ЛАБОРАТОРНАЯ РАБОТА №7.
КУРСОРЫ. ОСНОВЫ УПРАВЛЕНИЯ ТРАНЗАКЦИЯМИ В MS SQL SERVER.

Цель: сформировать знания и умения по управлению курсорами и транзакциями в среде Management Studio MS SQL Server, изучить основные операторы для управления транзакциями и курсорами.

Содержание лабораторной работы:

1. Изучить теоретические сведения лабораторной работы.
2. Открыть базу данных.
3. Решить две задачи с помощью курсора:
    a. предусмотреть использование курсора в простом пакете;
    b. включить курсор в хранимую процедуру.
4. Выполнить задание по варианту (стр. 11).

Краткий вспомогательный материал

Операции в реляционной базе данных выполняются над множеством строк. Например, набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE инструкции. Такой полный набор строк, возвращаемых инструкцией, называется результирующим набором. процедурные языки обрабатывают данные построчно. Им нужен механизм, позволяющий обрабатывать одну строку или небольшое их число за один раз. Курсоры (CURrent Set Of Records) являются расширением результирующих наборов, которые предоставляют такой механизм.

После того как курсор сохранен, приложения могут прокручивать данные в прямом или обратном направлении по одной или несколько строк.

Основные типы курсоров в Transact-SQL:

- *статические курсоры* – изолированы от изменений данных. Сервер сохраняет весь итоговый набор курсора при первой выборке в базе *tempdb*. Доступны только для чтения;
- *динамические курсоры* – изменения данных отображаются при перемещении курсора. Используют минимум ресурсов сервера за пределами базы данных;
- *ключевые курсоры* – при открытии курсора сервер запоминает в базе *tempdb* только ключи возвращаемых записей. Новые записи не включаются в курсор, а при обращении к записи, удаленной другим пользователем, происходит ошибка.

Работу с курсором можно разделить на несколько этапов.

- Прежде чем курсор может быть использован, его следует объявить (создать). При этом выборка данных не производится, а определяется используемый оператор *SELECT* и задаются некоторые параметры курсора.
- После объявления курсор может быть открыт для использования. При этом производится выборка данных согласно заданному оператору *SELECT*.
- После того как курсор заполнен данными, из него могут быть извлечены

(прочитаны) необходимые строки.

- По окончании работы курсор должен быть закрыт и, возможно, должны быть освобождены занятые им ресурсы.

После того как курсор объявлен, его можно открывать и закрывать столько раз, сколько необходимо. Если курсор открыт, извлекать из него строки можно произвольное число раз.

Создание курсора. Синтаксис оператора:

```
DECLARE <имя_курсора> CURSOR
                                [LOCAL|GLOBAL]
                                [FORWARD_ONLY|SCROLL]
                                [STATIC|KEYSET|DYNAMIC|AST_FORWARD]
                                [READ_ONLY|SCROLL_LOCKS|OPTIMISTIC]
FOR команда_select
                                [FOR UPDATE [OF <список_столбцов>]]
```

Опции оператора создания курсора:

- *LOCAL* – локальный курсор (видим в пределах пакета, хранимой процедуры или пользовательской функции);
- *GLOBAL* – глобальный курсор, существующий до закрытия соединения (в текущем соединении можно ссылаться на него из любого модуля);
- *FORWARD_ONLY* – последовательный курсор (выборка данных осуществляется только в направлении от первой строки к последней);
- *SCROLL* – прокручиваемый курсор (можно обращаться к нему в любом порядке и направлении);
- *STATIC* – статический курсор (содержимое курсора формируется один раз и далее не обновляется);
- *KEYSET* – ключевой курсор (набор ключей, идентифицирующих строки полного результирующего набора курсора);
- *DYNAMIC* – динамический курсор (каждый раз при обращении к нему, происходит выборка из таблиц);

- *FAST_FORWARD* – курсор «только для чтения» (оптимизирован для быстрого доступа к данным). Опцию нельзя использовать с опциями *FOR- WARD_ONLY* и *OPTIMISTIC*;
- *SCROLL_LOCKS* – обновления и удаления, производимые над строками курсора, будут гарантированно успешны. Не совместима с *FAST_FORWARD*;
- *OPTIMISTIC* – в курсоре запрещено изменение или удаление строк. Для отслеживания изменений сервер использует столбец *timestamp*. Если столбец отсутствует, то сервер генерирует контрольные суммы строк;
- *команда_select* – стандартный запрос на выборку;
- *список_столбцов* – указываются обновляемые столбцы курсора. Если конструкция без столбцов, то можно обновлять все столбцы (если нет *READ_ONLY*).

