Задание 1



/\* Задание 1 \*/

select \* from Студент

use XAMEL24

set XACT\_ABORT on

begin transaction

insert into Студент

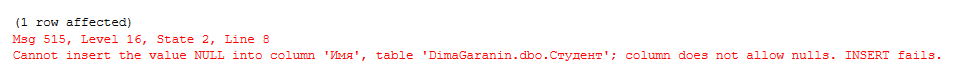
values ('342342', 'test', 'test', 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

insert into Студент

values ('342342', 'test', NULL, 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

commit transaction

--



select \* from Студент

set XACT\_ABORT off

begin transaction

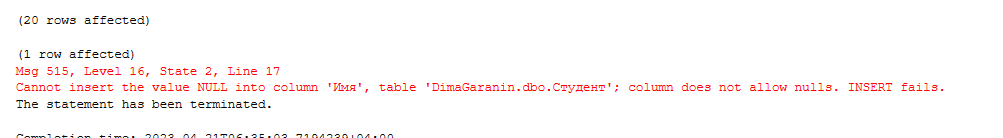
insert into Студент

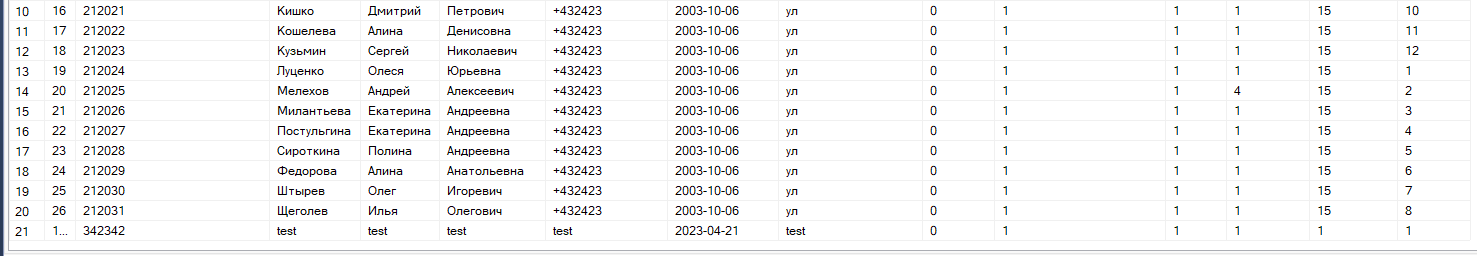
values ('342342', 'test', 'test', 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

insert into Студент

values ('342342', 'test', NULL, 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

commit transaction





Задание 2

set IMPLICIT\_TRANSACTIONS off

--явные

print N'Открываем явные транзакции'

begin transaction

print @@trancount

begin transaction

print @@trancount

print N'Закрываем явные транзакции'

commit transaction

print @@trancount

commit transaction

print @@trancount

--неявные

print cast (@@trancount as varchar) + N'Неявных транзакций нет'

--автоматические

select \* from Студент

print cast (@@trancount as varchar) + N'Автоматическое применение транзакции'

--вкл режим неявных

set IMPLICIT\_TRANSACTIONS on

--явные

print N'Открываем явные транзакции'

begin transaction

print @@trancount

begin transaction

print @@trancount

select \* from Студент

print @@trancount

print N'Закрываем явные транзакции'

commit transaction

print @@trancount

commit transaction

--неявные

print cast (@@trancount as varchar) + N' - Неявная транзакция'

print N'Закрываем неявные транзакции'

rollback transaction

print @@trancount

--автоматические

insert into Студент

values ('342342', 'test', 'test', 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

print cast (@@trancount as varchar) + N'Транзакция не применилась автоматически'

select \* from Студент

rollback transaction

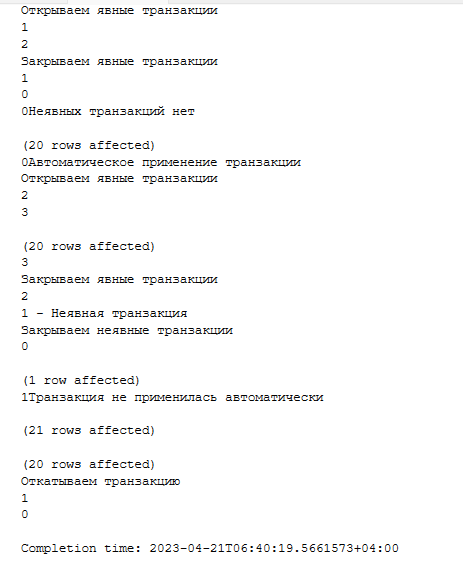
select \* from Студент

print N'Откатываем транзакцию'

