php?page=book&id=231519>
3. Секционированные таблицы и индексы SQL Server 2005, По материалам статьи Kimberly L. Tripp: SQL Server 2005. Partitioned Tables and Indexes, Перевод Алексея Сафонова, <http://www.sql.ru/articles/mssql/2005/073102PartitionedTablesAndIndexes.shtml>
4. Microsoft MSDN, <http://msdn.microsoft.com/ru-ru/>
5. Братченко, Н. Ю. (СевКавГТУ). Базы данных : учеб. пособие / Н. Ю. Братченко ; Сев.-Кав. гос. техн. ун-т. - Ставрополь : СевКавГТУ, 2011. - 195 с. - Библиогр.: с. 194
6. Братченко, Н. Ю. (СевКавГТУ). Базы данных. Основы работы с СУБД ORACLE : учеб. пособие (лабораторный практикум) / Н. Ю. Братченко ; ФГБОУ Сев.-Кав. гос. техн. ун-т. - Ставрополь : Издательство СевКавГТУ, 2011. - 129 с. : ил. - Библиогр.: с. 128-129
7. Характеристика средств администрирования баз данных и перспективы их развития [Электронный ресурс] / Н. И. Кусютин. - М.: Лаборатория книги, 2011. - 99 с. - Режим доступа:
8. Илюшечкин В. М. Основы использования и проектирования баз данных. Учебное пособие [Электронный ресурс] / В. М. Илюшечкин. - М.: ЮРАЙТ, 2011. - 214 с.
9. Советов, Б. Я. Базы данных : теория и практика : учебник / Б.Я. Советов, В.В. Цехановский, В.Д. Чертовской. - 2-е изд. - М. : Юрайт, 2012. - 464 с. - (Бакалавр). - Гриф: Рек. УМО. - Библиогр.: с. 459-460. - ISBN 978-5-9916-1479-
10. Шаша Д., Бонне Ф., Оптимизация баз данных. Принципы, практика, решение проблем, М.: КУДИЦ-Образ, 2004.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

по организации самостоятельной работы
по дисциплине «Тюнинг баз данных / Database tuning»
для магистрантов направления 09.04.03 «Прикладная
информатика» направленность (профиль) «Управление
знаниями»

Ставрополь
2022

Введение

Дисциплина «Тюнинг баз данных», формирует профессиональные навыки магистра по направлению подготовки 09.04.03 «Прикладная информатика».

Изучение дисциплины «Тюнинг баз данных» базируется на проведении лабораторных работ и индивидуальных занятий под руководством преподавателя, самостоятельной работе студентов

Задачами курса являются:

- развитие логического и алгоритмического мышления;
- изучение принципов работы программно-технических средств и организации данных в информационных системах, использующих БД;
- освоение работы с современными СУБД;
- выработка умения самостоятельного решения задач обработки текстовой и нетекстовой информации в БД;
- получение навыков в алгоритмизации задач, программировании на алгоритмическом языке, отладке и выполнении задач на персональном компьютере;
- изучение перспектив развития информационных технологий в информационных системах в предметной области;
- изучение рынков информационных ресурсов и особенностей их использования.

1. Общая характеристика самостоятельной работы студента при изучении дисциплины Тюнинг баз данных – формирование набора профессиональных компетенций будущего магистра по направлению подготовки 09.04.03 «Прикладная информатика».

2. Задачи освоения дисциплины:

- развитие логического и алгоритмического мышления;
- изучение принципов работы программно-технических средств и организации данных в информационных системах, использующих БД;
- освоение работы с современными СУБД;
- выработка умения самостоятельного решения задач обработки текстовой и нетекстовой информации в БД;
- получение навыков в алгоритмизации задач, программировании на алгоритмическом языке, отладке и выполнении задач на персональном компьютере;
- изучение перспектив развития информационных технологий в информационных системах в предметной области;
- изучение рынков информационных ресурсов и особенностей их использования.