Открытие курсора. Наполняет курсор данными и готовит его к выборке записей.

```
OPEN { { [ GLOBAL ] имя_курсора } | переменная }
```

Здесь: *GLOBAL* – указывается для глобального курсора; *имя_курсора* – имя курсора; *переменная* – переменная, в которой хранится имя курсора.

Чтение из курсора. Считывание записей курсора осуществляется оператором *FETCH*. Для выборки записей необходимо предварительно открыть курсор.

Синтаксис оператора:

```
FETCH [ направление FROM ]
{{[GLOBAL]имя_курсора}|переменная}[INTO список_переменных]
```

Если при объявлении курсора указано ключевое слово *SCROLL*, параметр *направление* может принимать следующие значения:

- *FIRST* – курсор перемещается к первой записи итогового набора;
- *LAST* – курсор перемещается к последней записи итогового набора;
- *NEXT* – курсор перемещается на одну запись вперед в текущем наборе (действие по умолчанию при выборке);
- *PRIOR* – курсор перемещается на одну запись назад в текущем наборе;
- *ABSOLUTE n /-n* – если $n > 0$, курсор перемещается к n-й записи от начала. Если $n < 0$, курсор перемещается к последней записи и отсчитывает n записей в обратном направлении. При $n = 0$ записи не возвращаются;
- *RELATIVE n /-n* – если $n > 0$, курсор перемещается на n записей вперед от текущей позиции. Если $n < 0$, курсор перемещается на n записей назад. При $n=0$ возвращается текущая запись.

При использовании параметров *ABSOLUTE* и *RELATIVE* курсор в качестве аргументов может получать целые переменные или параметры хранимых процедур.

Закрытие курсора. Оператор *CLOSE* переводит курсор в неактивное состояние и фактически удаляет его итоговый набор. Курсор можно открыть повторно, при этом все его указатели сбрасываются. Синтаксис оператора:

```
CLOSE {{[GLOBAL] имя_курсора}|переменная}
```

Освобождение курсора. При освобождении курсора удаляется вся информация о нем. Допускается освобождение курсора, даже если в нем остались незавершенные транзакции. Синтаксис оператора:

```
DEALLOCATE {{[GLOBAL]имя_курсора}|переменная }
```

После выполнения оператора имя курсора может быть использовано другим оператором *DECLARE CURSOR*. Если эта ссылка на курсор была последней, оператор освобождает всю память, занятую структурой курсора. Чтобы заново открыть курсор, необходимо повторно объявить его.

Контроль достижения конца курсора. Для контроля достижения конца курсора используют функцию *@@FETCH_STATUS*. Функция возвращает:

- 0, если выборка завершилась успешно;
- -1, если выборка завершилась неудачно из-за попытки выборки строки, находящейся за пределами курсора;
- -2, если выборка завершилась неудачно из-за попытки обращения к удаленной или измененной после открытия курсора строке.

Таким образом, работа с курсорами похожа на работу с файлами – сначала открытие курсора, затем чтение и после закрытие.

С учетом изложенного работа с курсорами происходит по следующему алгоритму:

1. При помощи оператора *DECLARE … CURSOR FOR* связывается имя курсора с выполняемым запросом.

2. Оператор *OPEN* выполняет запрос, связанный с курсором, и устанавливает курсор перед первой записью результирующей таблицы.

3. Оператор *FETCH* помещает курсор на первую запись результирующей таблицы и извлекает данные из записи в локальные переменные пакета или хранимой процедуры. Повторный вызов оператора *FETCH* приводит к перемещению курсора к следующей записи, и так до тех пор, пока записи в результирующей таблице не будут исчерпаны. Эту операцию удобно осуществлять в цикле.

4. Оператор *CLOSE* прекращает доступ к результирующей таблице и ликвидирует связь между курсором и результирующей таблицей.

Изменение и удаление записей с помощью курсора. Для выполнения изменений с помощью курсора необходимо выполнить следующий оператор *UPDATE*:

```
UPDATE имя_таблицы SET { имя_столбца =
```

```
{DEFAULT | NULL |выражение}} [ , ... n ]
WHERE CURRENT OF
{{ [ GLOBAL] имя_курсора}|@имя_переменной курсора }
```

За одну операцию могут быть изменены несколько столбцов текущей строки курсора, но все они должны принадлежать одной таблице.

