МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра Прикладная математика и информатика
(наименование)

09.04.03 Прикладная информатика
(код и наименование направления подготовки)

Информационные системы и технологии корпоративного управления
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Моделирование системы сбора и обработки больших массивов данных»

Студент С.А. Кондрусева
(И.О. Фамилия) (личная подпись)

Научный руководитель д.т.н, доцент, С.В. Мкртычев
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Оглавление

Введение

Как известно, для сбора и обработки информации используются системы оперативной транзакционной обработки данных – OLTP-системы.

В настоящее время предъявляются повышенные требования к эффективности указанных систем, приближая их характеристики к системам реального времени.

Решения данной проблемы существенно усложняется, если объектом обработки являются большие массивы данных, неограниченных источником которых является Интернет вещей.

Под большими массивами данных в рассматриваемом контексте понимаются большие структурированные данные.

Следует также отметить, что некоторые предприятия и компании социально-экономической сферы могут использовать различные источники внешних данных и объединять их со своей транзакционными данными.

Как показывает практика, эффективность OLTP-системы зависит от модели, положенной в ее основу в процессе проектирования.

Актуальность темы исследования обусловлена необходимостью разработки модели системы сбора и обработки больших массивов данных, обеспечивающей повышение эффективности указанной системы.

Объектом исследования магистерской диссертации являются системы сбора и обработки больших массивов данных.

Предметом исследования является модель системы сбора и обработки больших массивов данных.

Целью работы является разработка модели эффективной системы сбора и обработки больших массивов данных.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать современное состояние проблемы исследования.
2. Проанализировать и выбрать методологию разработки высокоэффективных систем сбора и обработки больших массивов

данных.

3. Разработать модель системы сбора и обработки больших массивов данных.

4. Проверить адекватность предлагаемой модели.

Гипотеза исследования: применение предлагаемой модели в качестве основы для построения системы сбора и обработки больших массивов данных позволит повысить эффективность последней.

Методы исследования. В процессе исследования будут использованы следующие положения и методы: системный анализ, методологии и технологии построения OLTP-систем.

Новизна исследования заключается в разработке модели эффективной системы сбора и обработки больших массивов данных.

Практическая значимость исследования заключается в возможности практического применения предлагаемой модели для построения эффективной системы сбора и обработки больших массивов данных.

Теоретической основой диссертационного исследования являются научные труды российских и зарубежных ученых, занимающихся проблемами моделирования и повышения эффективности OLTP-систем.

Основные этапы исследования: исследование проводилось с 2018 по 2020 год в несколько этапов:

На первом этапе (констатирующем этапе) – формулировалась тема исследования, выполнялся сбор информации по теме исследования из различных источников, проводилась формулировка гипотезы, определялись постановка цели, задач, предмета исследования, объекта исследования и выполнялось определение проблематики данного исследования.

Второй этап (поисковый этап) – в ходе проведения данного этапа осуществлялся анализ методологий моделирования OLTP-систем, была разработана модель эффективной системы сбора и обработки больших массивов данных, подготовлены и опубликованы научные статьи по теме исследования в научных журналах и сборниках.

Третий этап (оценка эффективности) – на данном этапе осуществлялась оценка эффективности и проверка адекватности предлагаемой модели системы сбора и обработки больших массивов данных, сформулированы выводы о полученных результатах по проведенному исследованию.

На защиту выносятся:

1. Модель эффективной системы сбора и обработки больших массивов данных.
2. Результаты проверки адекватности предлагаемой модели системы сбора и обработки больших массивов данных.

По теме исследования опубликованы 2 статьи:

1. Кондрусева С.А. Технологии сбора и обработки больших массивов информации // В сборнике: Прикладная математика и информатика: современные исследования в области естественных и технических наук. Материалы VI Международной научно-практической конференции (школы-семинара) молодых ученых. 2020 (принята к публикации).
2. Кондрусева С.А. Методы повышения эффективности OLTP-систем // Вестник научных конференций. 2020. N 5-3(57). С. 52-53. https://ukonf.com/doc/cn.2020.05.03.pdf

Диссертация состоит из введения, трех глав, заключения и списка литературы.

В первой главе проанализировано современное состояния проблемы повышения эффективности систем сбора и обработки больших массивов данных. Рассмотрено современное представление о системах транзакционной обработки данных. Описаны принципы построения систем транзакционной обработки данных и методологические основы их моделирования.

Вторая глава посвящена анализу и выбору методологии разработки высокоэффективных систем сбора и обработки больших массивов данных.

Даны обзор и анализ технологий управления эффективностью систем сбора и обработки больших массивов данных. Рассмотрены технологии распределенных баз данных, NoSQL, NewSQL и In-Memory. Дан

сравнительный анализ представленных технологий. Дан сравнительный анализ и выбран подход к моделированию предметно-ориентированной OLTP-системы.

Третья глава посвящена непосредственно разработке логической и физической моделей эффективной системы сбора и обработки больших массивов данных и оценка ее эффективности.

Произведен выбор методологии разработки логической модели. На основе созданной модели разработана физическая модель OLTP-системы, представляющая ее программную реализацию.

Проведен эксперимент для оценки эффективности OLTP-системы и проверке адекватности предлагаемой модели.

В заключении приводятся результаты исследования.

Работа изложена на 73 страницах и включает 26 рисунков, 6 таблиц, 37 источников.

Глава 1 Анализ современного состояния проблемы повышения эффективности систем сбора и обработки больших массивов данных

1.1 Современное представление о системах транзакционной обработки данных

Для транзакционной обработки данных используются OLTP-системы.

OLTP (OnLine Transaction Processing) – оперативная транзакционная обработка данных – это режим сбора и обработки данных, который характеризуется короткими транзакциями, записывающими бизнес-события.

Согласно современному определению OLTP-системы компании Gartner,Inc. «OLTP – это режим сбора и обработки данных, который характеризуется короткими транзакциями, записывающими бизнес-события. Обычно требует высокой доступности и согласованного, короткого времени отклика. Эта категория приложений требует, чтобы запрос на обслуживание выполнялся в течение предсказуемого периода, приближающегося к «реальному времени». База данных мгновенно отражает изменения по мере их возникновения. Другими словами, обработка данных моделирует реальный бизнес в реальном времени, а транзакция преобразует эту модель из одного бизнес-состояния в другое» [22].

Для хранения данных в OLTP-системе используется база данных, которая является ее обязательным компонентом.

База данных (БД) должна мгновенно отражать изменения данных по мере их возникновения.

Другими словами, обработка данных моделирует реальный бизнес в реальном времени, а транзакция преобразует эту модель из одного бизнес-состояния в другое.

В основу работы OLTP положена концепция транзакции.

Транзакция – это неделимая последовательность операций над данными БД, которая отслеживается СУБД от начала и до завершения.

Транзакционные данные – это информация, которая отслеживает взаимодействия, связанные с деятельностью организации.

Эти взаимодействия обычно представляют собой бизнес-операции, такие как платежи, полученные от клиентов, платежи поставщикам, перемещение продуктов через запасы, принятые заказы или предоставленные услуги. Транзакционные события, которые представляют сами транзакции, обычно содержат измерение времени, некоторые числовые значения и ссылки на другие данные [35].

Свойства транзакции, представленные на слайде, описываются акронимом ACID, что в переводе на русский язык означает «кислота» [17].

ACID – это Atomicity (Атомарность), Consistency (Согласованность), Isolation (Изоляция), Durability (Долговечность).

Другими словами, транзакции должны быть атомарными и последовательными.

Атомарность означает, что вся транзакция всегда завершается успешно или терпит неудачу как одна единица работы и никогда не остается в полузавершенном состоянии. Если транзакция не может быть завершена, система базы данных должна откатить все шаги, которые уже были выполнены как часть этой транзакции.

В традиционной СУБД этот откат происходит автоматически, если транзакция не может быть завершена. Согласованность означает, что транзакции всегда оставляют данные в допустимом состоянии.

Транзакционные базы данных могут поддерживать строгую согласованность транзакций с использованием различных стратегий блокировки, таких как пессимистическая блокировка, чтобы гарантировать, что все данные строго согласованы в контексте предприятия для всех пользователей и процессов.

Транзакцией считается любая операция баз данных, в том числе чтение или запись данных. Для OLTP-систем характерны большие объемы запросов с интенсивным обновлением.

В системе OLTP-системах точность и целостность транзакций БД имеют высший приоритет.

Почему это важно? Потому что OLTP-системы предназначены для одновременного использования сотнями пользователей.

Если один пользователь обновляет запись, система должна гарантировать, что все последующие запросы будут отражать последнее состояние.

В противном случае данные будут неточными и не синхронизированными.

Типовая OLTP -система реализует следующую функциональность:

– сбор и представление информации в форме, удобной для обработки и хранения в режиме онлайн;

– хранение данных о выполняемых бизнес-транзакциях. OLTP-транзакции обычно очень специфичны для задачи, которую они выполняют, и обычно охватывают одну или небольшой набор записей БД;

– формирование операционной аналитической отчетности. Данная функция обусловлена принадлежностью OLTP-системы к первому уровню архитектуры информационно-аналитических систем.

Важно отметить, что OLTP-системы должны иметь чрезвычайно высокую доступность.

Это связано с тем, что указанные системы часто имеют дело с критически важными данными и обслуживают большое количество пользователей.

OLTP-системы имеют следующие характеристики:

– короткие транзакции. OLTP-системы обычно считывают и обрабатывают очень избирательные небольшие объемы данных. Обработка данных в основном проста, а сложные соединения

– относительно редки;

– большое количество пользователей. В OLTP-системах могут быть очень большие группы пользователей, причем многие пользователи пытаются

получить доступ к одним и тем же данным одновременно;

– большая частота обновлений;

– простые запросы: INSERT, UPDATE, DELETE;

– минимальное время отклика на запрос;

– высокая степень параллелизма. Из-за большого количества пользователей, короткого времени отклика и небольших транзакций параллелизм в OLTP-системах очень высок.

Таблица 1.1 – Требования к OLTP-системе по ключевым характеристикам

Характеристика	Требования к OLTP-системе
Степень детализации хранимых данных	Хранение только детализированных данных
Качество данных	Допускаются неверные данные из-за ошибок ввода
Формат хранения данных	Может содержать данные в разных форматах в зависимости от приложений
Допущение избыточных данных	Должна обеспечиваться максимальная нормализация
Управление данными	Должна быть возможность в любое время добавлять, удалять и изменять данные
Количество хранимых данных	Должны быть доступны все оперативные данные, требующиеся в данный момент
Характер запросов к данным	Доступ к данным пользователей осуществляется по заранее составленным запросам
Время обработки обращений к данным	Время отклика системы измеряется в секундах
Характер вычислительной нагрузки на систему	Постоянно средняя загрузка процессора
Приоритетность характеристик системы	Основными приоритетами являются высокая производительность и доступность

Эффективность систем транзакционной обработки данных определяется количеством транзакций в секунду [32].

Проблемы обеспечения эффективности OLTP-систем:

– OLTP-системы не всегда подходят для обработки агрегатов по большим объемам данных, хотя есть исключения, такие как хорошо спланированное решение на основе SQL Server. Аналитика данных, основанная на совокупных вычислениях миллионов отдельных транзакций,

очень ресурсоемка для системы OLTP. Они могут выполняться медленно и могут вызывать замедление из-за блокировки других транзакций в базе данных.

– При проведении аналитики и составлении отчетов по данным, которые сильно нормализованы, запросы, как правило, являются сложными, потому что большинство запросов требует денормализации данных с помощью объединений. Кроме того, соглашения об именах для объектов базы данных в OLTP-системах, как правило, лаконичны. Повышенная нормализация в сочетании с краткими соглашениями об именах затрудняет выполнение запросов бизнес-пользователями к OLTP-системам без помощи администратора баз данных или разработчика данных.

– Хранение истории транзакций на неопределенный срок и хранение слишком большого количества данных в какой-либо одной таблице может привести к снижению производительности запросов в зависимости от количества сохраненных транзакций. Распространенным решением является поддержание соответствующего временного окна (например, текущего финансового года) в системе OLTP и выгрузка исторических данных в другие системы, такие как витрина или хранилище данных.

На основе технологии OLTP строятся предметно-ориентированные учетные системы, биллинговые системы, автоматизированные банковские системы и др.

1.2 Принципы построения систем транзакционной обработки данных

Принципы построения OLTP-систем представлены в публикациях специалистов компаний-вендоров современных СУБД: Microsoft, Oracle, IBM и др.

Как показал обзор источников, OLTP-системы строятся на основе архитектуры «клиент-сервер» с применением в качестве СУБД сервера баз

данных (СБД).

При этом часть бизнес-логики OLTP-системы выполняется на СБД и реализуется с помощью соответствующих программных объектов: хранимых процедур и триггеров.

На практике наиболее распространенной архитектурой развертывания, использующей транзакционные данные, является уровень хранилища данных в трехуровневой архитектуре (рисунок 1.1.) [2].

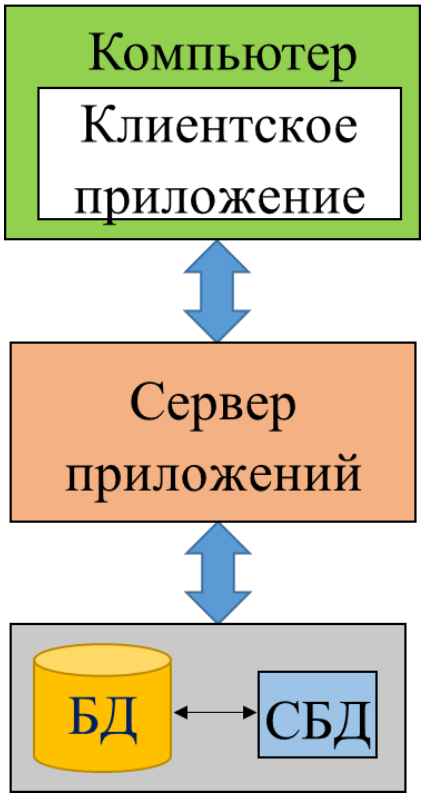


Рисунок 1.1 – Трехзвенная модель архитектуры «клиент-сервер» OLTP-системы

На основе данной архитектуры строятся веб-приложения.

Трехуровневая архитектура обычно состоит из уровня представления (клиент), уровня бизнес-логики (сервер приложений, функции которого выполняет веб-сервер) и уровня хранилища данных (СБД).

Помимо применения архитектуры «клиент-сервер» выделяются следующие принципы построения OLTP-систем [31]:

– использование реляционной БД;

– сильная нормализация данных, преимущественно нормальная форма Бойса-Кодда (3Ф+);

– обеспечение целостности данных;

– преимущественная децентрализация для предотвращения единых точек отказа (единая точка отказа – это узел, линия связи или объект системы доступности данных, отказ которого может вывести из строя всю систему, или вызвать недоступность данных).

Перечислим преимущества сильной нормализации данных для OLTP декларируемые корпорацией Майкрософт [8]:

– более быстрая сортировка и создание индексов;

– большее количество кластерных индексов в БД, построенных на первичных ключах. Так как таблица может иметь только один кластерный индекс, строки данных могут быть отсортированы только в одном порядке;

– более узкие и компактные индексы;

– меньшее количество индексов в таблице. Это улучшает производительность инструкций INSERT, UPDATE и DELETE;

– меньшее количество NULL-значений и более низкая вероятность несогласованности. Это повышает компактность БД.

При усилении нормализации количество и сложность соединений, необходимых для получения данных, также возрастают.

Слишком большое количество сложных реляционных соединений между слишком большим количеством таблиц может снизить производительность.

Рациональная нормализация часто включает несколько регулярно выполняемых запросов, которые используют соединения, включающие более четырех таблиц.

Следует отметить, вышеперечисленные принципы построения OLTP-систем распространяются на решения, предназначенные для обработки больших массивов данных.

1.3 Методологические основы моделирования систем транзакционной обработки данных

Методологические основы моделирования OLTP-систем рассмотрены в работах К.Дж. Дейта, В. Маккарти, У. Мерти, Г. Гиртса, С.В. Мкртычева и др.

В большинстве известных работ по данной проблематике рассматриваются вопросы, связанные с моделированием данных OLTP-систем, в которых рассматриваются такие методологии моделирования данных, как «сущность-связь» (ER-model), анкерное моделирование (Anchor model) и др. [7, 33].

Так, в работе [16] перечисляются основные требования к моделям данных систем обработки Big Data:

– производительность: хорошие модели данных могут помочь нам быстро запросить необходимые данные и снизить пропускную способность ввода-вывода;

– стоимость. Хорошие модели данных могут значительно сократить ненужную избыточность данных, повторно использовать результаты вычислений и снизить затраты на хранение и вычисления для системы больших данных;

– эффективность: хорошие модели данных могут значительно улучшить взаимодействие с пользователем и повысить эффективность использования данных;

– качество: хорошие модели данных делают статистику данных более согласованной и уменьшают вероятность ошибок вычислений.

С проектированием OLTP тесно связаны методологии моделирования автоматизированных учетных систем.

В основу современных учетных систем положена модель REA (Resources – Events – Agents, Ресурсы-События-Агенты), предложенная В. Маккарти [28].

REA – онтология описывает учетную систему как виртуальное представление реального бизнес-процесса.

Пример REA-модели представлен на рисунке 1.2.

