МИНИСТЕРСТВО ПРОСВЕЩЕНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ПЕДАГОГИЧЕСКИЙ УНИВЕРСИТЕТ им. А. И. ГЕРЦЕНА»

Направление подготовки/специальность
09.03.01 Информатика и вычислительная техника

направленность (профиль)/специализация
«Технологии разработки программного обеспечения»

Выпускная квалификационная работа

«Разработка решения для очистки и архивирования данных в секционированных таблицах СУБД»

Обучающегося 4 курса
очной формы обучения
Шардта Максима Александровича

Руководитель выпускной квалификационной работы:
кандидат физ.-мат. наук, доцент
Жуков Николай Николаевич

Санкт-Петербург

2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современным информационным системам требуется хранить большие объемы данных, которые необходимы для поддержки операционной деятельности, аналитики и соблюдения регуляторных требований. Однако значительная часть этих данных со временем устаревает и перестает использоваться, продолжая занимать высокопроизводительное оборудование, что приводит к неоправданному росту эксплуатационных затрат. Для уменьшения стоимости эксплуатации требуется удалять или переносить в архив часть устаревших данных. Так как одним из эффективных способов управления большими данными является секционирование — разделение на логические и физические блоки (сегменты) по указанным критериям, а ручное удаление или архивация затруднены из-за сложных связей между таблицами и риска нарушения целостности данных, необходимо разработать автоматическую очистку таких сегментов. Тем не менее, существующие решения для очистки и архивации данных сталкиваются с рядом ограничений: несогласованность данных в связанных таблицах после очистки, невозможность работы с многоуровневыми иерархиями секций, а также жесткая привязка к кратности секционирования.

Целью работы является создание решения для автоматизированной очистки и архивации данных в секционированных таблицах СУБД PostgreSQL, обеспечивающего согласованность данных, минимальную нагрузку на систему и поддержку сложных сценариев работы.

Для достижения поставленной цели необходимо решить следующие **задачи**:

1. Провести анализ существующих подходов к жизненному циклу данных в секционированных таблицах.
2. Сформировать требования к решению.
3. Реализовать механизмы очистки простых, секционированных и связных таблиц, включая удаление сегментов и удаление данных пересозданием таблиц в СУБД PostgreSQL, обеспечив обработку зависимостей.

4. Провести тестирование работоспособности решения и согласованности данных после очистки и/или архивирования.

Практическая значимость работы заключается в снижении затрат на хранение данных за счет автоматизированного удаления устаревших записей и их переноса в архивные системы, обеспечение согласованности данных в распределенных системах, предотвращая ситуации, когда удаление информации в родительских таблицах ведет к возникновению некорректных ссылок в дочерних таблицах.

Результаты работы могут быть применены информационных системах, работающих с большими объемами данных в PostgreSQL, где требуется управление большими объемами секционированных данных. Это расширяет область использования решения в областях, где критически важны оптимизация ресурсов и соответствие требованиям к долгосрочному хранению информации.

1. Теоретические основы управления данными в секционированных таблицах

1.1. Особенности хранения и управления данными в реляционных СУБД 😕

1.2. Механизмы секционирования таблиц в PostgreSQL

Управление большим объемом данных в PostgreSQL требует комплексного подхода, сочетающего различные методы хранения информации. В рамках работы рассматриваются стандартные подходы к организации данных, включая как базовые механизмы ручного управления, так и современные технологии секционирования.

Традиционные методы работы с данными в PostgreSQL предполагают ручное удаление устаревших записей с помощью оператора DELETE. Такой подход прост в реализации, но обладает существенными недостатками при работе с крупными таблицами. При выполнении DELETE система не освобождает место на диске немедленно, а лишь помечает записи как удаленные, что приводит к фрагментации данных. Для полного освобождения пространства требуется выполнение команды VACUUM FULL, которая создает новую физическую версию таблицы без удаленных записей. Этот процесс требует эксклюзивной блокировки таблицы и может занимать значительное время при больших объемах данных.

Альтернативой ручному удалению выступает использование временных таблиц или таблиц-наследников (inheritance). В этом случае данные распределяются между несколькими физическими таблицами по определенным критериям, что позволяет удалять целые таблицы при необходимости очистки. Однако такой подход требует сложной логики в приложении для маршрутизации запросов и поддержания целостности данных. Несмотря на свою гибкость, данный метод требует значительных трудозатрат на поддержку и сопровождение, что в сочетании с появлением более современных решений привело к его постепенному вытеснению из современных информационных систем.

Современным решением для управления большими объемами информации стало declarative partitioning, введенное в PostgreSQL 10. Данный механизм позволяет создавать секционированные таблицы, где данные автоматически распределяются по дочерним секциям в соответствии с заданными правилами. Поддерживаются три стратегии секционирования: по диапазону (range), по списку (list) и по хэшу (hash). Каждая из стратегий применяется в зависимости от специфики данных и требований к их обработке.

Диапазонное партиционирование предполагает разделение данных на секции по заданным интервалам значений. Оно особенно эффективено для временных данных, где естественным критерием разбиения выступает дата или временная метка. Например, таблицы с логов событий часто секционируются по месяцам или годам. Это позволяет быстро удалять или архивировать устаревшие данные путем отсоединения (DETACH) целых секций. При этом операция выполняется практически мгновенно и не требует перезаписи оставшихся данных. Преимущество этого подхода заключается в простоте управления жизненным циклом данных: добавление новых секций и удаление старых выполняется декларативно. Однако при неравномерном распределении данных возможен дисбаланс, когда одни партиции оказываются перегруженными, а другие остаются почти пустыми.

Списковое партиционирование основано на явном перечислении значений, которые должны попадать в каждую секцию. Оно применяется, когда данные имеют дискретные категории, такие как регионы, статусы заказов или типы событий. В этом случае каждая секция содержит записи с определенным набором значений ключевого столбца. Такой подход обеспечивает гибкость при работе с нечисловыми критериями, однако поддержка спискового партиционирования требует ручного управления при изменении ключевых значений: если появляется новый регион, необходимо явно создать для него секцию.

Хэш-партиционирование распределяет данные по секциям с использованием хэш-функции от указанного столбца. Такой подход обеспечивает равномерное распределение записей, что полезно для балансировки нагрузки при частых операциях вставки и выборки. Однако хэш-партиционирование плохо подходит для задач очистки и архивации, поскольку данные не имеют логической группировки по времени или категориям. Удаление устаревших записей в этом случае требует полного сканирования таблицы или сложных условий фильтрации, что снижает эффективность.

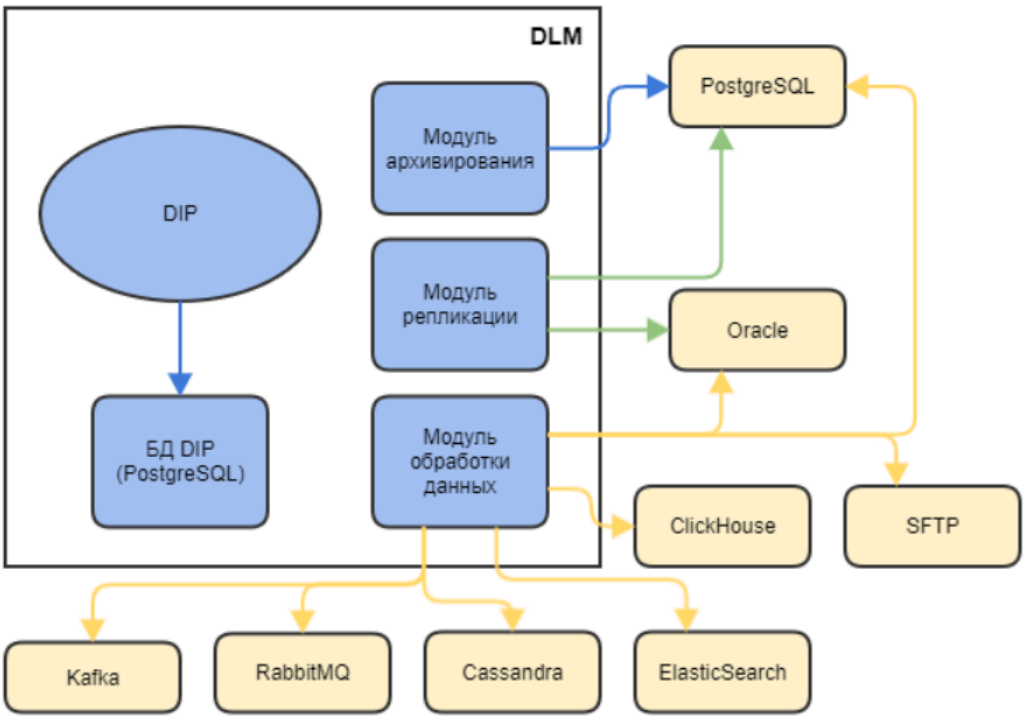
Важным аспектом работы с секционированными таблицами является управление индексами. В PostgreSQL 11 появилась возможность создания первичных ключей для секционированных таблиц. Индексы могут создаваться как для всей таблицы (глобальные), так и для отдельных секций (локальные). Глобальные индексы обеспечивают уникальность значений во всех секциях, но требуют больше ресурсов для поддержания. Локальные индексы проще в обслуживании, но не гарантируют уникальность на уровне всей таблицы.