3. Формируемые компетенции - ОПК-5

4. План-график выполнения самостоятельной работы

№ п/п	Раздел дисциплины	Неделя семестра	СРС в часах	Формы текущего контроля успеваемости
1	Лабораторная работа 1. Индексные структуры	1.	3	Собеседования по лабораторным работам
2	Лабораторная работа 2. Соединение таблиц данных	2.	3	Собеседования по лабораторным работам
3	Лабораторная работа 3. Алгоритмы соединения таблиц	3.	3	Собеседования по лабораторным работам
4	Лабораторная работа 4. Запросы с подзапросами	4.	3	Собеседования по лабораторным работам
5	Лабораторная работа 5. Планы выполнения запросов	5.	3	Собеседования по лабораторным работам
6	Лабораторная работа 6. Транзакции	6.	3	Собеседования по лабораторным работам
7	Лабораторная работа 7. Работа с XML. Часть 1	7.	3	Собеседования по лабораторным работам
8	Лабораторная работа 8. Работа с XML. Часть 2	8.	3	Собеседования по лабораторным работам
9	Лабораторная работа 9. Иерархически организованные данные	9.	3	Собеседования по лабораторным работам

				работам
10	Лабораторная работа №10. Секционирование таблиц	10.	3	Собеседования по лабораторным работам
11	Лабораторная работа №11. Принципы построения статистической и динамической модели	11.	3	Собеседования по лабораторным работам
12	Лабораторная работа №12. Конструкции языка XML	12.	3	Собеседования по лабораторным работам
13	Лабораторная работа №13. Способы создания Web-сервисов	13.	2,5	Собеседования по лабораторным работам
14	Лабораторная работа №14. Технологии создания и валидации XML-документов	14.	2	Собеседования по лабораторным работам
ИТОГО			40,5	

5. Методические рекомендации по изучению теоретического материала

На первом этапе необходимо ознакомиться с рабочей программой дисциплины, в которой рассмотрено содержание тем дисциплины лекционного курса, взаимосвязь тем лекций с практическими занятиями, темы и виды самостоятельной работы. По каждому виду самостоятельной работы предусмотрены определённые формы отчетности.

Лабораторная работа №1 Индексные структуры

7. Проверьте наличие индексов по ключевым полям таблицы. При необходимости создайте кластеризованные индексы. Для создания нового индекса воспользуйтесь командой `CREATE INDEX`, или в среде Microsoft SQL Management Studio в разделе Tables/имя_таблицы/Indexes используйте команду New Index...

8. Создайте некластеризованные индексы по полям внешних ключей таблиц базы данных. Объясните, для чего нужны такие индексы?

9. Создайте некластеризованные индексы по информационным полям: Name и Date во всех таблицах базы данных. Объясните, для чего нужны такие индексы?

10. Для кластерного индекса и индекса по полю Date таблицы записей в чеке получите сведения о расширенных свойствах индексов. Объяснить значение информации, представленной в разделе «Fragmentation» на странице «Properties». Объясните, как вычислена глубина дерева индекса, число листьев, коэффициент фрагментации.

11. Перестройте кластеризованный индекс таблицы BillItem, используя команду `ALTER INDEX` или с помощью команды Rebuild в контекстном меню индекса.

12. Подготовьте материал для включения в отчетную презентацию по курсу «Тьюнинг баз данных».

Лабораторная работа №2 Соединение таблиц данных

1. Фильтр по отдельному названию товара. Используйте одиночное условие фильтрации «=». Название товара выберите самостоятельно.
2. Фильтр по множеству названий товаров. Используйте условие фильтрации in. Перечень товаров выберите самостоятельно.
3. Фильтр по отдельной дате. Используйте одиночное условие фильтрации «=». Дату чека выберите самостоятельно.
4. Фильтр с ограничением диапазона по датам чека. Используйте фильтрацию по диапазону between или совместно «<=», «>=». Диапазон дат выберите самостоятельно.
5. Фильтр по полуоткрытому интервалу дат чека. Используйте фильтрацию по условию «<» или «>». Диапазон дат выберите самостоятельно.