print @@trancount

commit transaction

print @@trancount



Задание 3

Uncommitted

-- uncommitted

select \* from Студент

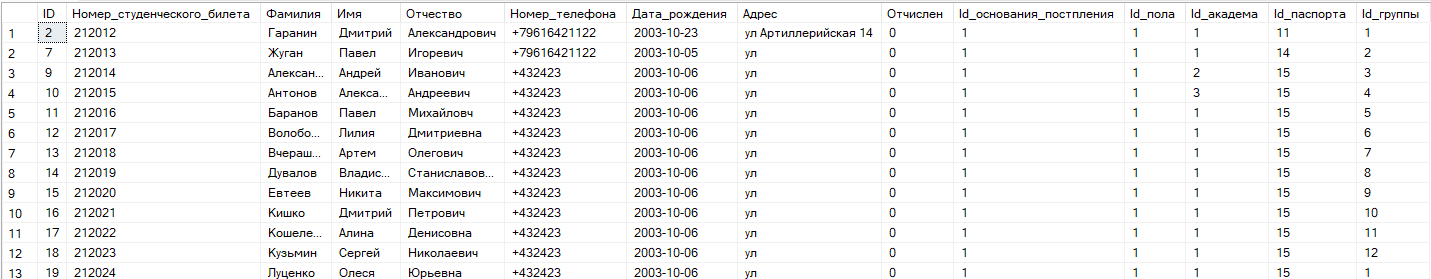
set transaction isolation level read uncommitted

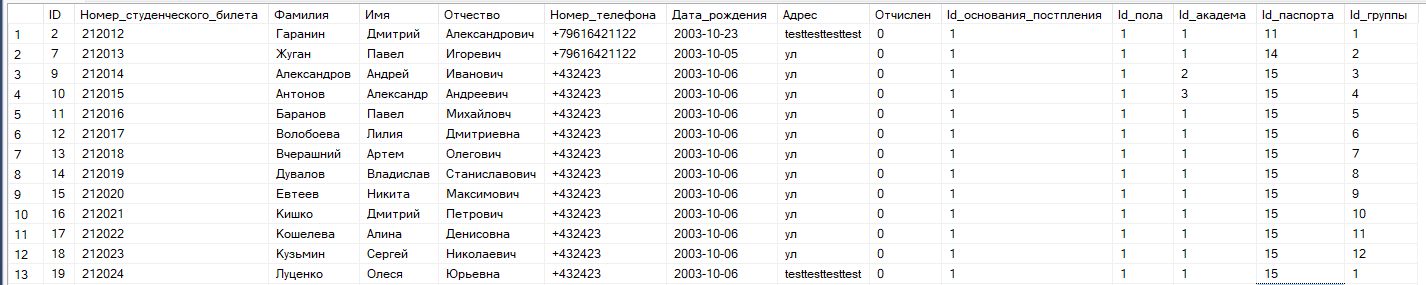
begin transaction

update Студент set Адрес = 'testtesttesttest' where Студент.Id\_группы = 1

select \* from Студент

rollback transaction





Committed(Решение “грязного чтения”)

1.

select \* from Студент

set transaction isolation level read committed

begin transaction

update Студент set Адрес = 'testtesttesttest' where Студент.Id\_группы = 1

rollback transaction

2.

set transaction isolation level read committed

select \* from Студент

Остается проблема неправильного повторого чтения строк

1.

begin transaction

select \* from Студенты

select \* from Студенты

commit transaction

2.

update Студент set Адрес = 'testtesttesttest' where Студент.Id\_группы = 1

Repeatable read(Решение проблемы “грязного чтения” и неправильного повторного чтения строк)

1.

set transaction isolation level repeatable read

begin transaction

select \* from Студент

select \* from Студент

commit transaction

2.

set transaction isolation level repeatable read

update Студент set Адрес = 'testtesttesttest' where Студент.Id\_группы = 1

Остается пробелма фантомных строк

1.