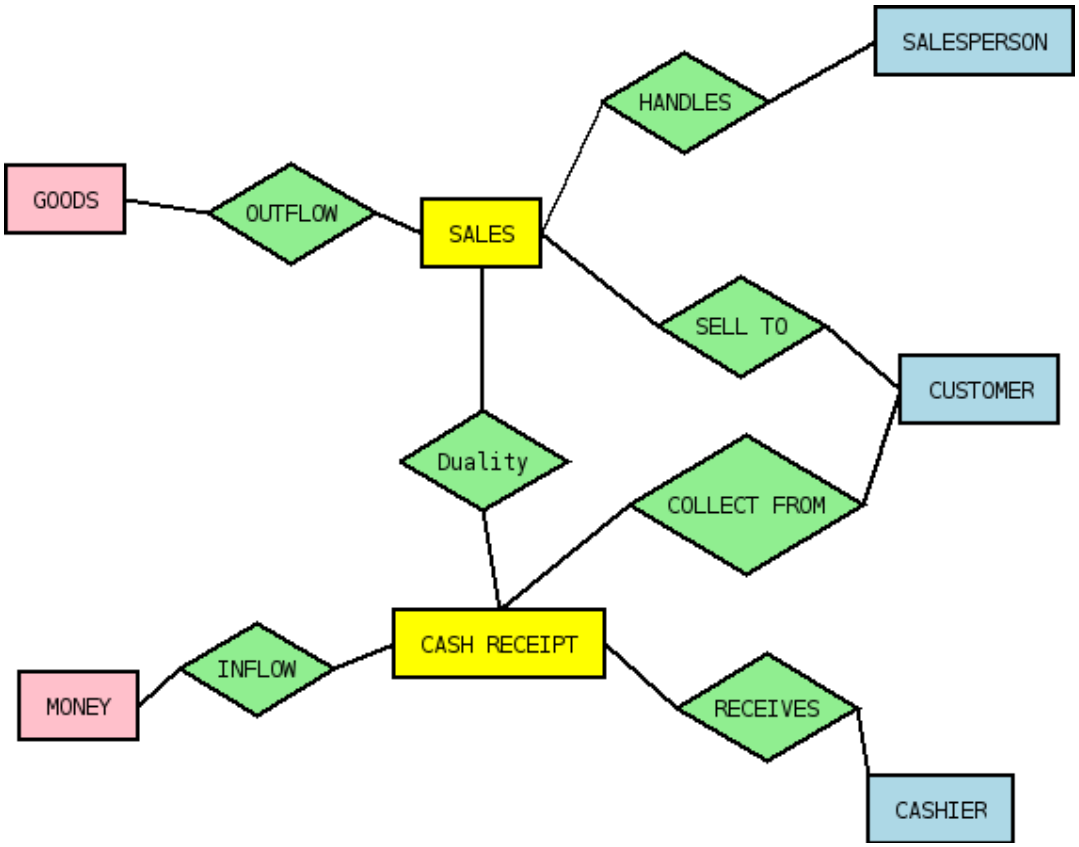


Рисунок 1.2 – REA-модель

Здесь:

Товары, деньги или услуги – ресурсы.

Бизнес-транзакции или соглашения, влияющие на ресурсы – события.

Люди или компании – агенты.

Следует отметить, что в работе [30] описана модель на основе онтологии REA для отображения больших данных в элементы информационных систем учета.

По мнению авторов, модель и методы, представленные в этой статье, могут использоваться организациями для формализации связей между внешними элементами больших данных в их среде и их артефактами бухгалтерской информации, для создания архитектур, извлекающих

информацию из внешних источников больших данных для использования в контексте бухгалтерского учета, и чтобы использовать возможности аналитики для более эффективного принятия решений.

На рисунке 1.3 представлена модель, отображающая связи между этими пятью относительно структурированными внешними источниками больших данных и примитивами REA в автоматизированную информационную систему (AIS).

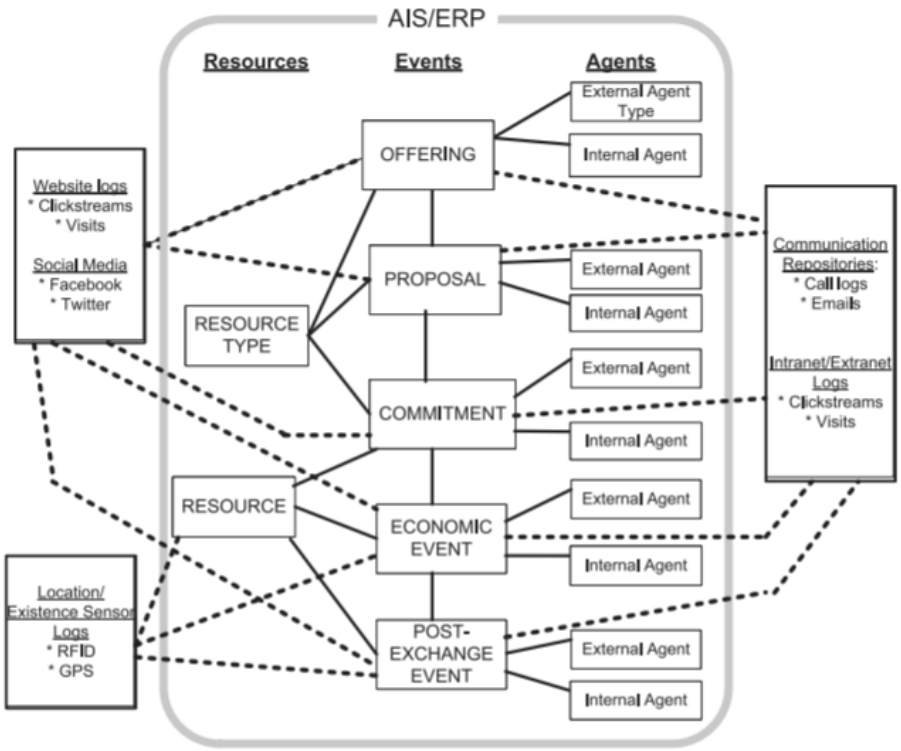


Рисунок 1.3 – Модель, отображающая связи между элементами больших данных и примитивами онтологии REA

К достоинствам REA-модели можно отнести простоту преобразования в ER-модель реляционной базы данных OLTP-системы.

Согласно зарубежным источникам, REA-модель широко применяется в ERP-системах.

Недостатки REA-модели: существует проблема идентификации и формализации объектов, представляемых концептами REA в рамках исследуемой предметной области, что ограничивает возможности REA онтологии при моделировании OLTP-систем для предприятий с ярко выраженной производственной спецификой.

Мкртычев С.В. предложил методологию построения информационных систем управленческого учета (ИСУУ), которая состоит из следующих стадий [29].

1. Объектно-структурное моделирование ИСУУ.

Объектно-структурная модель (ОСМ) системы N-этапной обработки учетной информации имеет вид ориентированного графа O(W,S,D), изображенного на рисунок 1.4.

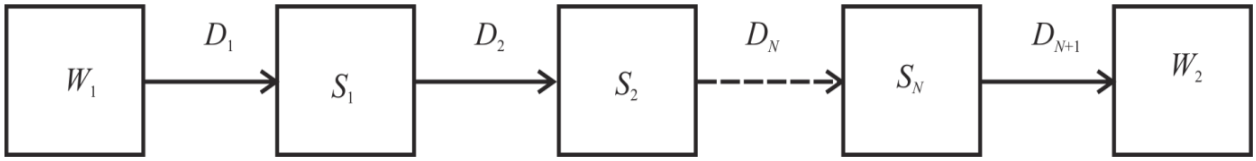


Рисунок 1.4 – Объектно-структурная модель системы N-этапной обработки учетной информации

На рисунке:

$W = \{W_1, W_2\}$ – узлы, обозначающие объекты класса “Склад”.

$S = \{S_1, S_2,..., S_N\}$ – множество узлов, обозначающие объекты класса “Этап (Склад+Агрегат+Контролер)”.

$D = \{D_1, D_2,..., D_{N+1}\}$ – множество дуг, нагруженных элементами информационного потока.

Физическим представлением ОСМ являет бизнес-транзакция.

2. Формализация элементов ОСМ

Основана на автоматном подходе, предложенным А. Шалыто.

Заключается в представлении объектно-структурной модели ИСУУ в виде системы взаимодействующих автоматов, управляющих статусом обрабатываемого элемента материального (информационного) потока, который описывается как конечный автомат (КА) в соответствии с его жизненным циклом, представленным в таблице 1.2.

Таблица 1.2 – Жизненный цикл продукта в многоэтапном производстве

Статус	Описание
1	Сырье
2	Полуфабрикат
3	Готовая продукция

3. Разработка UML паттернов проектирования ИСУУ

Паттерны проектирования строятся в нотации UML на основе объектных моделей КА, сгруппированных по суперклассам – классам технологической онтологии конкретной предметной области.

Примеры паттернов проектирования ИСУУ представлены на рисунке 1.5.

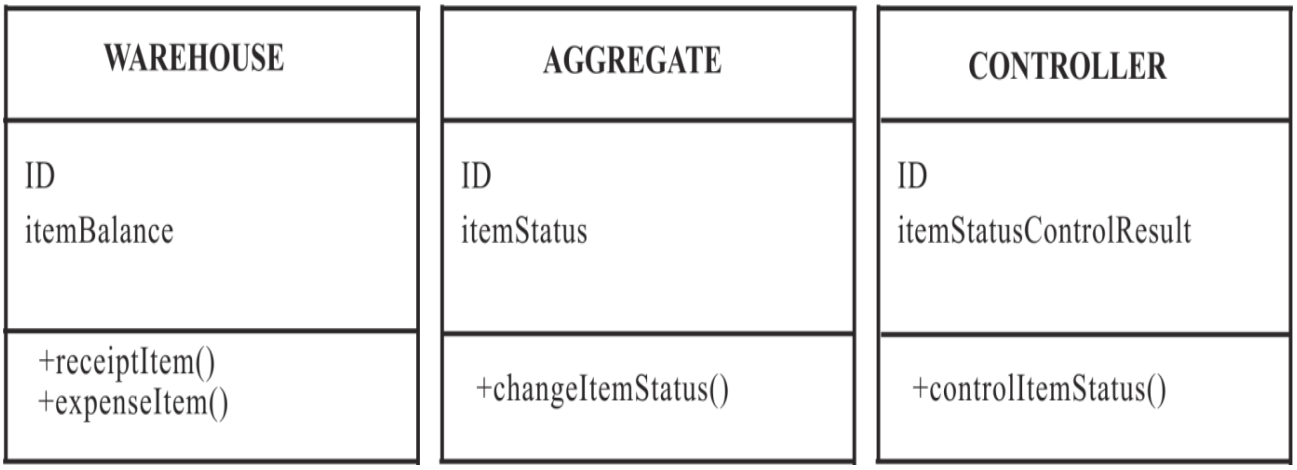


Рисунок 1.5 – Паттерны проектирования ИСУУ

Преимуществами предлагаемой методологии проектирования ИСУУ являются:

– Универсальность объектно-структурных моделей ИСУУ.

– Простота адаптации объектно-структурных моделей ИСУУ к специфике управленческого учета в конкретной организации.

– Простота интеграции ИСУУ в корпоративную информационную систему (КИС) предприятия.

Одним из таких паттернов является «Контролёр», обеспечивающий проверку целостности данных учетную систему.

К недостаткам можно отнести то, что область применения данной методологии ограничена OLTP-системами для многоэтапных производств.

Вместе с тем, необходимо констатировать недостаточность работ по проблематике моделирования OLTP-систем, включая транзакционную обработку больших данных.

Во многом это связано с тем известные работы больше ориентированы на описание проблем, связанных с моделированием предметно-ориентированных учетных систем с высокой эффективностью использования, а не OLTP-систем, на основе которых последние будут реализованы.

Выводы к главе 1

1. Для транзакционной обработки данных используются OLTP-системы.
2. Принципы построения OLTP-систем распространяются на решения, предназначенные для обработки больших массивов данных.
3. В большинстве известных работ по проблеме исследования рассматриваются вопросы, связанные с моделированием данных OLTP-систем
4. К достоинствам REA-модели можно отнести простоту преобразования в ER-модель реляционной базы данных OLTP-системы.
5. Преимуществами методологии проектирования ИСУУ являются универсальность объектно-структурных моделей ИСУУ и простота адаптации последних к специфике управленческого учета в конкретной организации.
6. Следует констатировать недостаточность работ по проблематике моделирования OLTP-систем для обработки больших данных.

Глава 2 Анализ и выбор методологии разработки высокоэффективных систем сбора и обработки больших массивов данных

2.1 Обзор и анализ технологий управления эффективностью систем сбора и обработки больших массивов данных

Как было отмечено выше эффективность OLTP-систем определяется количеством транзакций в секунду.

Такие факторы, как загрузка процессора, задержки диска и сети, а также нагрузка на память, влияют на общую производительность OLTP-систем.

Помимо сильной нормализации данных для повышения эффективности OLTP-систем используются следующие методы:

– применение твердотельных накопителей информации (SSD);

– В OLTP или других систем реального времени SSD на каждом отдельном сервере часто используются в качестве кэшей для ускорения перемещения данных между сервером и централизованным хранилищем [24].

Данные сначала считываются или записываются на локальный SSD, затем в систему хранения данных. Транзакции данных на локальный SSD могут быть завершены очень быстро, прежде чем они автоматически передаются во внешнюю систему хранения.

Для абсолютной наименьшей задержки, а также стойкости при поддержке типичных нагрузок 24/7 в системах OLTP, SSD корпоративного класса PCIe или NVMe, такие как Samsung PM1725 или PM963, могут быть наиболее подходящими для установки на серверах. Эти твердотельные накопители не только обеспечивают скорость поддержки систем реального времени, но и потребляют меньше энергии и выделяют меньше тепла, чем старые твердотельные накопители или жесткие диски, что снижает общие эксплуатационные расходы и обеспечивает бесперебойную работу транзакций.

В работе [34] представлен подробный анализ возможностей оптимизации OLTP-систем за счет применения SSD.

– использование хранимых процедур при разработке программного обеспечения на СБД и другие рекомендации, предлагаемые, как правило, вендорами промышленных СУБД.

Хранимые процедуры являются программными объектами СУБД.

Хранимые процедуры представляют собой пакет SQL-операторов, которые могут быть выполнены несколькими способами. Большинство основных СУБД поддерживают хранимые процедуры.

Преимущества использования хранимых процедур являются:

- возможность использование модульного программирования;
- обеспечение высокой производительности выполнения кода;
- уменьшение сетевого трафика;
- возможность использования в качестве механизма безопасности.

Так, специалисты корпорации Oracle предлагают следующие рекомендации для повышения эффективности OLTP-системы [10]:

1. Используйте хранимые процедуры СБД при разработке ПО.
2. Возвращайте данные без избыточной информации.
3. Старайтесь уменьшить количество циклов между клиентом и СБД.
4. В больших единицах работы рассмотрите разбиение на более мелкие единицы, чтобы было много очень быстрых транзакций, а не одной большой транзакции. Крупные транзакции удерживают блокировки на более длительные периоды, что способствует проблемам с производительностью.
5. При большом количестве транзакций, способствующих избыточному обмену данными и сетевому трафику, рассмотрите возможность их пакетирования, чтобы между клиентом и СБД было меньше циклов.
6. Не используйте транзакцию для просмотра данных.
7. Рассмотрите возможность использования более низкого уровня изоляции транзакции, если это необходимо.

8. Не используйте неявные транзакции.

Как показывает практика, наилучшие результаты удается достичь при применении комбинации указанных методов и рекомендаций.

Следует также отметить, что в последнее время для повышения эффективности систем обработки больших массивов данных применяются технологии NoSQL, NewSQL и In-memory.

2.2 Архитектура распределенных баз данных

Распределенная база данных (Distributed Database) – единая база данных, объекты которой (таблицы, представления, столбцы и файлы) находятся в более чем одной системе в компьютерной сети и могут быть доступны или обновлены из любой системы в компьютерной сети [1].

Система управления распределенной БД (Distributed Database Management System) – СУБД, которая позволяет конечным пользователям или программистам приложений просматривать коллекцию физически отдельных баз данных как один логический образ единой системы. Концепция, которая является наиболее фундаментальной для распределенной СУБД, – это прозрачность местоположения, то есть пользователь не должен осознавать фактическое местоположение данных.

Шардинг (Sharding) – стратегия масштабирования приложений. В рамках шардинга информация из общей базы данных делится на блоки и распределяется по разным серверам

Партиционирование (Partitioning) – это часть шардинга, процесс разделения базы данных, перед выносом на отдельные сервера.

Физически распределенная база данных – это база данных, которая состоит из двух или более файлов, расположенных на разных сайтах либо в одной сети, либо в совершенно разных сетях. Части базы данных хранятся в нескольких физических местах, а обработка распределяется между несколькими узлами базы данных.

Распределенные базы данных в коллекции логически взаимосвязаны друг с другом и часто представляют собой одну логическую базу данных (рисунок 2.1).