Для эффективного управления жизненным циклом данных в PostgreSQL используются различные дополнительные механизмы. Декларативное секционирование таблиц вместе с наследованием таблиц может создавать сложные иерархии таблиц. Расширение pg_partman автоматизирует создание и управление секциями на основе временных меток.

Особого внимания заслуживает вопрос обеспечения целостности данных при работе с секционированными таблицами. Внешние ключи между секционированными таблицами требуют особого подхода, так как стандартные ограничения целостности не работают между разными секциями. Для решения этой проблемы могут использоваться триггеры или специальные расширения, такие как pg_pathman.

1.3. Обзор текущей реализации очистки и архивирования простых таблиц

Текущее решение для автоматизированной очистки и архивирования данных таблицах PostgreSQL строится на базе платформы IP (Nexign Data Integrator), в частности модуля DLM (Data Lifecycle Management). Выбор данной платформы в качестве основы обусловлен ее гибкостью, масштабируемостью и наличием готовых механизмов для работы с данными, что позволяет сосредоточиться на реализации специфической функциональности, связанной именно с очисткой секционированных таблиц. Решение включает два ключевых компонента: ядро DIP и модуль DLM, которые совместно обеспечивают выполнение всех требуемых функций.



подпис

Ядро DIP представляет собой центральный компонент, отвечающий за базовую функциональность обработки данных. В контексте решения для очистки и архивирования ядро выполняет несколько критически важных функций:

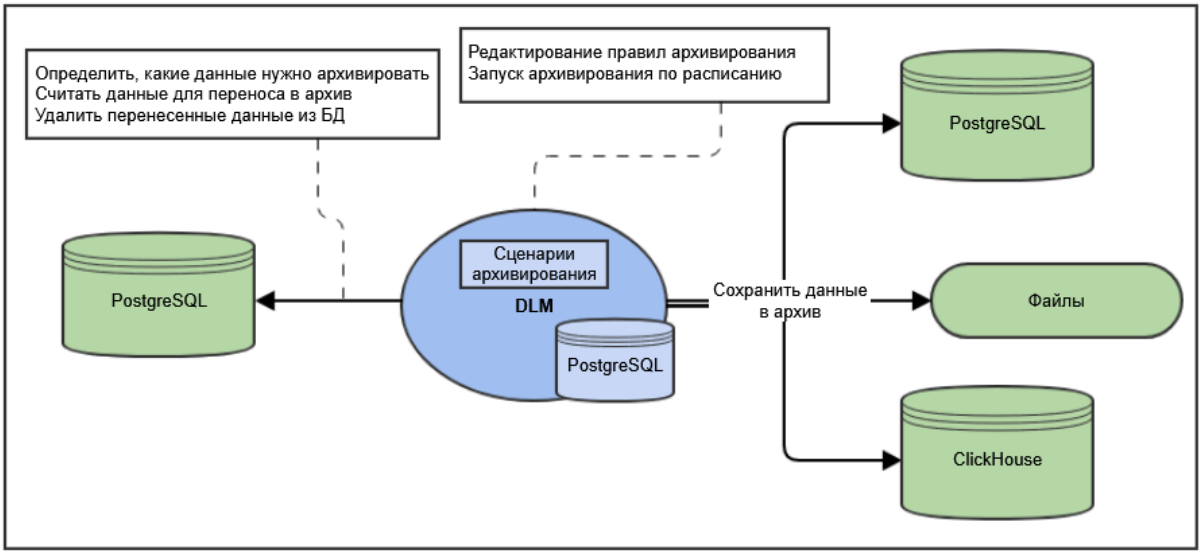
- Ядро обеспечивает взаимодействие с СУБД PostgreSQL, включая установление соединений, выполнение запросов и транзакционное управление операциями.

- Ядро предоставляет инфраструктуру для создания и выполнения сценариев обработки данных, что позволяет гибко настраивать логику работы системы под конкретные требования.
- В ядре реализованы механизмы планирования задач, которые используются для организации расписания выполнения операций очистки и архивирования.

Ядро DIP построено по модульному принципу, что позволяет расширять его функциональность за счет добавления специализированных модулей. В Архитектура ядра обеспечивает тесную интеграцию с этим модулем, предоставляя ему доступ к низкоуровневым функциям работы с СУБД и инфраструктуре выполнения задач. При этом ядро берет на себя такие общие задачи, как управление ресурсами и обработка ошибок, освобождая модуль архивирования от необходимости реализации этой стандартной функциональности.

Модуль DLM является специализированным компонентом, разработанным для решения задач управления жизненным циклом данных. Модуль состоит из нескольких логических блоков, каждый из которых отвечает за определенный аспект работы системы. В ходе работы модуль архивирования, отвечающий за очистку и архивирование данных в PostgreSQL, будет дополнен функциональностью, обеспечивающей очистку и архивирование секционированных таблиц.

Модуль архивирования включает несколько альтернативных стратегий выполнения очистки данных: удаление целых сегментов (таблиц или секций), выборочное удаление записей внутри сегментов и пересоздание сегментов с сохранением только актуальных данных. Каждый из этих методов применяется в зависимости от конфигурации задачи и особенностей структуры данных.



подпис

Для выполнения очистки и архивирования будут модифицированы существующие операторы и добавлены новые. В терминологии DIP, сценарии – это конструктор создания схем последовательного выполнения операций, состоящих из операторов, обеспечивающих решение пользовательских задач. Примером такого сценария является архивирование и очистка простых таблиц:

- В операторе «Подготовка задачи ЖЦ» указывается вся необходимая информация об очищаемой таблице. После запуска оператора создается задача на архивацию и очистку со всей информацией об очищаемых таблицах и периоде очистки
- Оператор «Архивация данных» загружает список таблиц, подлежащих архивации из задачи и возвращает данные из таблиц, которые подлежат архивации. Логика взаимодействия с данными, подлежащим архивации, может быть реализована операторами стандартного модуля DI, например, «Экспорт в БД по JDBC» для переноса записей в другую базу данных
- Оператор «Очистка» очищает таблицы из задачи по определенному заранее критерию. На этом сценарий «очистка и архивирование» завершается.

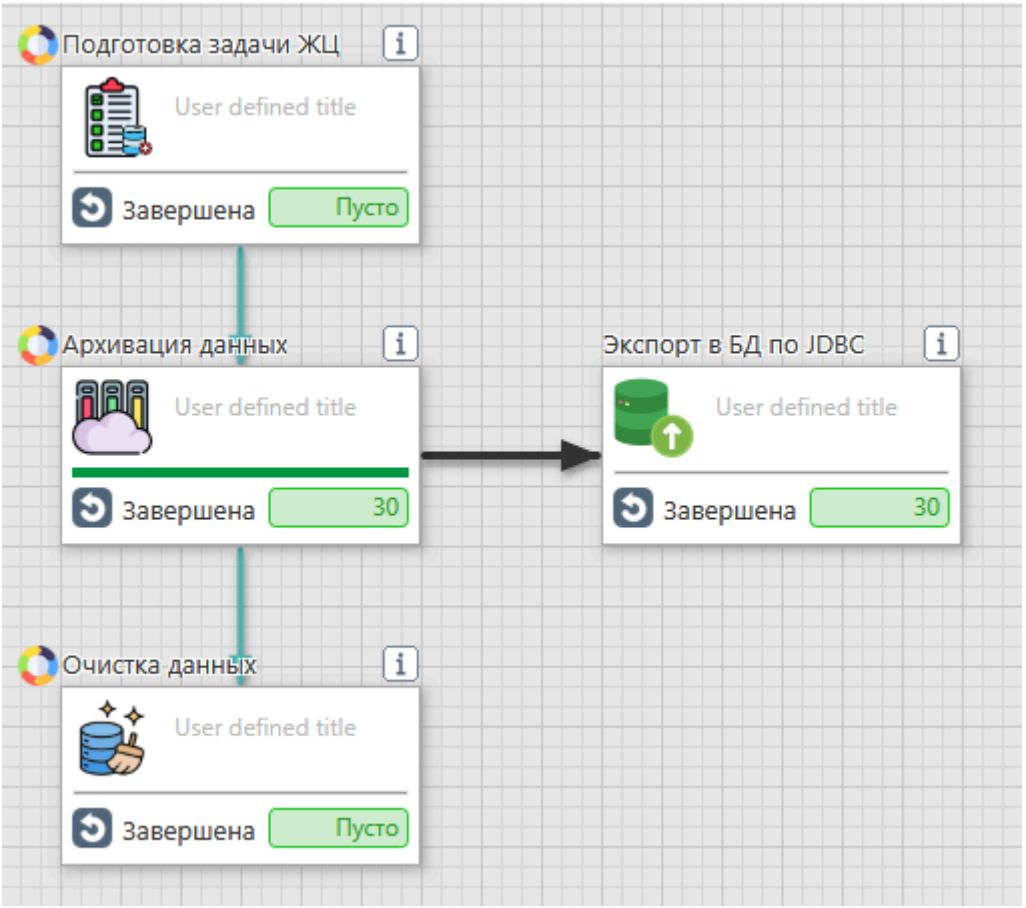


Рис. сценарий очистки

№	НАЗВАНИЕ ПОДЗАДАЧИ	СТРАТЕГИЯ	СТАТУС	ВРЕМЯ ВЫПОЛНЕНИЯ	ИНФОРМАЦИЯ О РАБОТЕ	ПАРАМЕТРЫ
0	test_child	АРХИВАЦИЯ И ОЧИСТКА	СОЗДАНО			ПОКАЗАТЬ

Рис. созданная задача

Таким образом, текущее решение будет дополнено функционалом для очистки секционированных таблиц.

