# Менеджер транзакций

# Компоненты менеджера транзакций

- Менеджер блокировок для конкурентного доступа.
- Менеджер восстановления.
- Буферный пул для хранения промежуточных состояний БД.
- Методы доступа для организации данных транзакций.

# Менеджер блокировок

Две стратегии:

- Блокировка
- Восстановление

# Механизмы управления согласованием в многопользовательской среде

- Multi-Granular Locking scheme (сокращённо **MGL**, также известна как **LSCC**) - схема гранулированных синхронизационных захватов;
- Multi-Versioning Concurrency Control (сокращённо **MVCC**) - многоверсионное управление параллелизмом;
- Optimistic Concurrency Control (**OCC**) - управление оптимистичным параллелизмом.

# LSCC - схема гранулированных синхронизационных захватов

- Одна строка – одна запись => эксклюзивная блокировка (W-lock) для защиты строк от таких операций записи как INSERT, UPDATE или DELETE.
- Одна строка – множественное чтение => совместно используемыte блокировки (R-lock), которые могут быть предоставлены для операции чтения нескольким одновременным клиентам.

| Когда транзакции необходима следующая блокировка строки | Когда другая транзакция уже имеет следующую блокировку в отношении той же строки | | Когда никакая другая транзакция не имеет никаких блокировок в отношении той же строки |
|---|---|---|---|
| | R-lock | W-lock | |
| R-lock | блокировка предоставлена | ожидание разблокировки | блокировка предоставлена |
| W-lock | ожидание разблокировки | ожидание разблокировки | блокировка предоставлена |

# READ UNCOMMITTED

- Уровень изолированности READ UNCOMMITTED не требует R-блокировки для защиты от чтения, но в случае других уровней изолированности, для чтения необходима R-блокировка, которая будет предоставлена, если никакая другая транзакция не имеет W-блокировки на строке (строках).
- Если транзакция READ UNCOMMITTED меняет данные, то ставится W-блокировка

# Снятие блокировок

- В случае уровня изолированности Read Committed, R-блокировка строки будет снята сразу после считывания строки
- При Repeatable Read и SERIALIZABLE R-блокировки будут сохранены до конца транзакции.
- Все блокировки транзакции будут сняты в конце транзакции независимо от того, как была завершена транзакция.

# Блокировка таблиц

- В некоторых продуктах СУБД диалект SQL включает явные команды LOCK TABLE, но снимаются эти блокировки в конце транзакции всегда неявно, а в случае уровня изолированности READ COMMITTED R-блокировка снимается раньше.
- Неявные команды UNLOCK TABLE обычно доступны в диалектах SQL, за исключением, например, MySQL/InnoDB.

# Гранулярность блокировок

| Ресурс | Описание |
|---|---|
| RID | Идентификатор строки, используемый для блокировки одной строки в куче. |
| KEY | Блокировка строки в индексе, используемая для защиты диапазонов значений ключа в сериализуемых транзакциях. |
| PAGE | Страница в базе данных, например страница данных или индекса. |
| EXTENT | Упорядоченная группа из восьми страниц, например страниц данных или индекса. |
| HoBT | Куча или сбалансированное дерево. Блокировка, защищающая сбалансированное дерево (индекс) или кучу страниц данных в таблице, не имеющей кластеризованного индекса. |
| TABLE | Таблица полностью, включая все данные и индексы. |
| DATABASE | База данных, полностью. |