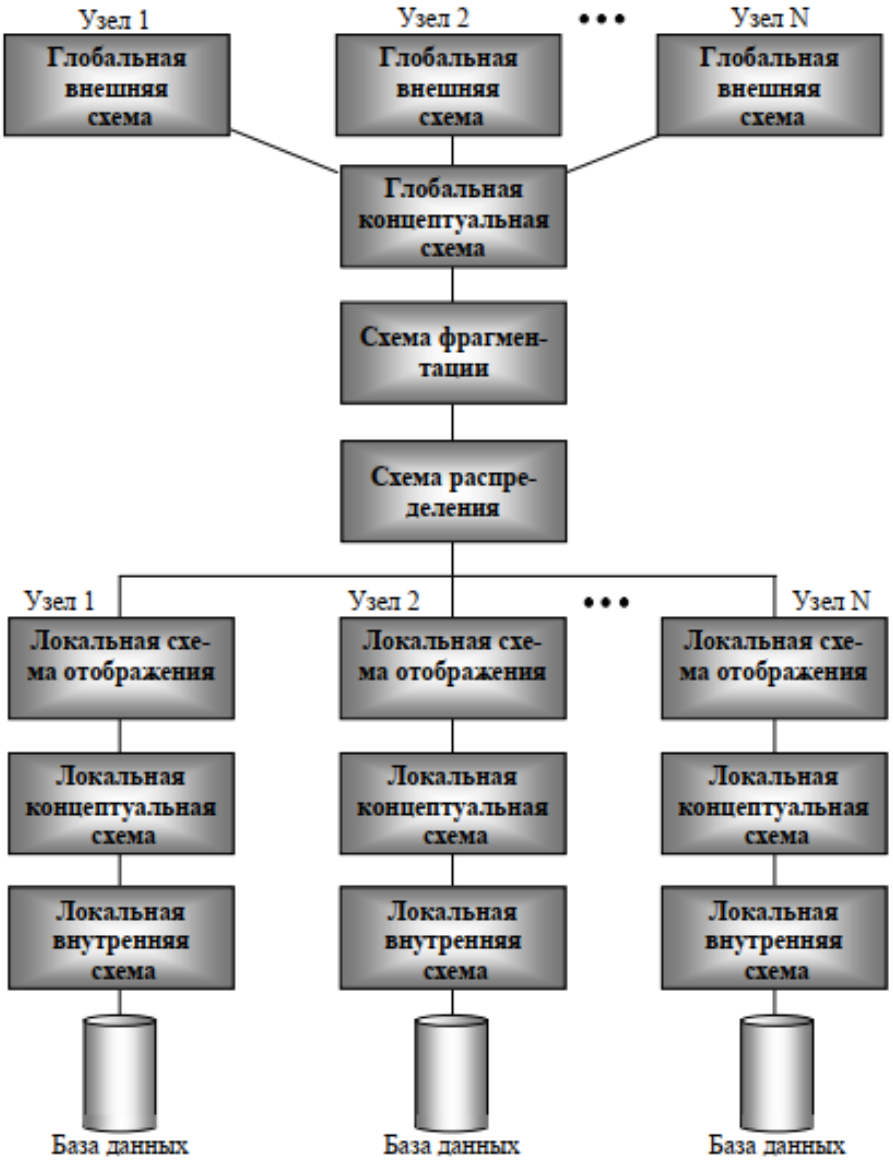


Рисунок 2.1 – Архитектура системы распределенной БД

Структурная схема распределенной СУБД представлена на рисунке 2.2

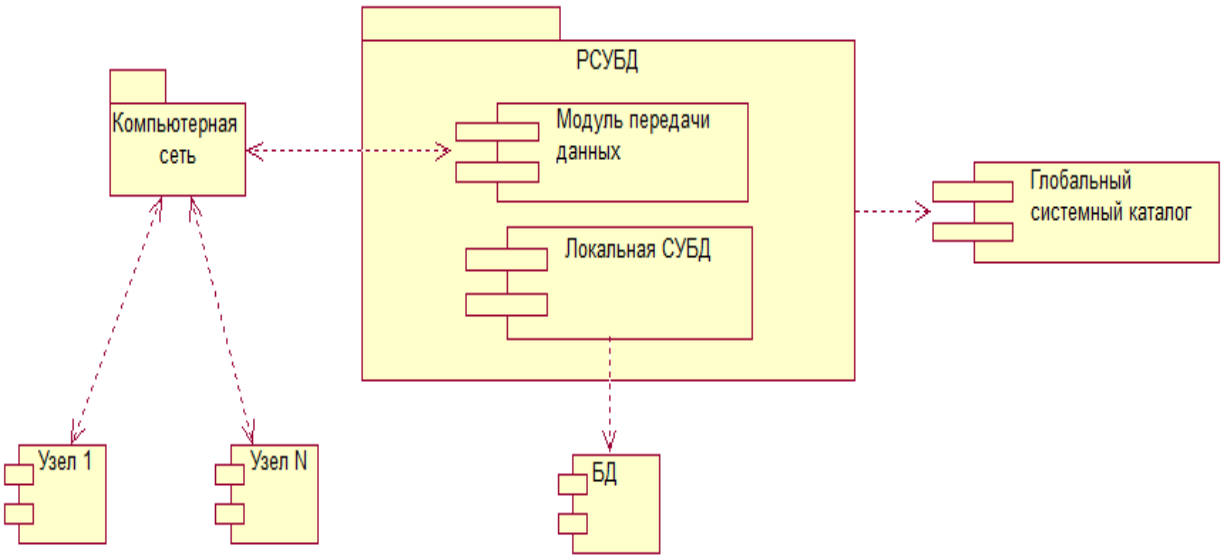


Рис. 2.2. – Диаграмма компонентов распределенной СУБД

В распределенных базах данных данные физически хранятся на нескольких сайтах и управляются независимо. Процессоры на каждом сайте соединены сетью, и у них нет многопроцессорной конфигурации.

Следует выделить следующие преимущества распределенных баз данных:

– распределенные базы данных способны к модульной разработке, что означает, что системы можно расширять, добавляя новые компьютеры и локальные данные на новый сайт и подключая их к распределенной системе без перерыва;

– когда происходят сбои в централизованных базах данных, система полностью останавливается. Однако в случае сбоя компонента в системах распределенных баз данных система будет продолжать работать с пониженной производительностью, пока ошибка не будет устранена;

– администраторы могут снизить расходы на связь для распределенных систем баз данных, если данные расположены рядом с тем местом, где они используются чаще всего. Это невозможно в централизованных системах.

Распространенным заблуждением является то, что распределенная база

данных является слабо связанной файловой системой. На самом деле все гораздо сложнее. Распределенные базы данных включают обработку транзакций, но не являются синонимами систем обработки транзакций.

В целом, распределенные базы данных имеют следующие особенности:

- независимость от расположения;
- распределенная обработка запросов;
- распределенное управление транзакциями;
- аппаратная независимость;
- независимость от операционной системы;
- независимый от особенностей компьютерной сети;
- прозрачность транзакций;
- независимость от СУБД.

По особенностям архитектуры распределенные базы данных могут быть однородными или разнородными.

В гомогенной (однородной) распределенной системе баз данных все физические местоположения имеют одинаковое базовое оборудование и работают с одинаковыми операционными системами и приложениями баз данных.

Гомогенные системы распределенных баз данных представляются пользователю как единая система, и их гораздо проще создавать и управлять ими. Чтобы система распределенных баз данных была гомогенной, структуры данных в каждом местоположении должны быть либо идентичными, либо совместимыми.

Приложение базы данных, используемое в каждом месте, также должно быть идентичным или совместимым.

В гетерогенной распределенной базе данных (БД) аппаратное обеспечение, операционные системы или приложения базы данных могут быть разными в каждом месте. Разные сайты могут использовать разные схемы и программное обеспечение, хотя различие в схеме может затруднить

обработку запросов и транзакций.

Разные узлы могут иметь разное оборудование, программное обеспечение и структуру данных, или они могут находиться в местах, которые несовместимы. Пользователи в одном месте могут читать данные в другом месте, но не могут загружать или изменять их.

Гетерогенные распределенные базы данных часто сложны в использовании, что делает их экономически невыгодными для многих предприятий.

Следует отметить, что технология распределенных БД обеспечивает масштабирование OLTP-системы, но повышает ее эффективность.

2.3 Технология NoSQL

Технология NoSQL (not only SQL, «не только SQL») предоставляет механизм для хранения и извлечения данных, которые моделируются другими способами, отличными от табличных отношений, используемых в реляционных базах данных.

Ключевые особенности технологии NoSQL:

– способность горизонтально масштабировать пропускную способность «простых операций» на многих серверах;

– возможность реплицировать и распределять (разбивать) данные по многим серверам;

– простой интерфейс или протокол уровня вызова (в отличие от SQL);

– более слабая модель параллелизма, чем транзакции ACID большинства реляционных (SQL) систем БД;

– эффективное использование распределенных индексов и оперативной памяти для хранения данных;

– возможность динамически добавлять новые атрибуты в записи данных.

В отличие от классических реляционных СУБД, поддерживающих

механизм управления транзакциями ACID, системы NoSQL поддерживают альтернативный механизм BASE:

– Basically Available (базовая доступность – каждый запрос гарантированно завершается (успешно или безуспешно));

– Soft state (гибкое состояние – состояние системы может изменяться со временем, даже без ввода новых данных, для достижения согласования данных);

– Eventually consistent (согласованность в конечном счёте – данные могут быть некоторое время не согласованы, но приходят к согласованию через некоторое время).

Представленный набор свойств NoSQL был выведен Э. Брюэром на основе предложенной им теоремы CAP, согласно которой распределенная система может иметь только два из трех следующих свойств [19]:

– Consistency (согласованность данных);

– Availability (доступность);

– Partition-tolerance (устойчивость к разделению).

По мнению специалистов, базы данных NoSQL по-прежнему выгодны для корпораций из-за недорогих стандартных серверов и высокой горизонтальной масштабируемости данных в среде облачных вычислений.

Базы данных NoSQL обеспечивают высокопроизводительные операции чтения и записи, поддерживая более слабую модель согласованности, в отличие от модели ACID, отказоустойчивости и балансировки нагрузки, а также со схемами, гибкими для моделирования данных.

Большинство баз данных NoSQL – это облачные базы данных, разработанные для обеспечения высокой производительности в облачных вычислительных средах с невероятной скоростью, обеспечивая производительность и доступность по согласованности.

Это помогает корпорациям экономить на издержках, связанных с производительностью, благодаря совместному использованию многопользовательской среды облачных вычислений и доступу к

распределенным данным для нескольких клиентов.

К СУБД, построенных на основе технологии NoSQL, относятся MongoDB, Apache Cassandra, OrientDB и др.

В качестве примера рассмотрим основанную на NoSQL технологию Hive [23].

Hive – это:

– инфраструктура хранилища данных, построенная на основе Hadoop для обобщения данных, построения запросов и анализа;

– ETL;

– структурирование данных;

– доступ к различным хранилищам;

– выполнение запросов через приложения Yarn (MapReduce, Tez, Spark).

Ключевые принципы построения Hive:

– SQL-знакомый язык;

– расширяемость – типы, функции, форматы, скрипты;

– масштабируемость.

Архитектура и компоненты технологии Hive представлены на рисунке 2.3.

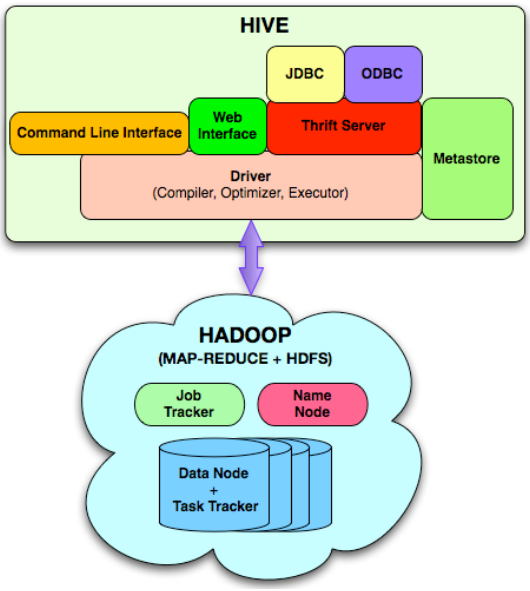


Рисунок 2.3 – Архитектура и компоненты технологии Hive

Вместе с тем, по мнению специалистов решения на основе технологии Hive не могут обеспечить эффективную транзакционную обработку больших массивов информации, так не поддерживают на требуемом уровне механизм ACID.

2.4 Технология NewSQL

NewSQL – это масштабируемые системы управления реляционными базами данных для оперативной обработки транзакций, которые обеспечивают масштабируемую производительность систем NoSQL для рабочих нагрузок чтения-записи, а также поддерживают свойства ACID систем транзакционных баз данных [37].

Эти системы преодолевают обычные ограничения масштабируемости, производительности и доступности СУБД и обеспечивают возможность частичного или полного запроса SQL.

NewSQL использует функции стиля NoSQL, такие как ориентированное на столбцы хранение данных и распределенные архитектуры, или с помощью таких технологий, как обработка в памяти, расширенные функции симметричной многопроцессорной обработки (SMP) или массивно-параллельной обработки (MPP), и интегрирует компоненты NoSQL или Search, разработанные для решения задач, связанных с объемом, разнообразием, скоростью и изменчивостью больших данных.

Следовательно, NewSQL следует рассматривать как альтернативу NoSQL и традиционной СУБД для новых приложений OLTP.

Техническими характеристиками СУБД класса NewSQL являются:

- язык SQL как способ взаимодействия между СУБД и приложением;
- поддержка транзакций ACID;
- неблокирующее управление параллелизмом, так что чтение и запись не вызывают конфликта друг с другом;
- архитектура, обеспечивающая более высокую производительность

на узел обработки;

– масштабируемая архитектура с распределенной памятью и способностью функционировать в кластере с большим количеством узлов;

– высокая производительность транзакционной обработки: NewSQL примерно в 50 раз быстрее традиционных СУБД для OLTP.

Категоризация NewSQL основана на существенных аспектах реализаций, принятых поставщиками.

Это:

– новые системы, созданные с нуля (базы данных новых архитектур);

– повторное внедрение той же шардовой инфраструктуры, разработанной другими вендорами (Middleware);

– база данных как услуга от поставщиков облачных вычислений, которые также основаны на новых архитектурах.

Архитектуры СУБД NewSQL основаны на новой кодовой базе для достижения масштабируемости и повышения производительности.

СУБД этой категории основаны на распределенных архитектурах, которые работают на ресурсах без общего доступа и содержат компоненты для поддержки многоузлового управления параллелизмом, отказоустойчивости посредством репликации, управления потоком и распределенной обработки запросов.

Репликация базы данных по кластерам позволяет легко внедрять новые машины в уже запущенной системе, которая импортирует характеристику масштабируемости. Это было невозможно реализовать в традиционных СУБД.

Все части системы могут быть оптимизированы для многоузловой среды для СУБД, построенной на распределенной среде.

Например, большинство СУБД NewSQL могут отправлять данные внутри запроса напрямую между узлами, не перенаправляя их в центральное местоположение.

Эти СУБД также управляют своим собственным основным хранилищем,

как в оперативной памяти, так и на диске, чтобы повысить производительность.

На рисунке 2.4 представлен пример архитектуры OLTP-системы на основе технологии NewSQL [18].

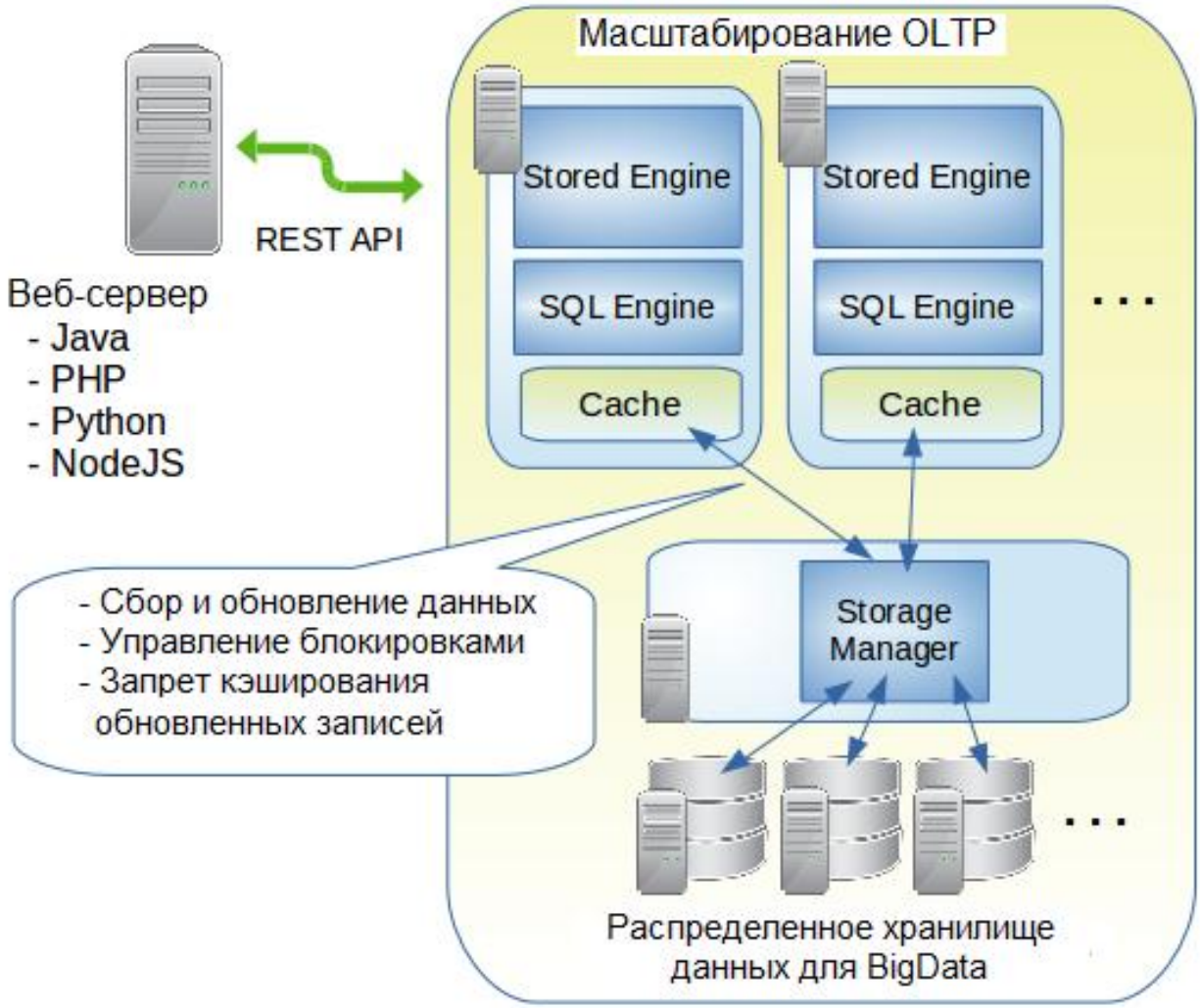


Рисунок 2.4 – Пример архитектуры OLTP-системы на основе технологии NewSQL

СУБД отвечает за распределение базы данных по своим ресурсам с помощью специального механизма, вместо того чтобы полагаться на готовую распределенную файловую систему или структуру хранения.

Это позволяет СУБД отправлять запрос к данным, что приводит к значительно меньшему сетевому трафику, чем передача данных для вычислений. Например, передача данных для вычислений приносит кортежи

вместе с индексами и материализованными представлениями, что увеличивает сетевой трафик. Это лучший вариант для тех, кто хочет разрабатывать высоко масштабируемые и эффективные системы OLTP.

Принятие этих новых решений требует миграции данных и некоторых изменений в коде. Это также означает, что организация может потерять доступ к существующим инструментам администрирования и отчетности.

Решения в этой категории могут быть только программными, такими как VoltDB, NuoDB и Drizzle, или поддерживаться в качестве устройства, такого как Clustrix и Translattice.

Прозрачное шардинговое промежуточное ПО позволяет организации разделить базу данных на несколько сегментов, которые хранятся в кластере СУБД с одним узлом, для повышения масштабируемости.

Одним из примеров прозрачного шардинга является ScaleBase.

Компонент централизованного промежуточного программного обеспечения направляет запросы, координирует транзакции, а также управляет управлением данными, репликацией и распределением по узлам.

Каждый узел СУБД содержит промежуточный слой, который связывается с промежуточным программным обеспечением и отвечает за выполнение запросов от имени промежуточного программного обеспечения в своем локальном экземпляре СУБД и выдачу результатов, следовательно, представляя единую логическую базу данных приложению без необходимости изменения базовой СУБД.

Прозрачное промежуточное ПО для сегментирования позволяет повторно использовать существующие наборы навыков, а также экосистему, и избавляет от необходимости писать код или выполнять какую-либо миграцию данных. Однако такие системы все еще требуют традиционных СУБД на каждом узле, таких как MySQL, PostgreSQL и Oracle.