1.4. Формирование требований к разрабатываемому решению

Анализ существующих решений и требований к системе управления жизненным циклом данных позволил сформулировать ключевые критерии для проектируемого решения. Эти критерии охватывают три основных аспекта: обеспечение согласованности данных при выполнении операций очистки и архивирования, поддержку сложных связей между таблицами в иерархиях секционирования, а также гибкость в настройке и адаптации к различным сценариям работы с данными. Каждый из этих аспектов требует детальной проработки, так как именно их сочетание позволит создать

решение, превосходящее существующие аналоги по эффективности и надежности.

Согласованность данных выступает как фундаментальный критерий проектируемого решения. Особенно это важно при работе с секционированными таблицами, где связи могут пересекать границы секций и даже уровни иерархии. Проектируемое решение должно гарантировать, что удаление данных в одной таблице автоматически влечет соответствующие изменения во всех связанных таблицах. Это требует реализации механизмов отслеживания зависимостей между таблицами и выполнения операций в правильной последовательности. Например, при удалении записей из родительской таблицы необходимо сначала обработать все дочерние таблицы, содержащие ссылки на эти записи, либо обеспечить каскадное удаление.

Поддержка сложных связей между таблицами является вторым ключевым критерием. В реальных системах данные организованы в сложные иерархии, где таблицы могут быть связаны как напрямую, так и через промежуточные таблицы. Причем эти связи могут пересекать границы секций и уровни секционирования. Например, родительская таблица может быть секционирована по месяцам, дочерняя - по неделям, а таблицы следующего уровня - по дням. При этом связи между ними могут быть организованы через несколько промежуточных таблиц с различной структурой. Проектируемое решение должно уметь анализировать такие сложные схемы связей и автоматически определять порядок обработки таблиц при выполнении операций очистки и архивирования.

Особое внимание должно быть уделено случаям, когда таблицы имеют разные схемы секционирования. Решение должно уметь определять соответствие между секциями разных таблиц, даже если их границы не совпадают. Например, при очистке месячной секции родительской таблицы необходимо корректно обработать все недельные и дневные секции дочерних таблиц, которые полностью или частично попадают в этот временной интервал. Для этого потребуется реализовать механизмы анализа метаданных

секционирования и определения пересечений временных интервалов. Важно также учитывать возможность динамического изменения схем секционирования - например, когда в разные периоды использовались разные интервалы разбиения.

Гибкость решения проявляется в нескольких аспектах. Во-первых, это возможность настройки различных стратегий очистки и архивирования в зависимости от специфики данных и требований системы. Решение должно поддерживать как полное удаление секций (например, через оператор DETACH PARTITION), так и выборочное удаление записей внутри секций. Во-вторых, решение должно предоставлять гибкие механизмы определения критериев устаревания данных - не только по временным меткам, но и по другим атрибутам записей, таким как статусы или категории.

Сформулированные критерии - согласованность, поддержка сложных связей и гибкость - образуют основу для проектирования архитектуры решения. Их реализация потребует разработки специализированных алгоритмов анализа связей между таблицами, механизмов планирования и выполнения операций очистки, а также интерфейсов для интеграции с внешними системами. При этом важно соблюсти баланс между сложностью реализации и производительностью решения, особенно при работе с большими объемами данных. Последующие этапы работы будут посвящены детальной проработке архитектуры и алгоритмов, обеспечивающих выполнение всех указанных критериев.

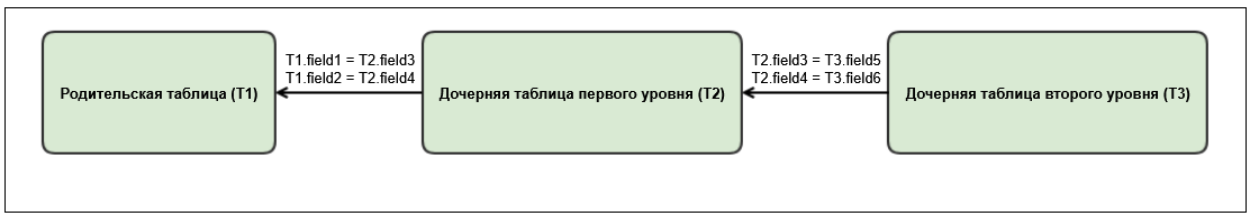
Связка одиночных таблиц

Система предоставляет средства для конфигурации очистки одиночных (несекционированных) таблиц. Такой тип таблиц представляет собой логическую структуру без физического деления на секции, что упрощает работу с данными при отсутствии необходимости в горизонтальном разбиении. Работа с одиночными таблицами осуществляется с использованием тех же операторов жизненного цикла, что и для секционированных, но с учётом специфики их конфигурации.

Ключевой особенностью реализации данного подхода является способ определения временной актуальности записей. В отличие от секционированных таблиц, где границы хранения задаются структурой секций, в одиночных таблицах принадлежность данных к хранимому периоду определяется значением поля, указанного в параметре «Колонка с датой записи». Это поле, как правило, содержит дату создания, модификации или удаления записи, и используется как основной критерий фильтрации устаревшей информации при выполнении задачи очистки.

Следует отметить, что режим работы с одиночными таблицами может быть применён в том числе и к секционированным таблицам, если по тем или иным причинам необходимо игнорировать их внутреннюю секционную структуру. В этом случае очистка будет производиться не по секциям, а построчно, на основании значения даты в указанной колонке. Такой подход может быть полезен, например, при необходимости реализовать универсальный сценарий очистки, применимый ко всем таблицам вне зависимости от их физического устройства.

На Рисунке 1 приведена обобщённая схема взаимодействия между таблицами в рамках задачи очистки, демонстрирующая, как одиночные и секционированные таблицы могут быть включены в единый сценарий. На схеме видно, что даже при наличии иерархической связи между таблицами (например, родительская таблица — дочерняя таблица), каждая из них может быть настроена как одиночная, если соответствующий параметр установлен. Это повышает гибкость конфигурации и позволяет использовать единый механизм управления жизненным циклом данных в рамках разнородной структуры хранения.



рис

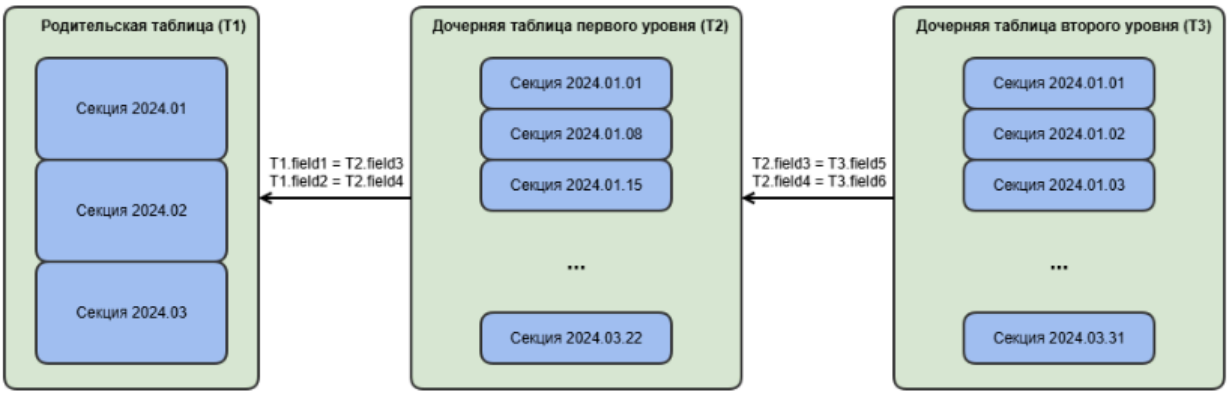
Кроме того, при работе с одиночными таблицами сохраняются все прочие параметры конфигурации, аналогичные секционированным структурам: стратегия и режим очистки, фильтрация по сущностям, направление удаления данных (родитель → дочерние или дочерние → родитель), использование буферных таблиц и т. д.

Связка секционированных таблиц

Связка секционированных таблиц представляет собой один из ключевых сценариев конфигурации очистки. Этот подход используется для работы с таблицами, реализованными с применением механизма секционирования, при котором данные внутри таблицы распределяются по секциям на основании заданного критерия.

Если таблица секционирована декларативно, то очистка может быть выполнена более эффективно и предсказуемо, с возможностью автоматического вычисления границ секций, что задаётся в параметре «Автоматическое вычисление границы секции». При таком подходе система гарантирует, что дочерние таблицы будут очищены синхронно с родительской, строго по секциям, и только в пределах тех данных, которые были признаны устаревшими в контексте родительской структуры. Это позволяет не только поддерживать консистентность данных в иерархии, но и повышает производительность операций очистки за счёт пакетной обработки секций.