set transaction isolation level repeatable read

begin transaction

select \* from Студент

select \* from Студент

commit transaction

2.

insert into Студент

values ('342342', 'test', 'test', 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

Serializable(Решает все возможные проблемы “ грязного чтения”, неправильного повторного чтения строк, фантомных строк)

1.

set transaction isolation level serializable

begin transaction

select \* from Студент

select \* from Студент

commit transaction

select \* from Студент

2.

set transaction isolation level serializable

insert into Студент

values ('342342', 'test', 'test', 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

Snapshot

1.

set ALLOW\_SNAPSHOT\_ISOLATION on

set transaction isolation level snapshot

begin transaction

select \* from Студент

select \* from Студент

commit transaction

select \* from Студент

2.

set ALLOW\_SNAPSHOT\_ISOLATION on

set transaction isolation level snapshot

insert into Студент

values ('342342', 'test', 'test', 'test', 'test', CURRENT\_TIMESTAMP, 'test', 0, 1, 1, 1, 1, 1)

Задание 4

Подключение номер 1

select \* from Студент

set transaction isolation level read committed

begin transaction

update Студент

set Адрес = 'test1'

where ID = 2

update Студент

set Адрес = 'test2'

where Id = 14

commit transaction

Подключение номер 2  
set transaction isolation level read committed

begin transaction

update Студент

set Адрес = 'test1'

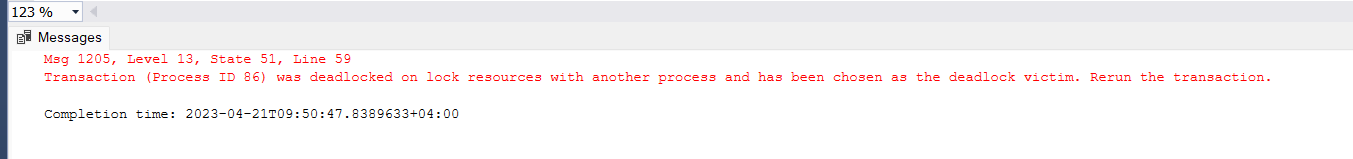
where Id = 14

update Студент

set Адрес = 'test2'

where Id = 2

commit transaction



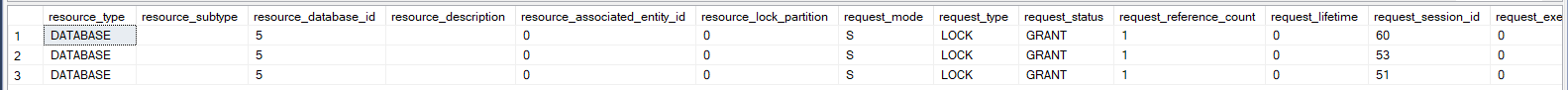
Задание 5

begin transaction

select \* from Студент

select \* from sys.dm\_tran\_locks

commit transaction



Ответы на вопрсы:

**№1. Сущность и предназначение транзакции.**

**Транзакция** – это команда или блок команд (инструкций), которые успешно завершаются как единое целое, при этом в базе данных все внесенные изменения фиксируются на постоянной основе, или отменяются, т.е. все изменения, внесенные любой командой, входящей в транзакцию, будут отменены. Другими словами, если одна команда или инструкция внутри транзакции завершилась с ошибкой, то все, что было отработано перед ней, также отменяется, даже если предыдущие команды завершились успешно.

**№2. Базовые свойства транзакции.**

* **Атомарность** – все команды в транзакции либо полностью выполняются, и соответственно, фиксируются все изменения данных, либо ничего не выполняется и ничего не фиксируется;
* **Согласованность** – данные, в случае успешного выполнения транзакции, должны соблюдать все установленные правила в части различных ограничений, первичных и внешних ключей, определенных в базе данных;
* **Изоляция** – механизм предоставления доступа к данным. Транзакция изолирует данные, с которыми она работает, для того чтобы другие транзакции получали только согласованные данные;
* **Надежность** – все внесенные изменения фиксируются в журнале транзакций и данные считаются надежными, если транзакция была успешно завершена. В случае сбоя SQL Server сверяет данные, записанные в базе данных, с журналом транзакций, если есть успешно завершенные транзакции, которые не закончили процесс записи всех изменений в базу данных, они будут выполнены повторно. Все действия, выполненные не подтвержденными транзакциями, отменяются.