Эти СУБД основаны на дисковой архитектуре, которая не может использовать схему управления параллелизмом или диспетчер хранения, оптимизированный для ориентированной на память системы.

Продукты баз данных NewSQL как услуга (DBaaS) предлагаются поставщиками облачных вычислений. Как правило, это сервис, отвечающий за поддержание физической конфигурации базы данных, то есть репликации, резервного копирования и настройки системы (изменение размера пула буферов).

Рассмотрим характеристики наиболее популярны СУБД класса NewSQL.

2.5 Обзор и анализ СУБД класса NewSQL

СУБД NuoDB можно классифицировать как NewSQL базу данных, которая сохраняет характеристики традиционных SQL баз данных, а также включает функции поддержки масштабирования в средах облачных вычислений [14].

Основные характеристики СУБД NuoDB приведены в таблице 2.1.

Таблица 2.1 – Характеристики СУБД NuoDB

Характеристика	Описание
Первичная модель БД	Реляционная
Вендор	NuoDB, Inc.
Лицензия	Коммерческая
Язык разработки	C++
Серверные ОС	Linux OS X Windows
Схема данных	+
Типизация	+
Поддержка XML	-
Вторичные индексы	+
Поддержка SQL	+

Продолжение таблицы 2.1

API и др. методы доступа	ADO.NET JDBC ODBC
Поддерживаемые языки программирования	.Net C C++ Go Java JavaScript JavaScript (Node.js) PHP Python Ruby
Скрипты на стороне сервера	Java, SQL
Триггеры	+
Методы партиционирования	Данные динамически сохраняются/ кэшируются на узлах при записи-чтении
Репликация	+
Концепция согласованности	Немедленная согласованность
Внешние ключи	+
Концепция транзакции	ACID
Параллелизм	+
Надежность	+
In-memory технология	+
Управление пользователями	Standard SQL roles/ privileges, Administrative Users Стандартные SQL-роли и привилегии, администрирование пользователями
Область использования	Оперативные / OLTP-базы данных рабочих нагрузок в центрах обработки данных или публичных облаках

Прикладные программы взаимодействуют с NuoDB с операторами SQL, как и в реляционной базе данных, а также поддерживает модель ACID для надежности транзакций.

Архитектура СУБД NuoDB представлена на рисунке 2.5.

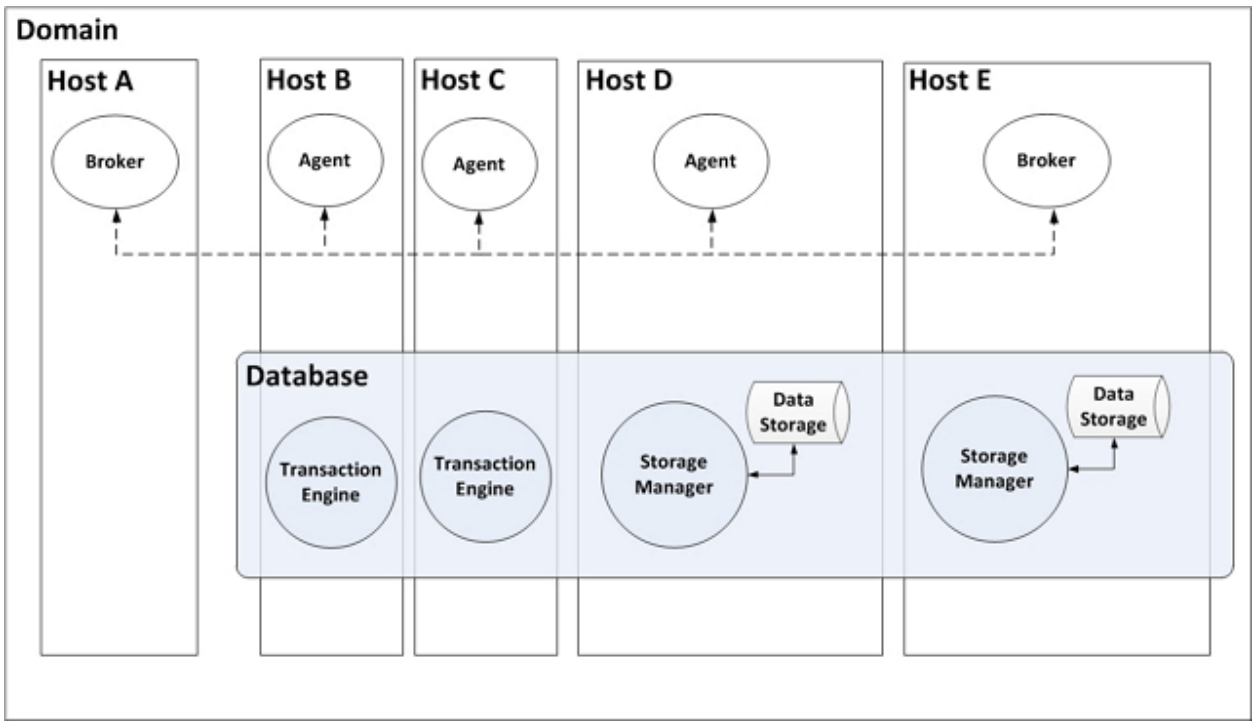


Рисунок 2.5 – Архитектура СУБД NuoDB

Архитектура NuoDB включает три уровня: уровень управления, уровень SQL и уровень данных. Уровень управления состоит из агента, контролирующего процессы NuoDB на конкретном компьютере. Он их запускает и останавливает, собирает статистику с транзакционных модулей и модулей хранения данных.

Отдельные агенты могут выступать в качестве брокеров (посредников).

Брокеры сначала устанавливают связь с клиентом, а потом подключают клиента к транзакционному движку. Клиент может напрямую взаимодействовать с транзакционными движками.

Кроме того, уровень управления в NuoDB предоставляет командную строку и веб-инструмент управления базами данных, а также загрузчик для экспорта и импорта данных.

На уровне SQL транзакционные движки обеспечивают доступ к единственной БД. Они проводят синтаксический анализ, компилируют, оптимизируют и выполняют SQL-запросы от имени клиентов.

На уровне данных блок управления хранилищем обеспечивает хранение данных. Он использует пары "ключ-значение" для хранения информации, но

может использовать и более продвинутые хранилища (например, HDFS).

В случае минимальной конфигурации, можно запускать каждый компонент (брокер, транзакционный движок и блок управления памятью) на одной машине. NuoDB можно с легкостью масштабировать горизонтально и сделать избыточной, добавив несколько брокеров, транзакционных движков и блоков управления хранилищем. Более сложные сценарии предполагают запуск NuoDB в облаке AWS или в корпоративных дата-центрах, чтобы обеспечить географическую избыточность.

СУБД VoltDB – это In-Memory, многораздельная, однопоточная, распределенная, совместимая с ACID СУБД со встроенным машинным обучением. In-memory обеспечивает гораздо более быстрый доступ, чем устаревшие дисковые системы [15].

Характеристики СУБД VoltDB представлены в таблице 2.2.

Таблица 2.2 – Характеристики СУБД VoltDB

Характеристики	Описание
Первичная модель БД	Реляционная
Вендор	VoltDB Inc.
Лицензия	Коммерческая
Язык разработки	Java, C++
Серверные ОС	Linux OS X
Схема данных	+
Типизация	+
Вторичные индексы	+
Поддержка SQL	+
API и др. методы доступа	Java API JDBC RESTful HTTP/JSON API

Продолжение таблицы 2.2

Поддерживаемые языки программирования	C# C++ Erlang Go Java JavaScript PHP Python
Скрипты на стороне сервера	Java
Триггеры	-
Методы партиционирования	Шардинг
Методы репликации	репликация Master-master репликация Master-slave
Внешние ключи	-
Концепция транзакции	ACID
Параллелизм	+
Надежность	+
Управление пользователями	Пользователи и роли с доступом к хранимым процедурам

Архитектура СУБД VoltDB представлена на рисунке 2.6.

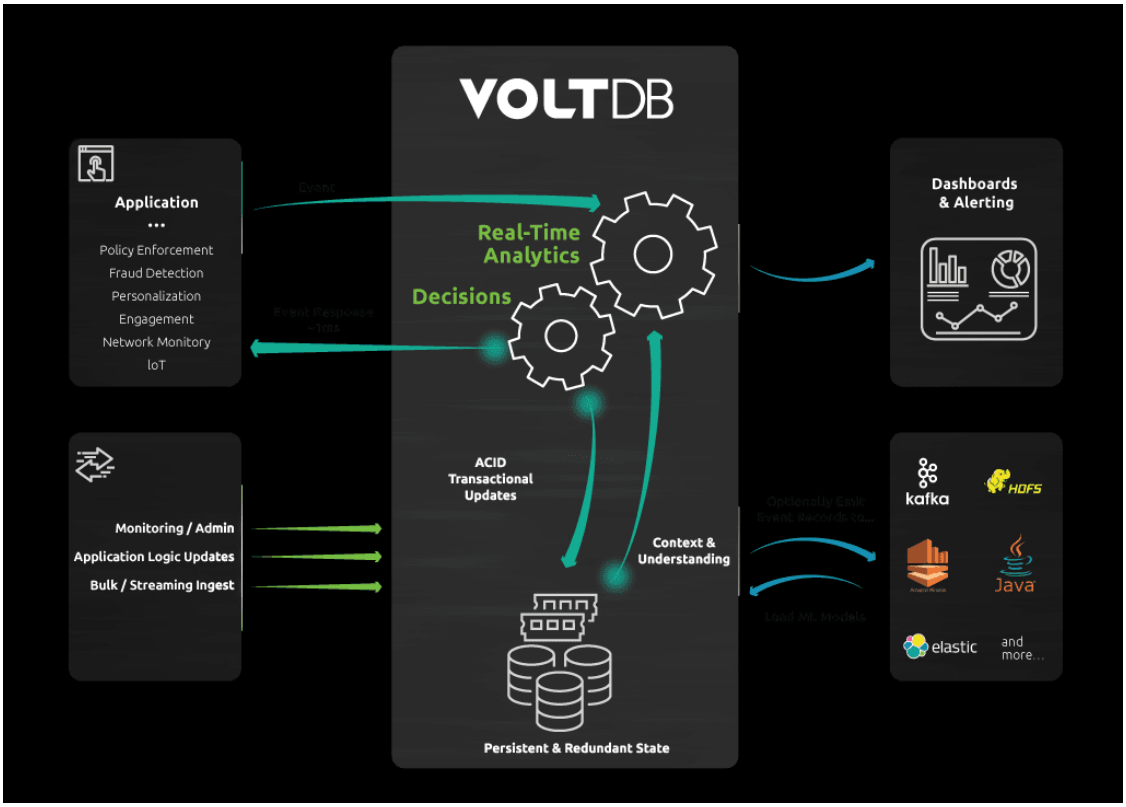


Рисунок 2.6 – Архитектура СУБД VoltDB

VoltDB поддерживает функцию VoltML (машинное обучение): пользователи могут создавать пользовательские функции VoltDB (UDF) из своих моделей PMML и использовать их в хранимых процедурах для оценки в реальном времени потоковых данных.

На рисунке 2.7 представлена архитектура принятия решений в режиме реального времени в СУБД VoltDB.

Рисунок 2.7 – Архитектура принятия решений в режиме реального времени в СУБД VoltDB

СУБД VoltDB пользуется мировым доверием у поставщиков телекоммуникационных программных решений и используется для поддержки их критически важных приложений, развернутых более чем у 100 операторов по всему миру.

VoltDB выбирают за ее производительность и возможности для решения не только текущих задач, но и для поддержки быстрого развития систем в конкретной отрасли.

2.6 Технология In-memory

Технология In-memory – это технология выполнения компьютерных вычислений полностью в памяти компьютера (например, в ОЗУ). Этот термин обычно подразумевает крупномасштабные сложные вычисления, которые требуют специального системного программного обеспечения для выполнения вычислений на компьютерах, работающих вместе в кластере [26].

В качестве кластера компьютеры объединяют свою оперативную память, поэтому расчет по существу выполняется между компьютерами и использует общее пространство ОЗУ всех компьютеров вместе.

Обработка в памяти устраняет все медленные обращения к данным и полагается исключительно на данные, хранящиеся в ОЗУ. Общая производительность обработки не снижается из-за задержки, обычно наблюдаемой при доступе к жестким дискам или твердотельным накопителям. Программное обеспечение, работающее на одном или нескольких компьютерах, управляет обработкой, а также данными в памяти, а в случае нескольких компьютеров программное обеспечение делит обработку на более мелкие задачи, которые распределяются на каждый компьютер для параллельной работы.

Обработка в памяти часто выполняется с помощью технологии, известной как сетки данных в памяти (IMDG).

Im-memory база данных (IМБД) – это компьютерная система, которая хранит и извлекает записи данных, которые находятся в основной памяти компьютера, например, в оперативной памяти (ОЗУ).

Имея данные в ОЗУ, IМБД имеют преимущество в скорости по сравнению с традиционными дисковыми базами данных, которые влекут за собой задержки доступа, поскольку носители данных, такие как жесткие диски и твердотельные накопители (SSD), имеют значительно более медленное время доступа, чем ОЗУ.

Это означает, что IМБД полезны, когда критически важно быстрое

чтение и запись данных.

Обработка в памяти сегодня чрезвычайно популярна из-за ее огромного преимущества в производительности по сравнению с методами обработки, требующими чтения и записи на более медленные носители.

Чтение и запись на более медленный носитель часто приводит к узкому месту доступа к данным (явление, известное как «ограничение ввода-вывода», где «ввод-вывод» относится к вводу / выводу). Из-за своей скорости обработка в памяти часто описывается как обработка в реальном времени или, по крайней мере, применяется к вариантам использования в реальном времени. А поскольку цены на оперативную память продолжают падать, появляется больше возможностей обработки в оперативной памяти, поскольку они становятся более экономически привлекательными.

Архитектура In-memory OLTP-системы с распределенной памятью представлена на рисунке 2.8.

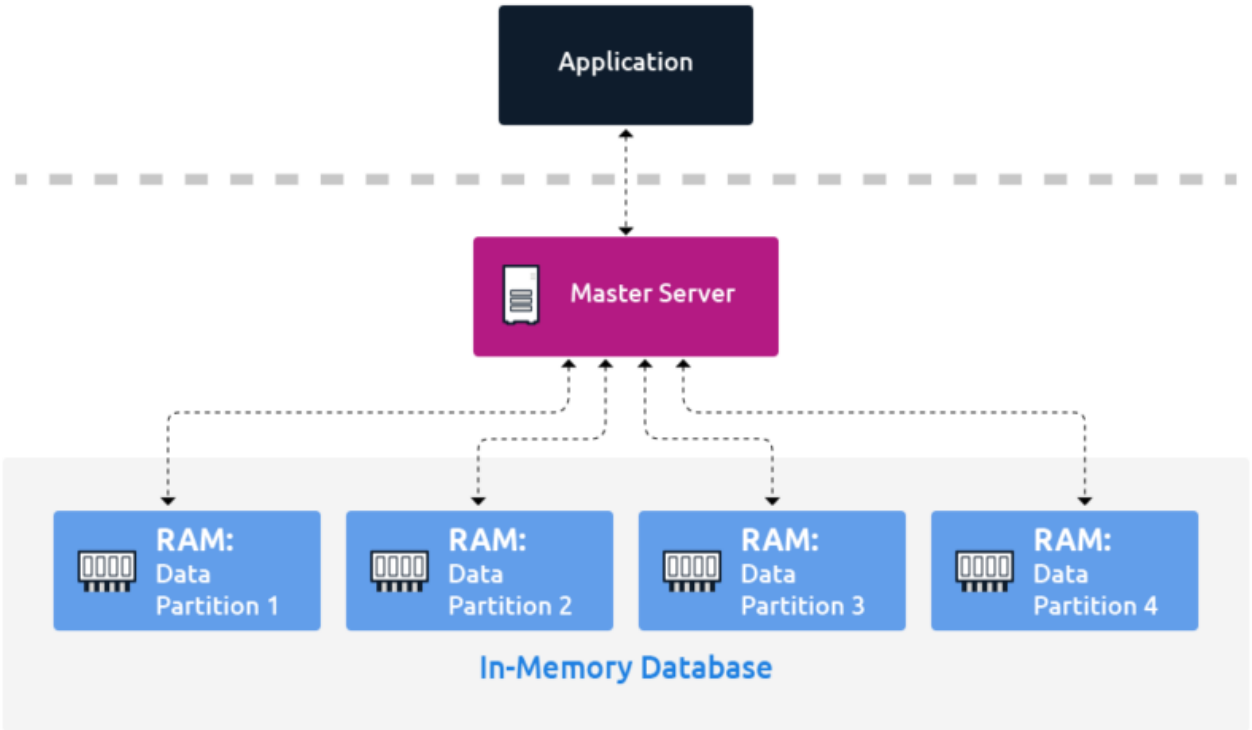


Рисунок 2.8 – Архитектура In-memory OLTP-системы с распределенной памятью

Следует отметить, что упомянутые здесь функции не являются исключительными ноу-хау систем БД NewSQL, и не обязательно, чтобы все они считались продуктом NewSQL.

Многие существующие реляционные СУБД поддерживают такой механизм, как In-Memory (например, поздние версии MS SQL Server и Oracle Database).

Как было отмечено выше, механизм In-Memory может значительно повысить производительность обработки транзакций, приема и загрузки данных, а также сценариев переходных данных.

Рассмотрим реализацию данного механизма в MS SQL Server,

Архитектура MS SQL Server с In-Memory OLTP представлена на рисунке 2.9 [27].