На Рисунке 2 показан пример иерархической структуры связанных секционированных таблиц, где каждая дочерняя таблица логически и физически связана с родительской по определённому признаку секционирования.



Связка таблиц как секций

Связка таблиц как секций представляет собой альтернативный подход к реализации очистки данных, в котором каждая таблица физически остаётся самостоятельной, но логически рассматривается как часть единой секционированной структуры.

В отличие от классической конфигурации секционированных таблиц, где используется встроенный механизм секционирования в рамках одной таблицы (например, декларативное секционирование), данный подход предполагает, что каждая таблица представляет собой отдельную секцию. При этом объединение таблиц в единую логическую структуру выполняется на уровне приложения, что позволяет управлять ими как с единой точки зрения — как если бы это была одна секционированная таблица.

Такой подход особенно актуален в тех случаях, когда по организационным, техническим или историческим причинам невозможно реализовать физическое секционирование в СУБД. Например, это может быть полезно при работе с наборами однотипных таблиц, разделённых по периоду (например, log_2022, log_2023, log_2024), но не объединённых в единую секционированную таблицу.

Очистка в этом режиме осуществляется по тем же правилам, что и в случае классических секционированных таблиц. Система гарантирует, что каждая таблица, включённая в конфигурацию как часть логической секции, будет обработана согласно установленным правилам и срокам хранения, аналогично тому, как если бы она являлась частью секционированной

таблицы. Это позволяет сохранить высокую степень управляемости и производительности даже без использования встроенных механизмов секционирования в СУБД.

На Рисунке 3 представлен пример связки таблиц как секций, где несколько самостоятельных таблиц объединены в логическую структуру, охватывающую разные временные диапазоны.

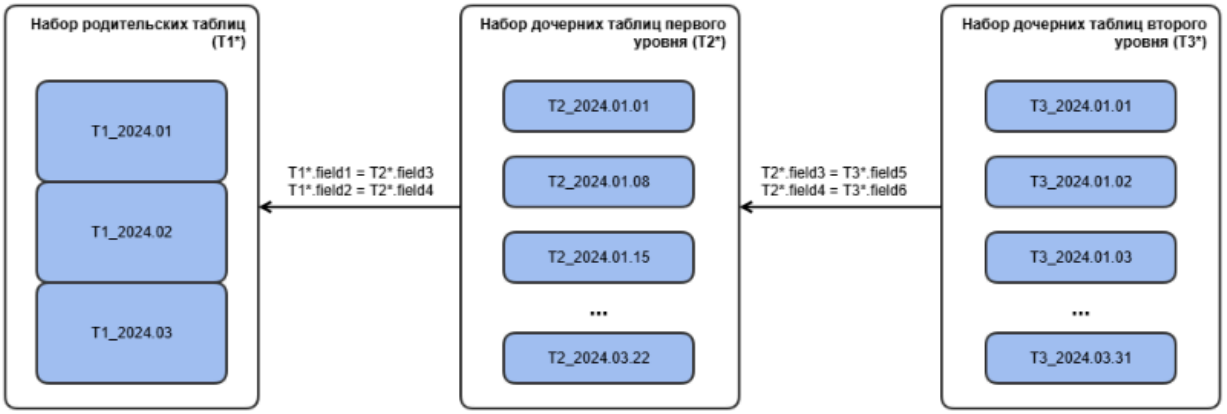


рис 3

Использование буферных таблиц с идентификаторами сущностей для очистки и архивирования

Буферные таблицы используются для обеспечения согласованной очистки и архивирования данных, хранящихся в связанных секционированных таблицах, особенно при наличии сложных иерархий. Основная задача буферных таблиц — зафиксировать набор идентификаторов сущностей, подлежащих удалению или переносу, на момент подготовки сценария очистки, тем самым исключив возможность расхождений между связанными таблицами из-за изменений, произошедших в процессе выполнения сценария.

Использование буферных таблиц оправдано в случаях, когда:

- требуется выполнять очистку и архивирование связанных таблиц в разное время;
- очистка охватывает лишь часть секций, удовлетворяющих условиям хранения;

- необходимо обеспечить атомарную привязку данных из дочерних таблиц к сущностям, уже определённым к удалению в родительской.

На этапе подготовки очистки производится выборка идентификаторов сущностей из родительской таблицы на основании заданных критериев (например, истечение срока хранения). Эти значения (например, пары полей OBJ_ID, OBJ_TYPE из таблицы OBJECTS) заносятся в специальную буферную таблицу. Далее, вместо непосредственного выполнения операций очистки или архивирования с применением условий к основной таблице, сценарий обращается к буферной таблице, обеспечивая строгую детерминированность множества обрабатываемых записей.

При наличии многоуровневой иерархии таблиц может потребоваться создание нескольких буферных таблиц. Это обусловлено необходимостью фиксировать идентификаторы на каждом уровне зависимости. Пример:

- Таблица OBJECTS содержит поля OBJ_ID, OBJ_TYPE;
- Таблица OBJECT_HISTORY ссылается на OBJECTS через OBJ_OBJ_ID, OBJ_TYPE;
- Таблица OBJECT_HISTORY_DATA ссылается на OBJECT_HISTORY по OBJ_HISTORY_HISTORY_ID.

В данном случае требуется использовать две буферные таблицы:

- для хранения пар OBJ_ID, OBJ_TYPE, полученных из OBJECTS;
- для хранения значений HISTORY_ID, соответствующих строкам из OBJECT_HISTORY, удовлетворяющим условию принадлежности к объектам из первой буферной таблицы.

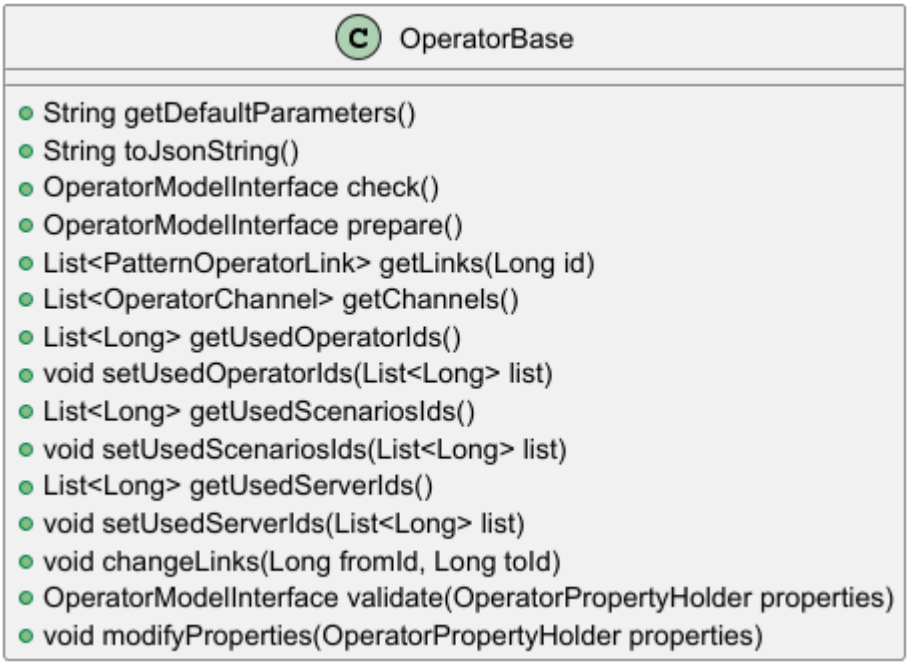
Таким образом, обеспечивается каскадное распространение признака подлежащих удалению сущностей по всей иерархии, без необходимости выполнять многократные вложенные запросы в процессе выполнения сценария. Буферные таблицы особенно полезны в сценариях с отложенной или поэтапной обработкой, в которых важна согласованность между таблицами, а также в случаях, когда очистка производится по нестандартным условиям или с применением специфических стратегий удаления.

2. Разработка и реализация решения по очистке и архивированию данных

2.1. Реализация конфигурации очистки и интеграция в редактор сценариев

Для расширения функциональности текущей реализации были дополнены ключевые операторы: «Подготовка задачи ЖЦ», «Очистка» и «Архивирование». В рамках этих операторов реализована поддержка работы с секционированными таблицами. Одновременно с этим были добавлены новые операторы-параметры, не обладающие собственной логикой, но предназначенные для визуальной настройки параметров других операторов. Такими компонентами стали «Таблица DLM» и «Дочерняя таблица», которые обеспечивают задание структуры таблиц, настройку связей между родительскими и дочерними таблицами, а также определение сущностей, подлежащих очистке. Эти расширения позволили построить архитектуру, пригодную для работы с различными типами таблиц, включая одиночные, секционированные и таблицы как секции.