Для удаления данных посредством курсора используется команда *DELETE* в следующем формате:

```
DELETE имя_таблицы WHERE CURRENT OF
{{[GLOBAL] имя_курсора} | @имя_переменной курсора }
```

В результате будет удалена строка, установленная текущей в курсоре. Пример использования курсора для модификации данных. Разработать прокручиваемый курсор для клиентов. Если фамилия клиента начинается на букву *К*, удалить клиента с такой фамилией. В первой записи курсора заменить первую цифру в номере телефона на 1. После выполнения запроса необходимо восстановить первоначальное заполнение таблиц БД.

```
1  USE Book;
2  GO
3  DECLARE @id INT, @name VARCHAR(20), @surname VARCHAR(20), @phone VARCHAR(15);
4  DECLARE user_cursor CURSOR GLOBAL SCROLL
5  KEYSET FOR
6      SELECT user_ID, u_name, u_surname, u_phone
7      FROM Users
8  FOR UPDATE;
9  OPEN user_cursor;
10 FETCH NEXT FROM user_cursor
11     INTO @id, @name, @surname, @phone;
12 WHILE @@FETCH_STATUS = 0
13 BEGIN
14     IF @surname LIKE 'К%'
15     DELETE Users WHERE CURRENT OF user_cursor;
16     FETCH NEXT FROM user_cursor
17         INTO @id, @name, @surname, @phone;
18 END;
19 FETCH ABSOLUTE 1 FROM user_cursor
20     INTO @id, @name, @surname, @phone;
21 UPDATE Users
22 SET u_phone = '1' + RIGHT(@phone, LEN(@phone)-1)
23     WHERE CURRENT OF user_cursor;
24 CLOSE user_cursor;
25 DEALLOCATE user_cursor;
26 SELECT * FROM Users;
```

100 %

Результаты | Сообщения

	user_ID	u_name	u_patronymic	u_surname	u_phone	u_email	u_status
1	1	Александр	Валерьевич	Иванов	18-98-78	ivanov@email.ru	active
2	2	Сергей	Иванович	Посев	90-57-77	losev@email.ru	passive
3	3	Нина	Сергеевна	Грачева	95-66-61	gracheva@email.ru	active
4	5	Анатолий	Юрьевич	Петров	NULL	NULL	lock

Под **транзакцией** понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации), приводящая к одному из двух возможных результатов: либо последовательность выполняется, если все операторы правильные, либо вся транзакция откатывается, если хотя бы один оператор не может быть успешно выполнен. Обработка транзакций гарантирует целостность информации в базе данных. Таким образом, транзакция переводит базу данных из одного целостного состояния в другое.

Поддержание механизма транзакций – показатель уровня развитости СУБД. Корректное поддержание транзакций одновременно является основой обеспечения целостности БД. Транзакции также составляют основу изолированности в многопользовательских системах, где с одной БД параллельно могут работать несколько пользователей или прикладных программ. Одна из основных задач СУБД – обеспечение изолированности, т.е. создание такого режима функционирования, при котором каждому пользователю, казалось бы, что БД доступна только ему. Такую задачу СУБД принято называть параллелизмом транзакций.

Большинство выполняемых действий производится в теле транзакций. По умолчанию каждая команда выполняется как самостоятельная транзакция. При необходимости пользователь может явно указать ее начало и конец, чтобы иметь возможность включить в нее несколько команд.

При выполнении транзакции система управления базами данных должна придерживаться определенных правил обработки набора команд, входящих в транзакцию. В частности, разработано четыре правила, известные как требования ACID, они гарантируют правильность и надежность работы системы.

ACID-свойства транзакций.

Характеристики транзакций описываются в терминах ACID (Atomicity, Consistency, Isolation, Durability – неделимость, согласованность, изолированность, устойчивость).

Транзакция неделима в том смысле, что представляет собой единое целое. Все ее компоненты либо имеют место, либо нет. Не бывает частичной транзакции. Если может быть выполнена лишь часть транзакции, она отклоняется.

Транзакция является согласованной, потому что не нарушает бизнес-логику и отношения между элементами данных. Это свойство очень важно при разработке клиент-серверных систем, поскольку в хранилище данных поступает большое количество транзакций от разных систем и объектов. Если хотя бы одна из них нарушит целостность данных, то все остальные могут выдать неверные результаты.