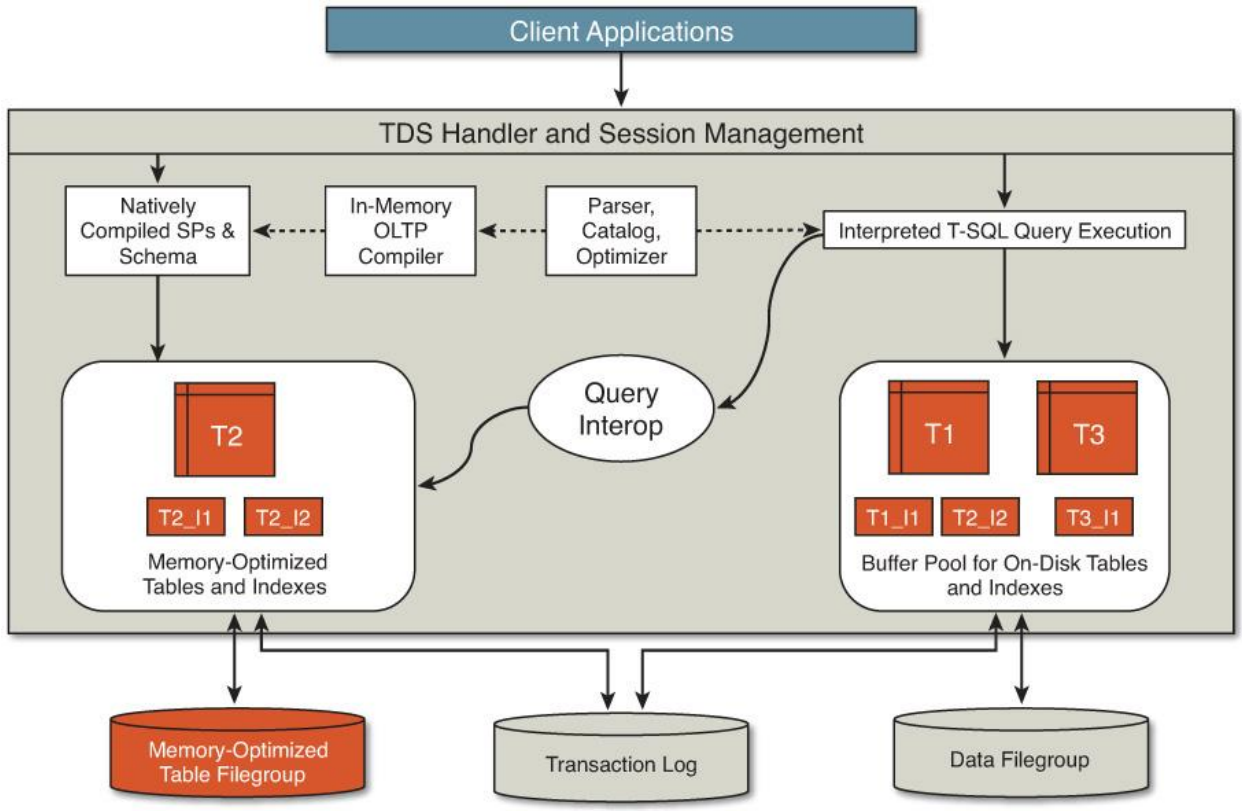


Рисунок 2.9 – Архитектура MS SQL Server с In-Memory OLTP

Технология In-Memory OLTP, также известная как «Hekaton» и «In-Memory оптимизация», является новейшей технологией обработки в памяти

Microsoft.

Механизм In-Memory оптимизирован для оперативной обработки транзакций. Он интегрирован в ядро СУБД SQL Server и может использоваться точно так же, как и любой другой компонент СУБД.

Механизм In-Memory изначально поставлялся с SQL Server 2014 и в основном содержит две новые структуры данных: таблицы с оптимизированной памятью и хранимые процедуры с собственной компиляцией.

Оптимизированные для памяти таблицы хранят свои данные в памяти, используя несколько версий данных каждой строки. Этот метод характеризуется как «неблокирующее многоуровневое оптимистическое управление параллелизмом» и устраняет как блокировки, так и защелки, тем самым достигая значительных преимуществ производительности.

Основными функциями таблиц, оптимизированных для памяти, являются:

− строки в таблице считываются из памяти и записываются в нее;

− вся таблица находится в памяти;

– неблокирующее многоверсионное оптимистичное управление параллелизмом;

− опция долговечных и недолговечных данных;

– вторая копия сохраняется на диске для долговечности (если включена);

− данные в оптимизированных для памяти таблицах читаются только с диска во время восстановления базы данных;

− совместимость с дисковыми таблицами.

Нативно-скомпилированные хранимые процедуры – это объект SQL Server, который может обращаться только к оптимизированным для памяти структурам данных, таким как оптимизированные для памяти таблицы, переменные таблиц и т. д.

Основными функциями скомпилированной хранимой процедуры

являются:

– компиляция в собственный код (DLL) при создании (интерпретированные хранимые процедуры компилируются при первом выполнении);

– агрессивная оптимизация требует времени во время компиляции;

– возможность взаимодействовать только с таблицами, оптимизированными для памяти;

– вызов скомпилированной хранимой процедуре фактически является вызовом ее точки входа DLL.

Следует напомнить, что механизм In-Memory в SQL Server оптимизирован для обработки OLTP.

Это означает, что он лучше всего работает для определенных конкретных типов нагрузки.

Это не означает, однако, что, если он используется для других типов рабочей нагрузки, он не будет работать хорошо.

Вместе с тем, есть рекомендация от Microsoft, в которой указаны основные области рабочей нагрузки, для которых механизм In-Memory обеспечивает наибольшую эффективность.

Для выбора методологии разработки высокоэффективных эффективностью систем сбора и обработки больших массивов данных используем таблицу 2.3.

Для сравнительного анализа характеристик существующих методологии разработки высокоэффективных эффективностью систем сбора и обработки больших массивов данных используем таблицу 2.3 и введем критерии оценивания:

0 – полное несоответствие требованиям;

1 – значительное несоответствие требованиям;

2 – для соответствия требования необходима значительная доработка;

3 – для соответствия требования необходима незначительная доработка;

4 – незначительное несоответствие требованиям;

5 – полное соответствие требованиям.

Таблица 2.3 – Сравнительный анализ методологии разработки высокоэффективных эффективностью систем сбора и обработки больших массивов данных

Характеристика/Аналог (макс. – 5)	DDMS	NoSql	NewSQL	In-memory
Поддержка ACID	5	2	4	5
Обеспечение высокого уровня эффективности	3	3	4	5
Низкая стоимость владения	2	3	3	4
Итого	10	8	11	14

На основании представленного анализа выбираем в качестве методологии разработки высокоэффективных систем сбора и обработки больших массивов данных методологию In-Memory.

Однако более целесообразным представляется использование комплексного подхода, использующего лучшие мировые практики обеспечения высокой эффективности OLTP-систем для больших массивов данных.

Выводы к главе 2

1. Следует отметить, что в последнее время для повышения эффективности систем обработки больших массивов данных применяются технологии NoSQL, NewSQL и In-memory.

2. Технология In-memory – это технология выполнения компьютерных вычислений полностью в памяти компьютера (например, в ОЗУ).

3. На основании представленного анализа выбираем в качестве методологии комплексный подход, использующий наилучшие мировые практики обеспечения высокой эффективности OLTP-систем для больших массивов данных.

Глава 3 Разработка модели эффективной системы сбора и обработки больших массивов данных и оценка ее эффективности

3.1 Выбор методологии моделирования OLTP-системы для сбора и обработки больших массивов данных

Модель OLTP-системы представлена на двух уровнях – логическом и физическом.

Для разработки логической модели OLTP-системы используем технологии, основанные на применении языка визуального моделирования UML [13].

Унифицированный язык моделирования (UML) – это объединение лучших практик унифицированного моделирования, которые были созданы на протяжении многих лет при использовании языков моделирования.

UML позволяет нам представлять широко варьирующиеся аспекты программной системы (например, требования, структуры данных, потоки данных и информационные потоки) в единой структуре с использованием объектно-ориентированных концепций.

UML не привязан к конкретному инструменту разработки, конкретному языку программирования или конкретной целевой платформе, на которой должна быть разработана система.

UML также не предлагает процесс разработки программного обеспечения. Он фактически разделяет язык моделирования и метод моделирования. Последний может быть определен на уровне проекта или предпочтениями разработчика.

Тем не менее, языковые концепции UML поддерживают итеративный и инкрементальный процесс.

Использование в программном обеспечении UML может использоваться последовательно во всем процессе разработки программного обеспечения.

На всех этапах разработки одни и те же языковые концепции могут использоваться в одних и тех же обозначениях.

Таким образом, модель может быть усовершенствована.

В UML модель представляется графически в форме диаграмм.

Диаграмма дает представление о той части реальности, которая описывается моделью. Есть диаграммы, которые показывают, какие пользователи используют какие функции и диаграммы, которые показывают структуру системы, но без указания конкретной реализации.

Несмотря на то, что стандартный набора языка UML включает в себя более десятка видов диаграмм, его основу составляют три диаграммы, позволяющие описать основные аспекты проектируемой информационной системы: диаграмма вариантов использования, диаграмма классов и диаграмма последовательности.

3.2 Разработка логической модели OLTP-системы для сбора и обработки больших массивов данных

Для разработки логической модели эффективной OLTP-системы для сбора и обработки больших массивов данных используем диаграмму компонентов UML.

Диаграммы компонентов используются для визуализации организации и отношений между компонентами в системе. Эти диаграммы также используются для создания исполняемых систем [36].

Диаграммы компонентов различны по своей природе и поведению. Диаграммы компонентов используются для моделирования физических аспектов системы. Теперь вопрос в том, что это за физические аспекты? Физические аспекты – это такие элементы, как исполняемые файлы, библиотеки, файлы, документы и т.д., которые находятся в узле.

Диаграмма компонентов – это особый вид диаграммы в UML. Ее назначение также отличается от всех других диаграмм. Она не описывает функциональные возможности системы, а только компоненты, используемые для реализации этих функций.

Таким образом, с этой точки зрения диаграммы компонентов используются для визуализации физических компонентов в системе. Эти компоненты представляют собой библиотеки пакеты, файлы и т. д.

Диаграммы компонентов также можно описать как статическое представление системы. Статическая реализация представляет собой организацию компонентов в определенный момент.

Для разработки модели использован продукт Rational Rose [11].

На рисунке 3.1 представлена модель эффективной системы сбора и обработки больших массивов данных в виде диаграммы пакетов данных.

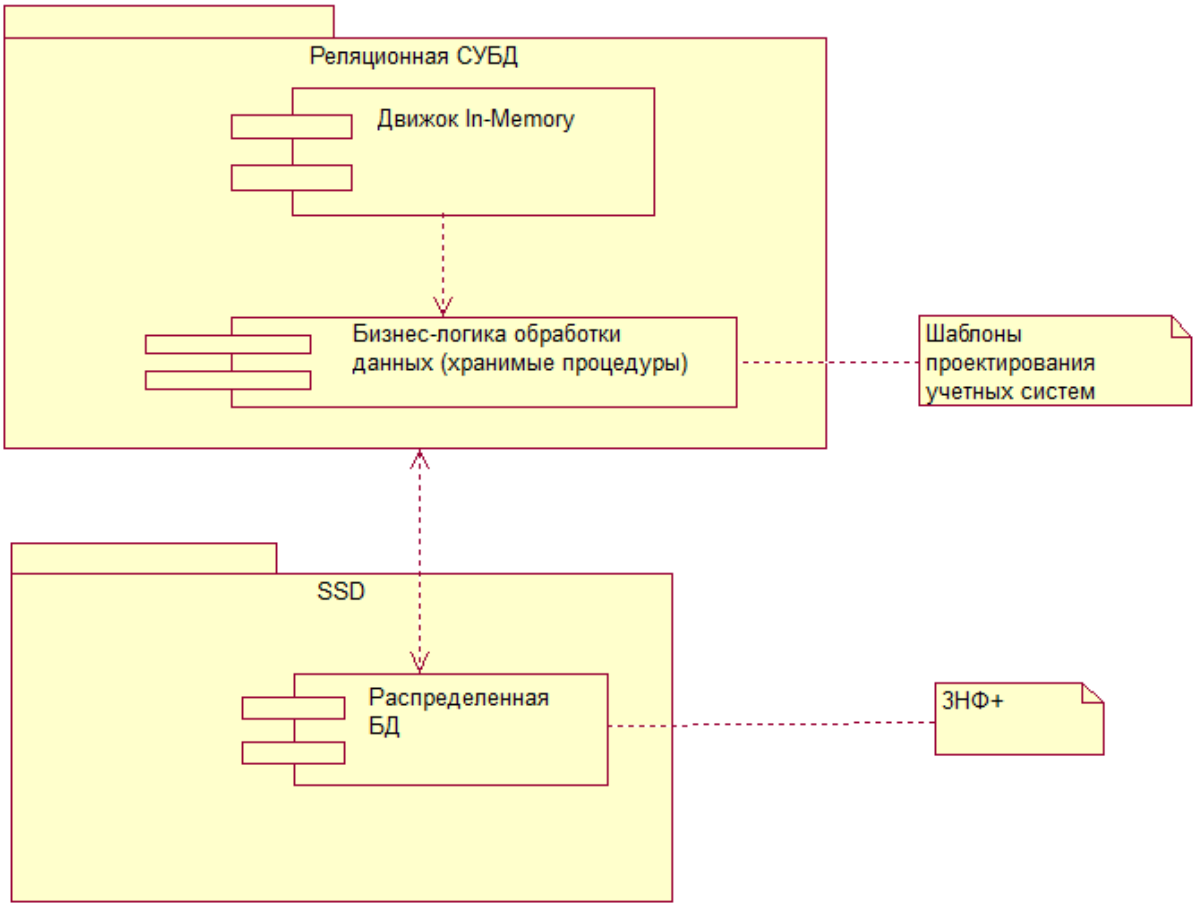


Рисунок 3.1 – Логическая модель системы сбора и обработки больших массивов данных

Рассмотрим компоненты предлагаемой модели.

OLTP-система построена на основе промышленной реляционной СУБД (РСУБД), поддерживающей механизм In-memory.

Управление данными осуществляется бизнес-логикой, размещенной на СБД с помощью программных объектов, разработанных на основе хранимых процедур.

Для сбора и хранения данных используется распределенное хранилище данных, размещенное на твердотельных (SSD) дисках.

В OLTP-системе используется сильная нормализация данных (3НФ+).

Предлагаемая модель разработана на основе комплексного подхода, использующего лучшие мировые практики обеспечения высокой эффективности OLTP-систем для больших массивов данных.

3.3 Разработка физической модели OLTP-системы для сбора и обработки больших массивов данных

Физическая модель представляет собой реализацию OLTP-системы для сбора и обработки больших массивов данных.

Технология In-Memory OLTP, также известная как «Hekaton» и «In-Memory оптимизация», является новейшей технологией обработки в памяти Microsoft.

На рисунке 3.2 представлена схема интеграции движка Hekaton в MS SQL Server [21].

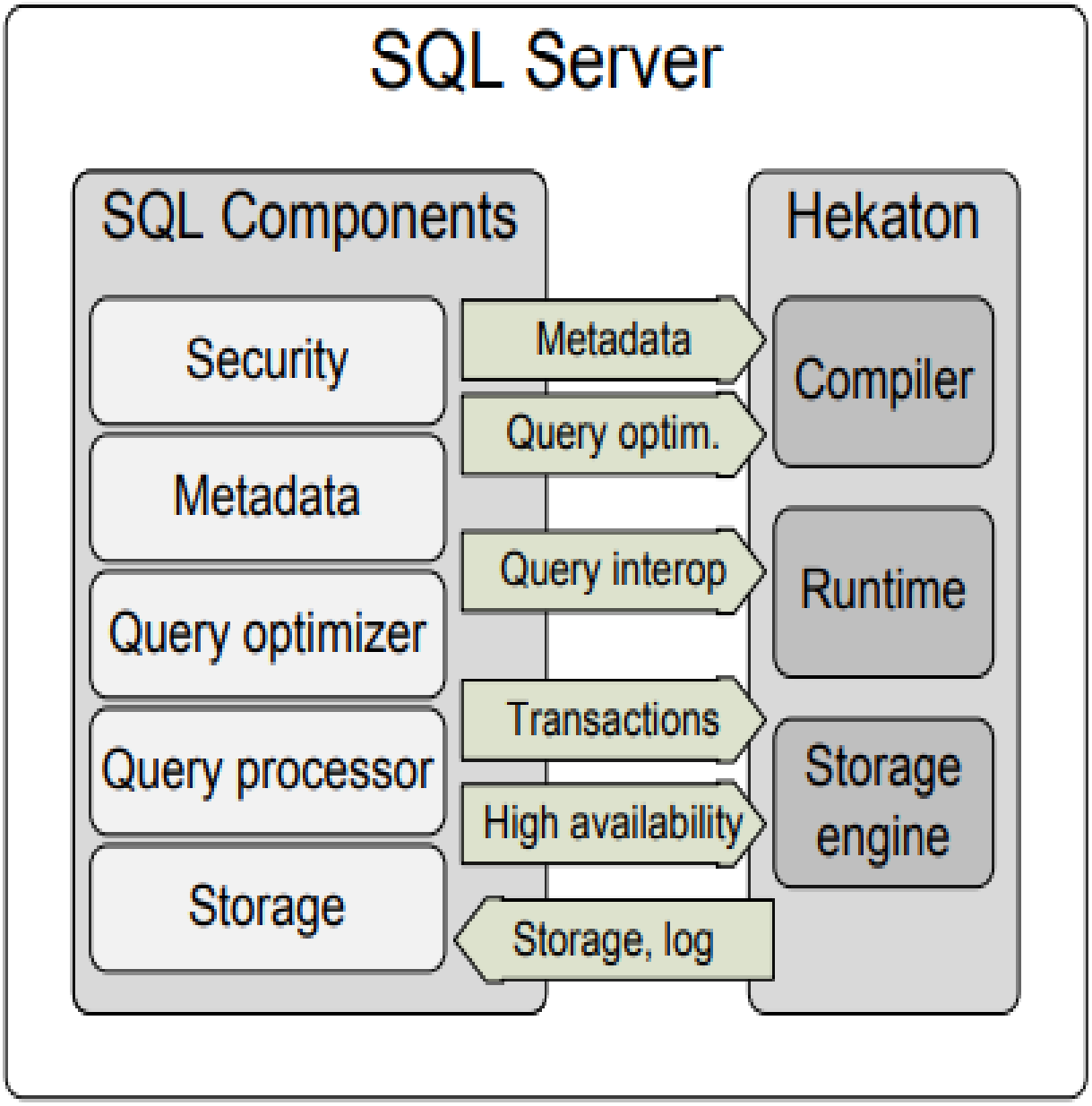


Рисунок 3.2 – Интеграция компонентов движка Hekaton в MS SQL Server

На рисунке 3.3 изображена структурная схема компилятора движка Hekaton.