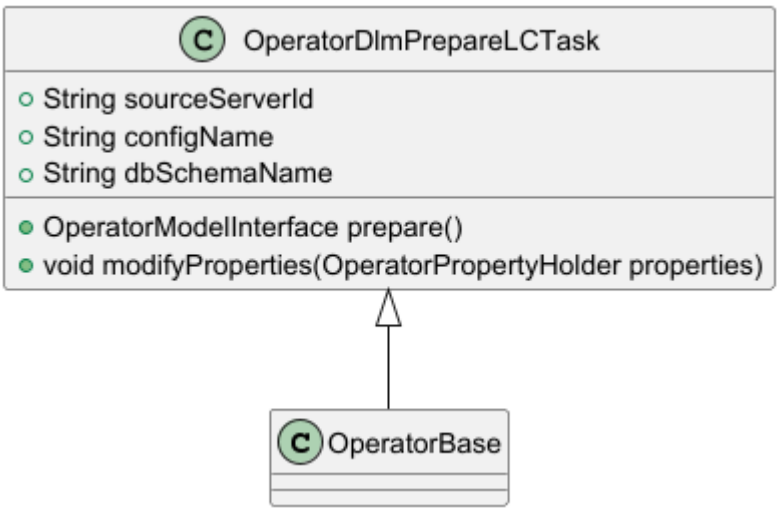
Структура операторов реализована на основе объектной модели, в которой все классы-операторы предназначены для интеграции с пользовательским интерфейсом и наследуются от базового класса OperatorBase. Данный базовый класс обеспечивает необходимый набор методов и интерфейсов, позволяющих включать операторы в общую систему конфигурации и исполнения сценариев. На диаграмме 1 представлена структура этого класса и его ключевых компонентов.



структура класса OperatorBase

Настройка конфигурации очистки начинается с оператора «Подготовка задачи ЖЦ». В общих свойствах данного оператора указывается подключение к целевой базе данных. На вкладке параметров подготовки задачи задаются два основных элемента конфигурации: наименование задачи очистки и имя схемы базы данных, в рамках которой будет производиться очистка. При этом непосредственно информация о таблицах, участвующих в процессе, больше не указывается в этом операторе. Вместо этого сведения о таблицах, их структуре, периодах хранения и иерархических связях описываются в специализированных операторах «Таблица DLM» и «Дочерняя таблица».

Оператор «Подготовка задачи ЖЦ» реализован в классе OperatorDlmPrepareLCTask, который наследуется от базового класса OperatorBase. Основной метод, переопределяемый в данном классе, — modifyProperties. Этот метод отвечает за формирование и модификацию набора параметров, доступных пользователю при настройке оператора в графическом интерфейсе. В частности, именно через modifyProperties добавляются такие параметры, как выбор сервера базы данных и указание имени схемы, в пределах которой будут определяться объекты для очистки. Структура класса имеет вид, показанный на диаграмме 1.



Структура OperatorDlmPrepareLCTask

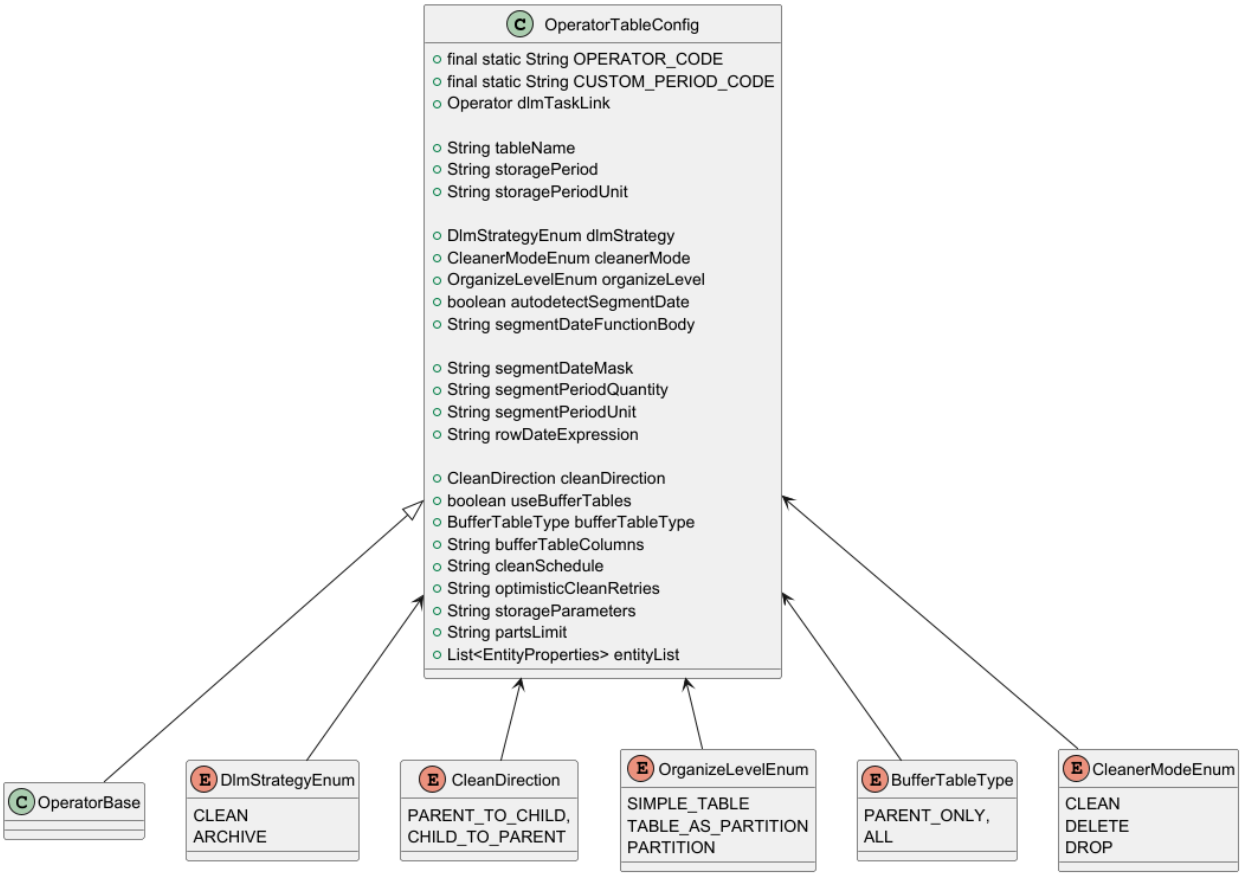
Оператор «Таблица DLM», представляющий собой основной механизм задания параметров очистки конкретной таблицы. Данный оператор включает в себя ряд обязательных параметров, без которых невозможно выполнение корректной процедуры очистки. Ключевой особенностью то, что правила, заданные для родительской таблицы, автоматически применяются ко всем связанным дочерним таблицам. При этом из дочерних таблиц удаляются только те записи, которые соответствуют очищаемым сущностям в родительской таблице согласно установленным связям. К основным параметрам, настраиваемым в операторе «Таблица DLM», относятся:

- Имя очищаемой таблицы;
- Период хранения — интервал времени, в течение которого данные считаются актуальными и подлежащими хранению. После истечения данного периода информация признается устаревшей;
- Стратегия очистки — определяет поведение системы в отношении устаревших данных. Возможны два подхода:
    - очистка (удаление без сохранения);
    - архивирование и очистка (предусматривает сохранение удаляемых данных в архивной структуре);
- Режим очистки — характеризует метод удаления данных:
    - удаление сегмента;

    - удаление пересозданием таблицы;
    - удаление построчно
- Структура таблицы — описывает физическую реализацию таблицы в СУБД, а также параметры очистки:
    - секции
    - таблица как секция
    - простая таблица
- Автоматическое вычисление границы секции — представляет собой механизм определения порогов хранения в секционированной таблице. В зависимости от конфигурации, границы могут рассчитываться автоматически либо быть заданы с использованием пользовательской функции;
- Направление очистки — определяет порядок удаления данных в иерархии таблиц:
    - от родительской к дочерним таблицам;
    - от дочерних к родительским;
- Использование буферных таблиц — опциональный параметр, активируемый при очистке в направлении от дочерних к родительским таблицам. Позволяет минимизировать блокировки и повысить надежность операций удаления;
- Список сущностей — набор записей с указанием:
    - названия сущности (идентификатора),
    - периода хранения (отличного от общего периода таблицы, при необходимости),
    - фильтрации по дополнительным критериям (например, по статусу или типу данных)

Структура класса, реализующего оператор «Таблица DLM», представлена в классе OperatorTableConfig, изображённом на диаграмме 1. Данный класс не содержит переопределений методов базового класса OperatorBase, вместо реализации собственной логики оператор служит

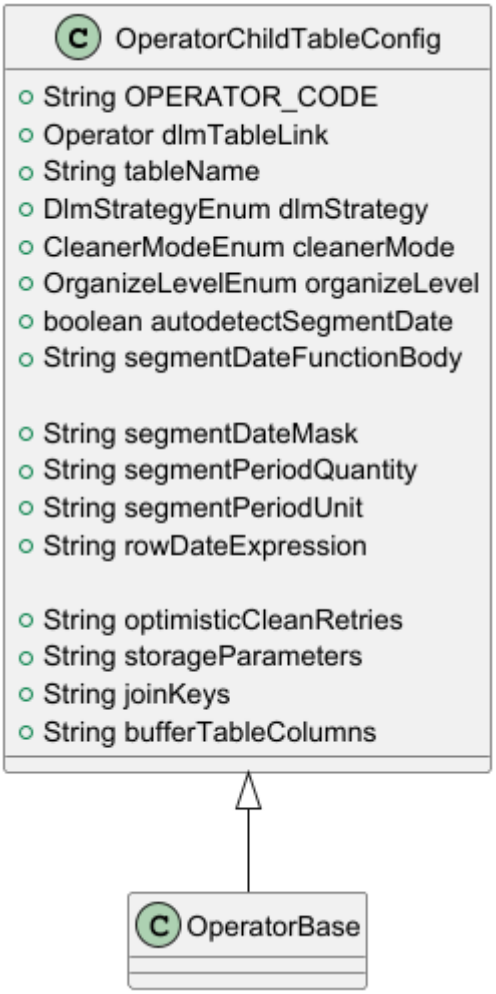
параметрическим контейнером, обеспечивая визуальную и программную конфигурацию в составе общего сценария очистки. Внутри класса определено поле dlmTaskLink, представляющее собой ссылку на другой оператор — OperatorDlmPrepareLCTask. Эта ссылка используется для установления связи с основной задачей очистки, обеспечивая как логическую связанность конфигурации, так и её визуальное отображение в интерфейсе проектирования. Такое решение позволяет формировать структурированную иерархию операторов, где каждый компонент конфигурации связан с соответствующим этапом выполнения сценария.