Транзакция всегда изолирована, поскольку ее результаты самодостаточны. Они не зависят от предыдущих или последующих транзакций – это свойство называется сериализуемостью и означает, что транзакции в последовательности независимы.

Транзакция устойчива. После своего завершения она сохраняется в системе, которую ничто не может вернуть в исходное (до начала транзакции) состояние, т.е. происходит фиксация транзакции, означающая, что ее действие постоянно даже при сбое системы. При этом подразумевается некая форма хранения информации в постоянной памяти как часть транзакции.

Указанные выше правила выполняет сервер. Программист лишь выбирает нужный уровень изоляции, заботится о соблюдении логической целостности данных и бизнес-правил. На него возлагаются обязанности по созданию эффективных и логически верных

алгоритмов обработки данных. Он решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций. Следует по возможности использовать небольшие транзакции, т.е. включающие как можно меньше команд и изменяющие минимум данных. Соблюдение этого требования позволит наиболее эффективным образом обеспечить одновременную работу с данными множества пользователей.

Управление транзакциями.

Под управлением транзакциями понимается способность управлять различными операциями над данными, которые выполняются внутри реляционной СУБД. Прежде всего, имеется в виду выполнение операторов INSERT, UPDATE и DELETE. Например, после создания таблицы (выполнения оператора CREATE TABLE) не нужно фиксировать результат: создание таблицы фиксируется в базе данных автоматически. Точно так же с помощью отмены транзакции не удастся восстановить только что удаленную оператором DROP TABLE таблицу.

После успешного выполнения команд, заключенных в тело одной транзакции, немедленного изменения данных не происходит. Для окончательного завершения транзакции существуют так называемые команды управления транзакциями, с помощью которых можно либо сохранить в базе данных все изменения, произошедшие в ходе ее выполнения, либо полностью их отменить.

Существуют три команды, которые используются для управления транзакциями:

- COMMIT – для сохранения изменений;
- ROLLBACK – для отмены изменений;
- SAVEPOINT – для установки особых точек возврата.

После завершения транзакции вся информация о произведенных изменениях хранится либо в специально выделенной оперативной памяти, либо во временной области отката в самой базе данных до тех пор, пока не будет выполнена одна из команд управления транзакциями. Затем все изменения или фиксируются в базе данных, или отбрасываются, а временная область отката освобождается.

Команда COMMIT предназначена для сохранения в базе данных всех изменений, произошедших в ходе выполнения транзакции. Она сохраняет результаты всех операций, которые имели место после выполнения последней команды COMMIT или ROLLBACK.

Команда ROLLBACK предназначена для отмены транзакций, еще не сохраненных в базе данных. Она отменяет только те транзакции, которые были выполнены с момента выдачи последней команды COMMIT или ROLLBACK.

Команда SAVEPOINT (точка сохранения) предназначена для установки в транзакции особых точек, куда в дальнейшем может быть произведен откат (при этом отката всей транзакции не происходит). Команда имеет следующий вид:

```
SAVEPOINT имя_точки_сохранения
```

Она служит исключительно для создания точек сохранения среди операторов, предназначенных для изменения данных. Имя точки сохранения в связанной с ней группе транзакций должно быть уникальным.

Для отмены действия группы транзакций, ограниченных точками сохранения, используется команда ROLLBACK со следующим синтаксисом:

```
ROLLBACK TO имя_точки_сохранения
```

Поскольку с помощью команды SAVEPOINT крупное число транзакций может быть разбито на меньшие и поэтому более управляемые группы, ее применение является одним из способов управления транзакциями.

Управление транзакциями в среде MS SQL Server.

Определение транзакций.

SQL Server предлагает множество средств управления поведением транзакций. Пользователи в основном должны указывать только начало и конец транзакции, используя команды SQL или API (прикладного интерфейса программирования). Транзакция определяется на уровне соединения с базой данных и при закрытии соединения автоматически закрывается. Если пользователь попытается установить соединение снова и продолжить выполнение транзакции, то это ему не удастся. Когда транзакция начинается, все команды, выполненные в соединении, считаются телом одной транзакции, пока не будет достигнут ее конец.

SQL Server поддерживает три вида определения транзакций:

- явное;
- автоматическое;
- подразумеваемое.

По умолчанию SQL Server работает в режиме автоматического начала транзакций, когда каждая команда рассматривается как отдельная транзакция. Если команда выполнена успешно, то ее изменения фиксируются. Если при выполнении команды произошла ошибка, то сделанные изменения отменяются, и система возвращается в первоначальное состояние.