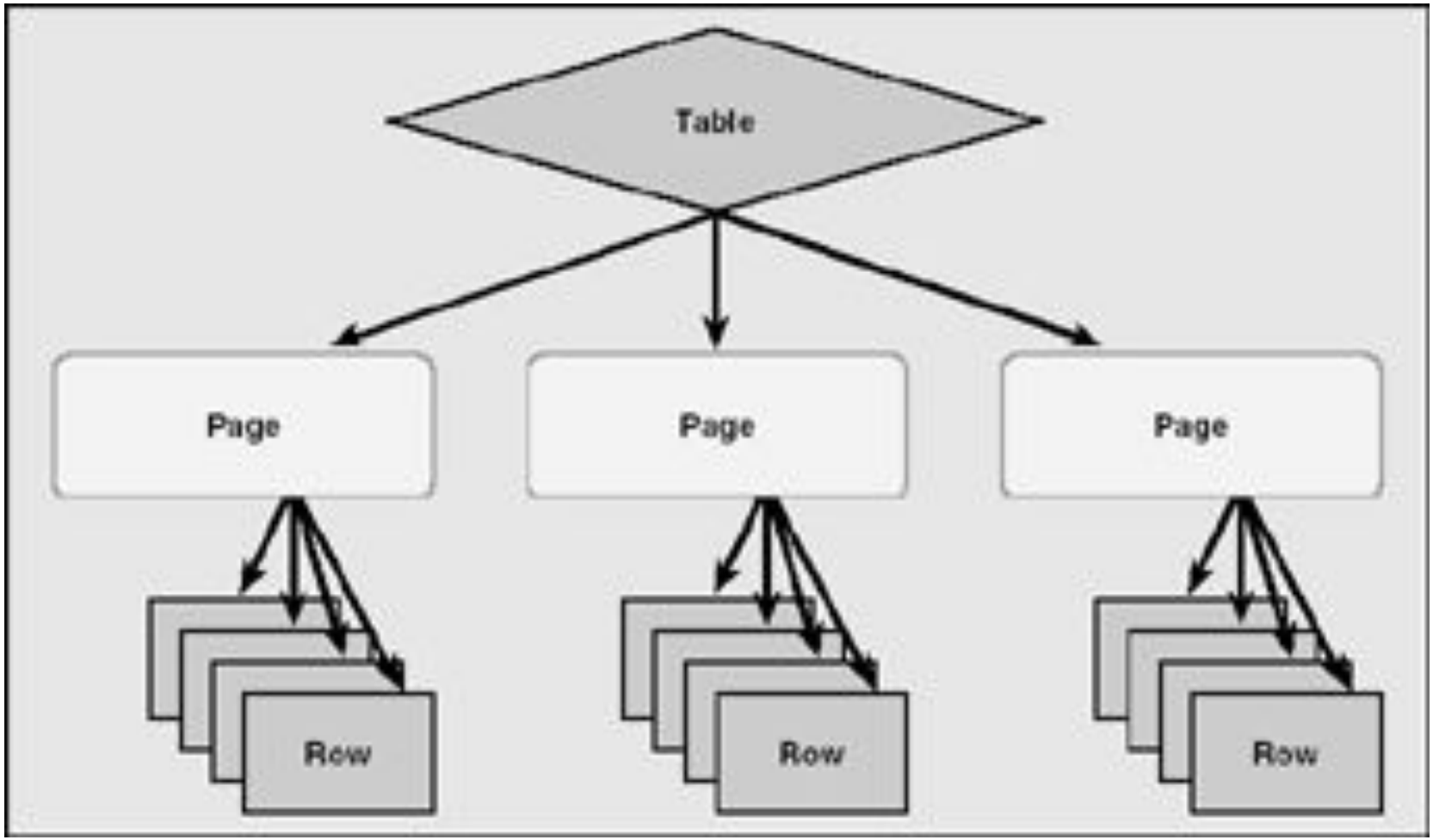
# Уровни блокировок

- Table (TAB): Это самая грубая логическая блокировка, которую может использовать SQL Server.
- Extent (EXT): Эти блокировки не используются для блокирования логических строк, а используются, когда SQL Server создает новые таблицы или расширяет существующие, когда файл увеличивается в размере.
- Page (PAG): Когда SQL Server требуется заблокировать одновременно множество строк, а свободные слоты блокировок заканчиваются, то он может использовать страничные блокировки.
- Key (KEY): Лучший уровень блокировки, возможный в SQL Server, вместе с RID блокировкой. KEY блокировки используются в индексах, а RID блокировки - в таблицах-кучах.

# Уровни блокировок

- Высокая конкурентность означает, что множество пользователей может работать одновременно. По возможности это достигается путем небольших блокировок, чтобы не блокировать без необходимости данные, нужные другим пользователям.
- С другой стороны, высокая скорость может быть достигнута при помощи больших блокировок, что быстрее, чем накладывание множества маленьких блокировок.

# Иерархия объектов



# Эскалация блокировок

- Эскалация блокировок – это процесс, при котором множество блокировок с маленькой гранулярностью, конвертируются в одну блокировку на более высоком уровне иерархии с большей гранулярностью.

# Проблемы маленьких блокировок

- **Locking overhead**. иногда выгоднее наложить одну блокировку с большей гранулярностью, чем несколько (или несколько тысяч) более мелких.
- **Data contention**. Наложение одной блокировки - действие атомарное, а наложение нескольких блокировок уже таковым не является.
- **Resource contention**. Блокировки занимают место + нуждаются так же и в некотором обслуживании, которое расходует системные ресурсы (проверка на тупики). + предел количества транзакций, которые могут выполняться в системе одновременно без ущерба для производительности.