Лабораторная работа №3 Алгоритмы соединения таблиц

1. Запросы, созданные в пунктах 1,2 предыдущей лабораторной работы переделайте таким образом, чтобы для соединения таблиц использовался алгоритмы соединения с использованием вложенных циклов, соединения хешированием и соединения слиянием.
2. Оцениваем время выполнения каждого из полученных запросов.
3. Полученные запросы сохраните в представлениях, добавляя к имени исходного представления имя используемого алгоритма.
4. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Лабораторная работа №4 Запросы с подзапросами

1. Выберите все чеки на определенную дату, которые не единственны и существуют другие чеки на данную дату. Для формирования запроса используйте оператор EXISTS.
2. Выберите всех покупателей, которые не делали покупки на определенную дату. Для формирования запроса используйте операторы NOT EXISTS.
3. Создайте временную таблицу (команда CREATE TABLE) или переменную типа таблица (команда DECLARE TABLE) со структурой как у таблицы покупатель, в которую добавьте сведения о пяти произвольных покупателях (используйте оператор INSERT). Выберите сведения о 5-ти добавленных покупателях и даты их покупок. Используйте соединение временной таблицы (переменной типа таблица) с таблицей чеков Bill. Для формирования запроса используйте оператор IN.
4. Сохраните полученные запросы.
5. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Лабораторная работа №5 Планы выполнения запросов

1. Для запросов, созданных в пунктах 1 и 2 лабораторной работы 2 и запросов, созданных в пункте 1, лабораторной работы 3 сформируйте графический и текстовый план выполнения запросов. Изучите планы выполнения и объясните порядок соединения таблиц.

2. Оцените трудоемкость отдельных узлов в дереве плана запроса. Выберите самые трудоемкие операции, изучите их и объясните, почему указанные операции самые трудоемкие по сравнению с остальными.

3. Оцените объем данных, передаваемых между операторами плана выполнения запроса.

4. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Лабораторная работа №6 Транзакции

1. Добавьте в таблицу товар (Product) поле количество (Quantity). Заполните поле произвольными значениями от 0 до 100.

2. Создайте хранимую процедуру AddBill, которая будет формировать чек, а затем возвращать его идентификатор. Входные параметры процедуры: BuyerID и EmployeeID. Все инструкции данной хранимой процедуры должны выполняться в транзакции.

3. Создайте хранимую процедуру добавления товара в чек. Назовите ее AddBillItem. Входные параметры процедуры: BillID, ProductID, Quantity. Установить уровень изоляции транзакции в режим SERIALIZABLE. После открытия транзакции (BEGIN TRANSACTION) необходимо проверить количество товара в магазине. В случае если товара в магазине недостаточно, то процедура должна завершить выполнение транзакции (ROLLBACK TRANSACTION). В противном случае хранимая процедура уменьшает доступное количество товара в магазине и добавляет запись о покупке в чек (добавляются записи в таблицу BillItem). Закрываем транзакцию (COMMIT TRANSACTION). Объясните выбор уровня изоляции транзакции.

4. Сделайте выборку содержимого чека с использованием запроса №1 из лабораторной работы №2.

5. Проверьте работоспособность написанных хранимых процедур. Попробуйте создать чек и добавить в него 2 товара с достаточным в магазине количеством и 1 товар с недостаточным количеством. Убедитесь, что в чеке присутствуют только 2 проданных товара.

6. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Лабораторная работа №7 Работа с XML. Часть 1

7. Создайте новую хранимую процедуру AddBillXML. Входным параметром данной хранимой процедуры будет XML документ, пример которого показан ниже.

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

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

8. С помощью `sp_xml_preparedocument`, `sp_xml_removedocument` и `OPENXML` проведите разбор XML-документа, выделяя для записи чека поля: `productId` и `quantity`. Сформируйте плоскую временную таблицу с содержимым XML-документа.

9. После разбора XML-документа и формирования плоской таблицы откройте транзакцию для формирования нового чека. Создайте новый чек (используйте хранимую процедуру `AddBill` из л.р.№6). Добавьте в созданный чек с помощью инструкции `INSERT...SELECT` только те товары, которых имеется достаточное количество. С помощью инструкции `UPDATE` обновляем поле `Quantity` таблицы `Product`, вычитая товары, которые были куплены. Самостоятельно выберите уровень изоляции транзакции и прокомментируйте свой выбор.

10. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Лабораторная работа №8 Работа с XML. Часть 2

1. Написать запрос, возвращающий номер чека, общую сумму по чеку, ФИО сотрудника, дату чека; информацию о каждом товаре, название, цену, количество, стоимость. Результат вернуть в виде XML, представленной в предыдущей лабораторной работе.

2. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Лабораторная работа №9 Иерархически организованные данные

1. Добавить в таблицу сотрудники столбец менеджер (внешний ключ к коду сотрудника). Добавить данные в таблицу так, чтобы все существующие сотрудники были подчинены 2-м новым менеджерам (каждый продавец только одному своему менеджеру), а 2 новых менеджера в свою очередь подчинены директору магазина.

2. Добавить в таблицу 2 новых столбца (правый ключ, левый ключ). Написать запрос на вычисление ключей и ввод ключей по методу Джо Селко.

3. Добавить в таблицу 1 новый столбец (материализованный путь). Написать запрос на вычисление и обновление материализованного пути.

4. Написать запрос на выборку всех подчиненных (на всех уровнях) некоторого сотрудника для каждого из трех способов представления иерархии.

5. Подготовьте материал для включения в отчетную презентацию по курсу «Тюнинг баз данных».

Лабораторная работа №10 Секционирование таблиц

1. Напишите 2 запроса соединения таблиц Bill и BillItem в первом случае с фильтрацией Bill.date between '05.02.2011' and '20.02.2011', во втором случае с фильтрацией Bill.date between '05.02.2011' and '20.03.2011'. Изучите планы выполнения запросов, объясните, почему они различны. Сохраните планы выполнения запросов.

2. Проведите секционирование таблицы BillItem по сценарию, описанному в данной лабораторной работе. В качестве ключа секционирования используйте BillItem.Date. Объясните, являются ли секционированные таблицы Bill и BillItem выровненными?

3. Выполните запросы из п.1 и изучите планы их выполнения. Объясните, что изменилось в планах выполнения запросов.

Лабораторная работа №11 Принципы построения статистической и динамической модели

1. Дать определение статической информационной модели.
2. Какие этапы необходимо пройти при построении статической информационной модели.
3. В чем заключается этап «идентификация понятий» при построении статической информационной модели.
4. В чем заключается этап «организация понятий в иерархию классов» при построении статической информационной модели.
5. В чем заключается этап «определение связей» при построении статической информационной модели.
6. В чем заключается этап «описания свойств» при построении статической информационной модели.
7. Дать определение динамической информационной модели.
8. Что представляют собой модели рабочих процессов.
9. Что представляют собой модели потоков данных.
10. Что представляют собой объектные модели.
11. Что описывают жизненные циклы объектов.
12. Что описывают варианты использования.
13. Что описывают диаграммы взаимодействия объектов.

Лабораторная работа №12 Конструкции языка XML

1. Что такое языки разметки. Стилистическая, структурная и семантическая разметки.
2. История развития языков разметки: SGML, HTML, XML.
3. Что такое XML. Как появился XML. XML в качестве данных
4. Структура XML документа. Правильно оформленные и верные документы.
5. Анализатор XML. Событийно управляемые анализаторы. Древовидные анализаторы.
6. Зачем нужно определение типа документа (DTD). Общие принципы написания определений DTD:
7. Недостатки и особенности определений DTD.

8. Моделирование данных с использованием XML. Информационная модель. Отображение на XML. Схемы.
9. Объектная модель документа (DOM).
10. Пространство имен и схемы.
11. Связи и запросы.
12. Трансформация документа XML.
13. Дизайн документов XML.

Лабораторная работа №13 Способы создания Web-сервисов

1. Web- службы. Основные понятия. Назначение.
2. История развития Web-служб. Модель COM/DCOM. .Стандарт CORBA/IIOP. Недостатки перечисленных технологий.
3. Web-службы .Net. Достоинства. Архитектура. Базовые технологии web-служб: SOAP, UDDI, WSDL.

6. Методические указания по подготовке к экзамену

Перед экзаменом студенту необходимо полностью выполнить все задания к лабораторным работам. При наличии задолженностей по текущей аттестации по данной дисциплине студент к экзамену не допускается. Экзамен по дисциплине предусмотрен в устной форме по билетам.