**№3. Типы транзакций. Режимы подтверждения транзакций сервера.**

1. Неявная транзакция - задает любую отдельную инструкцию INSERT, UPDATE или DELETE как единицу транзакции.
2. Явная транзакция - обычно это группа инструкций языка, начало и конец которой обозначаются такими инструкциями, как BEGIN TRANSACTION, COMMIT и ROLLBACK.

Режимы подтверждения транзакций сервера **set implicit\_transactions**:

* По умолчанию SQL Server работает в режиме автоматического начала транзакций, где 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.
* Если установлено значение ON, система находится в неявном режиме транзакции. Это означает, что если @@TRANCOUNT = 0, любая из следующих инструкций начинает новую транзакцию.
* Если задано значение OFF, каждая из предыдущих инструкций T-SQL ограничена невидимыми инструкциями BEGIN TRANSACTION и COMMIT TRANSACTION. При значении OFF транзакция выполняется в режиме автофиксации. Если ваш код T-SQL выдает видимую инструкцию BEGIN TRANSACTION, транзакция выполняется в явном режиме.

**№4. Уровни изолированности транзакций. Read uncommitted.**

Самый низкий уровень, при котором SQL сервер разрешает так называемое «грязное чтение». Грязным чтением называют считывание неподтвержденных данных, иными словами, если транзакция, которая изменяет данные, не завершена, другая транзакция может получить уже измененные данные, хотя они еще не зафиксированы и могут отмениться.

**№5. Уровни изолированности транзакций. Read committed.**

Этот уровень уже запрещает грязное чтение, в данном случае все процессы, запросившие данные, которые изменяются в тот же момент в другой транзакции, будут ждать завершения этой транзакции и подтверждения фиксации данных. Данный уровень по умолчанию используется SQL сервером.

**№6. Уровни изолированности транзакций. Repeatable read.**

На данном уровне изоляции запрещается изменение данных между двумя операциями чтения в одной транзакции. Здесь происходит запрет на так называемое «неповторяющееся чтение» или «несогласованный анализ». Другими словами, если в одной транзакции есть несколько операций чтения, данные будут блокированы и их нельзя будет изменить в другой транзакции. Таким образом, Вы избежите ситуации, когда вначале транзакции Вы запросили данные, провели их анализ (некое вычисление), в конце транзакции запросили те же самые данные, а они уже отличаются от первоначальных, так как они были изменены другой транзакцией.

Также уровень REPEATABLE READ, как и остальные, запрещает «Потерянное обновление» – это когда две транзакции сначала считывают одни и те же данные, а затем изменяют их на основе неких вычислений, в результате обе транзакции выполнятся, но данные будут те, которая зафиксировала последняя операция обновления. Это происходит потому, что данные в операциях чтения в начале этих транзакций не были заблокированы.

**№7. Уровни изолированности транзакций. Snapshot.**

SNAPSHOT – уровень хранит строки, подтверждённые на момент начала транзакции, соответственно, именно эти строки будут считаны в случае обращения к ним из другой транзакции. Данный уровень исключает повторяющееся и фантомное чтение примерно так же, как уровень SERIALIZABLE.

Иными словами, SQL Server делает снимок и хранит последние версии подтвержденных строк. В данном случае, клиенту не нужно ждать снятия блокировок, пока одна транзакция изменит данные, он сразу получает последнюю версию подтвержденных строк.

**№8. Уровни изолированности транзакций. Serializable.**

Данный уровень исключает чтение «фантомных» записей. Фантомные записи – это те записи, которые появились между началом и завершением транзакции. Иными словами, в начале транзакции Вы запросили определенные данные, в конце транзакции Вы запрашиваете их снова с тем же фильтром, но там уже есть и новые данные, которые добавлены другой транзакцией. Более низкие уровни изоляции не блокировали строки, которых еще нет в таблице, данный уровень блокирует все строки, соответствующие фильтру запроса, с которыми будет работать транзакция, как существующие, так и те, что могут быть добавлены.

**№9. Блокировки. Вид блокировок. Концепция менеджера блокировок.**

Одновременный конкурентный доступ может вызывать разные отрицательные эффекты, например чтение несуществующих данных или потерю модифицированных данных.