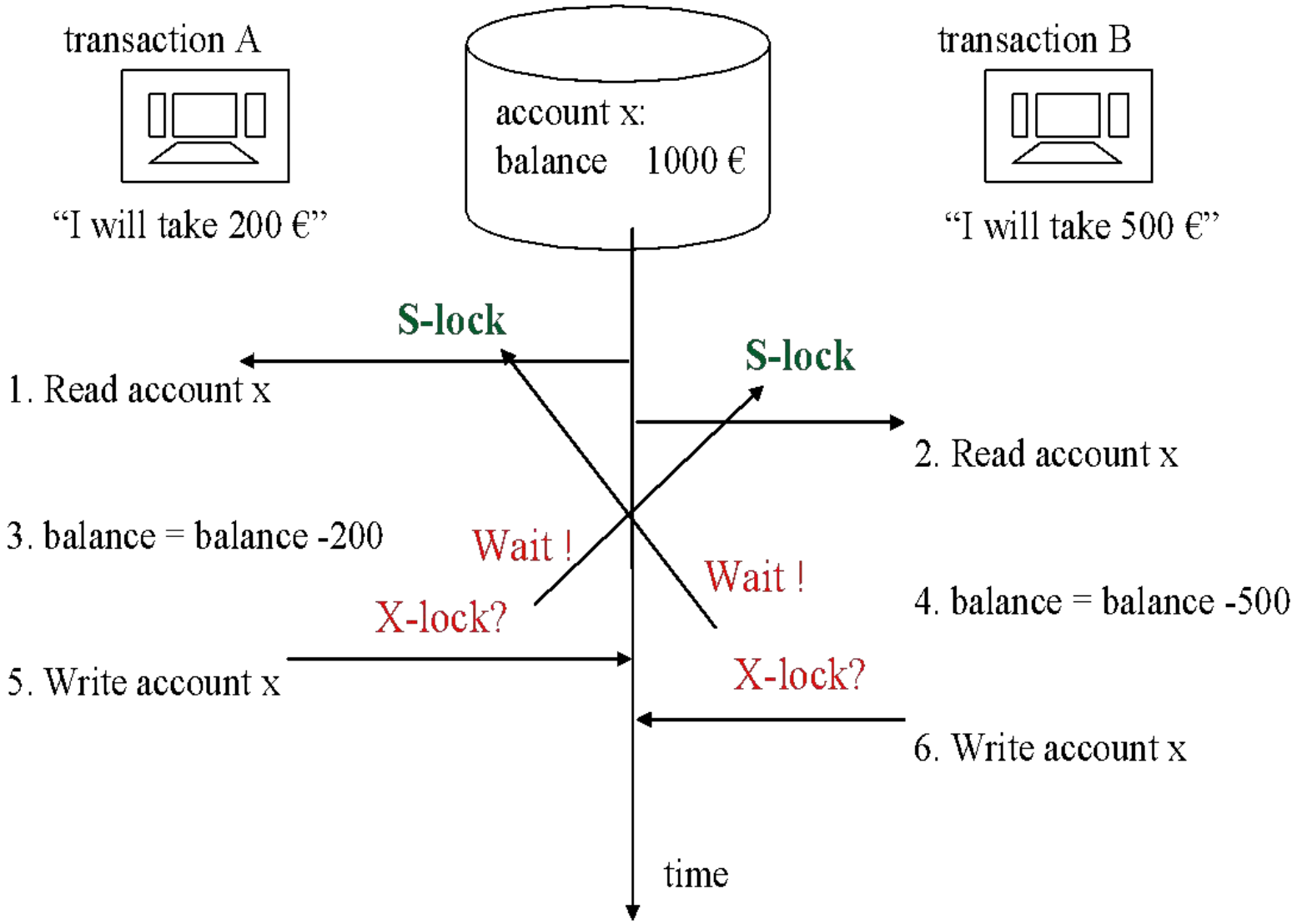
# Подсказки оптимизатору

- По умолчанию сервер старается наложить блокировку min гранулярности. Если сервер посчитает, что блокировать на уровне отдельной записи не самое оптимальное решение, то он заблокирует больший объем.
- Можно указать явно, с какой гранулярностью блокировать, с помощью специальных подсказок оптимизатору (hints) (в сторону увеличения!).
- ROWLOCK – блокировка на уровне записи.
- PGLOCK - блокировка на уровне страницы данных.
- TABLOCK – блокировка на уровне таблицы.