Структура класса OperatorTableConfig

Особое внимание следует уделить оператору «Дочерняя таблица», который используется при наличии иерархической связи между таблицами в базе данных. Структура данного оператора реализована в классе OperatorChildTableConfig и по своей архитектуре схожа с классом OperatorTableConfig, однако имеет ряд принципиальных отличий, связанных с особенностями конфигурации дочерних таблиц:

- параметр «Ключи объединения таблиц» обязателен для заполнения и определяет связь между дочерней и родительской таблицей, как правило, через внешний ключ;
- отсутствует параметр «Период хранения», так как этот период задается в родительской таблице;
- отсутствует список сущностей, поскольку сущности определяются исключительно на уровне родительской структуры.



Структура класса OperatorChildTableConfig

Операторы «Очистка» и «Архивирование», реализованные в классах OperatorDlmDataCleaning и OperatorDlmDataArchiving соответственно, остались практически без изменений в части интерфейсного поведения. Они продолжают выполнять свои функции в рамках сценария жизненного цикла данных, сохраняя прежний набор параметров и взаимодействие с другими

операторами. В частности, оба класса включают параметры подключения к базе данных и указание имени конфигурации очистки, что необходимо для корректной инициализации сценария, а OperatorDlmDataArchiving сохраняет выходной канал, предназначенный для передачи данных, подлежащих архивации. Диаграмма классов, представленных в виде OperatorDlmDataCleaning и OperatorDlmDataArchiving, приведена на диаграмме 1.

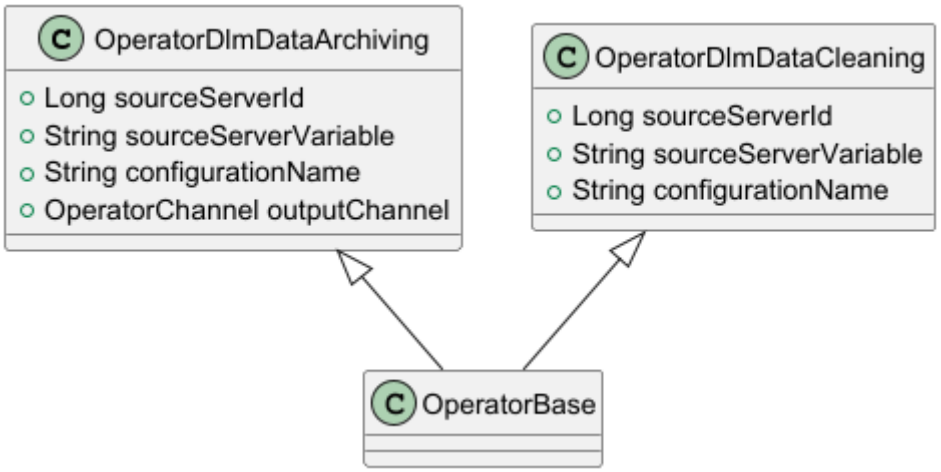


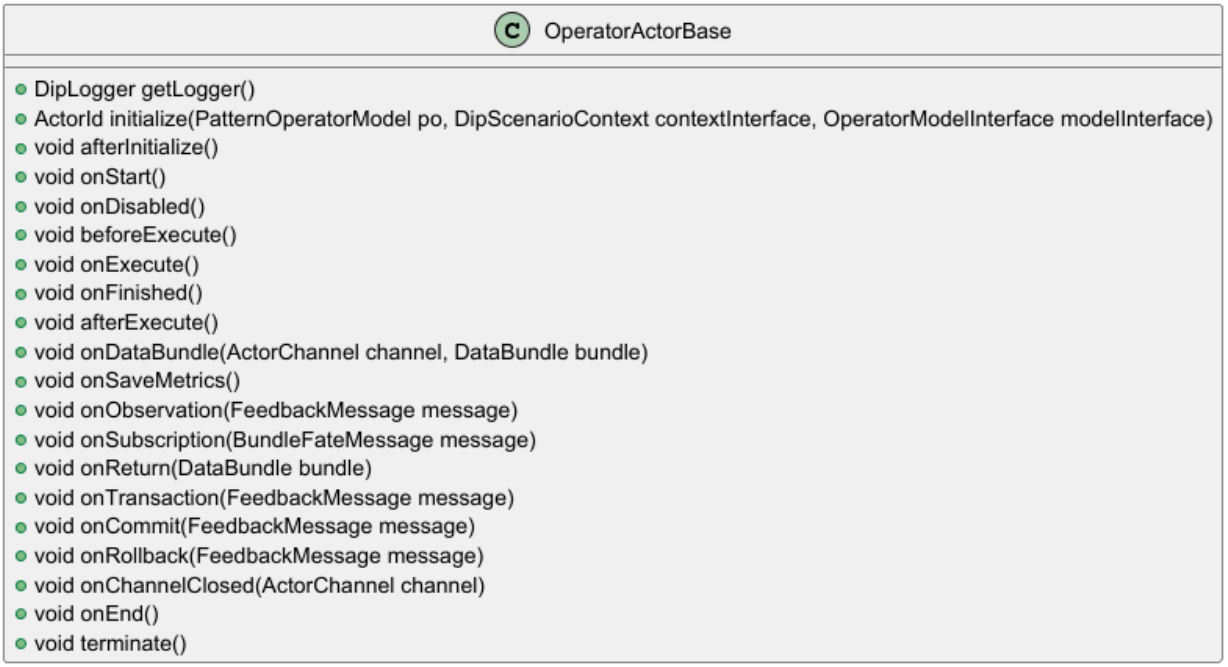
Диаграмма классов OperatorDlmDataCleaning и OperatorDlmDataCleaning

Таким образом, при корректной конфигурации вышеописанных операторов формируется связная система, где каждая задача очистки полностью определена: от структуры данных до методов их удаления и расписания выполнения. Вся совокупность параметров передается в механизм исполнения, где реализуется логика создания задач, очистки и архивирования.

2.2. Реализация логики подготовки задач, очистки и архивирования

Структура акторов, реализующих исполняемую единицу логики в системе, построена на наследовании от базового класса ActorOperatorBase. Этот класс обеспечивает подключение к жизненному циклу оператора и предоставляет набор методов, вызываемых на различных этапах его выполнения. В частности, метод onStart используется при запуске оператора, а метод onCommit — при завершении транзакции базы данных. Кроме того,

каждый актор содержит ссылку на соответствующий ему оператор, что позволяет синхронизировать состояние пользовательского интерфейса с выполняемой логикой и обеспечивает целостность взаимодействия между визуальной конфигурацией и исполняемым процессом. На диаграмме 1 представлена структура этого класса.



структура класса OperatorActorBase

Класс ActorDlmPrepareLCTask реализует актор оператора «Подготовка задачи ЖЦ». Класс наследуется от класса OperatorActorBase. Основное назначение класса — организация выполнения этапов подготовки конфигурации для последующей очистки и/или архивирования данных в таблицах на основе параметров, заданных оператором OperatorDlmPrepareLCTask. Актор обеспечивает полную обработку жизненного цикла подготовки задачи: от инициализации и валидации до создания задания и регистрации метрик.

При запуске в методе onStart() осуществляется инициализация ключевых компонентов, включая получение исходного сервера и проверку прав доступа, и загрузку соответствующего представления задачи.

Следующей частью подготовки является валидация конфигурации: проверяется наличие имени, схемы и хотя бы одной таблицы, а также

корректность описания каждой таблицы в списке. Невыполнение этих условий приводит к остановке выполнения с информативным сообщением об ошибке.

После инициализации и валидации, в методе onExecute() производится подключение к базе данных. Подключение повторяется до успешного выполнения, что позволяет избежать ошибок при временных сбоях соединения. Далее создаётся объект-композитор CompositorDlmTask, отвечающий за сборку и подготовку задачи DLM. Генерируется объект DlmTaskModel, содержащий описание всех подзадач. Если задача с таким именем уже активна, выполнение прекращается, чтобы не создавать дублирующих задач. В противном случае задача сохраняется через TasksService, а старые помечаются как неактуальные.