Для обеспечения согласованности данных в случае одновременного обращения к данным несколькими пользователями компонент Database Engine, подобно всем СУБД, применяет блокировки. Каждая прикладная программа блокирует требуемые ей данные, что гарантирует, что никакая другая программа не сможет модифицировать эти данные. Когда другая прикладная программа пытается получить доступ к заблокированным данным для их модификации, то система или завершает эту попытку ошибкой, или заставляет программу ожидать снятия блокировки.

* Database (DB): Это блокировка сессии - т.е. она не относится ни к какой транзакции, а только к пользователю, подключенному к определенной базе данных. Это нужно, чтобы предотвратить удаление базы данных, когда к ней подключены один или более пользователей.
* Page (PAG): Когда SQL Server требуется заблокировать одновременно множество строк, а свободные слоты блокировок заканчиваются, то он может использовать страничные блокировки.
* Эксклюзив (X); Для монопольной блокировки требуется, чтобы никакая другая активная транзакция (включая транзакции автоматической фиксации / отдельные операторы SQL) не могла касаться заблокированного объекта, пока удерживается монопольная блокировка. Эксклюзивные блокировки обычно получают путем вставки, обновления и удаления.
* Общий (S); Общие блокировки, как следует из названия, могут быть разделены между транзакциями, которые желают только доступа для чтения к рассматриваемым данным. Общие блокировки не позволяют использовать эксклюзивные блокировки, пока они активны, поэтому эксклюзивные блокировки должны ждать завершения общих блокировок, прежде чем они будут получены.
* Обновление (U); Блокировки обновления представляют собой комбинацию эксклюзивной и общей блокировки. Они обычно используются, когда запись или набор записей должны быть найдены и обновлены. Они допускают общие блокировки для диапазона сканируемых данных, но получают эксклюзивную блокировку для изменяемых данных.

Пользователю чаще всего не нужно предпринимать никаких действий по  управлению блокировками. Всю работу по установке, снятию и разрешению конфликтов выполняет специальный компонент сервера, называемый менеджером блокировок.  MS SQL Server поддерживает различные  уровни блокирования объектов (или детализацию блокировок), начиная с отдельной строки таблицы и заканчивая базой данных в целом. Менеджер блокировок автоматически оценивает, какое количество данных необходимо блокировать, и устанавливает соответствующий тип  блокировки.

**№10. Взаимоблокировка. Примеры.**

Взаимоблокировка (deadlock) - это особая проблема одновременного конкурентного доступа, в которой две транзакции блокируют друг друга. В частности, первая транзакция блокирует объект базы данных, доступ к которому хочет получить другая транзакция, и наоборот. В общем, взаимоблокировка может быть вызвана несколькими транзакциями, которые создают цикл зависимостей.

**№11. Понятие связанного сервера и распределённого запроса.**

**Связанные серверы** – это своего рода подключение к другому источнику данных, которым может выступать как сервер баз данных, так и простой файл xls или dbf. Используя это подключение можно посылать запросы к данному источнику данных.

**Распределенные запросы** используются для доступа к данным из нескольких разнородных источников данных. Эти источники данных могут храниться на одном или различных компьютерах. Microsoft SQL Server поддерживает распределенные запросы с использованием OLE DB.

Пользователи SQL Server могут применять распределенные запросы для доступа к следующим данным:

* Распределенные данные, хранящиеся в нескольких экземплярах SQL Server.
* Разнородные данные, хранящиеся в различных реляционных и нереляционных источниках данных, доступ к которым осуществляется с использованием поставщика OLE DB.

Поставщики OLE DB представляют данные в табличных объектах, именуемых «наборами строк». SQL Server позволяет ссылаться в инструкциях Transact-SQL на наборы строк из поставщиков OLE DB так, как если бы эти наборы строк являлись таблицами SQL Server.

**№12. Структура файлов журналов. Понятие логического и физического журналов, LSN, minLSN, контрольной точки.**

Журнал транзакций выполнен как отдельный файл или набор файлов в базе данных. Кэш журнала управляется отдельно от буферного кэша для страниц данных, что приводит к простому, быстрому и устойчивому коду в пределах компонента Компонент SQL Server Database Engine.

Формат записей журнала и страниц не обязан следовать формату страниц данных.