# Бесконечное ожидание

- Протокол блокировки уладит проблему потерянных обновлений, но если конкурирующие транзакции используют уровень изолированности, который не снимает R-блокировки до конца транзакции, то это может привести к бесконечному ожиданию. Транзакции будут ожидать друг друга в бесконечном цикле, называемом **взаимная блокировка (Deadlock)**. В ранних продуктах баз данных это было серьёзной проблемой, но современные СУБД включают в себя находящийся в спящем режиме поток выполнения, называемый **детектор взаимных блокировок (Deadlock Detector)**, который «просыпается», как правило, каждые 2 секунды (продолжительность «сна» можно изменять) для поиска взаимных блокировок, а после нахождения таковой выберет одну из ожидающих транзакций в качестве «жертвы» и произведёт этой «жертве» автоматический откат.

”Tellers”


transaction A
account x:
balance 1000 €
transaction B
“I will take 200 €”
“I will take 500 €”
S-lock
S-lock
1. Read account x
2. Read account x
3. balance = balance -200
Wait !
Wait !
4. balance = balance -500
X-lock?
X-lock?
5. Write account x
6. Write account x
time

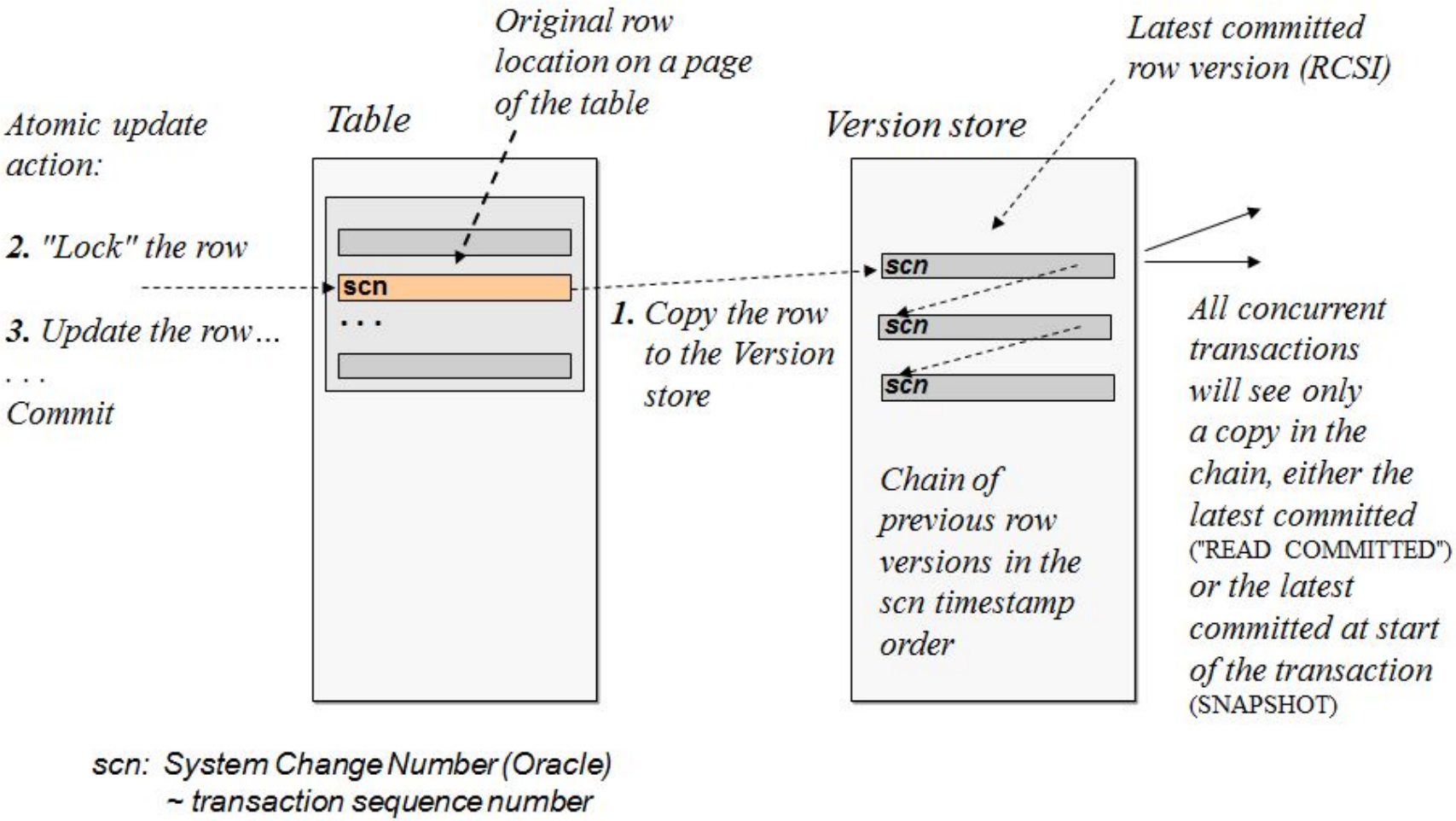
- Важно помнить, что никакая СУБД не может автоматически перезапустить прерванную взаимной блокировкой «жертву», за это несёт ответственность код приложения или сервер приложений, на который был установлен компонент клиента доступа к данным. Важно также понимать, что взаимная блокировка не является ошибкой, а прерывание транзакции-«жертвы» - это сервис, предоставляемый сервером для того, чтобы клиентские приложения могли продолжить работу в случае, когда выполнение одновременных транзакций не может быть продолжено.