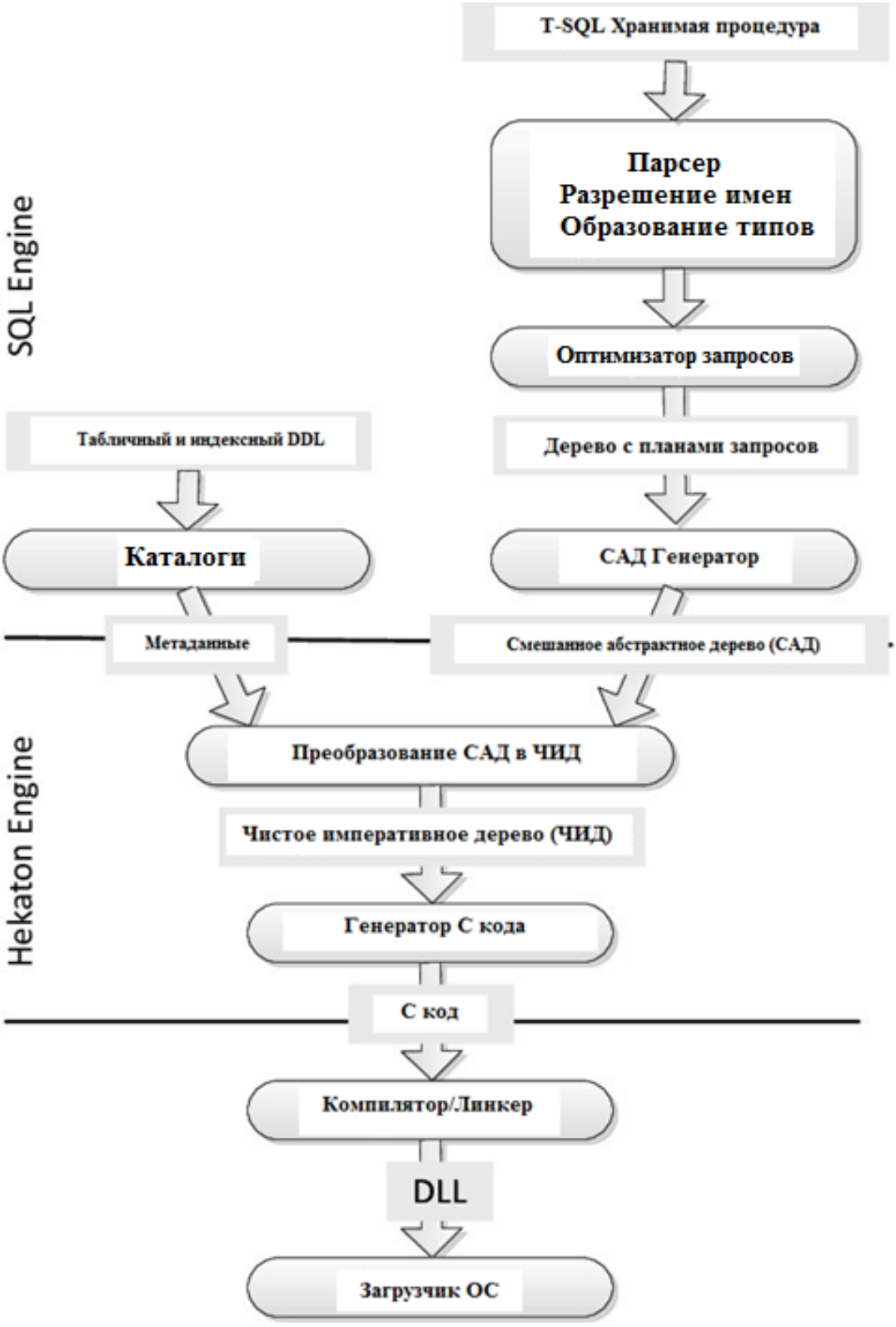


Рисунок 3.3 – Структурная схема компилятора движка Hekaton

Механизм In-Memory оптимизирован для оперативной обработки транзакций. Он интегрирован в ядро СУБД SQL Server и может использоваться точно так же, как и любой другой компонент СУБД.

Механизм In-Memory поставлялся, начиная с версии SQL Server 2014, и в основном содержит две новые структуры данных: таблицы с оптимизированной памятью и хранимые процедуры с собственной компиляцией.

Для представления физических компонентов OLTP-системы и их спецификации используем диаграмму развертывания UML.

В контексте UML диаграмма развертывания относится к семейству структурных диаграмм, поскольку она описывает аспект самой системы. В этом случае диаграмма развертывания описывает физическое развертывание информации, созданной программным обеспечением, на компонентах оборудования. Информация, которую генерирует программа, называется артефактом [20].

Диаграммы развертывания состоят из нескольких форм UML. Трехмерные блоки, известные как узлы, представляют собой основные программные или аппаратные элементы, или узлы в системе. Линии от узла к узлу указывают на отношения, а меньшие фигуры, содержащиеся в прямоугольниках, представляют развернутые программные артефакты.

Назначение диаграмм развертывания:

− показать, какие элементы программного обеспечения развертываются с помощью каких аппаратных элементов;

− показать работу аппаратного обеспечения;

− обеспечить представление о топологии аппаратной системы.

На рисунке 3.4 представлена диаграмма развертывания OLTP-системы, построенной на основе предлагаемой модели.

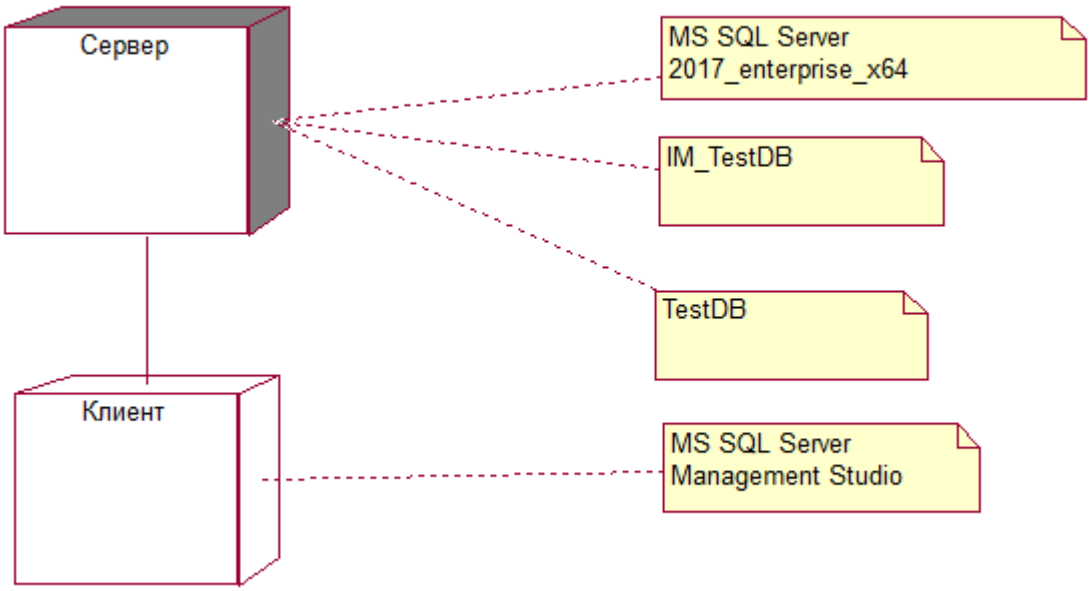


Рисунок 3.4 – Диаграмма развертывания OLTP-системы, построенной на основе предлагаемой модели

Физическая модель OLTP-системы реализована по двухзвенной модели «клиент-сервер».

В качестве сервера баз данных OLTP-системы использована версия промышленной СУБД MS SQL Server 2017.

В качестве клиента используется среда MS SQL Server Management Studio 2017.

По мнению вендора в СУБД SQL Server 2017 встроено все: лучшая в отрасли производительность в памяти, надежная безопасность, революционная расширенная аналитика в базе данных и гибкость, позволяющая запускать все ваши данные в любой среде с любыми данными. Все это с меньшими затратами благодаря непревзойденной совокупной стоимости владения [12].

СУБД выполняет запросы до 100 раз быстрее, чем диск, с хранением столбцов в памяти, а транзакции – до 30 раз быстрее с OLTP в памяти.

Используются уровни защиты, включая инновационные функции, такие как шифрование в состоянии покоя и в движении, и все это из наименее

уязвимой базы данных за последние семь лет.

Можно получить трансформирующую информацию с помощью до 1 миллиона прогнозов в секунду, используя встроенную интеграцию Python и R, и получите сквозную мобильную бизнес-аналитику на любой платформе.

Версии:

Enterprise – самый полный выпуск, включает все возможности SQL Server 2017, предназначен для крупных баз данных, которые требуют максимальной производительности, надежности, масштабируемости и доступности, а также имеют очень строгие требования по бизнес-аналитике. Конечно же, данная редакция самая дорогая.

Standard – самая распространенная редакция, включает ключевые возможности управления данными и бизнес-аналитики.

В отличие от выпуска Enterprise у Standard имеются ограничения, например:

– максимальное количество ядер, которое можно задействовать, это 24;

– максимальный объем данных в оптимизированной памяти на базу данных 32 ГБ;

– максимальный кэш сегмента Columnstore на экземпляр 32 ГБ;

– недоступна расширенная высокая доступность: группы доступности Always On (Always On Availability Groups), отработка отказа нескольких баз данных, доступные для чтения вторичные реплики;

– недоступно прозрачное шифрование данных;

– недоступны мобильные отчеты (Mobile Reports);

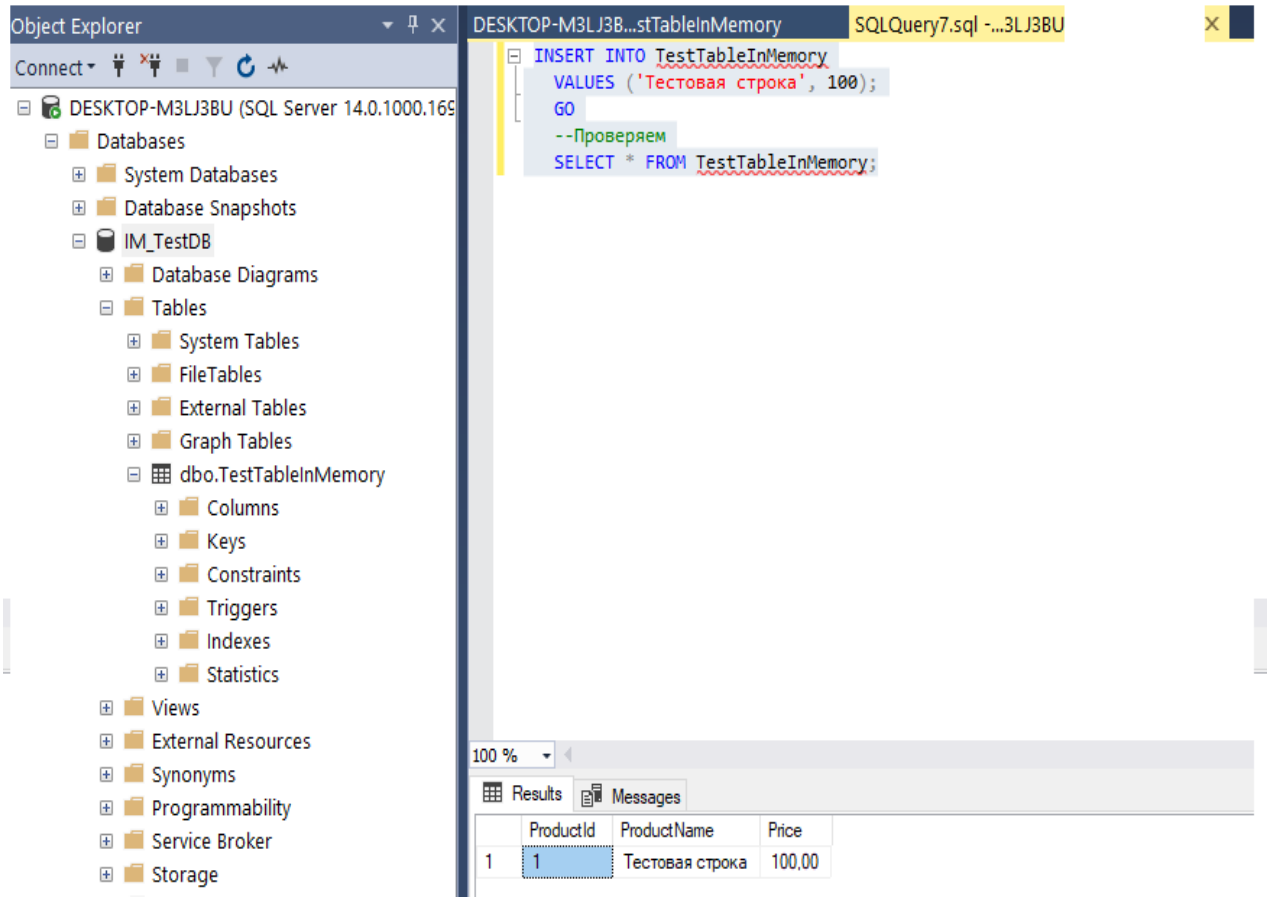
– недоступна расширенная интеграция машинного обучения: полный параллелизм аналитики R и Python, возможность запуска на графических процессорах;

– и некоторые другие ограничения.

Если отсутствие вышеперечисленного функционала Вам некритично, то можете использовать ее, т.е. приобрести данную редакцию.

Developer – редакция для разработчиков программного обеспечения, которая включает полный функционал SQL Server, она позволяет создавать и тестировать приложения на основе SQL Server без ограничений. Она бесплатна, но ее могут использовать только программисты для разработки и демонстрации приложений, иными словами, в качестве сервера баз данных на предприятии ее использовать нельзя.

На рисунке 3.5 представлен скриншот таблицы TestTableInMemory, входящей в тестовую БД IM_TestDB



Рисунке 3.5 – Скриншот таблицы TestTableInMemory, входящей в тестовую БД IM_TestDB

В процессе разработки таблицы In-memory использованы рекомендации, представленные в работе [9].

Итоговый скрипт представлен на рисунке 3.6.

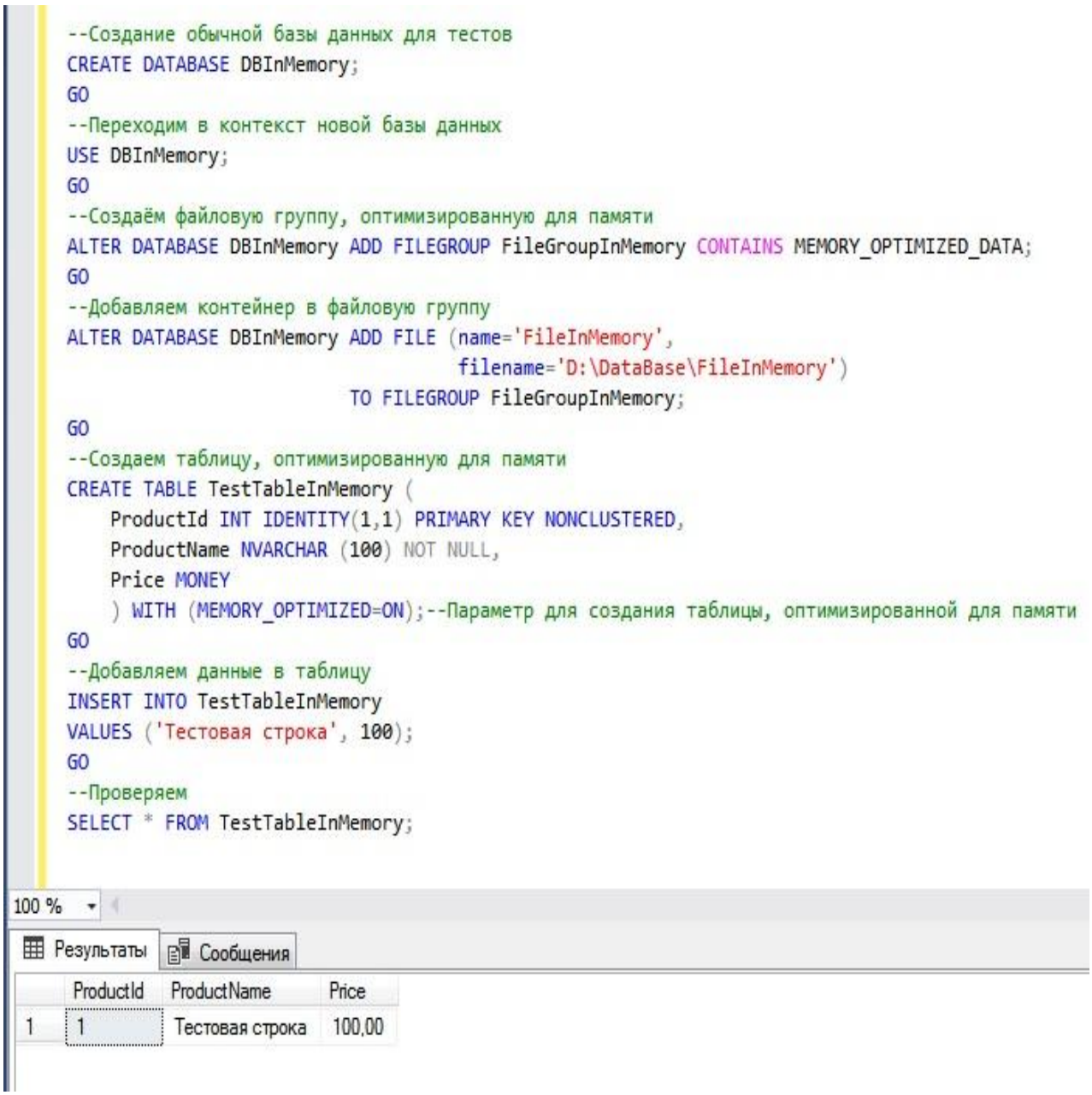


Рисунок 3.6 – Итоговый скрипт разработки таблицы, оптимизированной для памяти в Microsoft SQL Server 2017

Разработаны 3 хранимые процедуры для управления процессом сбора и обработки данных OLTP-системы.