Когда пользователю понадобится создать транзакцию, включающую несколько команд, он должен явно указать транзакцию.

Сервер работает только в одном из двух режимов определения транзакций: автоматическом или подразумевающемся. Он не может находиться в режиме исключительно явного определения транзакций. Этот режим работает поверх двух других. Для установки режима автоматического определения транзакций используется команда:

```
SET IMPLICIT_TRANSACTIONS OFF
```

При работе в режиме неявного (подразумевающегося) начала транзакций SQL Server автоматически начинает новую транзакцию, как только завершена предыдущая. Установка режима подразумевающегося определения транзакций выполняется посредством другой команды:

```
SET IMPLICIT_TRANSACTIONS  ON
```

Явные транзакции требуют, чтобы пользователь указал начало и конец транзакции, используя следующие команды:

- начало транзакции: в журнале транзакций фиксируются первоначальные значения изменяемых данных и момент начала транзакции;

```
BEGIN TRAN[SACTION]
     [имя_транзакции |
     @имя_переменной_транзакции
     [WITH MARK ['описание_транзакции']]]
```

- конец транзакции: если в теле транзакции не было ошибок, то эта команда предписывает серверу зафиксировать все изменения, сделанные в транзакции, после чего в журнале транзакций помечается, что изменения зафиксированы и транзакция завершена;

```
COMMIT [TRAN[SACTION]
     [имя_транзакции |
     @имя_переменной_транзакции]]
```

- создание внутри транзакции точки сохранения: СУБД сохраняет состояние БД в текущей точке и присваивает сохраненному состоянию имя точки сохранения;

```
SAVE TRAN[SACTION]
  {имя_точки_сохранеия |
  @имя_переменной_точки_сохранения}
```

- прерывание транзакции ; когда сервер встречает эту команду, происходит откат транзакции, восстанавливается первоначальное состояние системы и в журнале транзакций отмечается, что транзакция была отменена. Приведенная ниже команда отменяет все изменения, сделанные в БД после оператора BEGIN TRANSACTION или отменяет изменения, сделанные в БД после точки сохранения, возвращая транзакцию к месту, где был выполнен оператор SAVE TRANSACTION.

```
ROLLBACK [TRAN[SACTION]
  [имя_транзакции |
  @имя_переменной_транзакции
  | имя_точки_сохранения
 |@имя_переменной_точки_сохранения]]
```

Функция @@TRANCOUNT возвращает количество активных транзакций.
Функция @@NESTLEVEL возвращает уровень вложенности транзакций.

Пример задания 1. Использование точек сохранения.

```
BEGIN TRAN
SAVE TRANSACTION  point1
```

В точке point1 сохраняется первоначальное состояние таблицы Stock.

Пример задания 2. Использование точек сохранения.

```
DELETE FROM Stock WHERE idStock=2
     SAVE TRANSACTION  point2
```

В точке point2 сохраняется состояние таблицы Stock без записи с кодом 2.

Пример задания 3. Использование точек сохранения.

```
DELETE FROM Stock WHERE idStock=3
     SAVE TRANSACTION  point3
```

В точке point3 сохраняется состояние таблицы Stock без записей с кодом 2 и 3.

Пример задания 4. Использование точек сохранения.

```
DELETE FROM Stock WHERE idStock<>1
     ROLLBACK TRANSACTION point3
```

Происходит возврат в состояние таблицы без записей с кодами 2 и 3, отменяется последнее удаление.

```
SELECT * FROM Товар
```

Оператор SELECT покажет таблицу Stock без записей с кодами 2 и 3.

Пример задания 5. Использование точек сохранения.

```
ROLLBACK TRANSACTION point1
```

Происходит возврат в первоначальное состояние таблицы.

```
SELECT * FROM Stock
COMMIT
```

Первоначальное состояние сохраняется.

Вложенные транзакции

Вложенными называются транзакции, выполнение которых инициируется из тела уже активной транзакции

Для создания вложенной транзакции пользователю не нужны какие-либо дополнительные команды. Он просто начинает новую транзакцию, не закрыв предыдущую. Завершение транзакции верхнего уровня откладывается до завершения вложенных транзакций. Если транзакция самого нижнего (вложенного) уровня завершена неудачно и отменена, то все транзакции верхнего уровня, включая транзакцию первого уровня, будут отменены. Кроме того, если несколько транзакций нижнего уровня были завершены успешно (но не зафиксированы), однако на среднем уровне (не самая верхняя транзакция) неудачно завершилась другая транзакция, то в соответствии с требованиями ACID произойдет откат всех транзакций всех уровней, включая успешно завершенные. Только когда все транзакции на всех уровнях завершены успешно, происходит фиксация всех сделанных изменений в результате успешного завершения транзакции верхнего уровня.