# MVCC - многоверсионное управление параллелизмом

- В MVCC техника такова, что сервер сохраняет цепь истории в некоторых метках порядка обновленных версий для данных всех строк, так что для любой обновленной строки в момент начала любой параллельной транзакции может быть найдена зафиксированная версия.
- Эта техника управления параллелизмом исключает время ожидания чтения и обеспечивает всего 2 уровня изолированности:
  - **READ COMMITTED**
  - **SNAPSHOT**

# MVCC - многоверсионное управление параллелизмом

- Любая транзакция с уровнем изолированности **READ COMMITTED** получит **последние зафиксированные версии строк** от цепи истории.
- Транзакция с уровнем изолированности **SNAPSHOT** будет видеть только **последние версии строк, зафиксированные во время начала транзакции** (или записанные самой транзакцией). В уровне изолированности SNAPSHOT, транзакция никогда не увидит фантомные строки и не может даже предотвратить запись фантомных строк параллельными транзакциями, тогда как изолированность SERIALIZABLE, основанная на механизме LSCC (MGL), препятствует тому, чтобы параллельные транзакции записывали фантомные строки в базу данных. Фактически, транзакция продолжает видеть в снимке фантомные строки, которые параллельные транзакции тем временем удалили из базы данных.
- Несмотря на уровень изолированности, как правило, запись также защищена в системах MVCC какой-либо блокировкой строк. Обновление строк, содержание которых тем временем был обновлено другими, будет генерировать ошибки «Your snapshot is too old» («Ваш снимок слишком старый»).


Original row location on a page of the table
Latest committed row version (RCSI)
Atomic update action:
Table
Version store
2. "Lock" the row
scn
. . .
3. Update the row…
. . .
Commit
1. Copy the row to the Version store
scn
scn
scn
Chain of previous row versions in the scn timestamp order
All concurrent transactions will see only a copy in the chain, either the latest committed ("READ COMMITTED") or the latest committed at start of the transaction (SNAPSHOT)
scn: System Change Number (Oracle)
~ transaction sequence number

- В MVCC Oracle первой транзакции для записи строки (то есть для выполнения вставки, обновления или удаления) будет предоставлена блокировка строки и приоритет записи, а конкурирующие записи будут помещены в очередь. Блокировки строки реализуются при помощи маркировки записанных строк посредством SCN (англ. «System Change Number» - системный номер изменения) транзакции, порядковыми номерами начатой транзакции. Поскольку SCN строки принадлежит активной транзакции, то эта строка будет зарезервирована для этой транзакции. Использование блокировки по записи означает, что взаимные блокировки возможны, но вместо автоматического прерывания «жертвы», Oracle немедленно находит блокировку строки, которая бы могла привести к взаимной блокировке, вызывает исключение в приложении клиента и ожидает клиента, чтобы разрешить взаимную блокировку явной командой отката (ROLLBACK).

- Технику управления параллелизмом в Oracle можно назвать гибридным CC, так как в дополнение к MVCC с неявной блокировкой строки, Oracle обеспечивает явные команды «LOCK TABLE», а также явную блокировку строк посредством команды «SELECT ... FOR UPDATE», которая обеспечивает средства для предотвращения невидимых фантомных строк. В Oracle транзакция также может быть объявлена как Read Only (только для чтения).
-
- Microsoft также отметил преимущества MVCC, а в SQL Server, начиная с версии 2005, стало возможным настроить в сервере использование управления версиями строк при помощи настроек свойств базы данных посредством команд Transact-SQL, а начиная с версии 2012 – посредством свойств базы данных, как это показано на рисунке 2.9.