1. Хранимая процедура для ввода данных spm_Ins_TestTableinmemory

```
USE [IM_TestDB]
GO
/****** Object:  StoredProcedure [dbo].[spm_Ins_TestTableinmemory]
Script Date: 04.11.2020 10:43:23 ******/
```

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:		<Author,,Name>
-- Create date: <Create Date,,>
-- Description:	<Description,,>
-- =============================================
ALTER PROCEDURE [dbo].[spm_Ins_TestTableinmemory]
@var1 nvarchar (100),
@var2 money
AS
BEGIN
	-- SET NOCOUNT ON added to prevent extra result sets from
	-- interfering with SELECT statements.
	SET NOCOUNT ON;
	 INSERT INTO TestTableInMemory
  VALUES (@var1, @var2);
  SELECT * FROM TestTableInMemory;
 END
```

Результат выполнения процедуры представлен на рисунке 3.7.

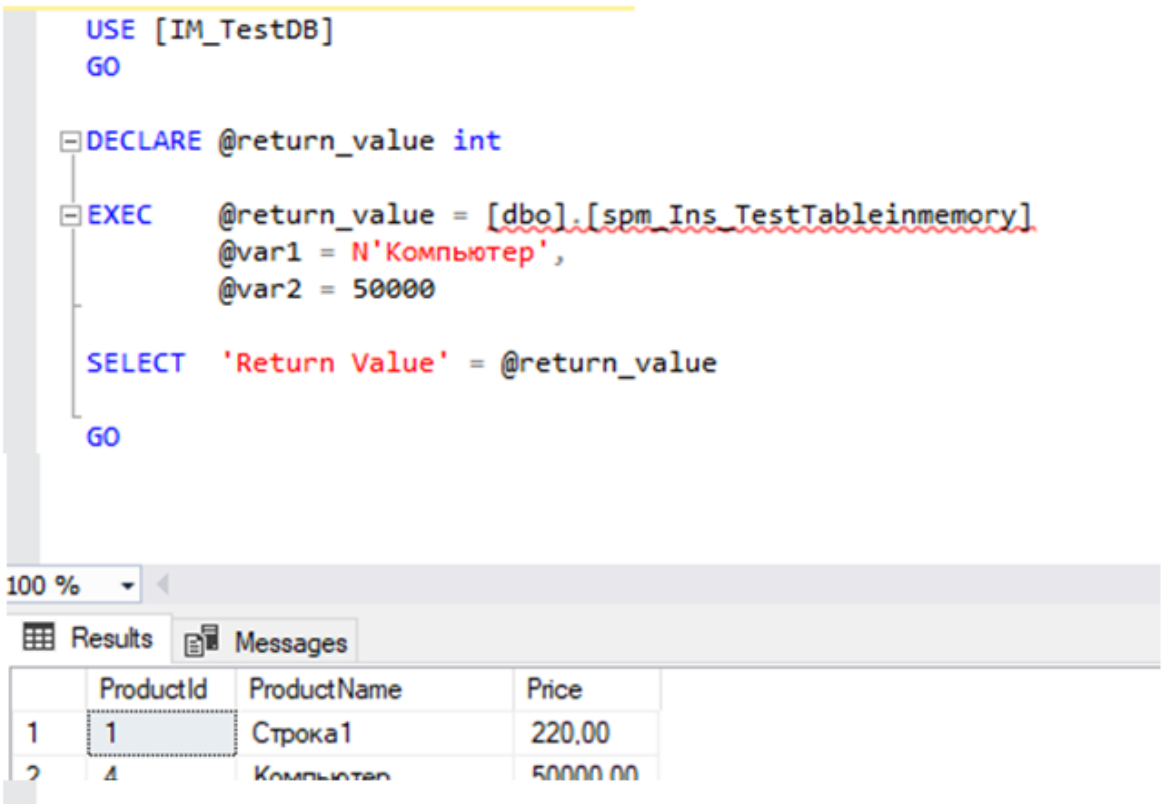


Рисунок 3.7 – Результат выполнения процедуры ввода данных в таблицу In_Memory

2. Хранимая процедура для редактирования данных spm_Upd_TestTableinmemory

```
USE [IM_TestDB]
GO
/****** Object:  StoredProcedure [dbo].[spm_Upd_TestTableInmemory]
Script Date: 04.11.2020 10:46:43 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:		<Author,,Name>
-- Create date: <Create Date,,>
-- Description:	<Description,,>
-- =============================================
```

```
ALTER PROCEDURE [dbo].[spm_Upd_TestTableInmemory]
        @var1 nvarchar(100),
        @var2 money,
        @var3 int
AS
BEGIN
        SET NOCOUNT ON;
        UPDATE TestTableInmemory SET ProductName=@var1,
Price=@var2 WHERE  ProductId=@var3
   -- Insert statements for procedure here
        SELECT * FROM TestTableInmemory WHERE  ProductId=@var3
END
```

Результат выполнения процедуры представлен на рисунке 3.8.

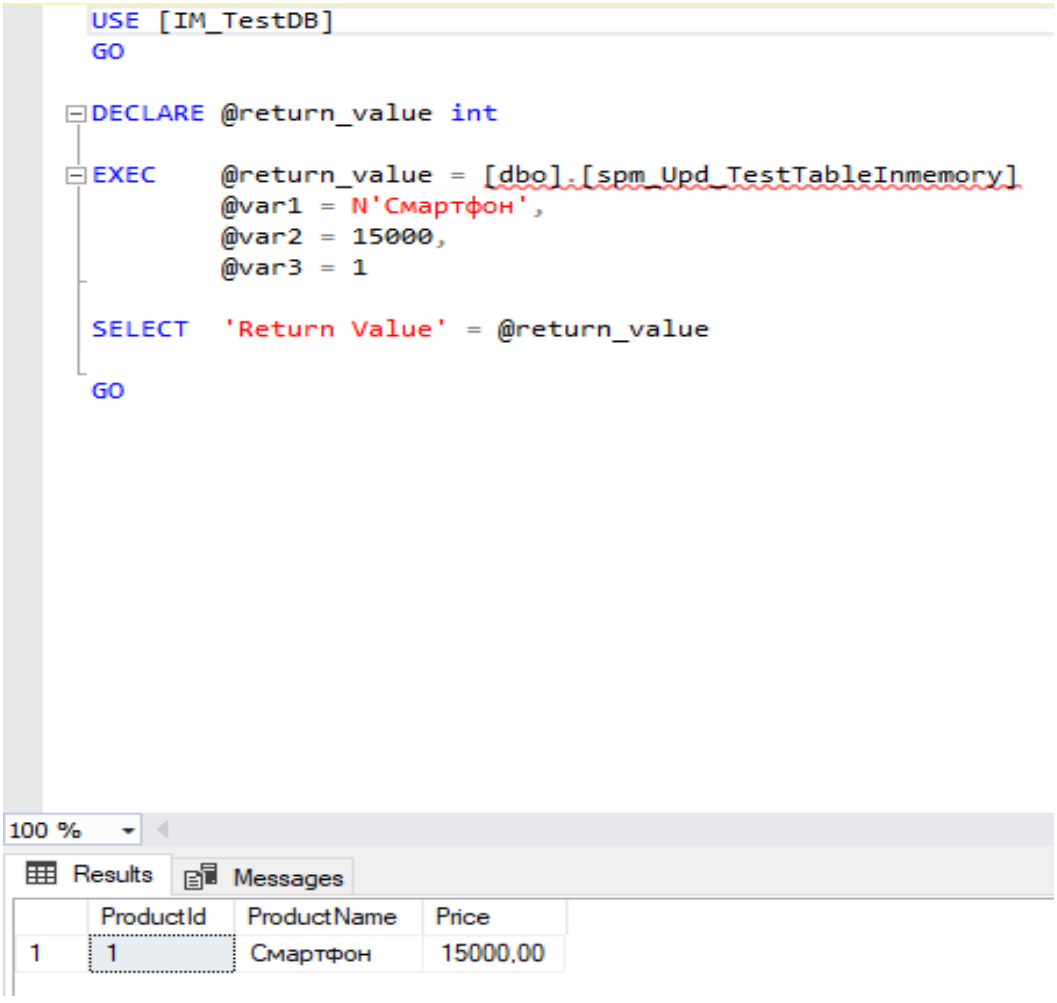


Рисунок 3.8 – Результат выполнения процедуры редактирования данных в таблице In_Memory

3. Хранимая процедура для удаление данных spm_Del_TestTableinmemory

```
USE [IM_TestDB]
	GO
	/****** Object:  StoredProcedure [dbo].[spm_Del_TestTableinMemory]    Script Date:
	04.11.2020 10:48:12 ******/
	SET ANSI_NULLS ON
	GO
	SET QUOTED_IDENTIFIER ON
	GO
	-- =============================================
	-- Author:		<Author,,Name>
	-- Create date: <Create Date,,>
	-- Description:<Description,,>
	-- =============================================
	ALTER PROCEDURE [dbo].[spm_Del_TestTableinMemory]
	@var1 int
	AS
	BEGIN
		-- SET NOCOUNT ON added to prevent extra result sets from
		-- interfering with SELECT statements.
		SET NOCOUNT ON;
		DELETE FROM TestTableinMemory WHERE ProductId=@var1
	  -- Insert statements for procedure here
		SELECT * FROM TestTableinMemory
	END
```

Результат выполнения процедуры представлен на рисунке 3.9

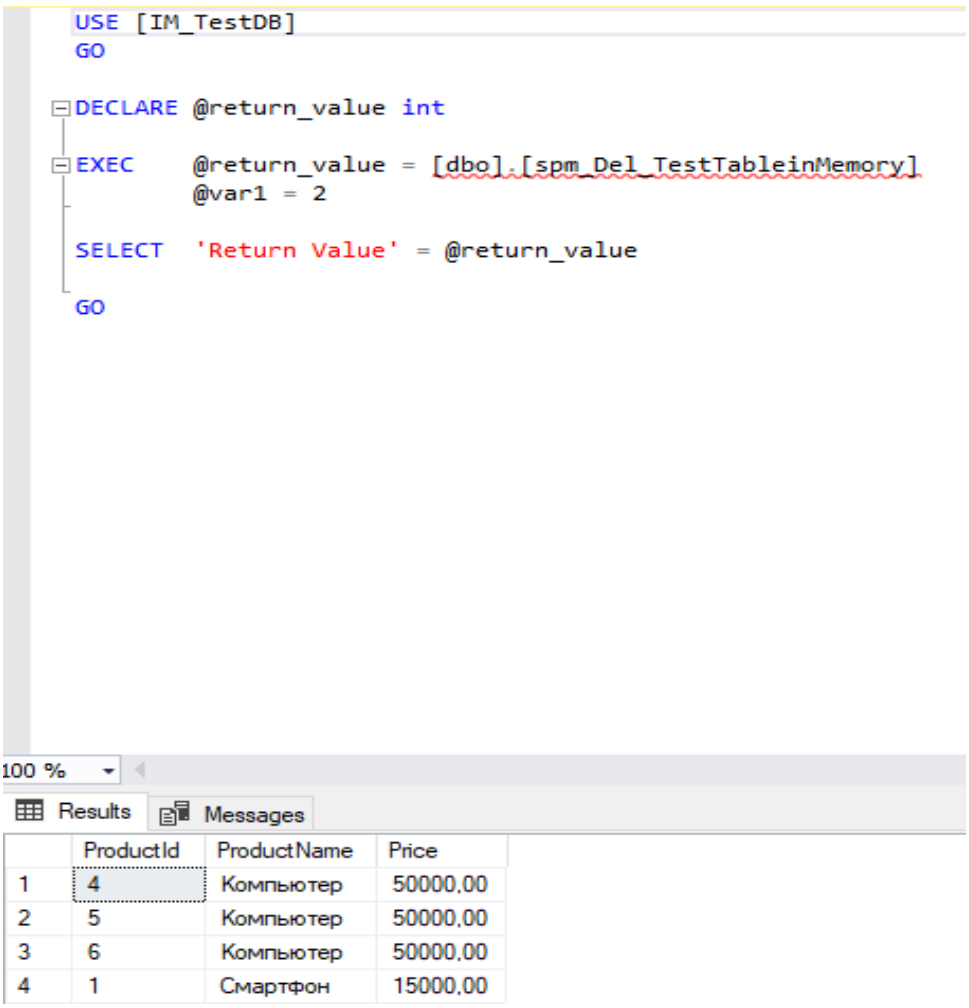


Рисунок 3.9 – Результат выполнения процедуры удаления данных в таблице In-Memory

3.4 Проверка адекватности модели системы сбора и обработки больших массивов данных

По мере того как информационные системы продолжают становиться все более и более сложными из-за более широкого внедрения и усложнения планового использования, проблемы, связанные с эффективностью таких программных проектов, становятся все более и более важными для ИТ-директора и его команды.

Кроме того, для поставщиков ИТ-услуг это становится более важным с растущим пониманием того, что простая реализация прогрессивных и амбициозных программных проектов в организациях не обязательно может

повысить производительность организации и привести к конкурентному преимуществу, которое не могут быть опережены конкурентами.

Именно здесь приходит более глубокое понимание критических показателей эффективности информационных систем, так что проблемы могут быть решены на низовом уровне.

Рассмотрим индикаторы, показывающие неэффективность информационных систем [25]:

– чрезмерные простои ИС;

– чрезмерные затраты на обслуживание ИС;

– ненадежные результаты по ИС и дедупликация данных;

– ошибочные данные по разным функциональным отделам предприятия;

– чрезмерные эксплуатационные расходы во время безотказной работы ИС;

– недовольство пользователей выходными данными, содержанием или своевременностью их предоставления.

Рассмотрим методы оценки эффективности информационных систем:

1) Относительная оценка.

Выполнение задачи: ожидается, что эффективный проект разработки ИС улучшит выполнение задач пользователями системы. Однако важно отметить, что предоставление конкретных показателей прошлых достижений, которые аудиторы могут использовать для оценки, может быть затруднено.

Показатели эффективности выполнения задач различаются для разных приложений, а иногда и для разных организаций. Например, для системы управления производством может быть количество выпущенных единиц, количество переработанных дефектных единиц, количество списанных единиц, время простоя / время простоя. Точно так же важно отслеживать выполнение задачи с течением времени. Может показаться, что система улучшилась в течение короткого времени после внедрения, но после этого впала в беспорядок.

2) Абсолютная оценка.

– Операционная эффективность: системный аудитор проверяет, насколько хорошо информационная система соответствует своим целям с точки зрения пользователя, который взаимодействует с системой на регулярной основе. Для оценки операционной эффективности используются четыре показателя: частота использования, характер использования, простота использования и удовлетворенность пользователей. Если система используется часто, она, вероятно, будет более эффективной с точки зрения конечного пользователя. Точно так же очень сложная система может иметь панели аналитики и аналитики, но ее можно использовать только для записи транзакций. Этот более низкий характер использования конечным пользователем будет указывать на более низкую эффективность внедренных информационных систем. Простота использования и удовлетворенность пользователей после использования оцениваются с точки зрения конечного пользователя. Это дает аудитору дополнительную информацию о том, есть ли потеря производительности, когда системы являются сложными и трудно управляемыми с точки зрения пользовательского интерфейса.

– Техническая эффективность: в этой оценке аудитор должен оценить, есть ли у внедренной информационной системы соответствующие аппаратные / программные технологии, используемые для поддержки системы, или же изменение технологии позволит системе лучше достичь поставленных целей. Производительность оборудования можно измерить с помощью аппаратных мониторов или более грубых показателей, таких как время отклика системы, время простоя. Эффективность программного обеспечения можно измерить, изучив историю обслуживания, модификации и потребления ресурсов во время выполнения программы. История ремонта программного или аппаратного обеспечения указывает на качество логики, существующей в программе. Обширное исправление ошибок часто подразумевает неправильный дизайн, кодирование или тестирование; отказ от использования структурированных подходов и т. д. Однако использование этих мер является

серьезной проблемой. Необходимо понимать, что оборудование и программное обеспечение не являются независимыми ресурсами, и при аудите эффективности необходимо учитывать возможные совпадения.

– Экономическая эффективность: это требует определения затрат и выгод и надлежащей оценки затрат и выгод. Иногда добиться этого сложно, поскольку затраты и выгоды зависят от характера проекта информационных систем. Не все затраты и выгоды можно отслеживать напрямую во многих бизнес-функциях. Некоторые из преимуществ, ожидаемых и извлекаемых из ИБ, будут основаны на контекстном использовании.

Для оценки эффективности разработанной OLTP-системы был проведен эксперимент.

На компьютере установлена СУБД MS SQL Server MS SQL Server 2017 Enterprise Edition, поддерживающая технологию In-Memory.

В компьютере установлены два диска – жесткий и твердотельный.

Тестовая БД размещена на жестком диске (базовый вариант), соответственно тестовая БД с таблицей In-Memory – на твердотельном (проектный вариант).

Разработана хранимая процедура вставки в тестовые таблицы 100000 записей с выводом времени на выполнении операции:

```
USE [DBInMemory]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[sp_LoopInsm]
    AS
BEGIN
DECLARE @ID1 AS BIGINT
SET @ID1=1
```