Базовый уровень.

Вопросы для проверки уровня обученности

Знать:

1. Кэширование в базах данных.
2. Физическая организация таблиц. Увеличение и старение физических таблиц.
3. Индексы в B-деревьях. B-деревья и B+-деревья.
4. Индексы с кластеризацией и без кластеризации. Хеш-таблицы.
5. Вычисление селективности. Селективность фильтра и селективность на диапазоне.
6. Типы соединений. Внутренние и внешние соединения.
7. Способы обработки соединений: вложенные циклы, соединение хеширование, соединение с сортировкой слиянием.
8. Блокировки. Параллельное выполнение транзакций.
9. Настройка блокировок.
10. Уровни изоляции транзакций.
11. Журналы транзакций и восстановление данных.
12. Уровни RAID.
13. Оптимизации размещения файлов базы данных.
14. Секционирование таблиц данных.
15. Организация иерархических данных в реляционных базах данных.
16. Управление XML-документами в реляционных базах данных.

17. Методы внутренней сортировки данных.
18. Методы внешней сортировки данных.
19. Методы поиска данных в основной памяти.
20. Методы поиска данных во внешней памяти.

Повышенный уровень:

Знать:

1. Преобразование сущностей и атрибутов
2. Преобразование бинарных связей
3. Проверка модели БД с помощью концепций последовательной нормализации
4. Проверка поддержки целостности данных
5. Физическая организация данных
6. Страничная организация данных в СУБД
7. Индексно-прямые файлы. Индексно-последовательные файлы
8. Организация индексов в виде Б-деревьев
9. Инвертированные списки
10. Реляционное исчисление
11. Целевой список и определяющее выражение
12. Квантор существования
13. Квантор всеобщности
14. Оператор выбора *select*. Формирование запросов к базе данных
15. Агрегатные функции языка
16. Вложенные запросы
17. Многотабличные запросы
18. Оператор ввода данных *INSERT*
19. Оператор удаления данных *DELETE*
20. Операция обновления данных *UPDATE*
21. Операторы создания и удаления индексов
22. Восстановление транзакции
23. Параллелизм БД
24. Блокировка БД

7. Методические материалы, определяющие процедуры оценивания знаний, умений, навыков и (или) опыта деятельности, характеризующих этапы формирования компетенций

Процедура проведения экзамена осуществляется в соответствии с Положением о проведении текущего контроля успеваемости и промежуточной аттестации обучающихся по образовательным программам высшего образования в СКФУ.

В экзаменационный билет включаются 3 вопроса: 2 теоретических вопроса (базовый уровень) и 1 теоретический вопрос (повышенный уровень).

Для подготовки по билету отводиться 40 минут.

Текущая аттестация магистров проводится преподавателями, ведущими лекционные и лабораторные работы по дисциплине в формах собеседование, письменный отчет.

Допуск к лабораторным работам происходит при наличии у магистранта письменного отчета в тетради для лабораторных работ. К занятию магистрант должен подготовить ответы на вопросы к лабораторной работе. На лабораторном занятии магистрант должен выполнить лабораторную работу целиком, включая индивидуальное задание. Защита отчета проходит в форме доклада магистранта по выполненной работе и ответов на вопросы преподавателя в рамках данной лабораторной работы.

Максимальное количество баллов магистрант получает, если он активно участвует в работе, владеет материалом, умеет логично и четко излагать мысли, творчески подходит к решению основных вопросов темы, показывает самостоятельность мышления.

Основанием для снижения оценки являются:

- не качественно выполненная работа на персональном компьютере
- письменный отчет выполнен в небрежной форме и с ошибками
- магистрант не дает исчерпывающий ответ на вопрос преподавателя
- путается или затрудняется в демонстрации работы на компьютере

Отчет может быть отправлен на доработку в следующих случаях:

- выполнен на отдельном листе
- небрежное выполнение и с ошибками
- сокращенное содержание отчета

Критерии оценивания собеседования и защите лабораторных работ, изучения тем дисциплины приведены в Фонде оценочных средств по дисциплине «Тюнинг баз данных».