- Механизм управления параллелизмом в MySQL/InnoDB является реальным гибридным CC, обеспечивающим для чтения четыре уровня изолированности:

- READ UNCOMMITTED считывает строки таблицы без R-блокировки;
- READ COMMITTED (фактически «Read Latest Committed») считывает строки таблицы даже когда они заблокированы посредством MVCC;
- REPEATABLE READ (фактически «Snapshot»), использующий MVCC;
- SERIALIZABLE, использующий MGL CC с S-блокировками для предотвращения появления фантомных строк.

- Примечание: независимо от уровня изолированности запись всегда будет нуждаться в защите исключительной блокировкой.

# OCC - управление оптимистичным параллелизмом

- **Begin**: Пометка о времени начала транзакции.
- **Modify**: Чтение и запись изменение данных.
- **Validate**: Поиск конфликтов - проверка изменений данных, которые читала и меняла транзакция, другими транзакциями.
- **Commit/Rollback**: Если конфликтов нет, изменения записываются. Если конфликт обнаружен, то откат транзакции.

# Восстановление

- Индивидуальный откат транзакции.
- Восстановление после внезапной потери содержимого оперативной памяти.
- Восстановление после поломки основного внешнего носителя базы данных.

# Отказы приложений

- завершение оператором ROLLBACK;
- аварийное завершение работы прикладной программы;
- принудительный откат транзакции в случае взаимной блокировки при параллельном выполнении транзакций.

# Технические проблемы

- Мягкий сбой
  - при аварийном выключении электрического питания;
  - при возникновении неустранимого сбоя процессора и т. д. Потеря той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.
- Жесткий сбой
  - Восстановление после поломки основного внешнего носителя базы данных. Основой восстановления является архивная копия и журнал изменений базы данных.

# Возобновление при отказах

- Отказы приложений/транзакции: транзакции обрываются
- Отказы системы: функционирование возобновляется после рестарта и восстановления согласованности БД
- Разрушение носителя: рестарт, восстановление с резервной копии, восстановление согласованности

# Восстановление оборванных транзакций: откат

- Обрывы транзакций могут быть вызваны ошибками при выполнении приложений или невозможностью выполнить транзакцию сериализуемым образом
- Операции обрыва можно заменить на выполнение отката для всех операций записи, включаемых в расписание в обратном порядке с последующей фиксацией

# Восстановимость

- Восстановимость расписаний: транзакции должны фиксироваться до того, как их результаты используются другими транзакциями
- Пример невосстановимого расписания
  - $w_1(x)\ w_2(x)\ c_2\ a_1$
- S2L (Двухфазный протокол с удержанием замков на запись до конца транзакции) гарантирует восстановимость

# Откат

- По команде rollback система откатит транзакцию на начало (на неявную точку отката)
- По команде commit – зафиксирует всё до неявной точки фиксации, которая соответствует последней завершённой команде в транзакции. Если в транзакции из нескольких команд во время выполнения очередной команды возникнет ошибка, то система откатит только эту ошибочную команду, т.е. отменит её результаты и сохранит прежнюю неявную точку фиксации.

# Оптимистический и пессимистический подходы

- Для обеспечения целостности транзакции СУБД может откладывать запись изменений в БД до момента успешного выполнения всех операций, входящих в транзакцию, и получения команды подтверждения транзакции (commit).
- Иногда используется другой подход: система записывает изменения в БД, не дожидаясь завершения транзакции, а старые значения данных сохраняет на время выполнения транзакции в сегментах отката.

# Сегмент отката (rollback segment, RBS)

- Сегмент отката – это специальная область памяти на диске, в которую записывается информация обо всех текущих изменениях. Обычно записывается "старое" и "новое" содержимое изменённых записей, чтобы можно было восстановить прежнее состояние БД при откате транзакции (по команде rollback) или при откате текущей операции (в случае возникновения ошибки).
- Данные в RBS хранятся до тех пор, пока транзакция, изменяющая эти данные, не будет завершена. Потом они могут быть перезаписаны данными более поздних транзакций.