Метод onSaveMetrics() занимается регистрацией метрик по каждой подзадаче, позволяя в дальнейшем отслеживать выполнение задач по очистке и архивированию. Метки метрик включают идентификатор сервера, имя конфигурации, схему и имя таблицы.

Методы afterExecute() и onEnd() предназначены для завершения выполнения: в частности, производится безопасное закрытие соединения с базой данных.

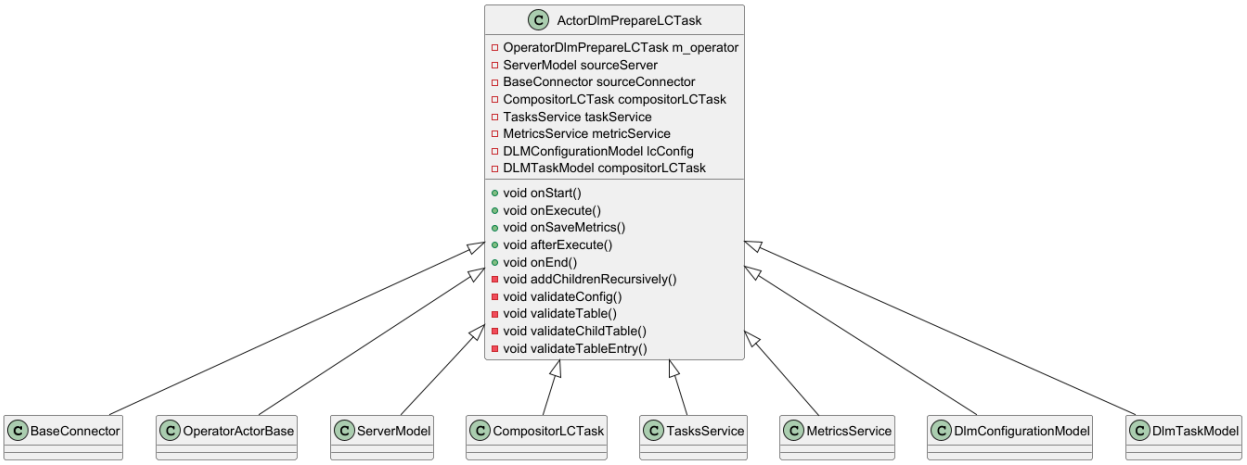


Рис. Диаграмма класса ActorDlmPrepareLCTask

Класс ActorDlmDataArchiving реализует актор обработки задач архивации данных. Основной задачей данного класса является поэтапное

выполнение архивных подзадач, каждая из которых связана с конкретной таблицей, секцией или подсекцией базы данных. Архивация осуществляется путём выборки данных по SQL-запросу и отправки их в выходной канал в виде структурированных пакетов данных.

Класс наследуется от OperatorActorBase. Основной логикой управления обладает метод onExecute, в котором последовательно обрабатываются подзадачи. Для каждой подзадачи формируется SQL-запрос в зависимости от уровня вложенности (таблица, секция, подсекция), после чего выполняется его исполнение. Чтение полученных данных производится построчно, с их последующей упаковкой и отправкой в выходной канал.

Внутри класса реализована система учёта статистик и метрик, сохранени метрик происходит в методе onSaveMetrics(). Также в классе поддерживается логирование событий выполнения с различным уровнем детализации, включая SQL-запросы, размер прочитанных пакетов и сообщения об ошибках.

Дополнительно реализована обработка обратной связи через методы onCommit и onRollback, позволяющие реагировать на подтверждение или отказ приёма пакетов с данными, что обеспечивает гарантию доставки и целостность архивации. В случае ошибок соответствующая подзадача переводится в состояние «неуспешно завершена», и метрики обновляются с учётом сбоя.

Реализация класса предусматривает механизм прерывания задач в случае остановки выполнения, а также корректное закрытие открытых ресурсов: соединений, курсоров и подготовленных выражений. Это гарантирует отсутствие утечек ресурсов при завершении работы, включая аварийные сценарии.

Диаграмма класса приведена в диаграмме 1.

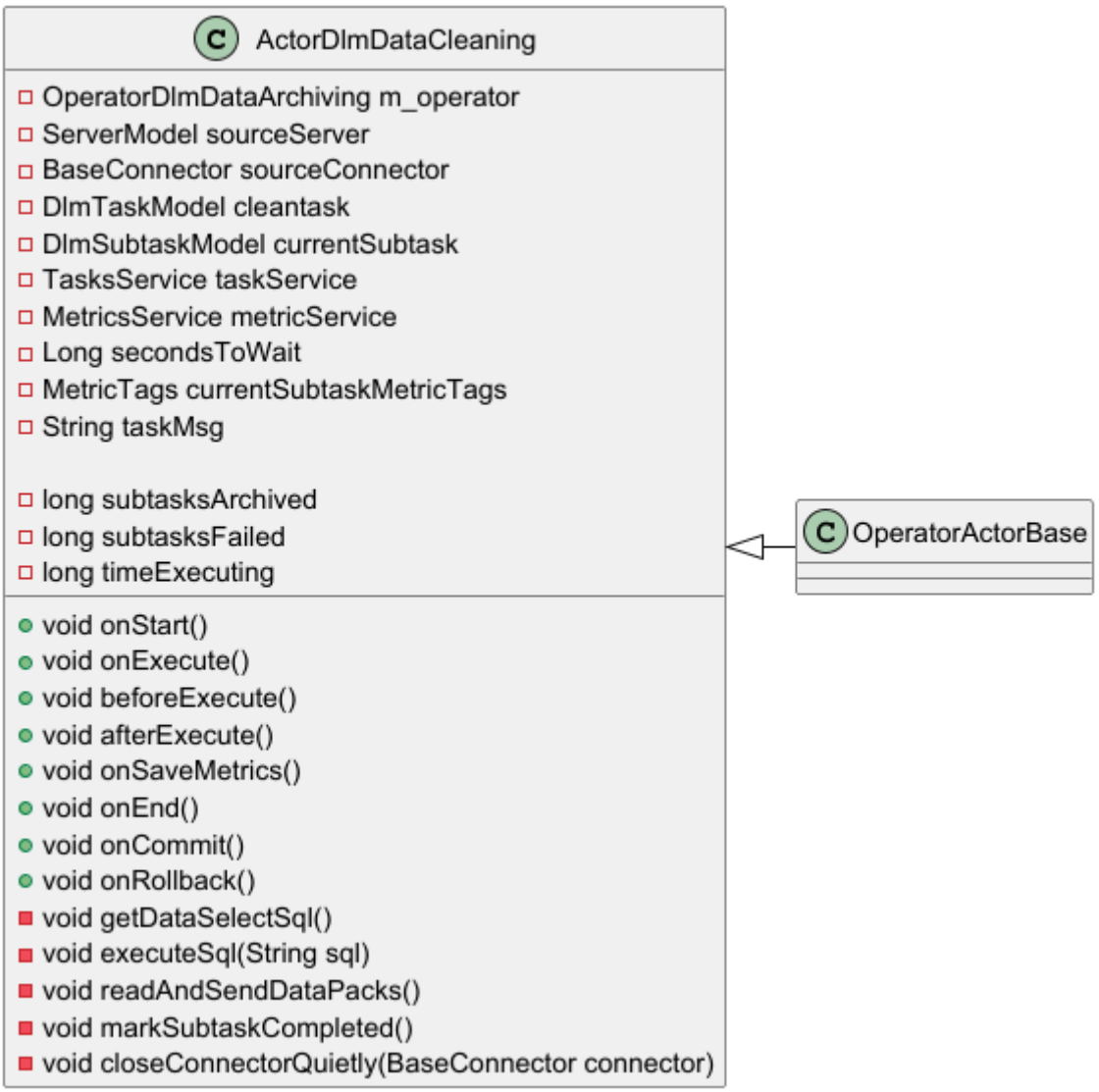


диаграмма ActorDlmDataArchiving

Класс ActorDlmDataCleaning реализует актор, отвечающий за выполнение задач очистки. Он наследуется от класса OperatorActorBase и во многом похож на ActorDlmDataArchiving, но этот актор обрабатывает задачи, связанные с удалением устаревших или более неактуальных данных из таблиц базы данных, в том числе секционированных. В реализации предусмотрены все этапы жизненного цикла задачи очистки: от инициализации до финализации и сбора метрик.

В методе onStart происходит начальная инициализация: проверяются права доступа к серверу, загружаются параметры конфигурации задачи и задается сообщение о начале выполнения. Также происходит валидация корректности параметров, в частности, задержки перед запуском очистки.

Метод onExecute содержит основную логику выполнения задачи очистки. В самом начале проверяется, требуется ли задержка перед очисткой, например, в случае, если до этого происходило архивирование данных или она указана эксплицитно. Если задержка необходима, выполнение временно приостанавливается. После этого начинается выполнение задачи очистки: статус задачи обновляется, инициализируется исполнитель ExecutorDlmCleaning, и производится последовательная обработка подзадач.

Каждая подзадача представляет собой очистку либо всей таблицы, либо её секции или подсекции, в зависимости от уровня вложенности. Для каждой подзадачи логируются ее параметры, фиксируются начальные метрики (например, предполагаемое количество строк перед очисткой), и запускается метод очистки.