Каждая команда COMMIT TRANSACTION работает только с последней начатой транзакцией. При завершении вложенной транзакции команда COMMIT применяется к наиболее "глубокой" вложенной транзакции. Даже если в команде COMMIT TRANSACTION указано имя транзакции более высокого уровня, будет завершена транзакция, начатая последней.

Если команда ROLLBACK TRANSACTION используется на любом уровне вложенности без указания имени транзакции, то откатываются все вложенные транзакции, включая транзакцию самого высокого (верхнего) уровня. В команде ROLLBACK TRANSACTION разрешается указывать только имя самой верхней транзакции. Имена любых вложенных транзакций игнорируются, и попытка их указания приведет к ошибке. Таким образом, при откате транзакции любого уровня вложенности всегда происходит откат всех транзакций. Если же требуется откатить лишь часть транзакций, можно использовать команду SAVE TRANSACTION, с помощью которой создается точка сохранения.

Задания лабораторной работы

Вариант	Задание
1-5, 16-20	1. Сохранить первоначальное состояние таблицы Customers 2. Сохранить состояние таблицы Customers без заказчиков из города Москва 3. Сохранить состояние таблицы Customers без заказчиков из городов Москва и Санкт-Петербург. 4. Удалить все записи. 5. Произвести возврат в состояние таблицы Customers без заказчиков из городов Москва и Санкт-Петербург, отменить последнее удаление. Вывести содержимое таблицы. 6. Произвести возврат в исходное состояние таблицы Customers, сохранить его.
6-10, 21-25	1. Сохранить первоначальное состояние таблицы Products 2. Сохранить состояние таблицы Products без товаров, которых меньше 5 3. Сохранить состояние таблицы Products без товаров, которых меньше 5 и больше 10000. 4. Удалить все записи. 5. Произвести возврат в состояние таблицы Products без товаров, которых меньше 5 и больше 10000, отменить последнее удаление. Вывести содержимое таблицы. 6. Произвести возврат в исходное состояние таблицы Products, сохранить его.
11-15, 26-30	1. Сохранить первоначальное состояние таблицы Orders 2. Сохранить состояние таблицы Orders без заказов, находящихся в статусе C 3. Сохранить состояние таблицы Orders без заказов, находящихся в статусе C и номера которых меньше 10. 4. Удалить все записи. 5. Произвести возврат в состояние таблицы Orders без, находящихся в статусе C и номера которых меньше 10, отменить последнее удаление. Вывести содержимое таблицы. 6. Произвести возврат в исходное состояние таблицы Orders, сохранить его.

Контрольные вопросы.

1. Что такое курсор?
2. Перечислить основные типы курсора.
3. Этапы работы с курсором.
4. Синтаксис оператора создания курсора.
5. Синтаксис оператора открытия курсора.
6. Синтаксис оператора чтения курсора.
7. Синтаксис оператора закрытия курсора.
8. Синтаксис оператора освобождения курсора.
9. Сформулируйте алгоритм работы курсора.
10. Что такое транзакция?
11. Перечислите и опишите ACID-свойства транзакций.
12. Что понимается под управлением транзакциями? Назовите команд управления транзакциями. Для чего они предназначены?
13. Какие виды определения транзакций поддерживает SQL Server? Опишите их.
14. Какие используются команды для установки режима автоматического/неявного определения транзакция?
15. Назовите и опишите команды, которые используются для описания явного вида транзакций.
16. Что такое вложенные транзакции? Требуются ли специальные команды для их создания?
17. Опишите механизм создания вложенных транзакций.
18. Что будет в результате выполнения данного кода? Берется произвольная таблица

```
BEGIN TRAN
INSERT TABLE1 (Val1, Val2)
VALUES ('v',40)
   BEGIN TRAN
   INSERT TABLE1 (Val1, Val2)
   VALUES ('n',50)
   BEGIN TRAN
      INSERT TABLE1 (Val1, Val2)
      VALUES ('m',60)
ROLLBACK TRAN
```