# Savepoint

- savepoint запоминает промежуточную "текущую копию" состояния базы данных для того, чтобы при необходимости можно было вернуться к состоянию БД в точке сохранения: откатить работу от текущего момента до точки сохранения (rollback to <имя_точки>) или зафиксировать работу от начала транзакции до точки сохранения (commit to <имя_точки>).
- На одну транзакцию может быть несколько точек сохранения (ограничение на их количество зависит от СУБД).

# Журнал транзакций

- Журнал транзакций – это часть БД, в которую поступают данные обо всех изменениях всех объектов БД. Журнал недоступен пользователям СУБД и поддерживается особо тщательно. Форма записи в журнал изменений зависит от СУБД. Но обычно там фиксируется:
  - номер транзакции (номера присваиваются автоматически по возрастанию);
  - состояние транзакции (завершена фиксацией или откатом, не завершена, находится в состоянии ожидания);
  - точки сохранения (явные и неявные);
  - команды, составляющие транзакцию, и проч.

# Ведение журнала

- Журнал ведется последовательно
- Опережающая запись в журнал (WAL, write-ahead log)
- Регистрируются операции записи, начала и конца транзакций
- Каждая запись содержит данные отката (undo)и «наката» (redo)
- При фиксации запись журнала обязательно «выталкивается» на диск

# Фиксация транзакции

- Изменения, внесённые транзакцией, помечаются как постоянные.
- Уничтожаются все точки сохранения для данной транзакции.
- Если выполнение транзакций осуществляется с помощью блокировок, то освобождаются объекты, заблокированные транзакцией.
- В журнале транзакций транзакция помечается как завершенная, уничтожаются системные записи о транзакции в оперативной памяти.

# Восстановление после системных отказов

- Необходим рестарт сервера БД
- Для того, чтобы восстановление было возможным, необходимо вести журнал обновлений
- При нормальной работе БД все изменения записываются в журнал
- При рестарте журнал используется для повторения операций и для отката незавершенных транзакций

# Общие принципы восстановления

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

# Что нужно сделать при восстановлении

- Определить, какие транзакции были зафиксированы до отказа и какие были активными (не завершены)
- Если необходимо, повторить изменения зафиксированных транзакций
- Оборвать незавершенные до отказа транзакции (выполнить откат)

# Журнал транзакций

- сохранение промежуточных состояний,
- подтверждение транзакции,
- отката транзакции

# Журнал транзакций поддерживает:

- восстановление отдельных транзакций;
- восстановление всех незавершенных транзакций при запуске SQL Server;
- откат восстановленной базы данных, файла, файловой группы или страницы до момента сбоя.
- поддержка репликации транзакций;
- поддержка решений высокого уровня доступности и аварийного восстановления

# Алгоритм восстановления при рестарте

- Фаза просмотра: найти все зафиксированные и активные транзакции (прямой просмотр журнала)
- Фаза наката (redo): при прямом просмотре, выполнить все операции зафиксированных транзакций
- Фаза отката (undo): обратный просмотр журнала, откат всех операций незавершенных транзакций

# Завершение восстановления

- После завершения фазы отката необходимо
  - Записать на диск все измененные блоки БД
  - После записи БД можно очистить журнал
- Возобновить нормальную работу системы

```
CREATE DATABASE Sales
ON ( NAME = Sales_dat, FILENAME =
    'C:\tmp\saledat.mdf',
SIZE = 10,
MAXSIZE = 50,
FILEGROWTH = 5 )
LOG ON ( NAME = Sales_log, FILENAME =
    'C:\tmp\salelog.ldf',
SIZE = 5MB,
MAXSIZE = 25MB,
FILEGROWTH = 5MB ) ;
```