Обработка подзадачи реализована в методе executeSubtask. Здесь выполняется пошаговая логика: установка статуса подзадачи, запуск подготовки, выполнение очистки в транзакции, фиксация успешного результата или откат при ошибке. В случае успеха обновляются метрики и статус подзадачи, в случае ошибки — происходит логирование сбоя, обновление метрик и установка статуса неудачи.

Отдельно реализована проверка на необходимость повторной попытки очистки, если определенные условия не были выполнены с первого раза. После выполнения последней подзадачи происходит финальное обновление статуса задачи. Если все подзадачи были выполнены успешно, задача считается завершенной.

Методы beforeExecute и afterExecute выполняют логгирование для отладки и подготовки к выполнению. Метод onSaveMetrics сохраняет значения метрик, связанных с успешными и неуспешными подзадачами, а также количеством очищенных строк. В методе onEnd производится корректное завершение работы и закрытие соединения с базой данных.

диаграмма ActorDlmDataCleaning

2.3. Реализация логики создания задач и очистки данных

Класс CompositorDlmTask реализует основную логику, связанную с созданием задач по очистке и архивированию данных в секционированных таблицах. Он играет ключевую роль в решении, поскольку именно здесь происходит интерпретация конфигурации, переданной из пользовательского интерфейса и формирование заданий, которые далее передаются в оператор очистки и/или архивации.

Основной метод класса — runPrepare. С его помощью на основании структуры и параметров, описанных в объекте конфигурации жизненного цикла, подготавливаются задачи на очистку и архивацию данных. Метод возвращает объект DlmTaskModel, включающий список подзадач по каждой

таблице, схеме и сегменту данных, участвующих в процессе. В рамках подготовки производится разбор конфигурации, включающей таблицы различных типов. Для каждой из категорий применяется отдельная логика обработки.

Для простых таблиц используется метод getDeleteTask, формирующий фильтрационное условие, по которому будет производиться очистка. Условие строится на базе пороговой даты хранения, вычисляемой как текущая дата, усечённая до суток, за вычетом значения периода хранения. Это позволяет определить «границу хранения» — временной момент, после которого данные считаются устаревшими. Дополнительно применяется логика исключения, позволяющая не удалять данные, если они всё ещё актуальны из-за наличия фильтра сущности. Конкретное SQL-выражение, формирующееся для фильтрации строк, представлено в приложении (см. Приложение А, Листинг 1).

В случае, когда таблицы в конфигурации описаны как «таблицы как секции», применяется другая стратегия. Здесь рассматриваются наборы таблиц, имитирующих поведение секционирования, например, с помощью шаблонных имен и определённой структуры в именовании. Для получения информации о таких таблицах используется SQL-запрос, приведённый в приложении (Приложение А, Листинг 2). Этот запрос позволяет определить, какие таблицы в схеме можно интерпретировать как секции по имени, извлекает информацию о границах дат (нижней и верхней), а также о схеме и имени таблицы. Таким образом, обеспечивается возможность динамически обнаруживать и обрабатывать такие таблицы без ручного задания списка.

Для полноценно секционированных таблиц — то есть тех, у которых в базе данных явно определена структура секций с помощью механизма pg_partitions — реализуется более сложная логика. Используется общий запрос, позволяющий извлечь полную информацию о структуре наследования, способе секционирования (по диапазону, списку или хешу), колонках, на которых осуществляется разбиение, типах данных, а также датах

начала и окончания для каждой секции. Этот запрос построен на основе представлений pg_inherits, pg_class, pg_partitioned_table и information_schema.columns. Он позволяет получить все секции заданной родительской таблицы, включая вложенные уровни. Запрос исключает секции, содержащие MAXVALUE, так как они предназначены для хранения «открытых» диапазонов и не подлежат удалению (Приложение А, Листинг 3).

После получения списка таблиц или секций в зависимости от типа, все они проходят этап валидации. На этом этапе проверяется, корректно ли определены ключевые параметры: имя таблицы, имя схемы, наличие ключа секционирования и границ хранения. Далее каждая таблица преобразуется во внутреннюю модель, позволяющую унифицировано обрабатывать все типы структур в следующих шагах жизненного цикла. Если в процессе подготовки определено, что потребуется создание буферных таблиц, они также создаются в рамках метода runPrepare.

Итогом выполнения runPrepare становится объект задачи, включающий полное описание всех необходимых действий по каждой из таблиц: имя таблицы и схемы, тип операции (очистка, архивация, комбинированный подход), метод удаления (по строкам, сегментам или пересоздание таблицы), способ идентификации устаревших данных, параметры секционирования и фильтрации. Все подзадачи агрегируются в единую структуру и передаются на следующий этап обработки — регистрацию задачи и её дальнейшее исполнение.

Схема классов CompositorLCTask, DlmTaskModel и DlmSubtaskModel

2.4. Тестирование и валидация разработанного решения

Для проверки корректности работы решения были разработаны тестовые сценарии, охватывающие различные способы организации данных: простые таблицы, таблицы, используемые как секции, и полноценные секционированные таблицы.

1. Очистка простых таблиц

Проверялась возможность удаления данных из связанных таблиц с использованием фильтров по дате и статусу. Тестовые таблицы имели следующую структуру:

- Родительская таблица содержала поля id, status и created_at.
- Дочерняя таблица включала id, status и ссылку на родительскую запись (parent_id).

Ожидаемый результат:

- Удаление всех записей старше 3 месяцев.
- Удаление записей старше 2 месяцев со статусом, отличным от 1.

2. Очистка таблиц как секций

Проверялась работа с таблицами как секций по датам. Структура тестовых таблиц:

- Родительская таблица с именем, включающим год и месяц (parent_table_YYYMM), содержала поля id и status.
- Дочерняя таблица (child_table_YYYMMDD) включала id и ссылку на родительскую запись (parent_id).

Ожидаемый результат:

- Удаление всех записей старше 3 месяцев.
- Удаление записей старше 2 месяцев со статусом, отличным от 1.

3. Очистка секционированных таблиц

Проверялась работа с таблицами, явно секционированными по диапазону дат. Структура тестовых таблиц:

- Родительская таблица содержала поля id, partition_low_date и status, с секционированием по partition_low_date.
- Дочерняя таблица также была секционирована по partition_low_date и включала id и ссылку на родительскую запись (parent_id).

Ожидаемый результат:

- Удаление всех записей старше 3 месяцев.

- Удаление записей старше 2 месяцев со статусом, отличным от 1.

Для генерации тестовых данных использовались SQL-запросы, приведённые в приложении.

В ходе тестирования подтверждена корректность работы решения для всех рассмотренных сценариев. Очистка данных выполнялась в соответствии с заданными условиями, включая фильтрацию по дате и статусу. Механизмы удаления, пересоздания и детача партиций функционировали без ошибок, обеспечивая целостность данных и минимальное время блокировки таблиц. Решение продемонстрировало устойчивость к различным конфигурациям данных и готово к использованию в промышленной среде.

ЗАКЛЮЧЕНИЕ

В ходе работы было разработано решение для автоматизированной очистки и архивирования данных в секционированных таблицах PostgreSQL, которое позволяет эффективно управлять жизненным циклом данных, снижая эксплуатационные затраты и обеспечивая согласованность данных. Решение поддерживает различные типы таблиц: простые, секционированные и таблицы как секции, что делает его универсальным инструментом для работы с разнородными структурами данных.

Проведенный анализ существующих подходов к управлению данными в секционированных таблицах, позволил выявить ключевые проблемы текущих решений. Были рассмотрены традиционные методы, такие как ручное удаление записей и использование временных таблиц, а также современные механизмы декларативного секционирования. В результате анализа определены ограничения, включая несогласованность данных после очистки, сложность работы с многоуровневыми иерархиями секций и отсутствие гибкости в настройке. Эти выводы легли в основу требований к разрабатываемому решению.

В работе были сформированы требования к решению. Основными критериями стали обеспечение согласованности данных, поддержка сложных связей между таблицами и гибкость настройки. Особое внимание уделено механизмам обработки зависимостей между таблицами, включая каскадное удаление и использование буферных таблиц для фиксации идентификаторов сущностей. Эти требования позволили создать архитектуру, способную работать с разнородными структурами данных и минимизировать влияние на производительность системы.

Была разработана реализация механизмов очистки и архивирования. Решение было интегрировано в существующую платформу DIP, расширено новыми операторами и акторами, такими как «Подготовка задачи ЖЦ», «Таблица DLM» и «Дочерняя таблица».

Четвертая задача посвящена тестированию и валидации решения. Проверка проводилась на всех видах таблиц, которые требовалось очищать исходя из сформированных требований к решению. Во всех сценариях подтверждена корректность работы механизмов очистки. Решение продемонстрировало устойчивость к различным конфигурациям данных и готовность к промышленному использованию.

Таким образом, поставленная цель достигнута: разработано решение, которое эффективно управляет жизненным циклом данных в секционированных таблицах PostgreSQL, сочетая гибкость, производительность и надежность. Дальнейшее развитие проекта может включать расширение функциональности для поддержки других СУБД и интеграцию с системами мониторинга для более детального анализа процессов очистки и архивирования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