Журнал транзакций может располагаться в нескольких файлах. Вы можете задать для этих файлов автоматическое расширение, установив для журнала значение FILEGROWTH. Это снижает вероятность исчерпания пространства журнала транзакций, в то же самое время уменьшая административные издержки.

**Логическая архитектура журнала транзакций**

**Логически журнал транзакций** SQL Server работает так, как если бы он являлся последовательностью записей в журнале. Каждая запись журнала идентифицируется регистрационным номером транзакции (номер LSN). Каждая новая запись добавляется в логический конец журнала с номером LSN, который больше номера LSN предыдущей записи. Записи журнала сохраняются в серийной последовательности по мере их создания, таким образом если LSN2 больше, чем LSN1, то изменение, описанное записью журнала, на которую ссылается LSN2, произошло после изменения, описанного записью журнала LSN1. Каждая запись журнала содержит идентификатор транзакции, к которой она относится. Все записи журнала, связанные с определенной транзакцией, с помощью обратных указателей связаны в цепочку, которая предназначена для ускорения отката транзакции.

MinLSN — это регистрационный номер транзакции самой старой записи в журнале, которая необходима для успешного отката на уровне всей базы данных.

**Физическая архитектура журнала транзакций**

Журнал транзакций в базе данных сопоставляет один или несколько **физических** файлов. По сути, файл журнала представляет собой строку записей журнала. Физически последовательность записей журнала эффективно хранится в наборе физических файлов, которые образуют журнал транзакций. Для каждой базы данных должен существовать хотя бы один файл журнала.

**Усечение журнала, контрольная точка**

Усечение журнала необходимо для предотвращения переполнения журнала. При усечении журнала удаляются неактивные виртуальные файлы журнала из логического журнала транзакций базы данных SQL Server , что приводит к освобождению пространства в логическом журнале для повторного использования физическим журналом транзакций. Если усечение журнала транзакций не выполняется, со временем он заполняет все доступное место на диске, отведенное для файлов физического журнала. Однако перед усечением журнала должна быть выполнена операция создания контрольной точки. Новая контрольная точка записывает текущие страницы, измененные в памяти (известные как измененные незафиксированные страницы), вместе со сведениями журнала транзакций из памяти на диск. При создании контрольной точки неактивная часть журнала транзакций помечается как неиспользуемая, после чего ее можно освободить путем усечения журнала.

**№13. Распределённые транзакции. Двухфазная фиксация. Компонент DTC.**

**Распределенные транзакции** охватывают два или более серверов - менеджеров ресурсов. Управление транзакцией должно координироваться диспетчером транзакций. Microsoft® SQL Server™ может работать менеджером ресурсов в распределенных транзакциях, координируемых диспетчерами транзакций типа **Microsoft Distributed Transaction Coordinator (MS DTC)**, или другими, поддерживающими спецификацию X/Open XA for Distributed Transaction Processing.

Для приложения, в сущности, нет разницы между локальной и распределенной транзакциями. В конце транзакции приложение требует либо ее завершения, либо отката. Но завершение распределенной транзакции должно осуществляться диспетчером транзакций так, чтобы минимизировать риск возникновения ситуации, в которой одни менеджеры ресурсов откатят транзакцию из-за сетевого сбоя, а другие завершат ее успешно. Это достигается применением **двухфазного** (фаза подготовки и фаза завершения) завершения транзакций (two-phase commit - 2PC).

**Фаза подготовки**

При получении требования завершения транзакции диспетчер транзакций посылает команду подготовки всем менеджерам ресурсов, вовлеченным в транзакцию. Каждый менеджер ресурсов делает все, что требуется для сохранения результатов транзакции и сбрасывает на диск все буферы, содержащие регистрационную информацию о транзакции. По мере готовности менеджеры ресурсов рапортуют диспетчеру транзакций об удаче или провале подготовительной фазы.

**Фаза завершения**

Если все менеджеры ресурсов отрапортовали об удачном окончании подготовительной фазы, диспетчер транзакций посылает им команду завершения транзакции. Менеджеры ресурсов завершают транзакцию. Если все они докладывают об удаче, диспетчер транзакций посылает сообщение об успешном завершении транзакции приложению. Если же какой-то менеджер ресурсов доложил о провале подготовительной фазы, диспетчер транзакций посылает всем команду отката и сообщает приложению о неудачном завершении транзакции.