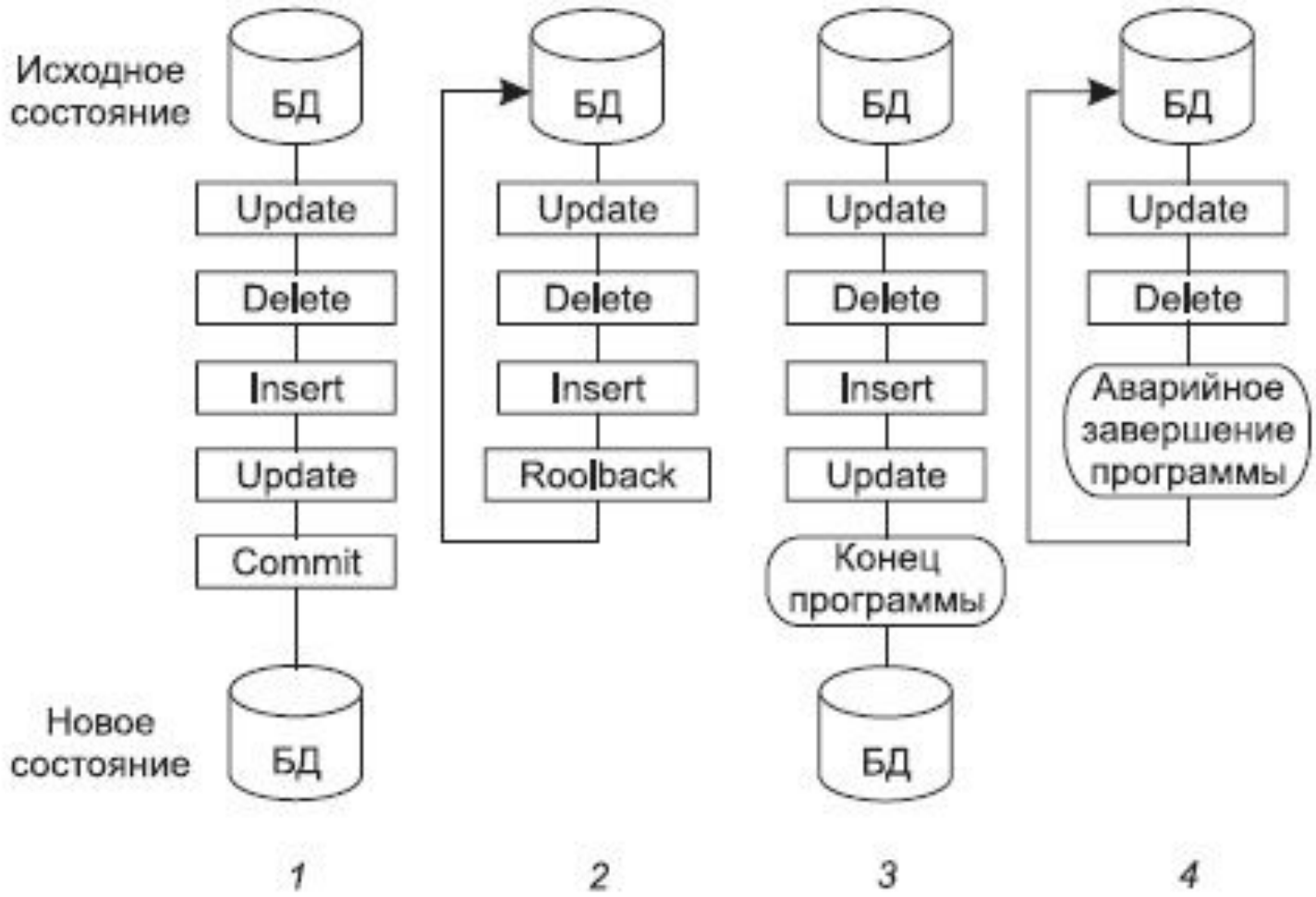
# Усечение журнала транзакций

- Процесс усечения журнала освобождает место в файле журнала для повторного использования журналом транзакций.
- Усечение журнала необходимо для предотвращения переполнения журнала.
- При усечении журнала удаляются неактивные виртуальные файлы журнала из логического журнала транзакций базы данных SQL Server.
- Если усечение журнала транзакций не выполняется, со временем он заполняет все доступное место на диске, отведенное для файлов физического журнала.
- Усечение журнала выполняется автоматически

# Сжатие журнала

- Усечение журнала не приводит к уменьшению размера физического файла журнала.
- Для уменьшения реального размера физического файла журнала необходимо выполнить его сжатие.
- DBCC SHRINKFILE (DataFile1, 7);
  DataFile1 – имя файла, 7 – размер в МБ);

# Модель транзакций



# Модель транзакций

1. оператор COMMIT означает успешное завершение транзакции; его использование делает постоянными изменения, внесенные в базу данных в рамках текущей транзакции;
2. оператор ROLLBACK прерывает транзакцию, отменяя изменения, сделанные в базе данных в рамках этой транзакции; новая транзакция начинается непосредственно после использования ROLLBACK;
3. успешное завершение программы, в которой была инициирована текущая транзакция, означает успешное завершение транзакции (как будто был использован оператор COMMIT);
4. ошибочное завершение программы прерывает транзакцию (как будто был использован оператор ROLLBACK).