```
        SET NOCOUNT ON;
DELETE FROM test_tblm
SELECT CONVERT (time, SYSDATETIME())
   ,CONVERT (time, CURRENT_TIMESTAMP)
   ,CONVERT (time, GETDATE());
WHILE (SELECT count(*) FROM test_tblM) < 100000
BEGIN
  INSERT INTO test_tblm (id1,city1, city2) VALUES
(@ID1,'Тольятти','Москва')
  SET @ID1=@ID1+1
END
SELECT CONVERT (time, SYSDATETIME())
   ,CONVERT (time, CURRENT_TIMESTAMP)
   ,CONVERT (time, GETDATE());
END
```

Полученные значения времени выполнения операции сравниваются.

Результаты эксперимента представлены на рисунке 3.10 (а, б) и сведены в таблицу 3.1.

Хранимая процедура ввода данных реализована на основе шаблона контролёра учетной системы, разработанного на основе соответствующего паттерна проектирования.

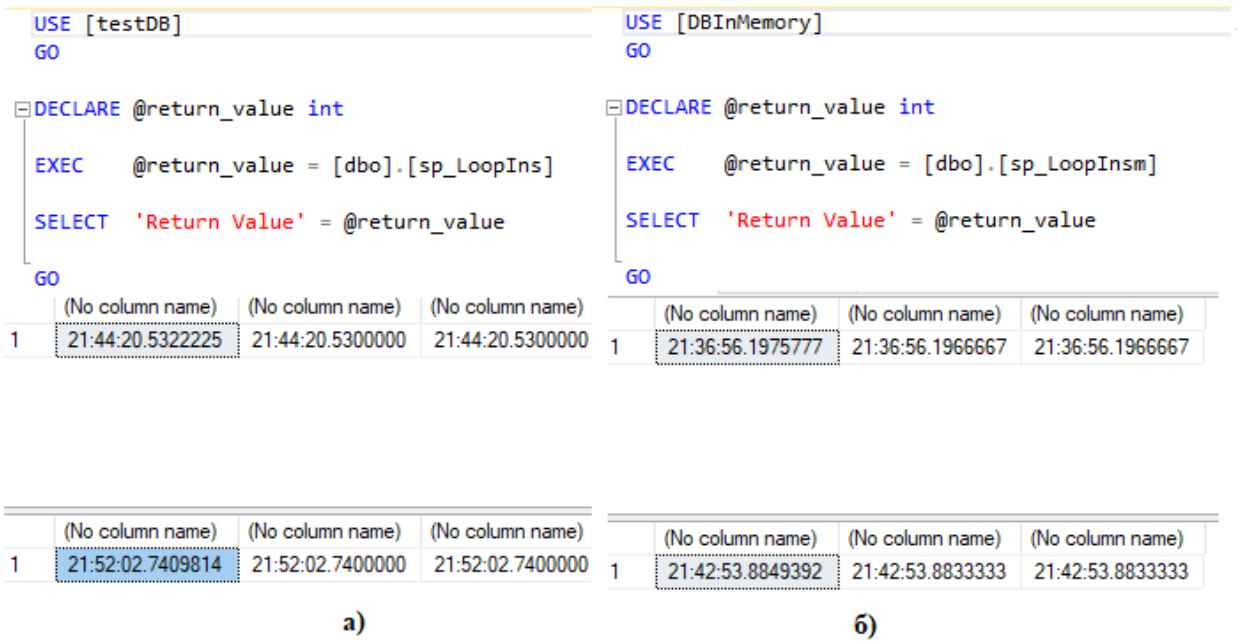


Рисунок 3.10 – Отчеты выполнения хранимой процедуры в базовом (а) и проектном (б) вариантах

Таблица 3.1 – Таблица условий и результатов эксперимента

Характеристики	Базовый вариант	Проектный вариант
Конфигурация компьютера	Процессор: Intel ® Core ™ i5-7400 CPU @ 3ГГц ОЗУ: 8 ГБ	
ОС	64-разрядная Windows 10 Pro	
СУБД	MS SQL Server 2017 Enterprise Edition	
Область размещения тестовой БД	Жесткий диск, 1 ТБ	SSD, 256 МБ
Таблица для загрузки данных	Обычная	In-Memory
Источник данных	Хранимая процедура для вставки 100000 записей	
Время выполнении операции	7 мин 40 с	6 мин

Как показал, эксперимент обработка данных в проектном варианте выполнена на 1мин и 40 с быстрее, чем в базовом.

Совершенно очевидно, что при больших объемах данных эта разница должна существенно вырасти.

На рисунках 3.11, 3.12 показана производительность на диспетчере задач.

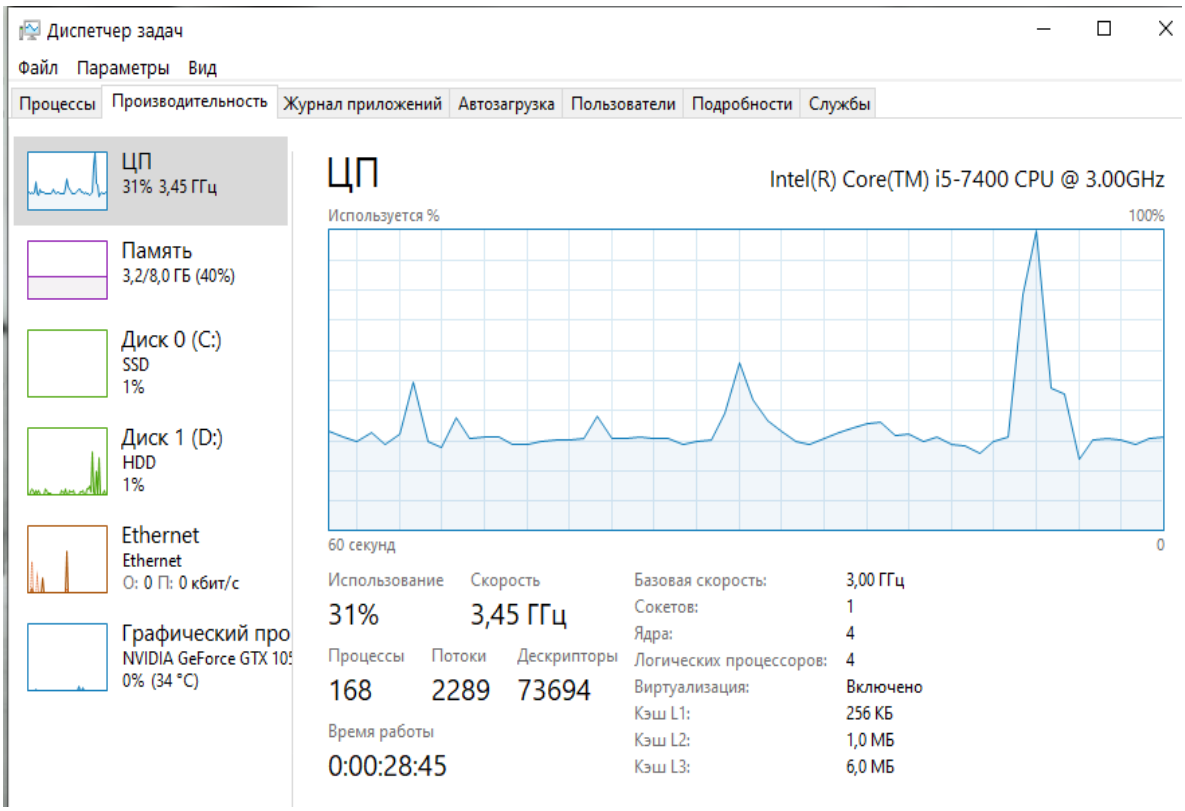


Рисунок 3.11 – Производительность базового варианта

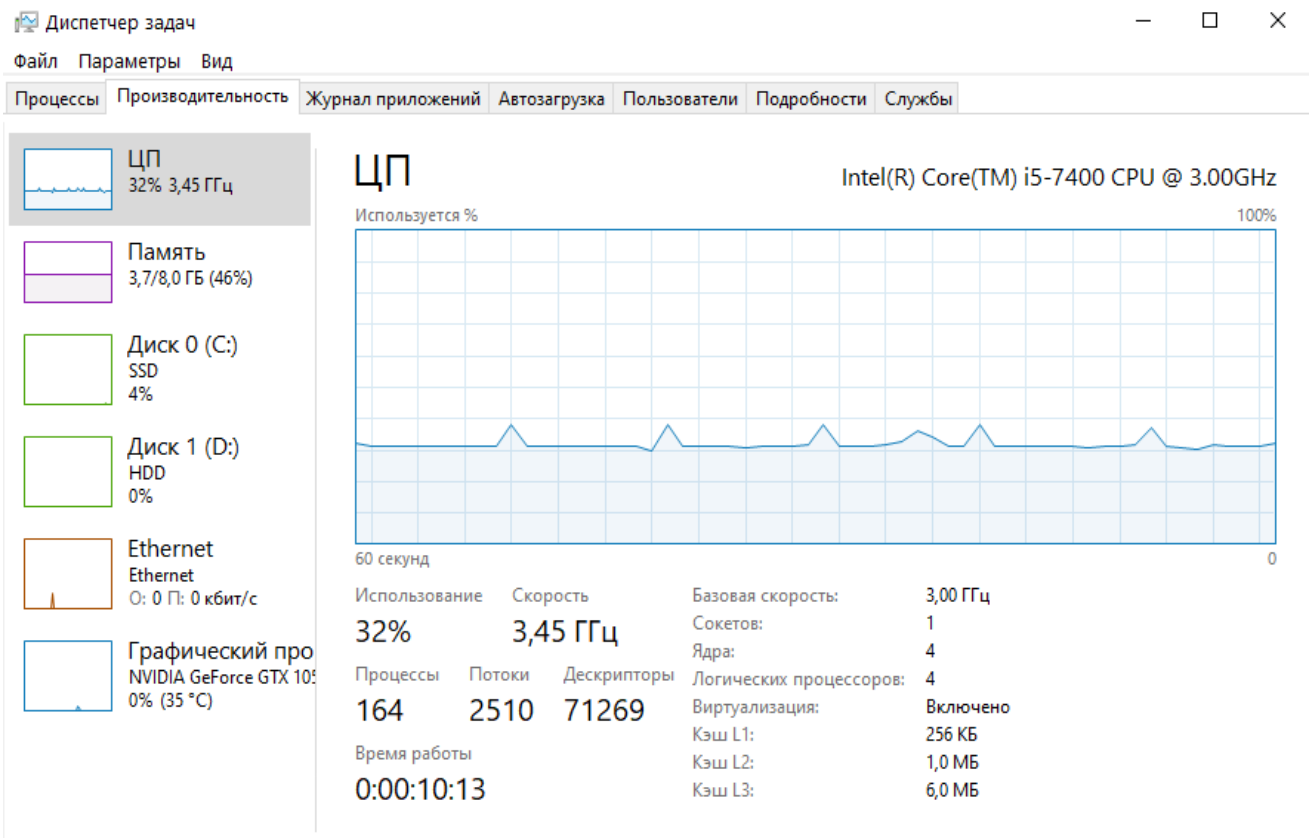


Рисунок 3.12 – Производительность проектного варианта

Таким образом, оценка эффективности OLTP-системы для сбора и обработки больших массивов данных на основе предлагаемой модели подтвердила адекватной последней.

Выводы к главе 3

1. Для разработки логической модели эффективной OLTP-системы для сбора и обработки больших массивов данных использованы диаграммы компонентов и развертывания UML, отражающие соответственно компоненты системы и связи между ними, а также ее топологию.
2. Физическая модель системы OLTP-системы представляет собой ее реализацию на основе СУБД MS SQL Server 2017.
3. Оценка эффективности OLTP-системы для сбора и обработки больших массивов данных на основе предлагаемой модели подтвердила адекватной последней.

Заключение

В настоящее время к OLTP-системам предъявляются повышенные требования к эффективности указанных систем, приближая их характеристики к системам реального времени.

Решения данной проблемы существенно усложняется, если объектом обработки являются большие массивы данных, неограниченных источником которых является Интернет вещей.

Магистерская диссертация посвящена актуальной проблеме разработки модели системы сбора и обработки больших массивов данных, обеспечивающей повышение эффективности указанной системы.

Выполненные в работе научные исследования представлены следующими основными результатами:

1. Проанализировано современное состояние проблемы повышения эффективности OLTP-систем. Как, показал анализ принципы построения OLTP-систем распространяются на решения, предназначенные для обработки больших массивов данных. Вместе с тем анализ позволил констатировать недостаточность работ по проблематике моделирования OLTP-систем для обработки больших данных, что подтверждает актуальность темы магистерской диссертации.

2. Произведены анализ и выбор методологии и технологии моделирования системы сбора и обработки больших массивов данных. Отмечено, что в последнее время для повышения эффективности систем обработки больших массивов данных применяются технологии NoSQL, NewSQL и In-memory. На основании представленного анализа в качестве методологии разработки выбран комплексный подход, использующий лучшие мировые практики обеспечения высокой эффективности OLTP-систем для больших массивов данных.

3. Разработана модель системы сбора и обработки больших массивов данных. Для разработки логической модели эффективной OLTP-системы для сбора и обработки больших массивов данных использованы диаграммы

компонентов и развертывания UML, отражающие соответственно компоненты системы и связи между ними, а также ее топологию. На основе предложенной модели в двухзвенной архитектуре выполнена реализация OLTP-системы для сбора и обработки больших массивов данных. В качестве сервера баз данных OLTP-системы использована версия промышленной СУБД MS SQL Server 2017. В качестве клиента использована среда MS SQL Server Management Studio.

4. Для оценки эффективности системы сбора и обработки больших массивов данных, разработанной на основе предлагаемой модели, был проведен эксперимент, который подтвердил эффективность OLTP-системы, а, следовательно, – адекватность предложенной модели.

Таким образом, в работе решена актуальная научно-практическая проблема разработки модели системы сбора и обработки больших массивов данных, обеспечивающей повышение эффективности последней.

Гипотеза исследования подтверждена.

Значение диссертационной работы определяется тем, что в ее рамках исследованы возможности повышения эффективности системы сбора и обработки больших массивов данных.

Список используемой литературы

1. Братченко Н. Ю. Распределенные базы данных : учебное пособие. Ставрополь : Северо-Кавказский федеральный университет, 2015. 130 с. URL: http://www.iprbookshop.ru/63130.html (дата обращения: 30.10.2020).

2. Бурков А. В. Проектирование информационных систем в Microsoft SQL Server 2008 и Visual Studio 2008 : учебное пособие. Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. 310 с. [Электронный ресурс]. URL: http://www.iprbookshop.ru/89466.html (дата обращения: 23.09.2020).

3. ГОСТ 19781-90 Единая система программной документации. Обеспечение систем обработки информации программное. Термины и определения.

4. ГОСТ 20886-85 Организация данных в системах обработки данных. Термины и определения.

5. ГОСТ 34.003-90 Информационная технология (ИТ). Комплекс стандартов на автоматизированные системы. Термины и определения.

6. ГОСТ 34.321-96 Информационные технологии. Система стандартов по базам данных. Эталонная модель управления данными.

7. Дейт К.Дж. SQL и реляционная теория. М.: Символ-Плюс, 2010. 480с.

8. Зудилова Т.В., Шмелева Г.Ю. Создание запросов в Microsoft SQL Server 2008. СПб: НИУ ИТМО, 2013. 149 с.

9. Как создать таблицу, оптимизированную для памяти? Технология In-Memory OLTP в Microsoft SQL Server [Электронный ресурс]. URL: https://info-comp.ru/obucheniest/679-create-table-in-memory-oltp.html (дата обращения: 30.10.2020).

10. Корпорация Oracle [Электронный ресурс]. URL: https://www.oracle.com/index.html (дата обращения: 23.09.2020).

11. Леоненков А. В. Объектно-ориентированный анализ и

проектирование с использованием UML и IBM Rational Rose : учебное пособие. Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. 317 с. URL: http://www.iprbookshop.ru/97554.html (дата обращения: 30.10.2020).

12. Обзор основных нововведений в Microsoft SQL Server 2017 [Электронный ресурс]. URL: https://info-comp.ru/novosti/594-review-microsoft-sql-server-2017.html (дата обращения: 30.10.2020).

13. Самуйлов С.В. Объектно-ориентированное моделирование на основе UML [Электронный ресурс]: учебное пособие. Саратов: Вузовское образование, 2016. 37 с. URL: http://www.iprbookshop.ru/47277.html (дата обращения: 30.10.2020).

14. СУБД NuoDB Database [Электронный ресурс]. URL: http://doc.nuodb.com/Latest/Default.htm (дата обращения: 23.09.2020).

15. СУБД VoltDB [Электронный ресурс]. URL: https://www.voltdb.com/ (дата обращения: 23.09.2020).

16. A Comparison of Data Modeling Methods for Big Data [Электронный ресурс]. URL:https://dzone.com/articles/a-comparison-of-data-modeling-methods-for-big-data#:~:text=Modeling%20Methodology%20for%20OLTP%20and,and%20inconsistency%20in%20transaction%20processing (дата обращения: 23.09.2020).

17. ACID [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/ACID (дата обращения: 23.09.2020).

18. Alinous-elastic-db [Электронный ресурс]. URL: https://github.com/alinous-core/alinous-elastic-db (дата обращения: 23.09.2020).

19. Brewer Eric A. Towards robust distributed systems. (Invited Talk) Principles of Distributed Computing, Portland, Oregon, July 2000.

20. Deployment Diagram Tutorial [Электронный ресурс]. URL: https://www.lucidchart.com/pages/uml-deployment-diagram (дата обращения: 30.10.2020).

21. Diaconu C. Hekaton: SQL Server’s Memory-Optimized OLTP Engine,

Conference Paper, June 2013.

22. Gartner Glossary [Электронный ресурс]. URL: https://www.gartner.com/en/information-technology/glossary (дата обращения: 23.09.2020).

23. Hive как инструмент для ETL или ELT [Электронный ресурс]. URL: https://www.ibm.com/developerworks/ru/library/bd-hivetool/index.html (дата обращения: 23.09.2020).

24. Improving Online Transaction Processing Systems with SSDs [Электронный ресурс]. URL: https://insights.samsung.com/2018/01/23/improving-online-transaction-processing-systems-with-ssds/ (дата обращения: 23.09.2020).

25. Information Systems Effectiveness Measures [Электронный ресурс]. URL: https://tech-talk.org/2015/03/10/information-systems-effectiveness-measures/ (дата обращения: 30.10.2020).

26. In-Memory Processing [Электронный ресурс]. URL: https://hazelcast.com/glossary/in-memory-processing/ (дата обращения: 23.09.2020).

27. Introducing SQL Server In-Memory OLTP [Электронный ресурс]. URL: https://www.red-gate.com/simple-talk/sql/learn-sql-server/introducing-sql-server-in-memory-oltp/ (дата обращения: 23.09.2020).

28. McCarthy W. The REA Accounting Model: A Generalized Framework for Accounting System in a Shared Data Environment, 1982.

29. Mkrtychev S. Methodology to design management accounting information systems, CEUR Workshop Proceedings. 2018. №2258. P. 21-28.

30. Murthy U.S., Geerts G.L. An REA Ontology-Based Model for Mapping Big Data to Accounting Information Systems Elements, Journal of Information Systems 31(3), 2017.

31. OLTP (online transaction processing) [Электронный ресурс]. URL: https://searchdatacenter.techtarget.com/definition/OLTP (дата обращения: 23.09.2020).

32. OLTP [Электронный ресурс]. URL: http://sewiki.ru/OLTP (дата обращения: 23.09.2020).

33. Speelpenning J., Daux P., Gallus J. Data Modeling and Relational Database Design, Oracle Corporation, 1998, 1999,2001.

34. The Benefits of Optimizing OLTP Databases Using IBM eXFlash Solid-State Drives [Электронный ресурс]. URL: https://lenovopress.com/redp4849.pdf (дата обращения: 30.10.2020).

35. Transactional data [Электронный ресурс]. URL: https://docs.microsoft.com/en-us/azure/architecture/data-guide/relational-data/online-transaction-processing (дата обращения: 23.09.2020).

36. UML 2 Tutorial - Package Diagram [Электронный ресурс]. URL: https://sparxsystems.com/resources/tutorials/uml2/package-diagram.html (дата обращения: 30.10.2020).

37. What Is a NewSQL Database System? [Электронный ресурс]. URL: http://www.dbta.com/Columns/DBA-Corner/What-Is-a-NewSQL-Database-System-104489.aspx (дата обращения: 23.09.2020).